

# README.md

## The OpenVoiceOS Technical Manual

!

the OVOS project documentation is written and maintained by users just like you!

Think of these docs both as your starting point and also forever changing and incomplete

Please [open Issues and Pull Requests](#)!

User oriented docs are automatically published at  
<https://openvoiceos.github.io/community-docs>

Dev oriented docs (this repo) are automatically published at  
<https://openvoiceos.github.io/ovos-technical-manual>

## docs/001-release\_channels.md

## OVOS Release Channels & Installation Options

Open Voice OS (OVOS) is a **modular voice assistant platform** that lets you install only the components you need. Whether you're building a lightweight voice interface or a full-featured smart assistant, OVOS gives you flexibility through modular packages and optional feature sets called **extras**.

To manage updates and ensure system stability, OVOS uses **release channels** and **constraints files**, allowing users to pin versions based on their desired stability level.

---

## Choosing a Release Channel

OVOS follows **[\\*\\*semantic versioning\\*\\*](#)** (SemVer) with a **rolling release model** and

supports three release channels **stable**, **testing**, and **alpha** so you can pick the right balance between cutting-edge features and system reliability.

These channels are managed via the [constraints files](#) hosted in the [ovos-releases](#) repository

## 1. Stable Channel (Production-Ready)

Bug fixes only

No new features or breaking changes

Recommended for production or everyday use

```
pip install ovos-core[mycroft] -c https://raw.githubusercontent.com/OpenVoiceOS/ovos-releases
```

## 2. Testing Channel (Feature Updates)

Bug fixes and new features

Not as thoroughly tested as stable

Best for early adopters or development environments

```
pip install ovos-core[mycroft] -c https://raw.githubusercontent.com/OpenVoiceOS/ovos-releases
```

## 3. Alpha Channel (Bleeding Edge)

Experimental features

May include breaking changes

Not suitable for production use

```
pip install ovos-core[mycroft] --pre -c https://raw.githubusercontent.com/OpenVoiceOS/ovos-releases
```

> `constraints.txt` files act like version "filters". They don't install packages directly, but ensure only approved versions get installed.

---

# OVOS From Scratch: Custom Installation

Rather than using a full distro, you can manually pick which components to install:

`'ovos-messagebus'` internal messaging between services

`'ovos-core'` skill handling

`'ovos-audio'` text-to-speech (TTS), audio playback

`'ovos-dinkum-listener'` wake word, voice activation

`'ovos-gui'` GUI integration

`'ovos-PHAL'` hardware abstraction layer

This is useful if you're building something like a **Hivemind node** or **headless device**, where you might not need audio output or a GUI.

---

## What Are OVOS Extras?

OVOS uses Python extras (e.g., `[mymicroft]`) to let you install predefined groups of components based on your use case.

| Extra Name       | Purpose  |
|------------------|--|
|                  |  |
| -----            | -----  |
| -----            |  |
| mycroft          | Core services for full voice assistant experience      |
| lgpl             | Adds optional LGPL-licensed tools like Padatious       |
| plugins          | Includes various plugin interfaces                     |
| skills-essential | Must-have skills (like system control, clock, weather) |
| skills-audio     | Audio I/O-based skills                                 |
| skills-gui       | GUI-dependent skills                                   |
| skills-internet  | Skills that require an internet connection             |
| skills-media     | OCP (OpenVoiceOS Common Play) media playback skills    |
| skills-desktop   | Desktop environment integrations                       |
|                  |  |

## Full Installation Example

```
pip install ovos-core[mycroft,lgpl,plugins,skills-essential,skills-audio,skills-gui,skills
```

## Minimal Installation Example

```
pip install ovos-core[mycroft,plugins,skills-essential]
```

---

## Technical Notes

OVOS originally began as a fork of `mycroft-core`. Since version **0.0.8**, it has been **fully modularized**, with each major service in its own repository.

All packages follow [Semantic Versioning \(SemVer\)](#), so you can rely on versioning to understand stability and compatibility.

Constraints files are a **work in progress** and wont be finalized until the first official [codename release](#).

---

## Tips & Caveats

Using `--pre` installs pre-releases across all dependencies, not just OVOS-specific ones so use with caution.

You can mix and match extras based on your hardware or use case, e.g., omit GUI skills on a headless server.

When using constraints files, make sure all packages are pinned it avoids installing incompatible versions.

After installing you need to launch the individual ovos services, either manually or by creating a systemd service

---

## See Also

[OVOS Releases repo](#)

[Constraints files explanation \(pip docs\)](#)

[Semantic Versioning](#)

[OVOS Component Repos](#)

## docs/002-license.md

### License

We have a universal donor policy, our code should be able to be used anywhere by anyone, no ifs or conditions attached.

OVOS is predominately Apache2 or BSD licensed. There are only a few exceptions to this, which are all licensed under other compatible open source licenses.

Individual plugins or skills may have their own license, for example mimic3 is AGPL, so we can not change the license of our plugin.

We are committed to maintain all core components fully free, any code that we have no control over the license will live in an optional plugin and be flagged as such.

This includes avoiding LGPL code for reasons explained [here](#).

Our license policy has the following properties:

It gives you, the user of the software, complete and unrestrained access to the software, such that you may inspect, modify, and redistribute your changes

Inspection - Anyone may inspect the software for security vulnerabilities

Modification - Anyone may modify the software to fix issues or add features

Redistribution - Anyone may redistribute the software on their terms

It is compatible with GPL licenses - Projects licensed as GPL can be distributed with OVOS

It allows for the incorporation of GPL-incompatible free software, such as software that is CDDL licensed

The license does not restrict the software that may run on OVOS, however -- and thanks to the plugin architecture, even traditionally tightly-coupled components such as drivers can be distributed separately, so maintainers are free to choose whatever license they like for their projects.

## Notable licensing exceptions

The following repositories do not respect our universal donor policy, please ensure their licenses are compatible before you use them

| Repository                                   | License            | Reason   |
|--|--------------------|--|
| <a href="#">ovos-intent-plugin-padatious</a> | Apache2.0          | <a href="#">padatious</a> license might not be valid, depends on libfann2 (LGPL) |
| <a href="#">ovos-tts-plugin-mimic3</a>       | AGPL               | depends on <a href="#">mimic3</a> (AGPL)   |
| <a href="#">ovos-tts-plugin-espeakng</a>     | GPL                | depends on <a href="#">espeak-ng</a> (GPL)                                       |
| <a href="#">ovos-g2p-plugin-espeak</a>       | GPL                | depends on <a href="#">espeak-phonemizer</a> (GPL)                               |
| <a href="#">ovos-tts-plugin-SAM</a>          |                    | ?  |
|  | reverse engineered | abandonware  |

## docs/003-timeline.md

# Family Tree

!imagem

mycroft-core created

neon-core forked from mycroft-core

chatterbox forked from mycroft-core (closed source)

mycroft-lib forked from mycroft-core to become a library, it is only a properly packaged mycroft-core

chatterbox rewritten to use mycroft-lib internally, no longer a hard fork

neon-core rewritten to use mycroft-lib internally, no longer a hard fork

mycroft-lib renamed to HolmesIV to avoid trademark issues

HolmesV created to use HolmesIV internally, HolmesV gets features and HolmesIV bug fixes only

chatterbox updated to HolmesV

neon-core updated to HolmesV

HolmesIV+HolmesV abandoned, chatterbox focus on closed source product

ovos-core forked from HolmesV

neon-core updated to ovos-core

mycroft-dinkum forked from mycroft-core

## Events timeline

Aug 2015 - [MycroftAI](#) [kickstarter](#) launch

Feb 2016 - MycroftAI [Mimic](#) TTS released

May 2016 - [mycroft-core](#) repositories made public under GPL

Jun 2016 - @Aix releases [MycroftAI](#) [gnome shell](#) extension

Jun 2016 - @Aix becomes a contributor on the desktop-integration team

Aug 2016 - Steve Penrod becomes MycroftAI CTO

??? 2016 - NeonGecko begins working with Mycroft AI

Jan 2017 - @Aix release [MycroftAI](#) [plasmoid](#) for KDE Plasma desktop

Mar 2017 - @Aix incubates Mycroft plasmoid project under KDE

Apr 2017 - @JarbasAI forks `mycroft-core` as `jarbas-core`

May 2017 - @JarbasAI becomes a `mycroft-core` contributor

Sep 2017 - MycroftAI Mark 1 released / mycroft kickstarter campaign fulfilled

Sep 2017 - @Aix joins Blue Systems to work on MycroftAI and Plasma integration projects

Sep 2017 - `jarbas-core` tacotron TTS integration added

Aug 2017 - MycroftAI starts supporting KDE plasmoid project and installers

Oct 2017 - `mycroft-core` relicensed as Apache 2.0

Nov 2017 - @Aix becomes a `mycroft-core` contributor

Dec 2017 - `jarbas-core` abandoned

??? 2017 - Neon AI forks `mycroft-core` as `neon-core` to remove dependency on MycroftAI servers and work on other features

Jan 2018 - initial release of [personal-backend](#), reverse engineered MycroftAI backend by @JarbasAI, licensed as Apache 2.0

Jan 2018 - MycroftAI [mark 2 kickstarter](#) launch

Jul 2018 - `personal-backend` added to MycroftAI Roadmap

Aug 2018 - MycroftAI Mimic2 TTS based on tacotron released

Sep 2018 - [Mycroft-GUI](#) was created by collaboration between MycroftAI, Blue Systems based on KDE frameworks, maintained by @Aix

Oct 2018 - @j1nx creates "[MycroftOS](#)", the first version of what will eventually become the OpenVoiceOS smart speaker

Jan 2019 - @JarbasAI [personal-backend](#) implementation adopted by MycroftAI

Mar 2019 - MycroftAI mark 2 completely redesigned and based on different hardware

Apr 2019 - Steve Penrod and @JarbasAI create [lingua-franca](#) under MycroftAI with @JarbasAI as maintainer

??? 2019 - @JarbasAI partners with NeonAI to maintain `neon-core`

Jul 2019 - steve penrod leaves MycroftAI



Jul 2019 - [Chatterbox kickstarter](#) launched by @JarbasAI

Sep 2019 - [Mycroft on Plasma Automobile Demo](#) at Akademy 2019 by @aix

Oct 2019 - Official MycroftAI backend open sourced, licensed as AGPL

Dec 2019 - @Chance joins `lingua-franca` as a maintainer

Dec 2019 - Chatterbox released (closed source) / kickstarter campaign fulfilled

Dec 2019 - MycroftAI starts to work on Kivy based UI

??? 2020 - MycroftAI starts neglecting community contributions

Mar 2020 - MycroftAI abandons [personal-backend](#)

Mar 2020 - Michael Lewis becomes MycroftAI CEO

May 2020 - @JarbasAI and @AIX partner up to create a [VOIP skill](#)

Sep 2020 - Community takes stand against reimplementing of GUI using Kivy

Oct 2020 - Kivy has been dropped in support for QT Mycroft-GUI

Oct 2020 - @JarbasAI, @AIX and @j1nx form OpenVoiceOS community project around `mycroft-core`

Oct 2020 - [ovos-utils](#) transferred from @JarbasAI to OVOS and renamed from `jarbas-utils`

??? 2020 - Chatterbox forks `mycroft-core` as `mycroft-lib` (open source)

??? 2020 - `mycroft-lib` reorganizes `mycroft` imports cleanly separating each service

??? 2020 - Chatterbox (closed source) recreated on top of `mycroft-lib`

??? 2020 - `neon-core` recreated on top of `mycroft-lib`

Oct 2020 - MycroftOS renamed to OpenVoiceOS - Mycroft Edition due to trademark issues

??? 2020 - @JarbasAI leaves `lingua-franca`

??? 2020 - @Chance joins OpenVoiceOS

Nov 2020 - @Jarbas gives a talk about [translating mycroft](#) for [collectivats](#)  
Jornades de tecnologies linguistiques lliures en catala

Dec 2020 - [ovos-ww-plugin-pocketsphinx](#) released

Dec 2020 - [ovos-ww-plugin-snowboy](#) released

Dec 2020 - [ovos-ww-plugin-precise](#) released

Dec 2020 - [ovos-stt-plugin-vosk](#) released

Dec 2020 - [ovos-stt-plugin-chromium](#) released

Jan 2021 - [ovos-plugin-manager](#) released

Jan 2021 - [personal-backend](#) adopted by OpenVoiceOS, original repo unarchived  
and ownership transferred

Jan 2021 - [Mycroft embedded shell](#) adopted by OpenVoiceOS and renamed to  
[ovos-shell](#)

Jan 2021 - [skill-ovos-setup](#) forked from Mycroft to replace pairing

Jan 2021 - [ovos-skill-manager](#) released to support more skill Marketplaces

Feb 2021 - [skill-ovos-stop](#) forked from Mycroft

Mar 2021 - [skill-ovos-common-play](#) forked from Mycroft

Mar 2021 - MycroftAI mark 2 dev kits start shipping

Apr 2021 - OpenVoiceOS adopts `mycroft-lib` instead of `mycroft-core`

??? 202? - `mycroft-lib` renamed to [HolmesIV](#) to avoid trademark issues

Apr 2021 - [ovos-workshop](#) released, bringing OVOS features to individual skills

May 2021 - chatterbox forks `lingua-franca` as `lingua-nostra`

May 2021 . OpenVoiceOS deprecates `mycroft-lib` and adopts `HolmesIV`

Jun 2021 - [ovos-tts-plugin-espeakNG](#) released

Jun 2021 - [ovos-tts-plugin-mimic](#) released

Jun 2021 - [ovos-tts-plugin-mimic2](#) released

Jun 2021 - [ovos-tts-plugin-pico](#) released

Aug 2021 - [ovos-tts-plugin-google-tx](#) released

Aug 2021 - [ovos-ww-plugin-vosk](#) released

Aug 2021 - [precise-lite](#) forked from Mycroft, adding tflite support

Aug 2021 - [ovos-ww-plugin-precise-lite](#) released

Aug 2021 - [ovos-ww-plugin-nyumaya](#) released

Aug 2021 - [precise-lite-models](#) start being shared

Aug 2021 - [skill-ovos-volume](#) released

Sep 2021 - [VocalFusionDriver](#) released by OVOS to support the mk2

Sep 2021 - [ovos-tts-plugin-SAM](#) released

Sep 2021 - backend made optional in `HolmesIV`

Sep 2021 - msm made optional in `HolmesIV`

Oct 2021 - "instant\_listen" introduced in `HolmesIV`

Oct 2021 - `HolmesIV` abandoned by chatterbox

Oct 2021 - `lingua-nostra` abandoned by chatterbox

Oct 2021 - OpenVoiceOS forks `HolmesIV` as `ovos-core`

Oct 2021 - `ovos-core` becomes XDG compliant

Oct 2021 - `neon-core` deprecates `HolmesIV` and adopts `ovos-core`

Oct 2021 - [skill-ovos-common-play](#) deprecated in favor of [OCP](#)

??? 2021 - @Chance leaves `lingua-franca`

Nov 2021 - [ovos-plugin-manager](#) released

Nov 2021 - [skill-ovos-timer](#) forked from Mycroft

Nov 2021 - [skill-ovos-homescreen](#) forked from Mycroft

Nov 2021 - @JarbasAI leaves chatterbox

Nov 2021 - ovos-core version **0.0.1** released

Nov 2021 - [ovos-utils](#) adopted in ovos-core

Nov 2021 - [ovos-plugin-manager](#) adopted in ovos-core

Nov 2021 - multiple wake words support added

Nov 2021 - installable skills (setup.py) support added

Nov 2021 - [ovos-PHAL](#) released (mycroft compatible)

Nov 2021 - [skill-ovos-hello-world](#) forked from Mycroft

Nov 2021 - [skill-ovos-naptime](#) forked from Mycroft

??? - @NeonDaniel joins OpenVoiceOS

??? 2021 - NeonAI adopts [ovos-shell](#)

Dec 2021 - [ovos-PHAL-plugin-mk1](#) released

Dec 2021 - [skill-ovos-fallback-unknown](#) forked from Mycroft

Dec 2021 - [skill-ovos-weather](#) forked from Mycroft

Dec 2021 - [skill-ovos-common-query](#) forked from Mycroft

Dec 2021 - [skill-ovos-application-launcher](#) released

Jan 2022 - OpenVoiceOS forks [lingua-franca](#)

??? 2022 - OpenVoiceOS starts development in the open via [matrix chat](#)

Feb 2022 - [OCP](#) released as an audio plugin (mycroft compatible)

Feb 2022 - PHAL replaces `mycroft.client.enclosure` in ovos-core

Feb 2022 - [skill-ovos-date-time](#) forked from Mycroft

Mar 2022 - Fallback STT support added

Mar 2022 - VAD plugins support added

Mar 2022 - [ovos-vad-plugin-webrtcvad](#) released

Mar 2022 - [ovos-vad-plugin-silero](#) released

Mar 2022 - [OCP](#) adopted in `ovos-core` as default media handler

Mar 2022 - [ovos-PHAL-plugin-mk2](#) released

Mar 2022 - [ovos-PHAL-plugin-respeaker-2mic](#) released

Mar 2022 - [ovos-PHAL-plugin-respeaker-4mic](#) released

??? 2022 - OpenVoiceOS starts releasing buildroot images for rpi4 and Mark 2

??? 2022 - OpenVoiceOS starts releasing manjaro images for rpi4 and Mark 2

Apr 2022 - [ovos-stt-http-server](#) released

Apr 2022 - [ovos-stt-plugin-server](#) released

May 2022 - [ovos-tts-plugin-beepspeak](#) forked from chatterbox

May 2022 - [ovos-tts-plugin-marytts](#) released

May 2022 - [ovos-tts-plugin-polly](#) forked from chatterbox

May 2022 - [ovos-translate-server](#) released

May 2022 - `ovos-core` version **0.0.3** released

May 2022 - MycroftAI founders resign

Jun 2022 - [ovos-config](#) adopted in `ovos-core`

Jun 2022 - [skill-ovos-alarm](#) forked from Mycroft

Jun 2022 - [skill-ovos-qml-camera](#) forked from Mycroft

Jun 2022 - Plasma Bigscreen [drops mycroft-core in favor of ovos-core](#) version **0.0.4** (alpha)

Jul 2022 - MycroftAI starts work on [mycroft-dinkum](#) behind the scenes, a replacement/rewrite of mycroft-core

Jul 2022 - [ovos-tts-plugin-mimic3](#) forked from Mycroft

Jul 2022 - [skill-homescreen-lite](#) released

Aug 2022 - [padacioso](#) transferred from @JarbasAI to OVOS

Aug 2022 - adopt [padacioso](#) as a optional [padatious](#) alternative to avoid libfann (LGPL)

Aug 2022 - [ovos-core](#) version **0.0.4** released

Aug 2022 - experimental support for "continuous\_listening" and "hybrid\_listening" added

Sep 2022 - MycroftAI [Mimic3](#) TTS released, AGPL licensed

Sep 2022 . MycroftAI Mark 2 starts shipping

Sep 2022 - [skill-ovos-news](#) transferred from @JarbasAI to OVOS

Oct 2022 - [ovos-backend-manager](#) UI released

Oct 2022 - [ovos-stt-plugin-whispercpp](#) released

Oct 2022 - new [ask\\_yesno](#) parser added

Oct 2022 - [ovos-backend-client](#) adopted by [ovos-core](#), selectable backend support added (offline, personal, selene)

Oct 2022 - [ovos-tts-plugin-mimic3-server](#) released

Oct 2022 - [mycroft.blue-systems.com](#) mimic 3 public server added

Oct 2022 - [mimic3.ziggyai.online](#) mimic 3 public server added

Oct 2022 - @aix talks about [OVOS in KDE Academy](#)

Oct 2022 - [skill-ovos-soundcloud](#) transferred from @JarbasAI to OVOS

Oct 2022 - [skill-ovos-youtube](#) transferred from @JarbasAI to OVOS

??? 2022 - [mycroft-dinkum](#) source code made public, Apache2 licensed

??? 2022 - Extremely negative feedback from developer community, several key members announce no intention to support mycroft-dinkum

??? 2022 - NeonAI starts release NeonOS images for the Mark 2

Nov 2022 - `ovos-core` version **0.0.5** released

Nov 2022 - MycroftAI staff lay off, only a skeleton crew remaining

Nov 2022 - [ovos-bus-client](#) forked from Mycroft

Nov 2022 - [tts.smartgic.io/mimic3](https://tts.smartgic.io/mimic3) public Mimic3 server added

Dec 2022 - [Home Assistant PHAL](#) plugin initial release by @AIX , exposes HA devices to the messagebus

Dec 2022 - [skill-ovos-youtube-music](#) transferred from @JarbasAI to OVOS

Dec 2022 - [skill-ovos-bandcamp](#) transferred from @JarbasAI to OVOS

Jan 2023 - `ovos-core` version **0.0.6** released

Jan 2023 - negative user feedback from the community for Mark 2, lots of bugs and missing features

Jan 2023 - "fallback" wake word support added to `ovos-plugin-manager`

Jan 2023 - [skill-ovos-local-media](#) released

Jan 2023 - [ChatGPT skill](#) released

Feb 2023 - `ovos-config` cli tool by community member @sgee released

Feb 2023 - [ovos-solver-plugin-aiml](#) released

Feb 2023 - [ovos-solver-plugin-rivescript](#) released

Feb 2023 - [skill-ovos-somafm](#) transferred from @JarbasAI to OVOS

Feb 2023 - MycroftAI partners up with NeonAI to maintain mycroft-core(?)

Mar 2023 - Mimic3 TTS public servers become default OVOS voice (alan pope)

Mar 2023 - `ovos-core` version **0.0.7** released

Mar 2023 - [Fundraiser to form OpenVoiceOS](#) V.z.w. (Dutch: "Vereniging zonder winstoogmerk") completed in a couple days

Mar 2023 - First stretch goal of [fundraiser](#) reached and second stretch goal

announced

Mar 2023 - [raspbrian-ovos](#) images start being released, maintained by community member @builderjer

Mar 2023 - [community docs](#) start being maintained by community members

Mar 2023 - [ovos-ww-plugin-openWakeWord](#) released, maintained by author @dscripka

Mar 2023 - [skill-ovos-icanhazdadjokes](#) transferred from @JarbasAI to OVOS

Mar 2023 - [ovos-skill-alerts](#) forked from Neon, maintained by community member @sgee

Apr 2023 - [ovos-core](#) splits [ovos-audio](#), [ovos-listener](#), [ovos-gui](#) and [ovos-bus](#) into their own packages

Apr 2023 - [@Aix leaves](#) OpenVoiceOS

Apr 2023 - OpenVoiceOS stops releasing manjaro based images

Apr 2023 - [ovos-stt-plugin-fasterwhisper](#) released

Apr 2023 - [ovos-tts-plugin-piper](#) released

Apr 2023 - [precise-lite-trainer](#) released

Apr 2023 - [ovos-vad-plugin-precise](#) released

Apr 2023 - [ovos-dinkum-listener](#) released

Apr 2023 - [ovos-translate-plugin-deepl](#) released, maintained by community member @sgee

Apr 2023 - [mycroft-classic-listener](#) released, to preserve original mark 1 listener

Apr 2023 - [skill-ovos-tunein](#) transferred from @JarbasAI to OVOS, maintained by community member @sgee

Apr 2023 - [jurebes](#) intent parser released

May 2023 - [mycroft](#) import deprecated in favor of [ovos\\_core](#) module for skills service



May 2023 - [stt.openvoiceos.org](https://stt.openvoiceos.org) moves to whisper (small, cpu only)

May 2023 - [ovos-docker](#) released, maintained by community member @goldyfruit

May 2023 - Open Voice OS TTS/STT [status page](#) released, maintained by community member @goldyfruit

May 2023 - First successful run of OpenVoiceOS on Mac OS using containers with [ovos-docker](#)

May 2023 - [ovos-docker-stt](#) STT containers released, maintained by community member @goldyfruit

May 2023 - [ovos-microphone-plugin-sounddevice](#) released, which provides native Mac OS support, maintained by community member @goldyfruit

May 2023 - [ovos-persona](#) alpha release

May 2023 - [ovos-audio-transformer-plugin-speechbrain-langdetect](#) released

May 2023 - [ovos-skill-easter-eggs](#) transferred from @JarbasAI to OVOS, maintained by community member @mikejgray

May 2023 - [skill-ovos-dismissal](#) transferred from @ChanceNCounter to OVOS

May 2023 - [skill-ovos-dictation](#) transferred from @JarbasAI to OVOS

Jun 2023 - Home Assistant plugin starts being maintained by community member @mikejgray

Jun 2023 - [quebra\\_frases](#) transferred from @JarbasAI to OVOS

Jun 2023 - [ovos-translate-plugin-nllb](#) released

Jun 2023 - [fasterwhisper.ziggyai.online](https://fasterwhisper.ziggyai.online) public STT server added (large, GPU)

Jun 2023 - [Home Assistant Notify](#) integration released by community member @mikejgray

Jun 2023 - First (and second!) successful run of OpenVoiceOS on Windows, using WSL2 and [ovos-docker](#)

Jun 2023 - [ovos-docker-tts](#) TTS containers released, maintained by community member @goldyfruit

Jun 2023 - [ovos-tts-plugin-azure](#) released

Jun 2023 - [ovos-utterance-corrections-plugin](#) released

Jul 2023 - [mycroft-gui-qt6](#) forked from mycroft-gui at last commit supporting QT6 before license change to GPL (reverted shortly after)

Jul 2023 - [mycroft-gui-qt5](#) forked from mycroft-gui at last commit supporting QT5

Jul 2023 - [pipertts.ziggyai.online](#) public TTS server added

Jul 2023 - [tts.smartgic.io/piper](#) public TTS server added

Jul 2023 - piper TTS public servers become default OVOS voice (alan pope)

Jul 2023 - [skill-ovos-spotify](#) port of the mycroft-spotify skill by community member and original author @forslund

Aug 2023 - [ovos-translate-server-plugin](#) released

Aug 2023 - [ovos-docker-tx](#) translation containers released, maintained by community member @goldyfruit

Aug 2023 - [nllb.openvoiceos.org](#) public translation server added

Aug 2023 - [translator.smartgic.io/nllb](#) public translation server added

Aug 2023 - adopt NLLB public servers as default translation plugin

Aug 2023 - [skill-ovos-wolfie](#) transferred from @JarbasAI to OVOS

Aug 2023 - [skill-ovos-ddg](#) transferred from @JarbasAI to OVOS

Aug 2023 - [skill-ovos-wikipedia](#) transferred from @JarbasAI to OVOS

Aug 2023 - [ovos-stt-azure-plugin](#) released

Sep 2023 - [skill-ovos-parrot](#) transferred from @JarbasAI to OVOS

Sep 2023 - [stt.smartgic.io/fasterwhisper](#) public STT server (large, GPU)

Sep 2023 - GUI fully functional with [ovos-docker](#) containers

Sep 2023 - [persona-server](#) alpha version released

Sep 2023 - [ovos-audio-transformer-plugin-ggwave](#) released

Oct 2023 - [ovosnllb.ziggyai.online](#) public translation server added

Oct 2023 - [ovos-tts-plugin-mimic3-server](#) deprecated

Oct 2023 - [ovos-PHAL-sensors](#) released, exposing OVOS sensors in Home Assistant

Oct 2023 - [ovos-bidirectional-translation-plugin](#) released

Nov 2023 - Plasma Bigscreen [moves to QT6 and explicitly drops support for OVOS](#)

Dec 2023 - [ovos-installer](#) first release! codename Duke Nukem, maintained by community member @goldyfruit

Dec 2023 - `ovos-logs` cli tool by community member @sgee added to `ovos-utils`

Dec 2023 - [ovos-docs-viewer](#) cli tool released

Dec 2023 - [skill-ovos-spelling](#) forked from Mycroft

Dec 2023 - [skill-ovos-ip](#) forked from Mycroft

Dec 2023 - [skill-ovos-wallpapers](#) transferred to OVOS

Dec 2023 - [ovos-i2csound](#) released by community member @builderjer

??? 202? - [ovos-tts-plugin-mimic2](#) deprecated

Jan 2024 - [skill-ovos-boot-finished](#) forked from Neon

Jan 2024 - [skill-ovos-audio-recording](#) forked from Neon

Jan 2024 - [ovos-utterance-plugin-cancel](#) forked from Neon, deprecates dismissal skill

Jan 2024 - [ovos-mark1-utils](#) released

Jan 2024 - Mycroft forums move to [Open Conversational AI](#)

Jan 2024 - [ovos-vad-plugin-noise](#) released to support older platforms

Feb 2024 - [ovos-tts-plugin-edge-tts](#) released

Feb 2024 - Selene servers and Mycroft AI website [go down](#)

Feb 2024 - [skill-ovos-randomness](#) released, maintained by community member @mikejgray

Feb 2024 - [OVOSHatchery](#) created to incubate new projects

Feb 2024 - @Chance leaves OpenVoiceOS

Feb 2024 - [skill-ovos-wordnet](#) released

Mar 2024 - Community Mycroft [skills updated to OVOS](#) under OVOSHatchery

Mar 2024 - [OVOS Skill Store](#) released!

Mar 2024 - [Hatchery Skill Store](#) released!

Mar 2024 - First successful run of OpenVoiceOS natively on Mac OS with Apple Silicon

Mar 2024 - [ovos-installer](#) second release, codename `Doom` supports Mark II device, maintained by community member @goldyfruit

Apr 2024 - [Mark 2 demo](#) running Open Voice OS connected to a local A.I.

Apr 2024 - First successful run of [HiveMind Satellite](#) on Mark 1 device

Jun 2024 - [First successful run of OpenVoiceOS natively on Windows:](#)  
<https://drive.google.com/file/d/171801mbhbpG79BvIOIUCxVyMPcDGgnbM/view?usp=sharing>

Near Future - `ovos-core` version **0.0.8** released

## docs/100-bus\_service.md

### Bus Service

The **Message Bus** is the internal communication layer used by OVOS to allow independent components to interact using structured messages. It acts as a central nervous system, coordinating everything from speech recognition to skill execution.

---

# Overview

In the OVOS ecosystem, the messagebus is implemented as a **WebSocket** interface. Messages follow a structured format containing:

- A `type`: identifying the intent or action

- An optional `data` payload (JSON)

- An optional `context` dictionary for session or routing metadata

Some messages trigger actions; others act as notifications or state broadcasts.

Both core OVOS components and external systems (e.g., [HiveMind](#)) can interact with the bus.

---

# Configuration

The messagebus is configured in `mycroft.conf` under the `websocket` section:

```
"websocket": {  
  "host": "127.0.0.1",  
  "port": 8181,  
  "route": "/core",  
  "shared_connection": true  
}
```

> It is **strongly recommended** to keep the `host` set to `127.0.0.1` to prevent unauthorized remote access.

---

# Security

By default, all skills share a single bus connection. This can be exploited by malicious or poorly designed skills to interfere with others.

- To improve isolation, set `"shared_connection": false`. This ensures each skill uses a separate WebSocket connection.

- For a demonstration of potential vulnerabilities, see [BusBrickerSkill](#).

Security concerns are further documented in [Nhoya/MycroftAI-RCE](#).

> Never expose the messagebus to the public internet. It provides full control over the OVOS instance and the host system.

> For remote interaction, use [HiveMind](#), which offers secure proxy access to the bus.

---

## Message Structure

Each message sent on the bus consists of:

```
{
  "type": "message.type",
  "data": { /* arbitrary JSON payload */ },
  "context": { /* optional metadata */ }
}
```

`type`: Identifies the message (e.g., "recognizer\_loop:utterance")

`data`: Carries command-specific information

`context`: Session and routing information, used internally

For a complete index of known OVOS messages, refer to the [Message Spec documentation](#).

---

## Sessions

Messages can carry a "session" key inside their `context` to preserve request-specific state and user preferences. Sessions help enable:

Multi-user support

Conversational context

Remote device handling

Example session fields include:

Language and location

TTS/STT preferences

Active skills and follow-up intents

Pipeline settings

Site or device ID

Sessions are typically auto-managed by `ovos-core` for local interactions using the default session ID (`"default"`). External clients (e.g., HiveMind voice satellites) are expected to manage their own sessions.

See the [Session-Aware Skills](#) documentation for implementation guidelines.

> Skills that are not session-aware may behave unexpectedly when used with external voice clients.

---

## Message Targeting and Routing

OVOS uses `context["source"]` and `context["destination"]` to enable smart message routing across components and external devices.

The `Message` object includes utility methods:

`.forward()`: Sends the message onward while preserving the current context

`.reply()`: Sends a response back to the original source (swapping source/destination)

Example:

```
bus.emit(Message('recognizer_loop:utterance', data,
    context={
        'destination': ['audio', 'kde'],
        'source': 'remote_service'
    }))
```

OVOS itself does not implement any actual routing, everything connected to the ovos messagebus receives every message, however this metadata enables 3rd party applications to fully manage these messages and decide if and where to send them

> [HiveMind](#) uses these fields extensively to direct replies to the correct satellite.

---

## Internal Routing Overview

[!OVOS Message Flow Diagram](#)

The Intent Service replies directly to utterance messages.

Skill and intent interactions use `.forward()` to retain context.

Skills sending their own events should manually manage routing.

## docs/101-speech\_service.md

### Listener Service

The listener service is responsible for handling audio input, it understands speech and converts it into `utterances` to be handled by `ovos-core`

Different implementations of the listener service have been available during the years

[mycroft-classic-listener](#) the original listener from mycroft

mark1 extracted into a standalone component - **archived**

[ovos-listener](#) - an updated version of the mycroft listener with VAD

plugins and multiple hotwords support - **deprecated** in `ovos-core` version **0.0.8**

[ovos-dinkum-listener](#) - a listener rewrite based

on [mycroft-dinkum](#) - **NEW** in `ovos-core` version **0.0.8**

### Listener

You can modify microphone settings and enable additional features under the listener section such as wake word / utterance recording / uploading



```

"listener": {
  // NOTE, multiple hotwords are supported, these fields define the main wake_word,
  // this is equivalent to setting "active": true in the "hotwords" section
  "wake_word": "hey_mycroft",
  "stand_up_word": "wake_up",

  "microphone": {...},

  "VAD": {...},

  // Seconds of speech before voice command has begun
  "speech_begin": 0.1,
  // Seconds of silence before a voice command has finished
  "silence_end": 0.5,
  // Settings used by microphone to set recording timeout with and without speech detected
  "recording_timeout": 10.0,
  // Settings used by microphone to set recording timeout without speech detected.
  "recording_timeout_with_silence": 3.0,

  // Setting to remove all silence/noise from start and end of recorded speech (only non-s
  "remove_silence": true
}

```

## Microphone

**NEW** in ovos-core version **0.0.8**

Microphone plugins are responsible for feeding audio to the listener, different Operating Systems may require different plugins or otherwise have performance benefits

```

"listener": {
  "microphone": {
    "module": "ovos-microphone-plugin-alsa"
  }
}

```

## Hotwords

By default the listener is waiting for a hotword to do something in response

the most common usage of a hotword is as the assistant's name, instead of continuously transcribing audio the listener waits for a wake word, and then listens to the user speaking

OVOS allows you to load any number of hot words in parallel and trigger different actions when they are detected

each hotword can do one or more of the following:

- trigger listening, also called a **wake\_word**

- play a sound

- emit a bus event

- take ovos-core out of sleep mode, also called a **wakeup\_word** or **standup\_word**

- take ovos-core out of recording mode, also called a **stop\_word**

To add a new hotword add its configuration under "hotwords" section.

By default, all hotwords are disabled unless you set `"active": true`.

Under the `"listener"` setting a main wake word and stand up word are defined, those will be automatically enabled unless you set `"active": false`.

Users are expected to **only change** `listener.wake_word` if using a single wake word, setting `"active": true` is only intended for **extra** hotwords

```

"listener": {
    // Default wake_word and stand_up_word will be automatically set to active
    // unless explicitly disabled under "hotwords" section
    "wake_word": "hey mycroft",
    "stand_up_word": "wake up"
},
// Hotword configurations
"hotwords": {
    "hey_mycroft": {
        "module": "ovos-ww-plugin-precise-lite",
        "model": "https://github.com/OpenVoiceOS/precise-lite-models/raw/master/wakewords/",
        "expected_duration": 3,
        "trigger_level": 3,
        "sensitivity": 0.5,
        "listen": true
    },
    // default wakeup word to take ovos out of SLEEPING mode,
    "wake_up": {
        "module": "ovos-ww-plugin-pocketsphinx",
        "phonemes": "W EY K . AH P",
        "threshold": 1e-20,
        "lang": "en-us",
        "wakeup": true,
        "fallback_ww": "wake_up_vosk"
    }
}

```

## Sound Classifiers

hotwords can be used as generic sound classifiers that emit bus events for other systems to detect

Let's consider a model trained to recognize coughing, and a companion plugin to track how often it happens, this can be used as an indicator of disease

```

"hotwords": {
  "cough": {
    "module": "ovos-ww-plugin-precise",
    "version": "0.3",
    "model": "https://github.com/MycroftAI/precise-data/blob/models-dev/cough.tar.gz",
    "expected_duration": 3,
    "trigger_level": 3,
    "sensitivity": 0.5,
    "listen": false,
    "active": true,
    // on detection emit this msg_type
    "bus_event": "cough.detected"
  }
}

```

## Multilingualism

In multilingual homes a wake word can be configured for each language, by giving the assistant a different name in each we can assign a language to be used by STT

```

"listener": {
  "wake_word": "hey mycroft"
},
"hotwords": {
  // default wake word, in global language
  "hey_mycroft": {...},
  // extra wake word with lang assigned
  "android": {
    "module": "...",
    "model": "...",
    // set to active as extra wake word
    "active": true,
    "listen": true,
    // assign a language
    "stt_lang": "pt-pt"
  }
}

```

# Fallback Wake Words

**NEW** in ovos-core version **0.0.8**

hotword definitions can also include a "fallback\_ww", this indicates an alternative hotword config to load in case the original failed to load for any reason

```

    "threshold": 1e-90,
    "listener": {
        "lang": "en-us"
        // Default wake_word and stand_up_word will be automatically set to active
        "listen": true
        // unless explicitly disabled under "hotwords" section
    }
    "wake_word": "hey mycroft",
    // default wakeup word to take ovos out of SLEEPING mode,
    "stand_up_word": "wake up"
},
    "wake_up": {
        "module": "ovos-ww-plugin-pocketsphinx",
        // Hotword configurations
        "phonemes": "W EY K . AH P",
    "hotwords": {
        "threshold": 1e-20,
        "hey_mycroft": {
            "lang": "en-us",
            "module": "ovos-ww-plugin-precise-lite",
            "wakeup": true,
            "model": "https://github.com/OpenVoiceOS/precise-lite-models/raw/master/wakewords/
            "fallback_ww": "wake_up_vosk"
            "expected_duration": 3,
        },
        "trigger_level": 3,
        // in case pocketsphinx plugin is not installed, attempt to use vosk
        "sensitivity": 0.5,
        "wake_up_vosk": {
            "listen": true,
            "module": "ovos-ww-plugin-vosk",
            "fallback_ww": "hey_mycroft_precise"
        },
        "rule": "fuzzy",
    },
    "samples": ["wake up"],
    // in case precise-lite is not installed, attempt to use classic precise
    "lang": "en-us"
    "hey_mycroft_precise": {
        // makes this a wakeup word for usage in SLEEPING mode
        "module": "ovos-ww-plugin-precise",
        "wakeup": true,
        "version": "0.3",
    }
    "model": "https://github.com/MycroftAI/precise-data/raw/models-dev/hey-mycroft.tar
}
    "expected_duration": 3,
    "trigger_level": 3,
    "sensitivity": 0.5,
    "listen": true,
    "fallback_ww": "hey_mycroft_vosk"
},
// in case classic precise is not installed, attempt to use vosk
"hey_mycroft_vosk": {
    "module": "ovos-ww-plugin-vosk",
    "samples": ["hey mycroft", "hey microsoft", "hey mike roft", "hey minecraft"],
    "rule": "fuzzy",
    "listen": true,
    "fallback_ww": "hey_mycroft_pocketsphinx"
},
// in case vosk is not installed, attempt to use pocketsphinx
"hey_mycroft_pocketsphinx": {
    "module": "ovos-ww-plugin-pocketsphinx",
    "phonemes": "HH EY . M AY K R AO F T",

```

# VAD

Voice Activity Detection plugins have several functions under the listener service

- detect when user finished speaking

- remove silence before sending audio to STT - **NEW** in ovos-core version **0.0.8**

- detect when user is speaking during continuous mode (read below)

```

"listener": {
  // Setting to remove all silence/noise from start and end of recorded speech (only non-s
  "remove_silence": true,

  "VAD": {
    // recommended plugin: "ovos-vad-plugin-silero"
    "module": "ovos-vad-plugin-silero",
    "ovos-vad-plugin-silero": {"threshold": 0.2},
    "ovos-vad-plugin-webrtcvad": {"vad_mode": 3}
  }
}

```

## STT

Two STT plugins may be loaded at once, if the primary plugin fails for some reason the second plugin will be used.

This allows you to have a lower accuracy offline model as fallback to account for internet outages, this ensures your device never becomes fully unusable

```

"stt": {
  "module": "ovos-stt-plugin-server",
  "fallback_module": "ovos-stt-plugin-vosk",
  "ovos-stt-plugin-server": {"url": "https://stt.openvoiceos.com/stt"}
},

```

## Audio Transformers

**NEW** in ovos-core version **0.0.8**, originally developed for [Neon](#)

Similarly to utterance transformers in ovos-core, the listener exposes audio and message.context to a set of plugins that can transform it before STT stage

Audio transformer plugins can either transform the audio binary data itself (eg, denoise) or the context (eg, speaker recognition)

The audio is sent sequentially to all transformer plugins, ordered by priority (developer defined), until finally it is sent to the STT stage

[!imagem](#)



# Modes of Operation

There are 3 modes to run dinkum, wakeword, hybrid, or continuous (VAD only)

Additionally, there are 2 temporary modes that can be triggered via bus events / companion skills

## Wake Word mode

[!imagem](#)

## Sleep mode

Can be used via [Naptime skill](#)

[!imagem](#)

Be sure to enable a wakeup word to get out of sleep!

```
"listener": {
  "stand_up_word": "wake up"
},
"hotwords": {
  "wake up": {
    "module": "ovos-ww-plugin-pocketsphinx",
    "phonemes": "W EY K . AH P",
    "threshold": 1e-20,
    "lang": "en-us",
    "wakeup": true
  }
}
```

## Continuous mode

**EXPERIMENTAL - NEW** in ovos-core version **0.0.8**

[!imagem](#)

```
"listener": {  
  // continuous listen is an experimental setting, it removes the need for  
  // wake words and uses VAD only, a streaming STT is strongly recommended  
  // NOTE: depending on hardware this may cause mycroft to hear its own TTS responses as q  
  "continuous_listen": false  
}
```

## Hybrid mode

**EXPERIMENTAL - NEW** in ovos-core version **0.0.8**

[!imagem](#)

```
"listener": {  
  // hybrid listen is an experimental setting,  
  // it will not require a wake word for X seconds after a user interaction  
  // this means you dont need to say "hey mycroft" for follow up questions  
  "hybrid_listen": false,  
  // number of seconds to wait for an interaction before requiring wake word again  
  "listen_timeout": 45  
}
```

## Recording mode

**EXPERIMENTAL - NEW** in ovos-core version **0.0.8**

Can be used via [Recording skill](#)

[!imagem](#)

# docs/102-core.md

## OVOS-CORE

[OpenVoiceOS](#) is an open source platform for smart speakers and other voice-centric devices.

OpenVoiceOS is fully modular. Furthermore, common components have been repackaged as plugins. That means it isn't just a great assistant on its own, but also a

pretty small library!

`ovos-core` contains "the brains" of OpenVoiceOS, all the NLP components and skills are managed here

## Skills Service

The skills service is responsible for loading skills and intent parsers

All user queries are handled by the skills service, you can think of it as OVOS's brain

All Mycroft Skills should work normally with `ovos-core` until version **0.1.0**, after that modernization is required!

Under OpenVoiceOS skills are regular python packages, any installed skills will be loaded automatically by `ovos-core`

Since `ovos-core` **0.0.8** it is also possible to launch a skill standalone via `ovos-workshop`, this enables individual skill containers in [ovos-docker](#)

This can be also be helpful during skill development for quick testing before the skill is packaged

```
ovos-skill-launcher {skill_id} [path/to/my/skill_id]
```

## Configuration

```

"skills": {

    // blacklisted skills to not load
    // NB: This is the skill_id, usually the basename() of the directory where the skill lives
    // the skill you want to blacklist is in /usr/share/mycroft/skills/mycroft-alarm.mycroft
    // then you should write `["mycroft-alarm.mycroftai"]` below.
    "blacklisted_skills": [],

    // fallback skill configuration (see below)
    "fallbacks": {...},

    // converse stage configuration (see below)
    "converse": {...}

},

```

## Utterance Transformers

**NEW** in `ovos-core` version **0.0.8**, originally developed for [Neon](#)

when `ovos-core` receives a natural language query/ utterance from a user it is sent to a "preprocessing stage"

The utterance transformers framework consists of any number of plugins ordered by priority (developer defined), the `utterance` and `message.context` are sent sequentially to all transformer plugins, and can be mutated by any of those plugins

[!imagem](#)

to enable a utterance transformer simply add it to `mycroft.conf` after installing it

```
// To enable a utterance transformer plugin just add it's name with any relevant config
// these plugins can mutate the utterance between STT and the Intent stage
// they may also modify message.context with metadata
// plugins only load if they are installed and enabled in this section
"utterance_transformers": {

  "ovos-utterance-normalizer": {},

  // cancel utterances mid command
  "ovos-utterance-plugin-cancel": {},

  // define utterance fixes via fuzzy match ~/.local/share/mycroft/corrections.json
  // define unconditional replacements at word level ~/.local/share/mycroft/word_correction
  "ovos-utterance-corrections-plugin": {},

  // translation plugin
  "ovos-utterance-translation-plugin": {
    "bidirectional": true,
    "verify_lang": false,
    "ignore_invalid": true,
    "translate_secondary_langs": false
  }
},
```

## Metadata Transformers

**NEW** in ovos-core version **0.0.8**

Similar to utterance transformers, these plugins only transform the `message.context`

```
// To enable a metadata transformer plugin just add it's name with any relevant config
// these plugins can mutate the message.context between STT and the Intent stage
"metadata_transformers": {},
```

## Intent Pipelines

**NEW** in ovos-core version **0.0.8**

after the utterance has been transformed it is sent to various OVOS components

by priority order until one can handle the query

Pipelines include intent parsers, converse framework, common query framework and fallback skill framework

```
// Intent Pipeline / plugins config
"intents" : {
  // the pipeline is a ordered set of frameworks to send an utterance too
  // if one of the frameworks fails the next one is used, until an answer is found
  // NOTE: if padatious is not installed, it will be replaced with padacioso (much slower)
  // in the future these will become plugins, and new pipeline stages can be added by en
  "pipeline": [
    "ocp_high",
    "stop_high",
    "converse",
    "padatious_high",
    "adapt_high",
    "fallback_high",
    "stop_medium",
    "adapt_medium",
    "ovos-persona-pipeline-plugin-high",
    "adapt_low",
    "common_qa",
    "fallback_medium",
    "ovos-persona-pipeline-plugin-low",
    "fallback_low"
  ]
},
```

## docs/103-audio\_service.md

# Audio Service

The audio service is responsible for handling TTS and simple sounds playback

## TTS

Two TTS plugins may be loaded at once, if the primary plugin fails for some

reason the second plugin will be used.

This allows you to have a lower quality offline voice as fallback to account for internet outages, this ensures your device can always give you feedback

```
"tts": {
    "pulse_duck": false,

    // plugins to load
    "module": "ovos-tts-plugin-server",
    "fallback_module": "ovos-tts-plugin-mimic",

    // individual plugin configs
    "ovos-tts-plugin-server": {
        "host": "https://tts.smartgic.io/piper",
        "v2": true,
        "verify_ssl": true,
        "tts_timeout": 5,
    }
}
```

## Skill Methods

skills can use `self.play_audio`, `self.acknowledge`, `self.speak` and `self.speak_dialog` methods to interact with ovos-audio

```
def play_audio(self, filename: str, instant: bool = False):
    """
    Queue and audio file for playback
    @param filename: File to play
    @param instant: if True audio will be played instantly
                    instead of queued with TTS
    """
```

```
def acknowledge(self):
    """
    Acknowledge a successful request.

    This method plays a sound to acknowledge a request that does not
    require a verbal response. This is intended to provide simple feedback
    to the user that their request was handled successfully.
    """
```

```
def speak(self, utterance: str, expect_response: bool = False, wait: Union[bool, int] = False):
    """Speak a sentence.

    Args:
        utterance (str): sentence mycroft should speak
        expect_response (bool): set to True if Mycroft should listen
                                for a response immediately after
                                speaking the utterance.
        wait (Union[bool, int]): set to True to block while the text
                                is being spoken for 15 seconds. Alternatively, set
                                to an integer to specify a timeout in seconds.
    """
```

```
def speak_dialog(self, key: str, data: Optional[dict] = None,
                  expect_response: bool = False, wait: Union[bool, int] = False):
    """
    Speak a random sentence from a dialog file.

    Args:
        key (str): dialog file key (e.g. "hello" to speak from the file
                  "locale/en-us/hello.dialog")
        data (dict): information used to populate sentence
        expect_response (bool): set to True if Mycroft should listen
                                for a response immediately after
                                speaking the utterance.
        wait (Union[bool, int]): set to True to block while the text
                                is being spoken for 15 seconds. Alternatively, set
                                to an integer to specify a timeout in seconds.
    """
```



to play sounds via bus messages emit `"mycroft.audio.play_sound"` or `"mycroft.audio.queue"` with data `{ "uri": "path/sound.mp3" }`

## PlaybackThread

`ovos-audio` implements a queue for sounds, any OVOS component can queue a sound for playback.

Usually only TTS speech is queue for playback, but sounds effects may also be queued for richer experiences, for example in a story telling skill

The `PlaybackThread` ensures sounds don't play over each other but instead sequentially, listening might be triggered after TTS finishes playing if requested in the `"speak"` message

shorts sounds can be played outside the `PlaybackThread`, usually when instant feedback is required, such as in the listening sound or on error sounds

You can configure default sounds and the playback commands under `mycroft.conf`

```
// File locations of sounds to play for default events
"sounds": {
  "start_listening": "snd/start_listening.wav",
  "end_listening": "snd/end_listening.wav",
  "acknowledge": "snd/acknowledge.mp3",
  "error": "snd/error.mp3"
},

// Mechanism used to play WAV audio files
// by default ovos-utils will try to detect best player
"play_wav_cmdline": "paplay %1 --stream-name=mycroft-voice",

// Mechanism used to play MP3 audio files
// by default ovos-utils will try to detect best player
"play_mp3_cmdline": "mpg123 %1",

// Mechanism used to play OGG audio files
// by default ovos-utils will try to detect best player
"play_ogg_cmdline": "ogg123 -q %1",
```

> NOTE: by default the playback commands are not set and OVOS will try to

determine the best way to play a sound automatically

## Transformer Plugins

**NEW** in ovos-core version **0.0.8**

Similarly to audio transformers in ovos-dinkum-listener, the utterance and audio data generated by TTS are exposed to a set of plugins that can transform them before playback

[!imagem](#)

## Dialog Transformers

Similarly to utterance transformers in core, ovos-audio exposes utterance and message.context to a set of plugins that can transform it before TTS stage

The utterance to be spoken is sent sequentially to all transformer plugins, ordered by priority (developer defined), until finally it is sent to the TTS stage

To enable a transformer add it to mycroft.conf

```
// To enable a dialog transformer plugin just add it's name with any relevant config
// these plugins can mutate utterances before TTS
"dialog_transformers": {
  "ovos-dialog-translation-plugin": {},
  "ovos-dialog-transformer-openai-plugin": {
    "rewrite_prompt": "rewrite the text as if you were explaining it to a 5 year old"
  }
}
```

## TTS Transformers

The audio to be spoken is sent sequentially to all transformer plugins, ordered by priority (developer defined), until finally it played back to the user

> **NOTE:** Does not work with StreamingTTS

To enable a transformer add it to mycroft.conf

```
// To enable a tts transformer plugin just add it's name with any relevant config
// these plugins can mutate audio after TTS
"tts_transformers": {
  "ovos-tts-transformer-sox-plugin": {
    "default_effects": {
      "speed": {"factor": 1.1}
    }
  }
}
```

## docs/104-gui\_service.md

# GUI Service

OVOS devices with displays provide skill developers the opportunity to create skills that can be empowered by both voice and screen interaction.

`ovos-gui`, aka, The GUI Service, is responsible for keeping track of what should be rendered, but does not perform the rendering itself

The GUI service provides a websocket for gui clients to connect to, it is responsible for implementing the gui protocol under `ovos-core`.

You can find indepth documentation of [the GUI protocol](#) in the dedicated GUI section of these docs

## Architecture

The GUI state is defined by `namespaces`, usually corresponding to a `skill_id`, each with any number of `pages`.

users are expected to be able to "swipe left" and "swipe right" to switch between `pages` within a `namespace`

OpenVoiceOS components interact with the GUI by defining session data and active pages, gui-clients may also send back `events` to indicate interactions.

[!imagem](#)

`pages` are ordered and, usually, only 1 `page` is rendered at a time.

If the screen size allows it platform specific gui client applications are free

to render all `pages` into view.

The GUI clients may be implemented in any programming language, the default page templates provided to skills via [GUIInterface](#) should be implemented and provided by all alternative clients.

> **QML tip:** - set `fillWidth: true` under `Mycroft.Delegate` to have a page always using the full screen

## Active Namespaces

In the context of a smartspeaker, when the GUI is idle a `homescreen` may be displayed, eg. an animated face or clock

[!imagem](#)

Whenever a `page` is displayed by a skill, `ovos-gui` tracks it and sets it's `namespace` to active, then tells the gui clients to render it.

The active `namespace` and how long a page stays up are managed by `ovos-gui`, usually via platform specific plugins.

`ovos-gui` will decide when a `namespace` is no longer active, and then the next `namespace` will be rendered,

Skills using the [GUIInterface](#) can indicate how long they want a page to remain active

Example:

OVOS is idle - `homescreen` is the active `namespace`

you ask OVOS to play music and the music page shows up - music player page is the active `namespace`

you ask OVOS a question and wolfram alpha page shows up - wolfram page is the active `namespace`

wolfram alpha times out - music player page is the active `namespace`

music ends and page times out - `homescreen` is the active `namespace`

> **NOTE:** GUI does not yet support `Session`, in the future `namespaces` will be tracked per `Session` allowing remote clients to each have their own GUI state

## GUI Plugins

## NEW in ovos-core version 0.0.8

The active namespace is tracked by `ovos-gui` and manages the [homescreen skill](#), desktop implementations like Plasma Bigscreen do not have a homescreen, instead they manage each skill in their own window

A single GUI plugin can be loaded in `ovos-gui` to handle bus events and provide companion python code to GUI clients, this is usually done by your OS and specific to a use case.

```
| plugin
| description
| notes
|
|-----|
-----|-----
-----|-----
-----|-----
-----|
| ovos-gui-plugin-shell-companion | extra functionality for ovos-shell
| message specs
| | ovos-gui-plugin-bigscreen      | window management for Plasma Bigscreen
Voice Apps | UNMAINTED <br> message specs <br> backup fork in life support
ovos-plasma-bigscreen-qt5 | | ovos-gui-plugin-plasmoid      | Plasma
integration of different Mycroft AI services
| UNMAINTED <br> never migrated to OVOS
|
```

## Configuration

The gui service has a few sections in `mycroft.conf`

```
"gui": {
  "idle_display_skill": "skill-ovos-homescreen.openvoiceos",
  "extension": "generic",
  "generic": {
    "homescreen_supported": false
  }
},

"gui_websocket": {
  "host": "0.0.0.0",
  "base_port": 18181,
  "route": "/gui",
  "ssl": false
},
```

# docs/110-config.md

## Configuration Management

### Summary

The OVOS configuration loader merges settings from multiple sources default, system, remote, and userso you can customize only what you need without touching shipped files.

---

### Usage Guide

#### 1. Locate or create your user config

```
mkdir -p ~/.config/mycroft
nano ~/.config/mycroft/mycroft.conf
```

Add only the keys you want to override; everything else falls back to defaults.

#### 1. Override via environment variables (optional)

```
export OVOS_CONFIG_BASE_FOLDER="myfolder"
export OVOS_CONFIG_FILENAME="myconfig.yaml"
```

This changes paths such as:

```
~/.config/mycroft/mycroft.conf ~/.config/myfolder/mycroft.conf
```

```
~/.config/mycroft/mycroft.conf ~/.config/mycroft/myconfig.yaml
```

## 1. Use special flags (in system config)

```
{
  "disable_user_config": true,
  "disable_remote_config": true
}
```

Place these in `/etc/mycroft/mycroft.conf` (or your default package file) to turn off loading of remote or user settings.

---

# Technical Explanation

## Load Order & Overrides

OVOS loads **all** existing files in this sequence, with later files overriding earlier ones:

**Default** (ovos-config package)

**System** (/etc/mycroft/mycroft.conf)

**Remote** (~/.config/<base>/web\_cache.json)

**User** (~/.config/<base>/<filename>)

> Keys repeated in multiple files are overridden by the lastloaded file containing them.

## File Locations & Formats

**Base folder:** Controlled by `OVOS_CONFIG_BASE_FOLDER` (defaults to `mycroft`).

**Filename:** Controlled by `OVOS_CONFIG_FILENAME` (defaults to `mycroft.conf`).

**Formats:** JSON (`.json` or `.conf`) or YAML (`.yaml` or `.yml`).

## Protected Keys

Prevent certain settings from being overridden by remote or user configs.

```
{
  "protected_keys": {
    "user": [
      "gui_websocket.host",
      "websocket.host"
    ]
  }
}
```

> this example block users from exposing the messagebus accidentally

## Disabling Layers

**disable\_user\_config:** If `true`, XDG user configs are ignored.

**disable\_remote\_config:** If `true`, downloaded remote configs (`web_cache.json`) are ignored.

---

## Tips & Caveats

**Always use your user file** (`~/.config/.../mycroft.conf`) to override defaults never edit system or packageshipped files.

Ensure your JSON is valid; mixed file extensions may lead to load errors.

Remember that setting `disable_user_config` or `disable_remote_config` will silently skip those layers use with caution.

Admin PHAL is a special service that runs as root, this means it can **only** access `'/etc/mycroft/mycroft.conf'`

---

## References

[OVOS Config Loader \(GitHub\)](#)



## docs/150-advanced\_solvers.md

# Specialized Solver Plugins

Solver plugins also exist for specialized tasks, like regular question solvers  
these also benefit from automatic bidirectional translation for language support

---

## ReRankers / MultipleChoiceQuestionSolvers

A specialized kind of solver plugin that chooses the best answer out of several options

[!Untitled-2025-04-15-2340\(1\)](#)

These specialized solvers are used internally by  
[ovos-common-query-pipeline-plugin](#), some skills and even by other question solver plugins!

Example configuration of [ovos-flashrank-reranker-plugin](#) for usage with  
ovos-common-query-pipeline-plugin

```
"intents": {
  "common_query": {
    "min_self_confidence": 0.5,
    "min_reranker_score": 0.5,
    "reranker": "ovos-flashrank-reranker-plugin",
    "ovos-flashrank-reranker-plugin": {
      "model": "ms-marco-TinyBERT-L-2-v2"
    }
  }
}
```

---

## Evidence Solver

Evidence solvers accept not only a question but also a companion piece of text containing the answer.

Some question solver plugins like `ovos-solver-wikipedia-plugin` use evidence solvers internally, they are often helpful to generate a question out of a search result

[!Untitled-2025-04-15-2340\(9\)](#)

[!Untitled-2025-04-15-2340\(10\)](#)

---

## Summarizer

Some question solver plugin use summarizers internally, they are often helpful to shorten long text from web search results

[!Untitled-2025-04-15-2340\(11\)](#)

[!Untitled-2025-04-15-2340\(12\)](#)

---

## Collaborative Agents via MoS (Mixture of Solvers)

One of the most powerful features of the OVOS solver architecture is its ability to **orchestrate multiple agents collaboratively** through specialized **Mixture of Solvers (MoS)** plugins.

[!image](#)

These [MoS solvers](#) implement strategies that combine the strengths of various LLMs, rerankers, rule-based solvers, or even remote agents (like HiveMind nodes), allowing dynamic delegation and refinement of answers.

> **Flexible Plugin Design:** MoS strategies are implemented as standard solver plugins. This means they can be composed, nested, or swapped just like any other solver allowing advanced collaborative behavior with minimal integration effort.

## How It Works

Instead of relying on a single model or backend, a MoS solver delegates the query

to several specialized solvers (workers) and uses strategies like voting, reranking, or even further generation to decide the best final response.

Examples include:

**The King:** Uses a central "king" (reranker or LLM) to select or generate the best answer based on multiple solver outputs.

!Untitled-2025-04-15-2340(25)

---

**Democracy:** Implements a voting system among reranker solvers to choose the most agreed-upon response.

!Untitled-2025-04-15-2340(23)

---

**Duopoly:** A pair of collaborating LLMs generate and discuss answers before passing them to a final decider ("the president" solver).

!Untitled-2025-04-15-2340(24)

---

Each strategy enables different dynamics between solvers be it a single judge, a voting panel, or a back-and-forth discussion between agents.

> **Recursive Composition:** Any MoS strategy can recursively use another MoS as a sub-solver, allowing for arbitrarily deep collaboration trees.

## docs/150-personas.md

# AI Agents in OpenVoiceOS

OpenVoiceOS (OVOS) introduces a flexible and modular system for integrating AI agents into voice-first environments. This is made possible through a layered architecture built around **solvers**, **personas**, and **persona routing** components. This section explains how these parts work together to enable intelligent conversations with customizable behavior.

---

# Solver Plugins (Low-Level AI)

At the core of the AI agent system are **\*\*solver plugins\*\***. These are simple black-box components responsible for handling a single task: receiving a text input (typically a question) and returning a text output (typically an answer).

!Untitled-2025-04-15-2340

## Key Features:

**Input/Output:** Plain text in, plain text out.

**Functionality:** Usually question-answering, though more specialized solvers exist (e.g., summarization, multiple choice).

**Language Adaptation:** Solvers are automatically wrapped with a translation layer if they don't support the user's language. For instance, the Wolfram Alpha solver is English-only but can work with Portuguese through automatic bidirectional translation.

**Fallback Behavior:** If a solver cannot produce a result (returns `None`), higher-level systems will attempt fallback options.

---

# Personas (Agent Definition Layer)

A **persona** represents a higher-level abstraction over solver plugins. It behaves like an AI agent with a defined personality and behavior, built by combining one or more solvers in a specific order

## Key Features:

**Composition:** Each persona consists of a name, a list of solver plugins, and optional configuration for each.

**Chained Execution:** When a user question is received, the persona tries solvers one by one. If the first solver fails (returns `None`), the next one is tried until a response is generated.

**Customizable Behavior:** Different personas can emulate different personalities

or knowledge domains by varying their solver stack.

[!Untitled-2025-04-15-2340\(7\)](#)

[!Untitled-2025-04-15-2340\(8\)](#)

```
{
  "name": "OldSchoolBot",
  "solvers": [
    "ovos-solver-wikipedia-plugin",
    "ovos-solver-ddg-plugin",
    "ovos-solver-plugin-wolfram-alpha",
    "ovos-solver-wordnet-plugin",
    "ovos-solver-rivescript-plugin",
    "ovos-solver-failure-plugin"
  ],
  "ovos-solver-plugin-wolfram-alpha": {"appid": "Y7353-XXX"}
}
```

> personas don't need to use LLMs, you don't need a beefy GPU to use ovos-persona, any solver plugin can be used to define a persona

---

## Persona Pipeline (Runtime Routing in OVOS-Core)

Within `ovos-core`, the

**[persona-pipeline](<https://github.com/OpenVoiceOS/ovos-persona>)** plugin handles all runtime logic for managing user interaction with AI agents.

### Key Features:

**Persona Registry:** Supports multiple personas, defined by the user or discovered via installed plugins.

**Session Control:** The user can say "I want to talk with {persona\_name}" to route their dialog to a specific persona.

**Session End:** The user can disable the current persona at any time to return to normal assistant behavior.

**Fallback Handling:** If OpenVoiceOS can't answer, the system can ask the default persona instead of speaking an error.

**Extensible:** Potential for future enhancements via messagebus to adjust system behavior based on persona (e.g., dynamic prompt rewriting).

in your `mycroft.conf`

```
{
  "intents": {
    "persona": {
      "handle_fallback": true,
      "default_persona": "Remote Llama"
    },
    "pipeline": [
      "stop_high",
      "converse",
      "ocp_high",
      "padatious_high",
      "adapt_high",
      "ovos-persona-pipeline-plugin-high",
      "ocp_medium",
      "...",
      "fallback_medium",
      "ovos-persona-pipeline-plugin-low",
      "fallback_low"
    ]
  }
}
```

---

## OVOS as a Solver Plugin

An advanced trick: **'ovos-core' itself can act as a solver plugin**. This allows you to expose OVOS itself as an agent to other applications in localhost

[!Untitled-2025-04-15-2340\(3\)](#)

Good for chaining OVOS instances in docker.

Use skills in a collaborative AI / MoS (mixture-of-solvers) setup.

`ovos-bus-solver-plugin` makes **no sense inside a local persona** (infinite loop!), but is **great for standalone usage**.

Expose OVOS behind HTTP api via `ovos-persona-server` without exposing the messagebus directly

```
{
  "name": "Open Voice OS",
  "solvers": [
    "ovos-solver-bus-plugin",
    "ovos-solver-failure-plugin"
  ],
  "ovos-solver-bus-plugin": {
    "autoconnect": true,
    "host": "127.0.0.1",
    "port": 8181
  }
}
```

> if you are looking to access OVOS remotely or expose it as a service see [hivemind agents documentation](#) for a more secure alternative

---

## Summary Table

| Component               | Role   |
|-------------------------|--|
|                         |  |
|                         |  |
|                         |  |
|                         |  |
| <b>Solver Plugin</b>    | Stateless text-to-text inference (e.g., Q&A, summarization). |
| <b>Persona</b>          | Named agent composed of ordered solver plugins.              |
| <b>Persona Server</b>   | Expose personas to other Ollama/OpenAI compatible projects.  |
| <b>Persona Pipeline</b> | Handles persona activation and routing inside OVOS core.     |

By decoupling solvers, personas, and persona management, OVOS allows for powerful, customizable AI experiences, adaptable to both voice and text interactions across platforms.

# docs/151-llm-transformers.md

## Generative AI Transformer Plugins

**Transformer plugins** operate independently of personas and provide fine-grained control over OVOSs internal processing pipeline. They are not part of the persona framework but can synergize with it.

### Key Details:

**Scope:** Transformers apply within the OVOS core pipeline not inside personas (though solver plugins can use them internally if desired).

**Independence:** Transformers and personas are separate systems. However, future enhancements may allow dynamic interaction between the two (e.g., a persona adjusting transformer settings).

### Key Integration Points:

[Utterance

Transformers](<https://openvoiceos.github.io/ovos-technical-manual//102-core/#utterance-transformers>)

: Operate between **STT (Speech-to-Text)** and **NLP (Natural Language Processing)**.

[Dialog

Transformers]([https://openvoiceos.github.io/ovos-technical-manual//103-audio\\_service/#dialog-transformers](https://openvoiceos.github.io/ovos-technical-manual//103-audio_service/#dialog-transformers))

: Operate between **NLP** and **TTS (Text-to-Speech)**.

---

## Examples of Transformer Plugins Using AI Agents

### OVOS Transcription Validator



This plugin validates the output of STT engines using a language model to filter out incorrect or incoherent transcriptions *before* they are passed to NLP.

#### How It Works:

1. Receives an STT transcript and its language code.
2. Sends both to an LLM prompt (local or via Ollama).
3. Gets a True or False response based on utterance validity.

**Configuration Snippet (mycroft.conf):**

```
json "utterance_transformers": {  
  "ovos-transcription-validator-plugin": { "model": "gemma3:1b",  
  "ollama_url": "http://192.168.1.200:11434", "prompt_template":  
  "/path/to/template.txt", "error_sound": true, "mode": "reprompt" } }
```

**Use Case:** Prevent skills from being triggered by invalid STT output like "Potato stop green light now yes."

---

## Dialog Transformer

This plugin rewrites assistant responses based on a persona-style prompt, enabling tone or complexity adjustments.

#### Example Prompt Use Cases:

"Rewrite the text as if you were explaining it to a 5-year-old"

"Explain it like you're teaching a child"

"Make it sound like an angry old man"

"Add more 'dude'ness to it"

**Configuration Snippet (mycroft.conf):**

```
json "dialog_transformers": {  
  "ovos-dialog-transformer-openai-plugin": { "rewrite_prompt": "rewrite  
the text as if you were explaining it to a 5-year-old" } }
```

This plugin often leverages LLMs through solver plugins but operates *after* the main dialog logic, adjusting the final output.

## docs/152-hivemind-agents.md

# Remote Agents with OpenVoiceOS

While OpenVoiceOS is designed primarily for **local-first usage**, more advanced deployments like hosting agents in the cloud, connecting multiple voice satellites, or enabling multi-user access through a web frontend are made possible via the **HiveMind** companion project.

---

## HiveMind Server

**HiveMind** is a distributed voice assistant framework that allows you to expose **AI Agents** (either full ovos-core installs or just individual personas) over a secure protocol.

> Unlike the lightweight `persona-server`, HiveMind is designed for trusted, networked setups.

### Key Features:

**Secure Access:** Communicates over the **HiveMind protocol**, which supports authentication, encryption and granular permissions safe for exposing OVOS to remote clients or satellites.

**Agent Plugins:** Agent plugins integrate the **HiveMind protocol** with various frameworks, including OpenVoiceOS. Keep your existing infrastructure even when you totally replaces the brains!

**Multi-User Ready:** Great for use in **cloud hosting**, **web portals**, or **enterprise environments** where access control is critical.

**Composable:** Let **local personas delegate** questions to a smarter **remote OVOS instance**.

!

### Typical Use-cases:

Running OpenVoiceOS on a powerful server or in the cloud.

Connecting lightweight devices (satellites).

Remote access to OpenVoiceOS.

Serving multiple users or applications concurrently.

Check out the [HiveMind documentation](#) for more info

---

## HiveMind Personas

The `hivemind-persona-agent-plugin` project allows you to **expose a single persona** not the full OVOS stack through hivemind

This enables you to deploy AI agents for external use without needing a full OVOS assistant.

!Untitled-2025-04-15-2340(15)

## Why Use It?

Minimal attack surface (persona only, no full assistant features).

Can be queried remotely using the HiveMind protocol.

> This is **not** the same as `persona-server`. `hivemind-persona-agent-plugin` uses a **secure protocol** (HiveMind), while `ovos-persona-server` uses insecure HTTP.

## Server Configuration

in your hivemind config file `~/.config/hivemind-core/server.json`

```
{
  "agent_protocol": {
    "module": "hivemind-persona-agent-plugin",
    "hivemind-persona-agent-plugin": {
      "persona": {
        "name": "Llama",
        "solvers": [
          "ovos-solver-openai-plugin"
        ],
        "ovos-solver-openai-plugin": {
          "api_url": "https://llama.smartgic.io/v1",
          "key": "sk-xxxx",
          "persona": "helpful, creative, clever, and very friendly."
        }
      }
    }
  }
}
```

---

## HiveMind as a Solver Plugin

Want your local assistant to ask a remote one when it's stuck? You can!

The hivemind-bus-client can function as a solver plugin, allowing you to:

- Delegate processing to a more powerful/secure server for specific tasks.

- Handle outages: Handle intermittent local agent failures from other solver plugins in your persona definition

- Use remote hivemind agents in a collaborative AI / MoS (mixture-of-solvers) setup.

[!Untitled-2025-04-15-2340\(14\)](#)

> *When in doubt, ask a smarter OVOS.*

For usage with persona, use "ovos-solver-hivemind-plugin" for the solver id

```
{
  "name": "HiveMind Agent",
  "solvers": [
    "ovos-solver-hivemind-plugin"
  ],
  "ovos-solver-hivemind-plugin": {"autoconnect": true}
}
```

You can also use it in your own python projects

```
from ovos_hivemind_solver import HiveMindSolver

bot = HiveMindSolver()
bot.connect() # connection info from identity file
print(bot.spoken_answer("what is the speed of light?"))
```

---

## Chaining Components for Flexible Deployments

HiveMind and persona-server can be combined to bridge secure and insecure environments, depending on your needs:

expose existing OpenAI/Ollama servers to hivemind satellites **securely**

connect hivemind satellites directly to existing LLM apps (eg. ollama)

expose a remote `hivemind-core` to local **insecure** ollama/openai endpoints

eg. to integrate hivemind into HomeAssistant

expose a localhost `ovos-core/persona.json` to local **insecure** ollama/openai endpoints

half-way compromise, does not expose the full messagebus and does not require hivemind

easier to setup and configure

| Use Case | Tool | Secure? | API Type |
|----------|------|---------|----------|
| Notes    |      |         |          |

|   |  |                   |   |
|---|--|-------------------|---|
| Local interface + Persona                                 | ovos-persona-server + persona.json                           |                   |   |
| OpenAI-compatible   | Great for quick setups, not public exposure, HTTP, no auth   |                   |   |
| Local interface + OpenVoiceOS                             | ovos-persona-server + ovos-solver-bus-plugin                 | OpenAI-compatible | OpenVoiceOS   |
| bus must be exposed to ovos-persona-server, HTTP, no auth |  |                   |   |
| Local interface + HiveMind Agent                          | ovos-persona-server + ovos-solver-hivemind-plugin            | OpenAI-compatible | Same as above, but for any remote hivemind agent, HTTP, no auth |
| Secure remote OpenVoiceOS agent                           | hivemind-core + hivemind-ovos-agent-plugin + ovos-core       | HiveMind protocol | Auth, encryption, granular permissions, HTTP or Websockets      |
| Secure remote Persona agent                               | hivemind-core + hivemind-persona-agent-plugin + persona.json | HiveMind protocol | Auth, encryption, granular permissions, HTTP or Websockets      |

The first 3 examples allow us to integrate our Agents with HomeAssistant via the Ollama Integration

The last 2 examples allow us to integrate with HiveMind ecosystem and all the existing satellite implementations

---

## Related (Insecure) Alternatives

While useful for experimentation, some other persona access methods are **not secure** for remote use:

ovos-persona-server:

Compatible with **OpenAI/Ollama APIs**.

**HTTP only**, not encrypted or authenticated.

Useful to expose personas to **HomeAssistant**, **OpenWebUI**, and similar local network tools.

HomeAssistant + ovos-persona-server:

Can route HomeAssistant **wyoming satellites** to an OVOS persona.

Uses **Wyoming protocol**, which lacks hivemind's security features.

# docs/153-persona\_pipeline.md

## Persona Pipeline

The 'ovos-persona-pipeline-plugin' provides a dynamic way to integrate persona-based conversational behavior into the OVOS pipeline system. It allows you to route user utterances to AI personas instead of skill matchers, depending on context and configuration.

---

## Overview

The `persona_pipeline` is a plugin for the OVOS pipeline architecture. It dynamically delegates user utterances to a configured **Persona**, which attempts to resolve the intent using a sequence of **Solver Plugins** (e.g., LLMs, search tools, knowledge bases).

You can configure it to:

- Intercept all utterances and give full control to the persona.

- Fall back to the persona only if skills don't match.

- Operate based on confidence tiers (high/medium/low).

---

## Plugin Structure

The plugin is composed of two components:

| Plugin Name                       | Usage                           |
|-----------------------------------|---------------------------------|
| ovos-persona-pipeline-plugin-high | For active persona interactions |
| ovos-persona-pipeline-plugin-low  | Fallback persona handling       |

You must **insert these plugin IDs** in your `mycroft.conf` under the `intents.pipeline` key to activate persona handling at the appropriate tier.

---

# Configuration

```
{
  "intents": {
    "persona": {
      "handle_fallback": true,
      "default_persona": "Remote Llama",
      "short-term-memory": true
    },
    "pipeline": [
      // Depending on strategy, insert plugin here
    ]
  }
}
```

## persona section options:

| Key               | Description  |
|-------------------|--|
| handle_fallback   | Enables fallback routing when no persona is active           |
| default_persona   | Sets a persona to use by default (e.g., after boot or reset) |
| short-term-memory | Maintains conversation state within a session (boolean)      |

## Pipeline Strategies

### 1. Full Control (Persona-First)

In this mode, **personas override** all skills. The persona handles every utterance unless explicitly deactivated.



```
"pipeline": [
  "ovos-persona-pipeline-plugin-high",
  "stop_high",
  "converse",
  "padatious_high",
  "adapt_high",
  ...
]
```

Best for immersive chatbot experiences

Skills like music, alarms, and weather will not trigger unless persona is disabled

---

## 2. Hybrid Mode (Skills First)

Only unmatched or low-confidence utterances are routed to the persona.

```
"pipeline": [
  "stop_high",
  "converse",
  "padatious_high",
  "adapt_high",
  "ovos-persona-pipeline-plugin-high",
  "fallback_medium",
  ...
]
```

Preserves traditional voice assistant behavior

Persona fills in where skills fall short

---

## 3. Fallback Mode Only

Even when no persona is active, this mode allows the pipeline to fall back to a **default persona** for unmatched utterances.

```
"pipeline": [  
  ...  
  "fallback_medium",  
  "ovos-persona-pipeline-plugin-low",  
  "fallback_low"  
]
```

Replaces `skill-ovos-fallback-chatgpt`

Fallbacks to a default persona response for a consistent assistant feel

---

## Persona Resolution Flow

1. **Utterance Received**
2. Pipeline matchers are checked in order.
3. If `persona-pipeline` is reached:
4. If a persona is **active**, send utterance to that persona.
5. If no persona is active and `handle_fallback` is enabled, use the **default\_persona**.
6. The persona delegates to its configured **solvers** until one returns a response.
7. The pipeline returns the matched response back to the user.

---

## Persona Configuration

Personas are defined in:

```
~/.config/ovos_persona/*.json
```

## Example:

```
{
  "name": "Remote Llama",
  "solvers": [
    "ovos-solver-openai-plugin",
    "ovos-solver-failure-plugin"
  ],
  "ovos-solver-openai-plugin": {
    "api_url": "https://llama.smartgic.io/v1",
    "key": "sk-xxx",
    "persona": "friendly and concise assistant"
  }
}
```

Each persona defines a `solvers` list.

Solvers are attempted **in order**.

The first solver to return a valid result ends the search.

Include a `"ovos-solver-failure-plugin"` as a final fallback for graceful error handling.

---

## Persona Intents

`"ovos-persona-pipeline-plugin-high"` supports a set of core voice intents to manage persona interactions seamlessly.

These intents provide **out-of-the-box functionality** for controlling the Persona Service, ensuring smooth integration with the conversational pipeline and enhancing user experience.

## List Personas

**Example Utterances:**

"What personas are available?"

"Can you list the personas?"

"What personas can I use?"

## Check Active Persona

### Example Utterances:

"Who am I talking to right now?"

"Is there an active persona?"

"Which persona is in use?"

## Activate a Persona

### Example Utterances:

"Connect me to {persona}"

"Enable {persona}"

"Awaken the {persona} assistant"

"Start a conversation with {persona}"

"Let me chat with {persona}"

## Single-Shot Persona Questions

Enables users to query a persona directly without entering an interactive session.

### Example Utterances:

"Ask {persona} what they think about {utterance}"

"What does {persona} say about {utterance}?"

"Query {persona} for insights on {utterance}"

"Ask {persona} for their perspective on {utterance}"

## Stop Conversation

### Example Utterances:

"Stop the interaction"

"Terminate persona"

"Deactivate the chatbot"

"Go dormant"

"Enough talking"

"Shut up"

---

## docs/199-WIP\_ovos\_media.md

### ovos-media

> **EXPERIMENTAL - NEW** ovos-core version **0.0.8**

ovos-media is a work in progress, it does not yet ship with OVOS by default, but it can be manually enabled

In order to use ovos-media you need to enable the OCP pipeline in ovos-core and to disable the old audio service

disabling old audio service `json { "enable_old_audioservice": false }`

Enabling OCP pipeline

```

{
  // Intent Pipeline / plugins config
  "intents" : {
    // the pipeline is a ordered set of frameworks to send an utterance too
    // if one of the frameworks fails the next one is used, until an answer is found
    "pipeline": [
      "converse",
      "ocp_high",
      "...",
      "common_qa",
      "ocp_medium",
      "...",
      "ocp_fallback",
      "fallback_low"
    ]
  }
}

```

## OCP

!

**OCP** stands for OpenVoiceOS Common Play, it is a full-fledged media player service that can handle audio and video

> **DEPRECATION WARNING** OCP is in the process of migrating from a audio plugin to ovos-media service, this documentation is not valid for ovos-core version **0.0.7**

OCP provides a pipeline component specialized in matching media queries.

The pipeline classifies the media type (movie, music, podcast...) and queries OCP skills for results, you can read more about the OCP Pipeline docs

## Architecture

[!imagem](#)

## Media Intents

Before regular intent stage, taking into account current OCP state (media ready to play / playing)

"play {query} "  
"previous" (media needs to be loaded)  
"next" (media needs to be loaded)  
"pause" (media needs to be loaded)  
"play" / "resume" (media needs to be loaded)  
"stop" (media needs to be loaded)  
"I like that song" (music needs to be playing)

## MPRIS integration

OCP Integrates with MPRIS allows OCP to control external players

[!imagem](#)

## Sync with external players

Via MPRIS OCP can control and display data from external players, if using KDEConnect this includes playback in connected devices

See a demo here

[!demo video](https://www.youtube.com/watch?v=1KMFV0UVYEM)(https://www.youtube.com/watch?v=1KMFV0UVYEM)

This also includes voice intents, allowing you for example to voice control spotify

## Manage multiple players

If OCP is set to manage external players it will ensure only one of them is playing media at once, if using KDEConnect this includes playback in connected devices

See a demo here (**warning**: contains black metal)

[!demo video](https://www.youtube.com/watch?v=YzC7oFYCcRE)(https://www.youtube.com/watch?v=YzC7oFYCcRE)

## Skills Menu

Some skills provide featured\_media, you can access these from the OCP menu

!

## Homescreen widget

The homescreen skill that comes pre-installed with OpenVoiceOS also comes with a widget for the OCP framework.

!

## File Browser integration

selected files will be played in OCP

!

folders are considered playlists

!

## Favorite Songs

You can like a song that is currently playing via GUI and intent "I like that song"

[!like](#)

Liked songs can be played via intent "play my favorite songs" or GUI

[!favs](#)

## Configuration

under mycroft.conf



```

{ } // module: will be parsed by OCP and used to initiate
} // playback in the request playback handler
// Configure ovos-media service
} // similarly to wakewords, configure any number of playback handlers
// those will be parsed by OCP and used to initiate
// playback handlers might be local applications or even remote devices
"media": { // playback plugin by setting playback handler
    "aliases": ["GUI", "Graphical User Interface"],

    // order of preference to try playback handlers
    // gui deactivates a plugin by setting handler to false
    // if unavailable or unable to handle a uri, the next in list is used
    "gui": {
        "active": true
        // NB: users may request specific handlers in the utterance
    } // the plugin name

} // keys are the strings defined in "audio_players"

"preferred_audio_services": ["gui", "vlc", "mplayer", "cli"],
// PlaybackType.AUDIO handlers may use to refer to this playback handler

"web_players": { // those will be parsed by OCP and used to initiate
    // keys are the strings defined in "web_players"
    // open playback the live browser playback handler
    "preferred_web_services": ["gui", "browser"],
    "browser": {
        "aliases": ["GUI", "Graphical User Interface"],

        // the plugin name
    } // keys are the strings defined in "video_players"
    "preferred_video_services": ["gui", "vlc"],
    "active": true

} // friendly names a user may use to refer to this playback handler
// PlaybackType.AUDIO handlers
} // those will be parsed by OCP and used to initiate
"audio_players": {
    // playback in the request playback handler
    // vlc player uses a headless vlc instance to handle uris
    // PlaybackType.VIDEO handlers "Local Browser", "Default Browser",
    "vlc": {
        // the plugin name
        // vlc deactivates a plugin by setting handler to false
        "module": "ovos-media-audio-plugin-vlc",
        "vlc": {
            "active": true
        }, // the plugin name
        // friendly names a user may use to refer to this playback handler
        // module: "mycroft-media-audio-plugin-vlc"
        // those will be parsed by OCP and used to initiate
        "gui": {
            // playback in the request playback handler
            // friendly names a user may use to refer to this playback handler
            "aliases": ["VLC"],
            // module: will be parsed by OCP and used to initiate
            // playback in the request playback handler
            // deactivate a plugin by setting to false
            // friendly names a user may use to refer to this playback handler
            "active": true
        }, // those will be parsed by OCP and used to initiate
        // playback plugin by setting playback handler
        // command line player uses configurable shell commands with file uris as argument
        "cli": {
            "aliases": ["GUI", "Graphical User Interface"],
        }, // the plugin name
        // gui deactivates a plugin by setting handler to false
        "module": "ovos-media-audio-plugin-cli",
        "gui": {
            "active": true
        }, // the plugin name
        // friendly names a user may use to refer to this playback handler

```





# Troubleshooting

Having trouble getting OCP to run properly and be exposed as an MPRIS media player? Check the following:

The `DBUS_SESSION_BUS_ADDRESS` environment variable is what OCP uses to try to connect to 'dbus'. On an OVOS system it will look something like `unix:path=/run/user/1000/bus`. To get the right user ID, run `id -u`.

If `DBUS_SESSION_BUS_ADDRESS` is not set, the next place OCP checks is the `DISPLAY` environment variable. If this is set and looks similar to the value above, then you can probably exclude `DBUS_SESSION_BUS_ADDRESS`, but if neither are set then use `DBUS_SESSION_BUS_ADDRESS`.

Make sure your OCP settings in your config file like something like the following, taking note of the `dbus_type` value:

```
"media": {  
  "dbus_type": "session"  
}
```

If your `dbus_type` is set to `system` then OCP will still work, but since it requires root privileges to read from the system dbus, external systems or programs without root privileges cannot read the MPRIS data there.

You can confirm if the OCP player is registered with dbus using the following command: `dbus-send --session --dest=org.freedesktop.DBus --type=method_call --print-reply /org/freedesktop/DBus org.freedesktop.DBus.ListNames`

The output should look something like the following, if it is working:

```
method return time=1691467760.293397 sender=org.freedesktop.DBus -> destination=:1.10 seri
  array [
    string "org.freedesktop.DBus"
    string "org.freedesktop.systemd1"
    string ":1.10"
    string "org.mpris.MediaPlayer2.OCP"
    string ":1.9"
    string ":1.1"
  ]
```

The important part is the `org.mpris.MediaPlayer2.OCP` value.

If the above steps do not work, please reach out to the OVOS team on Matrix for assistance.

## docs/200-stt\_server.md

# OpenVoiceOS STT HTTP Server

**Lightweight HTTP microservice for any OVOS speechtotext plugin, with optional Gradio UI.**

The OpenVoiceOS STT HTTP Server wraps your chosen OVOS STT plugin inside a FastAPI service (complete with automatic language detection), making it easy to deploy on your local machine, in Docker, or behind a load balancer.

---

## Usage Guide

### Install the server

```
pip install ovos-stt-http-server
```

### Configure your STT plugin

In your `mycroft.conf` (or equivalent) under the `stt` section: `json { "stt": { "module": "ovos-stt-plugin-xxx", "ovos-stt-plugin-xxx": { "model": "xxx" } } }`

## Launch the server

```
ovos-stt-server \  
  --engine ovos-stt-plugin-xxx \  
  --host 0.0.0.0 \  
  --port 9666
```

## Verify its running

Visit <http://localhost:9666/status> in your browser or run:

```
curl http://localhost:9666/status
```

---

# CommandLine Options

```
$ ovos-stt-server --help
```

```
usage: ovos-stt-server [-h] --engine ENGINE [--lang-engine LANG_ENGINE] [--port PORT] [--h  
                        [--description DESCRIPTION] [--info INFO] [--badge BADGE]
```

options:

|                           |  |
|---------------------------|--|
| -h, --help                | show this help message and exit                          |
| --engine ENGINE           | stt plugin to be used                                    |
| --lang-engine LANG_ENGINE | audio language detection plugin to be used               |
| --port PORT               | port number  |
| --host HOST               | host   |
| --lang LANG               | default language supported by plugin                     |
| --multi                   | Load a plugin instance per language (force lang support) |
| --gradio                  | Enable Gradio Web UI                                     |
| --cache                   | Cache models for Gradio demo                             |
| --title TITLE             | Title for webUI  |
| --description DESCRIPTION | Text description to print in UI                          |
| --info INFO               | Text to display at end of UI                             |
| --badge BADGE             | URL of visitor badge                                     |

---

# Technical Explanation

## FastAPI core

The server spins up a FastAPI app exposing REST endpoints.

## Plugin wrapping

Any OVOS STT plugin (Deepgram, Whisper, etc.) is loaded dynamically via entry points.

## Language detection

If you enable `--lang-engine`, incoming audio is passed through the detector, falling back to `--lang` or plugin defaults.

## Scalability

Stateless design lets you run multiple instances behind a load balancer or in Kubernetes.

## Optional Gradio UI

Launches a simple web demo for testing without writing any frontend code.

---

# HTTP API Endpoints

| Endpoint         | Method | Description  |
|------------------|--------|--|
| -----            | -----  |  |
| -----    /status |        |  |
| GET              |        | Returns plugin names, versions, and Gradio status.    /stt |
| POST             |        | Transcribe audio plaintext transcript.                     |
| /lang_detect     | POST   | Detect language JSON { "lang": "en", "conf": 0.83          |
| }.    /docs      |        |  |
| GET              |        | Interactive FastAPI OpenAPI docs.                          |
|                  |        |  |

---

# Companion Plugin

To point a OpenVoiceOS (or compatible project) to a STT server you can use the

companion plugin

**Install** `bash pip install ovos-stt-plugin-server`

**Configure** `json "stt": { "module": "ovos-stt-plugin-server",  
"ovos-stt-plugin-server": { "urls": ["https://0.0.0.0:8080/stt"],  
"verify_ssl": true }, }`

for audio language detection

```
"listener": {  
  "audio_transformers": {  
    "ovos-audio-lang-server-plugin": {  
      "urls": ["https://0.0.0.0:8080/lang_detect"],  
      "verify_ssl": true  
    }  
  }  
}
```

---

## Docker Deployment

**Create a Dockerfile** `dockerfile FROM python:3.7-slim RUN pip install  
ovos-stt-http-server==0.0.1 RUN pip install {YOUR_STT_PLUGIN}  
ENTRYPOINT ["ovos-stt-http-server", "--engine", "{YOUR_STT_PLUGIN}"]`

**Build & Run** `bash docker build -t my-ovos-stt . docker run -p 8080:9666  
my-ovos-stt`

Pre-built containers are also available via the [ovos-docker-stt](#) repository.

---

## Tips & Caveats

**Audio Formats:** Ensure client sends PCMcompatible formats (.wav, .mp3 recommended).

**Securing Endpoints:** Consider putting a reverse proxy (NGINX, Traefik) in front for SSL or API keys.

**Plugin Dependencies:** Some STT engines require heavy native librariesbake



them into your Docker image.

---

## Links & References

OVOS STT HTTP Server GitHub:

<https://github.com/OpenVoiceOS/ovos-stt-http-server>

Companion Plugin: <https://github.com/OpenVoiceOS/ovos-stt-server-plugin>

Docker Images: <https://github.com/OpenVoiceOS/ovos-docker-stt>

OVOS Plugin Manager: <https://github.com/OpenVoiceOS/ovos-plugin-manager>

## docs/201-tts\_server.md

# OpenVoiceOS TTS Server

**Lightweight HTTP microservice for any OVOS texttospeech plugin, with optional caching.**

Wrap your favorite OVOS TTS engine in a FastAPI serviceready to deploy locally, in Docker, or behind a load balancer.

The OpenVoiceOS TTS HTTP Server exposes any OVOS TTS plugin over a simple HTTP API. Send text, receive audio no extra glue code required.

---

## Usage Guide

### Install the server

```
pip install ovos-tts-server
```

### Configure your TTS plugin

In your `mycroft.conf` (or equivalent) under the `tts` section: 

```
json { "tts": {  
  "module": "ovos-tts-plugin-xxx", "ovos-tts-plugin-xxx": { "voice":  
  "xxx" } } }
```

## Launch the server

```
ovos-stt-server \  
  --engine ovos-tts-plugin-xxx \  
  --host 0.0.0.0 \  
  --port 9666
```

## Verify its running

Visit <http://localhost:9666/status> in your browser or run:

```
curl http://localhost:9666/status
```

---

# CommandLine Options

```
$ ovos-tts-server --help
```

```
usage: ovos-tts-server [-h] [--engine ENGINE] [--port PORT] [--host HOST] [--cache] [--lang LANG] [--info INFO] [--badge BADGE]
```

options:

|                           |                                      |
|---------------------------|--------------------------------------|
| -h, --help                | show this help message and exit      |
| --engine ENGINE           | tts plugin to be used                |
| --port PORT               | port number                          |
| --host HOST               | host                                 |
| --cache                   | save every synth to disk             |
| --lang LANG               | default language supported by plugin |
| --radio                   | Enable Gradio Web UI                 |
| --title TITLE             | Title for webUI                      |
| --description DESCRIPTION | Text description to print in UI      |
| --info INFO               | Text to display at end of UI         |
| --badge BADGE             | URL of visitor badge                 |

---

## Technical Explanation

## FastAPI Core

Spins up a FastAPI application exposing RESTful endpoints for synthesis and status checks.

## Plugin Loading

Dynamically loads any OVOS TTS plugin via Python entry pointsno code changes needed when adding new voices.

## Caching

When `--cache` is enabled, every synthesis request is stored as a WAV file for debugging or reuse.

## Scalability

Stateless by designrun multiple instances behind NGINX, Traefik, or Kubernetes with roundrobin or loadbased routing.

---

# HTTP API Endpoints

| Endpoint                | Method | Description                                   |
|-------------------------|--------|---|
|                         |        |   |
| ----- ----- -----       |        |   |
| -----                   |        |   |
| /status                 | GET    | Returns loaded plugin names and versions.     |
| /synthesize/{utterance} | GET    | URLencoded text WAV audio bytes.              |
| /v2/synthesize          | GET    | JSON {utterance: string, voice?: string} WAV. |
| /docs                   | GET    | Interactive OpenAPI (Swagger) docs.           |
|                         |        |   |

> any query parameters passed to `/v2/synthesize` will be forwarded to the individual plugins `get_tts` method if they are defined as kwargs there. > This allows "voice" and "lang" to be defined at runtime and not by plugin config at load time (for plugins that support it)

---

# Companion Plugin

Point your OVOS instance at this TTS server:

```
pip install ovos-tts-server-plugin
```

**Configuration** mycroft.conf:

```
{
  "tts": {
    "module": "ovos-tts-plugin-server",
    "ovos-tts-plugin-server": {
      "host": "http://localhost:9667",
      "voice": "xxx",
      "verify_ssl": false,
      "tts_timeout": 5
    }
  }
}
```

---

## Docker Deployment

**Create a Dockerfile** dockerfile FROM python:3.7-slim RUN pip install ovos-tts-server RUN pip install {YOUR\_TTS\_PLUGIN} ENTRYPOINT ["ovos-tts-server", "--engine", "{YOUR\_TTS\_PLUGIN}"]

**Build & Run** bash docker build -t my-ovos-tts . docker run -p 8080:9666 my-ovos-tts

Pre-built containers are also available via the [ovos-docker-tts](#) repository.

---

## Tips & Caveats

**Audio Formats:** By default, outputs WAV (PCM). If you need MP3 or OGG, wrap with an external converter or check

plugin support.

**Disk Usage:** Caching every file can grow large; monitor `./cache/` or disable with `--no-cache`.

**Security:** Consider adding API keys or putting a reverse proxy (NGINX, Traefik)

in front for SSL termination and  
rate limiting.

**Plugin Dependencies:** Some voices require native libraries (e.g., TensorFlow).  
Bake them into your Docker image to  
avoid runtime surprises.

---

## Links & References

**TTS Server GitHub:** <https://github.com/OpenVoiceOS/ovos-tts-server>

**Companion Plugin:** <https://github.com/OpenVoiceOS/ovos-tts-server-plugin>

**Docker Images:** <https://github.com/OpenVoiceOS/ovos-docker-tts>

**OVOS Plugin Manager:** <https://github.com/OpenVoiceOS/ovos-plugin-manager>

## docs/202-persona\_server.md

### OVOS Persona Server

The OVOS Persona Server makes any defined persona available through an API compatible with OpenAI and Ollama, allowing you to use OVOS personas as drop-in replacements for traditional large language models (LLMs) in other tools and platforms.

---

### Usage Guide

To start the Persona Server with a specific persona file:

```
$ ovos-persona-server --persona my_persona.json
```

This will launch a local server (default: `http://localhost:8337`) that exposes the persona via OpenAI and Ollama-compatible endpoints.

---

## Technical Explanation

A **persona** in OVOS is a predefined character or assistant configuration that can respond to user inputs, leveraging OVOSs conversational tools. The **Persona Server** acts as a gateway that translates external API requests (like those from OpenAI or Ollama clients) into interactions with this persona.

This enables seamless integration with a variety of existing tools that expect LLM-like behavior, including frameworks, bots, or smart home assistants.

---

## OpenAI-Compatible API Example

You can use the `openai` Python SDK to interact with the Persona Server:

```
import openai

openai.api_key = "" # No API key required for local use
openai.api_base = "http://localhost:8337"

response = openai.ChatCompletion.create(
    model="", # Optional: some personas may define specific models
    messages=[{"role": "user", "content": "tell me a joke"}],
    stream=False,
)

if isinstance(response, dict):
    # Non-streaming response
    print(response.choices[0].message.content)
else:
    # Streaming response
    for token in response:
        content = token["choices"][0]["delta"].get("content")
        if content:
            print(content, end="", flush=True)
```

**Note:** Some persona solvers are **not LLMs** and do **not** maintain chat history. Only the last message in the `messages` list is processed in some cases.

---

## Ollama-Compatible API

The server is also fully compatible with tools expecting an Ollama API.

For example, the [Home Assistant Ollama integration](#) can connect directly to an OVOS Persona Server, treating it as a local LLM backend.

---

## Tips

Make sure your persona file (`.json`) includes all the configuration details required by the solver or conversational backend.

If using in a production setting, consider securing your endpoint and defining rate limits.

Since personas can be highly customized, capabilities may vary depending on the persona used.

---

## Related Links

[OVOS Personas](#)

[OpenAI Python SDK](#)

[Home Assistant Ollama Integration](#)

## docs/203-translate\_server.md

# OpenVoiceOS Translate Server

**Expose OVOS language detection and translation plugins over HTTP.**

The OVOS Translate Server allows any OpenVoiceOS-compatible translation or

language detection plugin to run as a lightweight web service. This makes it easy to integrate translation features into any application or device using simple HTTP requests.

Great for local or cloud deployments, and ideal for use with the OVOS companion plugin to provide translation capabilities to your voice assistant.

---

## Usage Guide

### Install the Server

```
pip install ovos-translate-server
```

### Run the Server

```
ovos-translate-server \  
  --tx-engine ovos-translate-plugin-nllb \  
  --detect-engine ovos-lang-detector-classics-plugin
```

### Make Requests

Once the server is running (default on `http://0.0.0.0:9686`), you can access endpoints like:

Auto-detect source language:

```
GET /translate/en/o meu nome e Casimiro  
"My name is Casimiro"
```

Specify source and target language:

```
GET /translate/pt/en/o meu nome e Casimiro  
"My name is Casimiro"
```

Language detection:



```
GET /detect/o meu nome e Casimiro
"pt"
```

---

## Command-Line Options

```
$ ovos-translate-server --help
usage: ovos-translate-server [-h] [--tx-engine TX_ENGINE] [--detect-engine DETECT_ENGINE]

options:
  -h, --help                show this help message and exit
  --tx-engine TX_ENGINE      translate plugin to be used
  --detect-engine DETECT_ENGINE
                             lang detection plugin to be used
  --port PORT               port number
  --host HOST               host
```

---

## Technical Overview

**Plugin-based:** Uses the OVOS Plugin Manager to dynamically load any compatible plugin by name.

**RESTful API:** Simple HTTP endpoints allow you to send and receive translations from any app.

**Language Detection Support:** Works with any OVOS lang-detection plugin.

**Easy Deployment:** Perfect for running locally, inside Docker, or on a small server.

---

## Docker Deployment

## Use Prebuilt Images

Check out [ovos-docker-tx](#) for prebuilt containers.

## Build Your Own

Create a Dockerfile:

```
FROM python:3.7

RUN pip install ovos-utils==0.0.15
RUN pip install ovos-plugin-manager==0.0.4
RUN pip install ovos-translate-server==0.0.1

# Install your plugins
RUN pip install {PLUGIN_HERE}

ENTRYPOINT ovos-translate-server --tx-engine {PLUGIN_HERE} --detect-engine {PLUGIN_HERE}
```

Build the image:

```
docker build . -t my_ovos_translate_plugin
```

Run the container:

```
docker run -p 8080:9686 my_ovos_translate_plugin
```

Each plugin can provide its own Dockerfile using `ovos-translate-server` as the entrypoint.

---

## Companion Plugin Integration

To use this server with an OVOS voice assistant instance, install:

```
pip install ovos-translate-server-plugin
```

Then configure your `mycroft.conf`:

```
{
  "language": {
    "detection_module": "ovos-lang-detector-plugin-server",
    "translation_module": "ovos-translate-plugin-server",
    "ovos-translate-plugin-server": {
      "host": "http://localhost:9686",
      "verify_ssl": false
    },
    "ovos-lang-detector-plugin-server": {
      "host": "http://localhost:9686",
      "verify_ssl": false
    }
  }
}
```

---

## Tips & Caveats

Some translation plugins auto-detect language; others require you to specify `source_lang`.

Network errors or unresponsive servers will trigger fallback plugins, if configured.

For production, consider placing the service behind a reverse proxy with HTTPS enabled.

---

## Related Projects

**Translate Server Plugin:**

[ovos-translate-server-plugin](#)

**Translate Server Source:**

[ovos-translate-server](#)

**Docker Templates:**

[ovos-docker-tx](#)

**Plugin Manager:**

[ovos-plugin-manager](#)

---

With the OVOS Translate Server, adding multi-language support to your voice assistant is just an HTTP request away.

## docs/300-plugin-manager.md

# OVOS Plugin Manager (OPM)

[!image](#)

## Summary

The OVOS Plugin Manager (OPM) is a base package designed to provide arbitrary plugins to the OVOS ecosystem. It standardizes the interface for plugins, allowing them to be easily portable and configurable, whether integrated into OVOS projects or used in standalone applications.

---

## Usage Guide

To install a plugin using OPM, you can typically follow this process:

**Install the plugin using pip:**

```
pip install ovos-plugin-name
```

**Edit your configuration file (e.g., 'mycroft.conf') to enable and configure the plugin.**

**Restart your OVOS service to apply the changes.**

> In some setups like `ovos-docker`, make sure you install plugins in the correct environment.

---

## Technical Explanation

OPM allows developers to create plugins that are decoupled from OVOS core functionality. By using OPM's standard interface, plugins can be easily integrated into a variety of OVOS services or other projects.

Each plugin can be classified according to its functionality, with its own entry point defined in `setup.py`. This approach ensures that plugins are portable and independent, allowing them to be reused in other projects.

---

## Plugin Types

OPM recognizes several plugin types, each serving a specific purpose within the OVOS ecosystem. These types help categorize plugins for easier integration and configuration:

```

class PluginTypes(str, Enum):
    PHAL = "ovos.plugin.phal"
    ADMIN = "ovos.plugin.phal.admin"
    SKILL = "ovos.plugin.skill"
    VAD = "ovos.plugin.VAD"
    PHONEME = "ovos.plugin.g2p"
    AUDIO = 'mycroft.plugin.audioservice'
    STT = 'mycroft.plugin.stt'
    TTS = 'mycroft.plugin.tts'
    WAKEWORD = 'mycroft.plugin.wake_word'
    TRANSLATE = "neon.plugin.lang.translate"
    LANG_DETECT = "neon.plugin.lang.detect"
    UTTERANCE_TRANSFORMER = "neon.plugin.text"
    METADATA_TRANSFORMER = "neon.plugin.metadata"
    AUDIO_TRANSFORMER = "neon.plugin.audio"
    QUESTION_SOLVER = "neon.plugin.solver"
    COREFERENCE_SOLVER = "intentbox.coreference"
    KEYWORD_EXTRACTION = "intentbox.keywords"
    UTTERANCE_SEGMENTATION = "intentbox.segmentation"
    TOKENIZATION = "intentbox.tokenization"
    POSTAG = "intentbox.postag"

```

Each plugin type has its own category, with the most common being `skill`, `stt` (speech-to-text), `tts` (text-to-speech), and `wake_word`.

---

## Plugin Packaging

When creating a plugin, you need to define an entry point for the plugin type and class in your `setup.py`. Heres a typical `setup.py` structure for packaging a plugin:

```

from setuptools import setup

PLUGIN_TYPE = "mycroft.plugin.stt" # Adjust based on the plugin type
PLUGIN_NAME = "ovos-stt-plugin-name"
PLUGIN_PKG = PLUGIN_NAME.replace("-", "_")
PLUGIN_CLAZZ = "MyPlugin"
PLUGIN_CONFIGS = "MyPluginConfig"

PLUGIN_ENTRY_POINT = f'{PLUGIN_NAME} = {PLUGIN_PKG}:{PLUGIN_CLAZZ}'
CONFIG_ENTRY_POINT = f'{PLUGIN_NAME}.config = {PLUGIN_PKG}:{PLUGIN_CONFIGS}'

setup(
    name=PLUGIN_NAME,
    version='0.1.0',
    packages=[PLUGIN_PKG],
    install_requires=["speechrecognition>=3.8.1", "ovos-plugin-manager>=0.0.1"],
    keywords='mycroft ovos plugin',
    entry_points={PLUGIN_TYPE: PLUGIN_ENTRY_POINT, f'{PLUGIN_TYPE}.config': CONFIG_ENTRY_POINT}
)

```

---

## Voice Satellites

HiveMind setups allow you to configure which plugins run server-side or satellite-side. Here are two examples:

**Skills Server:** In this setup, the HiveMind server runs only core services and skills, while the satellites handle their own STT/TTS.

[!Server Profile](#)

**Audio Server:** Here, the HiveMind server runs a full OVOS core, handling STT/TTS for the satellites.

[!Listener Profile](#)

These profiles help balance the workload between the server and satellites, improving performance based on the setup.

---

# Projects Using OPM

Several OVOS projects and tools support OPM plugins, either as dependencies or directly within their ecosystem:

[ovos-core](#)

[ovos-tts-server](#)

[ovos-stt-http-server](#)

[ovos-translate-server](#)

[neon-core](#)

[HiveMind voice satellite](#)

Additionally, some plugins like AudioService, WakeWord, TTS, and STT are backwards compatible with Mycroft-Core, ensuring broad compatibility.

---

## Related Links

[OVOS Plugin Manager Repository](#)

[OVOS Installer](#)

# docs/310-mic\_plugins.md

## Microphone Plugins in OVOS

Microphone plugins in Open Voice OS (OVOS) are responsible for capturing audio input and feeding it to the listener. Introduced in `ovos-core` version **0.0.8**, these plugins allow for flexible integration with different audio backends and platforms.

## Usage Guide

To use a microphone plugin in OVOS:



Install the desired plugin with `pip`:

```
pip install ovos-microphone-plugin-<name>
```

Update your `mycroft.conf` (or `ovos.conf`) to specify the plugin:

```
{
  "listener": {
    "microphone": {
      "module": "ovos-microphone-plugin-alsa" // or another plugin
    }
  }
}
```

Restart OVOS to apply the new microphone plugin configuration.

## Supported Microphone Plugins

| Plugin   | Description  | OS Compatibility      |
|--|--|-----------------------|
| <a href="#">ovos-microphone-plugin-alsa</a>        | Based on <a href="#">pyalsaaudio</a> . Offers low-latency and high performance on ALSA-compatible devices. | Linux                 |
| <a href="#">ovos-microphone-plugin-pyaudio</a>     | Uses <a href="#">PyAudio</a> . Good general-purpose plugin for Linux.                                      | Linux                 |
| <a href="#">ovos-microphone-plugin-sounddevice</a> | Built on <a href="#">python-sounddevice</a> . Offers cross-platform support.                               | Linux, macOS, Windows |
| <a href="#">ovos-microphone-plugin-files</a>       | Uses audio files as input instead of a live microphone ideal for testing and debugging.                    | Linux, macOS, Windows |
| <a href="#">ovos-microphone-plugin-arecord</a>     | Wraps <code>arecord</code> using subprocess calls. Simple and effective on systems with ALSA.              | Linux                 |
| <a href="#">ovos-microphone-plugin-socket</a>      | Receives audio over a socket connection. Useful for remote microphone setups.                              | Linux, macOS, Windows |

## Technical Explanation

OVOS uses a plugin architecture to decouple the audio input system from the rest of the voice stack. Microphone plugins implement a common interface, making it easy to swap between different audio sources or backends without changing application code.

Each plugin provides a stream of audio data that OVOS uses to detect wake words

and perform speech-to-text (STT) processing. The `microphone` module is configured in the main listener settings, and the selected plugin is loaded dynamically at runtime.

## Tips & Caveats

**Performance:** For best results on Linux, the ALSA plugin typically provides the lowest latency.

**Cross-platform development:** Use the `sounddevice` or `files` plugin when developing on non-Linux systems.

**Testing:** The `files` plugin is ideal for automated testing environments where live input isn't available.

**Remote audio:** The `socket` plugin is a proof-of-concept for networked microphones and is not recommended for production use without customization.

---

### Coming Soon

Standalone usage examples

How to create your own microphone plugin (plugin template)

## docs/311-vad\_plugins.md

# VAD Plugins in OVOS

## Overview

Voice Activity Detection (VAD) helps determine when a user has *finished* speaking. In OVOS, VAD plugins are used **after** the wake word is detected to decide when to stop recording and send the audio to speech-to-text (STT) engines. This reduces latency and avoids sending unnecessary silence.

While wake word detection typically starts the recording, VAD ensures it ends cleanly optimizing performance and responsiveness.

# Usage Guide

Install the desired VAD plugin:

```
pip install ovos-vad-plugin-<name>
```

Set your preferred plugin in the OVOS configuration file (`mycroft.conf`):

```
{
  "listener": {
    "VAD": {
      "module": "ovos-vad-plugin-silero"
    },
    // Setting to remove all silence/noise from start and end of recorded speech before STT
    "remove_silence": true
  }
}
```

Restart the OVOS service to apply changes.

> By default, VAD is only used **after** wake word activation. See below for optional continuous mode.

## Available VAD Plugins

| Plugin | Description | |-----|-----| | [ovos-vad-plugin-silero](#) | Uses [Silero VAD](#), a neural networkbased VAD offering excellent real-time accuracy. **Recommended.** | | [ovos-vad-plugin-webrtcvad](#) | Wraps Googles [WebRTC VAD](#), lightweight and fast, suited for short audio frames. | | [ovos-vad-plugin-noise](#) | Simple threshold-based VAD using volume levels. Useful for constrained devices, but less accurate. | | [ovos-vad-plugin-precise](#) | Uses a custom-trained model with [Mycroft Precise](#). Can be tailored for your environment. |

## Technical Explanation

In OVOS, VAD operates **after** the wake word engine triggers recording. Its main purpose is to detect **the end of the user's speech**. Without VAD, the system would use a fixed timeout (e.g., 3 seconds of silence), which can lead to

premature cutoffs or excessive silence that slows down transcription.

VAD plugins continuously monitor the audio during recording and tell the listener when the user has stopped talking. Once silence is detected for a defined threshold, OVOS stops recording and forwards the result to the STT engine.

This flow looks like:

```
[ Wake Word Detected ] [ Start Recording ] [ VAD detects end of speech ] [ Stop Recording ]
```

## Experimental Continuous Mode

OVOS also supports an **experimental continuous listening mode** in `ovos-dinkum-listener`, where wake word detection is bypassed entirely. In this mode, the listener uses **VAD alone** to decide when someone is speaking and triggers STT automatically.

To enable this behavior: `json { "listener": { "continuous_listen": false, "VAD": { "module": "ovos-vad-plugin-silero" } } }`

> This mode is **experimental**, it is not the default and is **unstable or prone to false triggers**. Use with caution. This may also cause OVOS to hear its own TTS responses as questions

> [ovos-transcription-validator](#) is extremely recommend as a companion plugin for this mode

## Tips & Caveats

**Silero** is the most accurate and works well across platforms.

**Noise-based** VAD can be too sensitive in environments with background sound.

VAD plugins may expose tunable settings like silence thresholds or sensitivity refer to each plugin's documentation.

Disabling the wake word and relying only on VAD is **experimental** and not recommended for production use (yet).

---

### Coming Soon

Standalone usage examples

How to build a custom VAD plugin

## docs/312-wake\_word\_plugins.md

# Wake Word Plugins

Wake Word plugins allow Open Voice OS to detect specific words or sounds, typically the assistants name (e.g., "Hey Mycroft"), but can be customized for various use cases. These plugins enable the system to listen for and react to activation commands or phrases.

## Available Plugins

OVOS supports different wake word detection plugins, each with its own strengths and use cases

The default OVOS plugins are:

**[ovos-ww-plugin-precise-lite](<https://github.com/OpenVoiceOS/ovos-ww-plugin-precise-lite>)**

: A model-based plugin that uses a trained machine learning model to detect wake words.

**[ovos-ww-plugin-vosk](<https://github.com/OpenVoiceOS/ovos-ww-plugin-vosk>)**

: A text-based plugin leveraging Vosk, which allows you to define a wake word without requiring a trained model. This is useful during the initial stages of data collection.

Each plugin has its pros and cons, with Vosk offering a faster setup for simple wakeword recognition without model training.

## Wakeword Configuration

The `hotwords` section in your `mycroft.conf` allows you to configure the wakeword detection parameters for each plugin. For instance:

```
"hotwords": {  
  "hey_mycroft": {  
    "module": "ovos-ww-plugin-precise-lite",  
    "model": "https://github.com/OpenVoiceOS/precise-lite-models/raw/master/wakewords/en/h",  
    "expected_duration": 3,  
    "trigger_level": 3,  
    "sensitivity": 0.5,  
    "listen": true  
  }  
}
```

> see the full docs for the [listener service](#)

## Tips and Caveats

**Vosk Plugin:** The Vosk plugin is useful when you need a simple setup that doesn't require training a wake word model. It's great for quickly gathering data during the development stage.

**Precision and Sensitivity:** Adjust the `sensitivity` and `trigger_level` settings carefully. Too high a sensitivity can lead to false positives, while too low may miss detection.

## Plugin Development

### Key Methods

When developing a custom wake word plugin, the following methods are essential:

**'found\_wake\_word(frame\_data)'**: This method must be defined. It checks whether a wake word is found in the provided audio data.

**'update(chunk)'**: An optional method for processing live audio chunks and making streaming predictions.

**'stop()'**: An optional method to shut down the plugin, like unloading data or halting external processes.

> `found_wake_word(frame_data)` should ignore `frame_data`, this has been deprecated and is only provided for backwards-compatibility. Plugins are now

expected to handle real time audio via update method

## Registering Your Plugin

To integrate your custom plugin, add it to OVOS via the following entry point:

```
setup([...], entry_points={'mycroft.plugin.wake_word': 'example_wake_word_plugin = my_exam
```

## Example Plugin

Heres a simple implementation of a wake word plugin:

```
from ovos_plugin_manager.templates.hotwords import HotWordEngine
from threading import Event

class MyWWPlugin(HotWordEngine):
    def __init__(self, key_phrase="hey mycroft", config=None, lang="en-us"):
        super().__init__(key_phrase, config, lang)
        self.detection = Event()
        self.engine = MyWW(key_phrase)

    def found_wake_word(self, frame_data):
        # NOTE: frame_data should be ignored, it is deprecated
        # inference happens via the self.update_method
        detected = self.detection.is_set()
        if detected:
            self.detection.clear()
        return detected

    def update(self, chunk):
        if self.engine.found_it(chunk):
            self.detection.set()

    def stop(self):
        self.engine.bye()
```

**docs/313-stt\_plugins.md**

# STT Plugins

STT plugins are responsible for converting spoken audio into text

## STT

The base STT, this handles the audio in "batch mode" taking a complete audio file, and returning the complete transcription.

Each STT plugin class needs to define the `execute()` method taking two arguments:

`audio` \([AudioData](#) object\) - the audio data to be transcribed.

`lang` \(\(str\) - `_optional_` - the BCP-47 language code

The bare minimum STT class will look something like

```
from ovos_plugin_manager.templates.stt import STT

class MySTT(STT):
    def execute(audio, language=None):
        # Handle audio data and return transcribed text
        [...]
        return text
```

## StreamingSTT

A more advanced STT class for streaming data to the STT. This will receive chunks of audio data as they become available and they are streamed to an STT engine.

The plugin author needs to implement the `create_streaming_thread()` method creating a thread for handling data sent through `self.queue`.

The thread this method creates should be based on the `StreamThread` class.

`handle_audio_data()` method also needs to be implemented.

## Entry point

To make the class detectable as an STT plugin, the package needs to provide an entry point under the `mycroft.plugin.stt` namespace.



```

setup([...],
      entry_points = {'mycroft.plugin.stt': 'example_stt = my_stt:mySTT'}
)

```

Where `example_stt` is the STT module name for the plugin, `my\_stt` is the Python module and `mySTT` is the class in the module to return.

## List of STT plugins

| Plugin   |  |                   |   |
|--|--|-------------------|---|
| Offline   Streaming   Type   |  |                   |   |
| ----- ----- -----  |  |                   |   |
| ----- ----- -----  |  |                   |   |
| ----- ----- -----  |  |                   |   |
| ----- ----- -----  |  |                   |   |
| <a href="#">ovos-stt-plugin-fasterwhisper</a>                          |  |                   |   |
|  |  | FOSS              | <a href="#">ovos-stt-plugin-whispercpp</a>          |
|  |  | FOSS              | <a href="#">ovos-stt-plugin-vosk</a>                |
|  |  | FOSS              | <a href="#">ovos-stt-plugin-chromium</a>            |
|  |  | API (free)        | <a href="#">ovos-stt-plugin-http-server</a>         |
|  |  | API (self hosted) | <a href="#">ovos-stt-plugin-pocketsphinx</a>        |
|  |  | FOSS              | <a href="#">ovos-stt-azure-plugin</a>               |
|  |  | API (key)         | <a href="#">!imagem</a>                             |
| <a href="#">neon-stt-plugin-google_cloud_streaming</a>       API (key) |  |                   |   |
| <a href="#">!imagem neon-stt-plugin-nemo</a>                           |  |                   |   |
|  |  | FOSS              | <a href="#">!imagem neon-stt-plugin-nemo-remote</a> |
|  |  | API (self hosted) |   |

## Standalone Usage

STT plugins can be used in your own projects as follows

```
from speech_recognition import Recognizer, AudioFile

plug = STTPlug()

# verify lang is supported
lang = "en-us"
assert lang in plug.available_languages

# read file
with AudioFile("test.wav") as source:
    audio = Recognizer().record(source)

# transcribe AudioData object
transcript = plug.execute(audio, lang)
```

## Plugin Template

```

from ovos_plugin_manager.templates.stt import STT

# base plugin class
class MySTTPlugin(STT):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # read config settings for your plugin
        lm = self.config.get("language-model")
        hmm = self.config.get("acoustic-model")

    def execute(self, audio, language=None):
        # TODO - convert audio into text and return string
        transcript = "You said this"
        return transcript

    @property
    def available_languages(self):
        """Return languages supported by this STT implementation in this state
        This property should be overridden by the derived class to advertise
        what languages that engine supports.
        Returns:
            set: supported languages
        """
        # TODO - what langs can this STT handle?
        return {"en-us", "es-es"}

# sample valid configurations per language
# "display_name" and "offline" provide metadata for UI
# "priority" is used to calculate position in selection dropdown
#     0 - top, 100-bottom
# all other keys represent an example valid config for the plugin
MySTTConfig = {
    lang: [{"lang": lang,
            "display_name": f"MySTT ({lang})",
            "priority": 70,
            "offline": True}]
    for lang in ["en-us", "es-es"]
}

```

# docs/320-tts\_plugins.md

## TTS Plugins

TTS plugins are responsible for converting text into audio for playback

## TTS

All Mycroft TTS plugins need to define a class based on the TTS base class from `ovos_plugin_manager`

```
from ovos_plugin_manager.templates.tts import TTS

class myTTS(TTS):
    def __init__(self, lang, config):
        super().__init__(lang, config, validator, audio_ext='wav',
                        phonetic_spelling=False, ssml_tags=None)
        # Any specific init code goes here
```

The `super()` call does some setup adding specific options to how Mycroft will preprocess the sentence.

`audio_ext`: filetype of output, possible options 'wav' and 'mp3'.

`phonetic_spelling`, True if Mycroft should preprocess some difficult to pronounce words (eg spotify) or provide the raw text to the TTS.

`ssml_tags`: list of valid SSML tags for the TTS if any, otherwise None.

`validator`: a special class that verifies that the TTS is working in the current configuration.

The `get_tts()` method will be called by Mycroft to generate audio and (optionally) phonemes. This is the main method that the plugin creator needs to implement. It is called with:

`sentence` (str): a piece of text to turn into audio.

`wav_file` (str): where the plugin should store the generated audio data.

This method should generate audio data and return a Tuple (wav\_file, visemes):

wav\_file \(\str\): path to written data \(\text{generally the input argument}\)

phonemes \(\text{list}\): phoneme list for synthesized audio

## TTS Validator

To check if the TTS can be used, a validator class is needed. This should inherit from `mycroft.tts.TTSValidator`.

It will be called with the TTS class as argument and will store it in `self.tts`.

The following is the bare minimum implementation:

```
class MyValidator(TTSValidator):
    def get_tts_class(self):
        # Should return a reference to the TTS class it's intended to validate.

    def validate_lang(self):
        # Raise exception if 'self.tts.lang' is not supported.

    def validate_connection(self):
        # Check that the software needed for the TTS is reachable,
        # be it a local executable, python module or remote server and
        # if not available raise an exception.
```

> NOTE: TTSValidator is optional

## Entry point

To make the class detectable as an TTS plugin, the package needs to provide an entry point under the `mycroft.plugin.tts` namespace.

```
setup([...],
      entry_points = {'mycroft.plugin.tts': 'example_tts = my_tts:myTTS'}
)
```

Where `example_tts` is the TTS module name for the plugin, `my_tts` is the Python module and `myTTS` is the class in the module to return.

# List of TTS plugins

| Plugin  |  |                   |   |                        |
|---|--|-------------------|---|------------------------|
| Streaming   |  | Offline           |   | Type                   |
| -----   |  |                   |   |                        |
| -----   |  |                   |   |                        |
| -----   |  | -----             |   | -----                  |
| --  |  |                   |   |                        |
| <a href="#">ovos-tts-plugin-mimic</a>                     |  |                   |   |                        |
|   |  | FOSS              | <a href="#">ovos-tts-plugin-mimic3</a>          |                        |
|   |  | FOSS              | <a href="#">ovos-tts-plugin-piper</a>           |                        |
|   |  | FOSS              | <a href="#">ovos-tts-plugin-marytts</a>         |                        |
|   |  | API (self hosted) | <a href="#">ovos-tts-server-plugin</a>          |                        |
|   |  | API (self hosted) | <a href="#">ovos-tts-plugin-pico</a>            |                        |
|   |  | FOSS              | <a href="#">ovos-tts-plugin-edge-tts</a>        |                        |
|   |  | API (free)        | <a href="#">ovos-tts-plugin-polly</a>           |                        |
|   |  | API (key)         | <a href="#">ovos-tts-plugin-voicerss</a>        |                        |
|   |  | API (key)         | <a href="#">ovos-tts-plugin-google-TX</a>       |                        |
|   |  | API (free)        | <a href="#">ovos-tts-plugin-responsivevoice</a> |                        |
|   |  | API (free)        | <a href="#">ovos-tts-plugin-espeakNG</a>        |                        |
|   |  | FOSS              | <a href="#">ovos-tts-plugin-cotovia</a>         |                        |
|   |  | FOSS              | <a href="#">ovos-tts-plugin-SAM</a>             |                        |
|   |  | Abandonware       | <a href="#">ovos-tts-plugin-beepspeak</a>       |                        |
|   |  | Fun               | <a href="#">!imagem</a>                         |                        |
| <a href="#">neon-tts-plugin-larynx_server</a>             |  |                   |   | API (self hosted)    ! |
| <a href="#">imagem neon-tts-plugin-coqui</a>              |  |                   |   | FOSS                   |
| <a href="#">!imagem neon-tts-plugin-coqui-remote</a>      |  |                   |   | API (self              |
| hosted)    <a href="#">!imagem neon-tts-plugin-glados</a> |  |                   |   |                        |
| FOSS  |  |                   |   |                        |

## Standalone Usage

TODO

## Plugin Template

```

    for lang in ["en-us", "es-es"]
from ovos_plugin_manager.templates.tts import TTS
}

# base plugin class
class MyTTSPugin(TTS):
    def __init__(self, *args, **kwargs):
        # in here you should specify if your plugin return wav or mp3 files
        # you should also specify any valid ssml tags
        ssml_tags = ["speak", "s", "w", "voice", "prosody",
                     "say-as", "break", "sub", "phoneme"]
        super().__init__(*args, **kwargs, audio_ext="wav", ssml_tags=ssml_tags)
        # read config settings for your plugin if any
        self.pitch = self.config.get("pitch", 0.5)

    def get_tts(self, sentence, wav_file):
        # TODO - create TTS audio @ wav_file (path)
        return wav_file, None

    @property
    def available_languages(self):
        """Return languages supported by this TTS implementation in this state
        This property should be overridden by the derived class to advertise
        what languages that engine supports.
        Returns:
            set: supported languages
        """
        # TODO - what langs can this TTS handle?
        return {"en-us", "es-es"}

# sample valid configurations per language
# "display_name" and "offline" provide metadata for UI
# "priority" is used to calculate position in selection dropdown
# 0 - top, 100-bottom
# all other keys represent an example valid config for the plugin
MyTTSConfig = {
    lang: [{"lang": lang,
            "display_name": f"MyTTS ({lang})",
            "priority": 70,
            "offline": True}]

```

# docs/321-g2p\_plugins.md

## Grapheme to Phoneme Plugins

Grapheme to Phoneme is the process of converting text into a set of "sound units" called phonemes

In `ovos-audio` these plugins are used to auto generate mouth movements / visemes in the TTS stage.

They can also be used to help configuring wake words or to facilitate training of TTS systems

These plugins can provide phonemes either in ARPA or IPA alphabets, an automatic conversion will happen behind the scenes when needed

## Visemes

Visemes are representations of the shape of a human mouth when speaking.

[!visemes](#)

Mouth movements are generated via a mapping of ARPA to VISEMES, TTS plugins may provide this natively, or a G2P plugin may be used to estimate it directly from text

Visemes are predefined mouth positions, timing per phonemes is crucial for a natural mouth movement

The Mycroft Mark 1 uses this to make his "lips" match his speech.

OpenVoiceOS uses six basic visemes.

|                  |                |             |      |             |        |
|------------------|----------------|-------------|------|-------------|--------|
| viseme           | mouth position | ----- ----- | 0    | wide open   |        |
| 1                | pursed         | 2           | open | 3           | narrow |
| lips             | 4              | closed lips | 5    | parted lips | 6      |
| barely open lips |                |             |      |             |        |

Mapping based on [Jeffers phoneme to viseme map, seen in table 1](#), partially based on the "12 mouth shapes visuals seen [here](#)



# List of G2P plugins

| Plugin                                  | Type                            | Duration |
|---|---------------------------------|----------|
| -----                                   | -----                           | -----    |
| ----- ----- -----                       |                                 |          |
| ovos-g2p-plugin-mimic                   |                                 |          |
| ARPA                                    | ovos-g2p-plugin-heuristic-arpa  |          |
| ARPA                                    | ovos-g2p-plugin-espeak          |          |
| IPA                                     | !imagem neon-g2p-cmudict-plugin | ARPA     |
| !imagem neon-g2p-phoneme-guesser-plugin | ARPA                            | !imagem  |
| neon-g2p-gruut-plugin                   | IPA                             |          |

## Standalone Usage

All G2P plugins can be used as follows

```

utterance = "hello world"
word = "hello"
lang = "en-us"

plug = G2pPlugin()

# convert a word into a list of phonemes
phones = plug.get_ipa(word, lang)
assert phones == ['h', '', 'l', 'o']

phones = plug.get_arpa(word, lang)
assert phones == ['HH', 'AH', 'L', 'OW']

# convert a utterance into a list of phonemes
phones = plug.utterance2arpa(utterance, lang)
assert phones == ['HH', 'AH', 'L', 'OW', '.', 'W', 'ER', 'L', 'D']

phones = plug.utterance2ipa(utterance, lang)
assert phones == ['h', '', 'l', 'o', '.', 'w', '', 'l', 'd']

# convert a utterance into a list of viseme, duration pairs
visemes = plug.utterance2visemes(utterance, lang)
assert visemes == [('0', 0.0775), ('0', 0.155), ('3', 0.2325), ('2', 0.31), ('2', 0.434),
                   ('3', 0.806)]

```

## Plugin Template

```

from ovos_plugin_manager.templates.g2p import Grapheme2PhonemePlugin
from ovos_utils.lang.visimes import VISIMES

# base plugin class
class MyARPAG2PPlugin(Grapheme2PhonemePlugin):
    def __init__(self, config=None):
        self.config = config or {}

    def get_arpa(self, word, lang, ignore_oov=False):
        phones = [] # TODO implement
        return phones

    def get_durations(self, utterance, lang="en", default_dur=0.4):
        words = utterance.split()
        phones = [self.get_arpa(w, lang) for w in utterance.split()]
        dur = default_dur # TODO this is plugin specific
        return [(pho, dur) for pho in phones]

    def utterance2visemes(self, utterance, lang="en", default_dur=0.4):
        phonemes = self.get_durations(utterance, lang, default_dur)
        return [(VISIMES.get(pho[0].lower(), '4'), float(pho[1]))
                for pho in phonemes]

```

If your plugin uses IPA instead of ARPA simply replace `get_arpa` with `get_ipa`

```

from ovos_plugin_manager.templates.g2p import Grapheme2PhonemePlugin
from ovos_utils.lang.visimes import VISIMES

# base plugin class
class MyIPAG2PPlugin(Grapheme2PhonemePlugin):
    def __init__(self, config=None):
        self.config = config or {}

    def get_ipa(self, word, lang, ignore_oov=False):
        phones = [] # TODO implement
        return phones

    def get_durations(self, utterance, lang="en", default_dur=0.4):
        # auto converted to arpa if ipa is implemented
        phones = [self.get_arpa(w, lang) for w in utterance.split()]
        dur = default_dur # TODO this is plugin specific
        return [(pho, dur) for pho in phones]

    def utterance2visemes(self, utterance, lang="en", default_dur=0.4):
        phonemes = self.get_durations(utterance, lang, default_dur)
        return [(VISIMES.get(pho[0].lower(), '4'), float(pho[1]))
                for pho in phonemes]

```

## docs/330-transformer\_plugins.md

# Transformer Plugins

Transformer plugins in Open Voice OS (OVOS) provide a flexible way to modify and enhance various types of data during processing. These plugins can transform audio data, text, metadata, and even dialog content.

## Audio Transformers

Audio transformers are designed to process and modify audio data. They can be used to detect languages from audio input or even decode data embedded within the

audio.

## Available Plugins

| Plugin   | Description  |
|--|--|
|  |  |
|  |  |
|  |  |
| <a href="#">ovos-audio-transformer-plugin-fasterwhisper</a>          | Detects language from audio to inform Speech-to-Text (STT) processing. |
| <a href="#">ovos-audio-transformer-plugin-speechbrain-langdetect</a> | Detects language from audio to inform STT.                             |
| <a href="#">ovos-audio-transformer-plugin-ggwave</a>                 | Decodes data over audio and emits bus events in response.              |

These plugins help automate language detection and data interpretation, which are especially useful for multilingual environments or when integrating specialized data streams into your voice assistant.

## Utterance Transformers

Utterance transformers modify the textual representation of speech, improving the quality of transcriptions and allowing for more advanced processing.

## Available Plugins

| Plugin  | Description   |
|---|---|
| Source  |   |
|   |   |
|   |   |
|   |   |
| <a href="#">ovos-utterance-normalizer</a>         | Normalizes text before it reaches the pipeline stage.   <a href="#">OpenVoiceOS/ovos-utterance-normalizer</a> |
| <a href="#">ovos-utterance-plugin-cancel</a>      | Cancels an utterance mid-transcription.   <a href="#">OpenVoiceOS/ovos-utterance-plugin-cancel</a>            |
| <a href="#">ovos-utterance-corrections-plugin</a> | Manually corrects bad transcriptions.   <a href="#">OpenVoiceOS/ovos-utterance-corrections-plugin</a>         |

ovos-utterance-translation-plugin | Automatically translates unsupported languages. | [OpenVoiceOS/ovos-bidirectional-translation-plugin](#) |

These plugins enhance the quality of speech recognition and allow real-time intervention for handling special cases, such as language translation or manual corrections.

## Metadata Transformers

Metadata transformers handle the transformation of metadata associated with audio or utterances. They help in structuring or enriching metadata for further use.

## Available Plugins

*Currently, no specific plugins are listed for metadata transformers.*

## Dialog Transformers

Dialog transformers modify conversational content, allowing you to rewrite speech or translate it into a different language before execution. These plugins are particularly useful for improving the interactivity and flexibility of voice-based dialogues.

## Available Plugins

| Plugin | Description |
|--------|-------------|
|--------|-------------|

|        |  |
|--------|--|
| Source |  |
|--------|--|

|  |  |
|--|--|
|  |  |
|--|--|

|       |       |
|-------|-------|
| ----- | ----- |
|-------|-------|

|       |       |
|-------|-------|
| ----- | ----- |
|-------|-------|

|       |  |
|-------|--|
| ----- |  |
|-------|--|

|                                       |  |
|---------------------------------------|--|
| ovos-dialog-transformer-openai-plugin | Rewrites speech with a large language model (LLM) before executing Text-to-Speech (TTS). |
|---------------------------------------|--|

|   |  |
|---|--|
| <a href="#">OpenVoiceOS/ovos-solver-plugin-openai-persona</a> |  |
|---|--|

|                                |  |
|--------------------------------|--|
| ovos-dialog-translation-plugin | Translates speech back into the user's language.   <a href="#">OpenVoiceOS/ovos-bidirectional-translation-plugin</a> |
|--------------------------------|--|

Dialog transformers enable more dynamic interactions, such as generating personalized responses or translating dialogues into multiple languages.

# TTS Transformers

TTS (Text-to-Speech) transformers allow you to apply various effects or modifications to the speech output generated by the assistant, such as sound effects or audio filtering.

## Available Plugins

| Plugin   | Description | Source |
|--|-------------|--------|
|  |             |        |
| -----  | -----       | -----  |
| -----  |             |        |
| ovos-tts-transformer-sox-plugin   Applies sound effects via <code>sox</code> (Sound eXchange).   <a href="#">OpenVoiceOS/ovos-tts-transformer-sox-plugin</a> |             |        |

These plugins are helpful for modifying the final audio output, such as adding special effects, changing pitch, or applying filters.

## Standalone Usage

*Details on standalone usage are coming soon.*

## Plugin Templates

*Details on plugin templates are coming soon.*

## Conclusion

Transformer plugins in OVOS offer versatile tools for transforming data at various stages of processing. Whether you're working with audio, text, metadata, or dialog, these plugins allow for a high degree of customization and enhancement. OVOS's flexible plugin system empowers developers to create powerful, tailored experiences for users. Stay tuned for more updates and templates to help you create your own custom plugins.

# docs/340-PHAL.md

# PHAL Platform/Hardware Abstraction Layer

The Platform/Hardware Abstraction Layer (PHAL) in OpenVoiceOS (OVOS) provides a flexible, plugin-based system for integrating hardware-specific and platform-level functionality.

---

## Usage Guide

PHAL plugins are loaded at runtime based on system compatibility and user configuration. You can:

- Install multiple PHAL plugins for system and hardware support.

- Rely on automatic hardware detection to load relevant plugins safely.

- Use AdminPHAL when elevated privileges are required.

---

## Technical Explanation

### PHAL Plugins

PHAL plugins dynamically extend your voice assistant's functionality by listening to events and integrating with system or hardware components. Examples include:

- System control:** Restart, shutdown, or factory reset via `ovos-PHAL-plugin-system`.

- Audio management:** Volume control with `ovos-PHAL-plugin-alsa`.

- Hardware support:** Mark 1 and Mark 2 integrations using hardware detection.

Plugins are validated before loading. For example, the `ovos-PHAL-plugin-mk2` checks for the presence of the SJ201 HAT before activating.

### AdminPHAL



AdminPHAL is a specialized version of PHAL that loads plugins with root privileges. This allows for deeper OS integration ideal for tasks like system configuration or device control. However, all admin plugins must be:

- Marked as admin in their entry point.

- Explicitly enabled in the config (`"enabled": true`).

- Carefully audited, as they can modify system state.

AdminPHAL and PHAL will not load each other's plugins.

---

## Developing a PHAL Plugin

PHAL plugins usually consist of a validator (to determine compatibility) and an event listener. Here's a minimal example:

```
from ovos_bus_client import Message
from ovos_plugin_manager.phal import PHALPlugin

class MyPHALPluginValidator:
    @staticmethod
    def validate(config=None):
        # Return False to prevent loading (e.g., missing hardware)
        return True

class MyPHALPlugin(PHALPlugin):
    validator = MyPHALPluginValidator

    def __init__(self, bus=None, config=None):
        super().__init__(bus=bus, name="ovos-PHAL-plugin-NAME", config=config)
        self.bus.on("my.event", self.handle_event)

    def handle_event(self, message):
        self.bus.emit(Message("my.event.response"))

    def shutdown(self):
        self.bus.remove("my.event", self.handle_event)
        super().shutdown()
```

More details on plugin packaging are available in the [OVOS Plugin Manager documentation](#).

---

## Choosing Between a PHAL Plugin and a Skill

Not sure whether to build a skill or a PHAL plugin? Here's a quick guideline:

Use **PHAL** for low-level system or hardware integration.

Use **skills** for voice interactions and user-facing features.

In some cases, both might be appropriate: a PHAL plugin for backend support and a skill as a frontend interface.

[!Should you use a skill or a PHAL plugin?](#)

---

## Available Plugins

| Plugin                                  |   |
|---|---|
| Description                             |   |
| -----                                   |   |
| ----- -----                             |   |
| <a href="#">ovos-PHAL-plugin-alsa</a>   | Volume control                                      |
| <a href="#">ovos-PHAL-plugin-system</a> | Reboot, shutdown, and                               |
| factory reset                           | <a href="#">ovos-PHAL-plugin-mk1</a>   Mycroft      |
| Mark 1 hardware integration             | <a href="#">ovos-PHAL-plugin-respeaker-2mic</a>     |
| Respeaker 2-mic HAT support             | <a href="#">ovos-PHAL-plugin-respeaker-4mic</a>     |
| Respeaker 4-mic HAT support             | <a href="#">ovos-PHAL-plugin-wifi-setup</a>         |
| Central Wi-Fi setup                     | <a href="#">ovos-PHAL-plugin-gui-network-client</a> |
| GUI-based Wi-Fi setup                   | <a href="#">ovos-PHAL-plugin-balena-wifi</a>        |
| Wi-Fi hotspot setup                     | <a href="#">ovos-PHAL-plugin-network-manager</a>    |
| Network Manager integration             | <a href="#">ovos-PHAL-plugin-ipgeo</a>              |
| Geolocation using IP address            | <a href="#">ovos-PHAL-plugin-gpsd</a>               |
| Geolocation using GPS                   | <a href="#">neon-phal-plugin-linear_led</a>         |
| LED control for Mycroft Mark 2          |   |

---

## Tips & Caveats

**Safe Defaults:** Most plugins are hardware-aware and won't load if the required device isn't detected.

**Admin Responsibility:** AdminPHAL plugins have full system access. Only use trusted sources and review their code.

**Extensibility:** PHAL is designed to grow with your system. Don't hesitate to build your own plugins for unique hardware.

---

## Related Documentation

[OVOS Plugin Manager \(OPM\) Docs](#) [Packaging Plugins](#)

[PHAL Plugins on GitHub](#)

--- By decoupling system and hardware features from core logic, PHAL makes OVOS more modular, secure, and adaptable to any platform from Raspberry Pi setups to full-featured smart assistants.

## docs/350-translation\_plugins.md

# Language Detection and Translation Plugins

Language detection and translation plugins in Open Voice OS (OVOS) enable the system to identify the language of text and translate it between different languages. These plugins are particularly useful in the context of [Universal Skills](#) and can be integrated with external tools like [solvers](#).

## Available Language Plugins

OVOS supports a variety of language detection and translation plugins, each with

different capabilities, such as language detection, text translation, offline functionality, and support for external APIs.

| Plugin   |           |         |   |           |
|--|-----------|---------|---|-----------|
| Detect   | Translate | Offline | Type  |           |
| -----  |           |         |   |           |
| -----  |           |         |   |           |
| -----  |           | -----   | -----   | -----     |
| -----  |           |         |   |           |
| <a href="#">ovos-translate-plugin-server</a>                 |           |         |   |           |
|  |           |         | API (self hosted)    <a href="#">ovos-translate-plugin-nllb</a> |           |
|  |           |         | FOSS  |           |
| <a href="#">ovos-lang-detector-fasttext-plugin</a>           |           |         |   |           |
|  |           |         | FOSS    <a href="#">ovos-lang-detect-ngram-lm</a>               |           |
|  |           |         | FOSS  |           |
| <a href="#">ovos-lang-detector-plugin-lingua-podre</a>       |           |         |   |           |
|  |           |         | FOSS  |           |
| <a href="#">ovos-lang-detector-plugin-voter</a>              |           |         |   |           |
|  |           |         | FOSS  |           |
| <a href="#">ovos-lang-detector-plugin-cld2</a>               |           |         |   |           |
|  |           |         | FOSS  |           |
| <a href="#">ovos-lang-detector-plugin-cld3</a>               |           |         |   |           |
|  |           |         | FOSS  |           |
| <a href="#">ovos-lang-detector-plugin-fastlang</a>           |           |         |   |           |
|  |           |         | FOSS  |           |
| <a href="#">ovos-lang-detector-plugin-langdetect</a>         |           |         |   |           |
|  |           |         | FOSS  |           |
| <a href="#">ovos-google-translate-plugin</a>                 |           |         |   |           |
|  |           |         | API (free)  |           |
| <a href="#">neon-lang-plugin-libretranslate</a>              |           |         |   | API (self |
| hosted)    <a href="#">neon-lang-plugin-amazon_translate</a> |           |         |   |           |
| API (key)  |           |         |   |           |

## Key Features of Language Plugins:

**Language Detection:** Plugins like `ovos-lang-detector-fasttext-plugin` automatically detect the language of the input text, which is crucial for multi-language support in voice assistants.

**Translation:** Plugins like `ovos-translate-plugin-nllb` and

`ovos-google-translate-plugin` can translate text from one language to another, enabling multilingual capabilities for OVOS.

**Offline Support:** Some plugins, such as `ovos-lang-detector-fasttext-plugin`, offer offline functionality, which is essential in environments where an internet connection may not be available.

**API-based Plugins:** Plugins like `ovos-translate-plugin-server` and `neon-lang-plugin-libretranslate` use external APIs and can be self-hosted or accessed with an API key for translation services.

## Standalone Usage

*TODO: Add standalone usage instructions for each plugin once ready.*

## Plugin Template

*TODO: Provide a template for developing custom language detection and translation plugins.*

By using these plugins, developers can easily integrate language detection and translation features into OVOS-based voice assistants, making it more versatile and capable of handling multiple languages.

# docs/360-solver\_plugins.md

## Solver Plugins

Solver plugins solve natural language queries, they define a unified api around specific kinds of questions and provide auto translation capabilities for language support

A plugin can define the language it works in, eg, wolfram alpha only accepts english input at the time of this writing

Bidirectional [translation](#) will be handled behind the scenes for other languages

Solvers are used by individual skills and by the [Persona Framework](#)

# Question Solvers

**NEW** in ovos-core version **0.0.8**

Given a free form natural language question, return an answer

Originally implemented for [Neon](#)

non-exhaustive reference table of question solver plugins

| plugin  |  |                            |
|---|--|----------------------------|
| description                                       |  | native language            |
| -----   |  |                            |
| ----- -----                                       |  |                            |
| ----- -----                                       |  |                            |
| <a href="#">ovos-solver-plugin-ddg</a>            |  | extract                    |
| keywords from query and search duck duck          | english  |                            |
| <a href="#">ovos-solver-plugin-wikipedia</a>      |  | extract keywords           |
| from query and search wikipedia                   | english  |                            |
| <a href="#">ovos-solver-plugin-wolfram-alpha</a>  |  | wolfram alpha              |
| spoken answers api                                | english  |                            |
| <a href="#">ovos-question-solver-wordnet</a>      | answer "what is" questions via wordnet         |                            |
| english   | <a href="#">ovos-solver-plugin-aiml</a>        |                            |
| AIML chatbot                                      | english  |                            |
| <a href="#">ovos-solver-plugin-rivescript</a>     |  | rivescript chatbot         |
| english   | <a href="#">ovos-solver-pandorabots-plugin</a> | old                        |
| school chatbots hosted around the web             | english  |                            |
| <a href="#">ovos-solver-plugin-openai-persona</a> |  | OpenAI API compatible LLMs |
| english   |  |                            |

## Example Usage - DuckDuckGo plugin

single answer

```
from skill_ovos_ddg import DuckDuckGoSolver

d = DuckDuckGoSolver()

query = "who is Isaac Newton"

# full answer
ans = d.spoken_answer(query)
print(ans)
# Sir Isaac Newton was an English mathematician, physicist, astronomer, alchemist, theologian, and
```

chunked answer, for conversational dialogs, ie "tell me more"

```

from skill_ovos_ddg import DuckDuckGoSolver

d = DuckDuckGoSolver()

query = "who is Isaac Newton"

# chunked answer
for sentence in d.long_answer(query):
    print(sentence["title"])
    print(sentence["summary"])
    print(sentence.get("img"))

# who is Isaac Newton
# Sir Isaac Newton was an English mathematician, physicist, astronomer, alchemist, the
# https://duckduckgo.com/i/ea7be744.jpg

# who is Isaac Newton
# He was a key figure in the philosophical revolution known as the Enlightenment.
# https://duckduckgo.com/i/ea7be744.jpg

# who is Isaac Newton
# His book Philosophi Naturalis Principia Mathematica, first published in 1687, establ
# https://duckduckgo.com/i/ea7be744.jpg

# who is Isaac Newton
# Newton also made seminal contributions to optics, and shares credit with German math
# https://duckduckgo.com/i/ea7be744.jpg

# who is Isaac Newton
# In the Principia, Newton formulated the laws of motion and universal gravitation tha
# https://duckduckgo.com/i/ea7be744.jpg

```

Auto translation, pass user language in context



```
from skill_ovos_ddg import DuckDuckGoSolver

d = DuckDuckGoSolver()

# bidirectional auto translate by passing lang context
sentence = d.spoken_answer("Quem e Isaac Newton", context={"lang": "pt"})
print(sentence)
# Sir Isaac Newton foi um matematico ingles, fisico, astronomo, alquimista, teologo e auto
```

Plugins are expected to implement the `get_xxx` methods and leave the user facing equivalents alone

```

from ovos_plugin_manager.templates.solvers import QuestionSolver
{
    "title": "optional",
    "summary": "speak this",
class MySolver(QuestionSolver):
    "img": "optional/path/or/url
    enable_tx = False # if True enables bidirectional translation
    }
    priority = 100
    :return:

    def __init__(self, config=None):
        steps = [
        config = config or {}
        { "title": "the question", "summary": "we forgot the question", "image": "404.j
        # set the "internal" language, defined by dev, not user
        { "title": "the answer", "summary": "but the answer is 42", "image": "42.jpg" }
        # this plugin internally only accepts and outputs english
        ]
        config["lang"] = "en"
        return steps
        super().__init__(config)

# expected solver methods to be implemented
def get_data(self, query, context):
    """
    query assured to be in self.default_lang
    return a dict response
    """
    return {"error": "404 answer not found"}

def get_image(self, query, context=None):
    """
    query assured to be in self.default_lang
    return path/url to a single image to accompany spoken_answer
    """
    return "http://stock.image.jpg"

def get_spoken_answer(self, query, context=None):
    """
    query assured to be in self.default_lang
    return a single sentence text response
    """
    return "The full answer is XXX"

def get_expanded_answer(self, query, context=None):
    """
    query assured to be in self.default_lang
    return a list of ordered steps to expand the answer, eg, "tell me more"

```

# Multiple Choice Solvers

**NEW** in `ovos-core` version **0.0.8**

given a question and multiple answers, select the best answer

non-exhaustive reference table of multiple choice solver plugins

| plugin |
|--------|
|--------|

| description |
|-------------|
|-------------|

| native language |
|-----------------|
|-----------------|

|  |
|--|
|  |
|--|

|  |
|--|
|  |
|--|

|  |
|--|
|  |
|--|

|  |
|--|
|  |
|--|

|  |
|--|
| <a href="#">ovos-choice-solver-bm25</a>   using <a href="#">Okapi BM25</a> ranking function to estimate the relevance of documents to a given search query |
|--|

Implementation

```

class MultipleChoiceSolver(AbstractSolver):
    """ select best answer from question + multiple choice
    handling automatic translation back and forth as needed"""

    # plugin methods to override
    @abc.abstractmethod
    def select_answer(self, query: str, options: List[str],
                      context: Optional[dict] = None) -> str:
        """
        query and options assured to be in self.default_lang
        return best answer from options list
        """
        raise NotImplementedError

```

## Evidence Solver

**NEW** in ovos-core version **0.0.8**

given a document and a question about it, select the best passage that answers the question

non-exhaustive reference table of evidence solver plugins

| plugin

| description

| native language |

|-----|

-----|-----|

-----|

-----|-----|

| [ovos-evidence-solver-bm25](#) | using [Okapi BM25](#) ranking function to estimate the relevance of documents to a given search query |

Implementation

```

class EvidenceSolver(AbstractSolver):
    """perform NLP reading comprehension task,
    handling automatic translation back and forth as needed"""

    # plugin methods to override
    @abc.abstractmethod
    def get_best_passage(self, evidence: str, question: str,
                        context: Optional[dict] = None) -> str:
        """
        evidence and question assured to be in self.default_lang
        returns summary of provided document
        """
        raise NotImplementedError

```

## Entailment Solver

**NEW** in ovos-core version **0.0.8**

Given a hypothesis and a premise, return True if the premise entails the hypothesis, False otherwise

```

class EntailmentSolver(AbstractSolver):
    """ select best answer from question + multiple choice
    handling automatic translation back and forth as needed"""

    # plugin methods to override
    @abc.abstractmethod
    def check_entailment(self, premise: str, hypothesis: str,
                        context: Optional[dict] = None) -> bool:
        """
        premise and hyopithesis assured to be in self.default_lang
        return Bool, True if premise entails the hypothesis False otherwise
        """
        raise NotImplementedError

```

## Summarization Solver

**NEW** in ovos-core version **0.0.8**

Given a document, return it's summary

non-exhaustive reference table of multiple choice solver plugins

| plugin  |                                       |
|---|---------------------------------------|
| description                                     | native language                       |
| -----   | -----                                 |
| ----- -----                                     | -----                                 |
| ----- -----                                     |                                       |
| <a href="#">ovos-summarizer-solver-wordfreq</a> | using word frequencies select the top |
| utterances                                      |                                       |

Implementation

```
class TldrSolver(AbstractSolver):
    """perform NLP summarization task,
    handling automatic translation back and forth as needed"""

    # plugin methods to override
    @abc.abstractmethod
    def get_tldr(self, document: str,
                 context: Optional[dict] = None) -> str:
        """
        document assured to be in self.default_lang
        returns summary of provided document
        """
        raise NotImplementedError
```

## docs/361-nlp\_plugins.md

### NLP plugins

Several NLP tasks are exposed as plugins, this allows to configure how to solve these tasks centrally

**NEW** in ovos-core version **0.0.8**

### Keyword Extraction

Extract keywords from utterances

```
| Plugin
| Description          |
|-----|
|-----|
| ovos-keyword-extractor-heuristic |
| ovos-keyword-extractor-rake    | nltk data dependent (stopwords) |
```

## Tokenization

Split utterances into tokens

```
| Plugin
| Description          |
|-----|
|-----|
| ovos-tokenization-plugin-quebrafrases | heuristic based tokenizer |
```

## Sentence Segmentation

Split utterances into sub-commands

```
| Plugin
| Description          |
|-----|
|-----|
| ovos-segmentation-plugin-quebrafrases | heuristic based sentence segmentation |
```

## Coreference Resolution

Replace coreferences (pronouns) with their entities

```
| Plugin
| Description          |
|-----|
|-----|
| ovos-coref-solver-heuristic | heuristic based coref solver      |
| ovos-classifiers-coref-solver | models trained with ovos-classifiers |
```

# Postag

| Plugin   | Description                          |
|--|--------------------------------------|
| -----  |                                      |
| ----- -----                                    |                                      |
| <a href="#">ovos-postag-plugin-nltk</a>        | using nltk default postag            |
| <a href="#">ovos-classifiers-postag-plugin</a> | models trained with ovos-classifiers |

## docs/370-ocp\_plugins.md

# OVOS Common Playback - Stream Extractor Plugins

OVOS Common Playback (OCP) Stream Extractor Plugins are designed to handle the extraction of playable streams and their associated metadata just before playback. This delegation allows skills to focus on their core functionality without having to worry about stream extraction, thus preventing additional latency during search or other operations. The relevant plugin is automatically invoked based on the **Stream Extractor Identifier (SEI)** or a matching URL pattern.

A SEI typically precedes the URI, which is used to access the stream. If the required plugin is missing, the corresponding request will be ignored.

## Available Plugins

Here are the key stream extractor plugins available in OVOS:

| Plugin  | Description           | Stream Extractor IDs (SEIs) | URL Pattern                     |
|---|-----------------------|-----------------------------|---------------------------------|
| ----- ----- ----- -----   |                       |                             |                                 |
| <a href="#">[ovos-ocp-rss-plugin](https://github.com/OpenVoiceOS/ovos-ocp-rss-plugin)</a>           | Handles RSS feed URLs | <code>rss//</code>          | <code>N/A</code>                |
| <a href="#">[ovos-ocp-bandcamp-plugin](https://github.com/OpenVoiceOS/ovos-ocp-bandcamp-plugin)</a> | Handles Bandcamp URLs | <code>bandcamp//</code>     | <code>"bandcamp." in url</code> |
| <a href="#">[ovos-ocp-youtube-plugin](https://github.com/OpenVoiceOS/ovos-ocp-youtube-p</a>         |                       |                             |                                 |



lugin)

| Handles YouTube URLs | youtube//, ydl//, youtube.channel.live// |  
"youtube.com/" in url or "youtu.be/" in url ||

**[ovos-ocp-m3u-plugin](https://github.com/OpenVoiceOS/ovos-ocp-m3u-plugin)** |

Handles .pls and .m3u file formats | m3u//, pls// | ".pls" in uri or ".m3u"  
in uri ||

**[ovos-ocp-news-plugin](https://github.com/OpenVoiceOS/ovos-ocp-news-plugin)**

| Handles dedicated news websites | news// | any([uri.startswith(url) for  
url in URL\_MAPPINGS]) |

Each plugin is designed to extract and process streams from specific types of  
content sources, ensuring seamless integration of services like YouTube,  
Bandcamp, RSS feeds, and more without introducing delays in skill interactions.

## Standalone Usage

*TODO: Instructions for using the plugins in a standalone setup*

## Plugin Template

*TODO: Template for creating a new stream extractor plugin*

## Summary

These plugins delegate the task of stream extraction to just before playback,  
relieving skills from the burden of handling it themselves and preventing latency  
during search or other tasks. They ensure OVOS can integrate various streaming  
services efficiently by using SEIs to identify the stream and process the  
corresponding URI automatically.

## docs/371-media\_plugins.md

## Media Playback Plugins

OVOS Media Plugins handle media playback, enabling OVOS to interact with popular  
streaming services and media players for audio, video, and remote control.

# Available Plugins

Here are the key media plugins available in OVOS:

| Plugin | Audio | Video | Web | Remote | Notes |

|-----|-----|-----|-----|-----|-----|

**[ovos-media-plugin-chromecast](<https://github.com/OpenVoiceOS/ovos-media-plugin-chromecast>)**

| | | | Extra: [cast\\_control](#) for MPRIS interface | |

**[ovos-media-plugin-spotify](<https://github.com/OpenVoiceOS/ovos-media-plugin-spotify>)**

| | | | Requires premium account  
Extra: [spotifyd](#) for native Spotify player |

Each plugin is designed for specific media platforms and devices, allowing OVOS to interact with popular streaming services and media players.

> ovos-media is a work in progress and has not yet been released, plugins support both ovos-audio and ovos-media

---

## ovos-media-plugin-spotify

The **ovos-media-plugin-spotify** allows OVOS to initiate playback on Spotify, enabling integration with OVOS systems.

> The [companion skill](#) is needed for voice search integration.

## Installation

To install the plugin, use the following command:

```
pip install ovos-media-plugin-spotify
```

> If you want to make the OVOS device itself a Spotify player, we recommend using [spotifyd](#).

## OAuth Setup

Currently, OAuth needs to be performed manually. After installing the plugin, run the following command:

```
$ ovos-spotify-oauth
```

This will prompt you to enter your Spotify developer credentials after you have created an application on [Spotify Developer Dashboard](#). Follow the instructions and enter the provided information.

Example output:

```
$ ovos-spotify-oauth
```

```
This script creates the token information needed for running spotify
with a set of personal developer credentials.
```

```
It requires the user to go to developer.spotify.com and set up a
developer account, create an "Application" and make sure to whitelist
"https://localhost:8888".
```

```
After you have done that enter the information when prompted and follow
the instructions given.
```

```
YOUR CLIENT ID: xxxxx
```

```
YOUR CLIENT SECRET: xxxxx
```

```
Go to the following URL: https://accounts.spotify.com/authorize?client_id=xxx&response_type=code
```

```
Enter the URL you were redirected to: https://localhost:8888/?code=.....
```

```
ocp_spotify oauth token saved
```

## Configuration

After OAuth setup, edit your `mycroft.conf` to expose your Spotify players.

Use the provided `ovos-spotify-autoconfigure` script to automatically configure all Spotify devices under your `mycroft.conf`:

```
$ ovos-spotify-autoconfigure
This script will auto configure ALL spotify devices under your mycroft.conf

    SPOTIFY PREMIUM is required!

    If you have not yet authenticated your spotify account, run 'ovos-spotify-oauth' f

Found device: OpenVoiceOS-TV

mycroft.conf updated!

# Legacy Audio Service:
{'backends': {'spotify-OpenVoiceOS-TV': {'active': True,
                                          'identifier': 'OpenVoiceOS-TV',
                                          'type': 'ovos_spotify'}}}

# ovos-media Service:
{'audio_players': {'spotify-OpenVoiceOS-TV': {'active': True,
                                              'aliases': ['OpenVoiceOS-TV'],
                                              'identifier': 'OpenVoiceOS-TV',
                                              'module': 'ovos-media-audio-plugin-spotify'}}}

---
```

## ovos-media-plugin-chromecast

The **ovos-media-plugin-chromecast** allows OVOS to initiate playback on Chromecast devices, enabling integration with OVOS systems.

## Installation

To install the plugin, use the following command:

```
pip install ovos-media-plugin-chromecast
```

### [!Chromecast Integration](#)

> If you want to control Chromecast playback externally, you can install [cast\\_control](#) to enable MPRIS interface integration.

# Configuration

Use the `ovos-chromecast-autoconfigure` script to automatically configure Chromecast devices under your `mycroft.conf`:

```
$ ovos-chromecast-autoconfigure
```

This script will discover Chromecast devices on your network and update `mycroft.conf` with the necessary configuration.

Example output:

```
$ ovos-chromecast-autoconfigure
Scanning...
  - Found Chromecast: Bedroom TV - 192.168.1.17:8009

Found devices: ['Bedroom TV']

mycroft.conf updated!

# Legacy Audio Service:
{'backends': {'chromecast-bedroom-tv': {'active': True,
                                         'identifier': 'Bedroom TV',
                                         'type': 'ovos_chromecast'}}}

# ovos-media Service:
{'audio_players': {'chromecast-bedroom-tv': {'active': True,
                                              'aliases': ['Bedroom TV'],
                                              'identifier': 'Bedroom TV',
                                              'module': 'ovos-media-audio-plugin-chromecast'},
                  'video_players': {'chromecast-bedroom-tv': {'active': True,
                                                                'aliases': ['Bedroom TV'],
                                                                'identifier': 'Bedroom TV',
                                                                'module': 'ovos-media-video-plugin-chromecast'}}}

---
```

## Summary

OVOS Media Plugins, like **ovos-media-plugin-spotify** and **ovos-media-plugin-chromecast**, provide seamless integration with popular media

platforms and devices, allowing you to control playback directly through OVOS. Whether it's streaming from Spotify, controlling Chromecast devices, or casting media, these plugins enhance the flexibility of the OVOS ecosystem for media playback.

## docs/399-intents.md

# Skills and Intents

At the heart of OVOS lies a powerful yet flexible **intent handling system** that enables voice-driven interaction. The system connects **user utterances** to **developer-defined behavior** through *intents*.

## Key Concept

**Skills register intent handlers.** In practice, this means that:

A **bus message** representing a user **intent** is **mapped to a specific piece of code** the **intent handler** within a skill.

When the system detects that an utterance matches a registered intent, it emits the relevant bus message, and the corresponding handler is invoked.

## How Intents Are Defined

Skill developers have **two main ways** to define intents:

### 1. Example Utterances

Developers write full example phrases that a user might say.

The engine learns patterns from these to match similar user utterances.

Example:

```
["what's the weather", "tell me the weather", "how's the forecast"]
```

## 2. Keyword Rules

Developers define combinations of required and optional **keywords**.

Rules are defined in a more structured way.

Example:

```
IntentBuilder("WeatherIntent")  
.require("weather_keyword")  
.optionally("location")
```

## How OVOS Handles This

The **OVOS Core** is responsible for interpreting user utterances and deciding **which (if any) intent** they match. This is done by comparing the input against the limited training data (example phrases or keyword rules) provided by skill developers.

## Modern Intent Pipelines

Historically:

**Adapt** was used for keyword-based matching.

**Padatious** was used for example-based matching.

These were inherited from Mycroft.

Now:

OVOS has evolved into a **highly configurable intent pipeline framework**.

Multiple intent engines can be used in **parallel or sequence**.

Skill developers and system integrators can choose or define:

Which engines to use

How to prioritize them

When to fall back or skip certain engines

## Example Flow:

1. User says: *"What's the weather like tomorrow in Lisbon?"*
2. OVOS pipelines the utterance through configured engines.
3. If an intent matches, a bus message like `intent:WeatherIntent` is emitted.
4. The matching skills handler for `WeatherIntent` is called with the parsed data.

## Summary

**Intent = Message + Handler**

Skills declare what they can handle; OVOS decides *when* to trigger them.

Intents are defined either via:

Full utterance examples

Structured keyword rules

Modern OVOS pipelines go beyond Padatious and Adapt, allowing advanced, modular configurations for intent parsing.

# docs/400-skill-design-guidelines.m d

## Voice User Interface Design Guidelines

Through these guidelines you will learn how to use principles of Voice User Interface Design to build more effective skills. These tools will help define and validate the features of the skill before diving deep into development.

This guide will cover some methods to use that can help plan, prototype and test



your skill during the early design stages.

> **CREDITS** - Voice User Interface Design Guidelines based on the [original work of Derick Schweppe](#)

## Interactions

### Intents

Let's start with an example. A user in Melbourne, Australia might want to know about the weather. To ask for this information, they might say:

> "Hey Mycroft, what's today's weather like?" > > "Hey Mycroft, what's the weather like in Melbourne?" > > "Hey Mycroft, weather"

Even though these are three different expressions, for most of us they probably have roughly the same meaning. In each case we would assume the user expects OVOS to respond with today's weather for their current location.

It is up to us as Skill creators to teach OVOS the variety of ways that a user might express the same intent. This is a key part of the design process. It is the key difference between a Skill that kind of works if you know what to say, and a Skill that feels intuitive and natural to talk to.

This is handled by an intent parser whose job it is to learn from your Skill what intents it can handle, and extract from the user's speech and key information that might be useful for your Skill. In this case it might include the specified date and location.

### Statements and Prompts

You can think of **Prompts** as questions and **Statements** as providing information to the user that does not need a follow-up response. For example a weather forecast like this would be considered a statement:

> Today's forecast is sunny with a high of 60 and a low of 45.

### Statements

For a lot of skills the conversation might end with a simple statement from OVOS, and no further action is necessary. Try to imagine what the user is trying to accomplish, if a simple statement gets the job done there is no reason to keep the conversation rolling, and in fact a follow-up might annoy the user with

unnecessary interaction.

## Prompts

It may be tempting to always give users specific instructions like traditional automated phone systems \(\Interactive Voice Response\). Many phone systems are notorious for being too verbose and difficult to follow.

With OVOS we were trying to break that mold and make the interaction natural. If you follow the phone system method you may be giving the user the exact phrasing to say, but you're also taking up valuable time and training them to think the system is very narrow in capability. In the event that the user does give a response that your skill can not handle, create follow-up prompts that steer the user back on track.

Remember, there are no errors in a cooperative conversation.

### Avoid

```
| Speaker   |  
|  
|:-----|:-----  
--|  
| Mycroft | _How many people are playing? For example, you can say 2 players._ |
```

### Better

```
| Speaker   |  
|:-----|:-----| | Mycroft | _How  
many players?_  
| | User   | _My Brother and Myself_  
| | Mycroft | _Im sorry, what was the number of players?_ | | User   | _Two_  
|
```

In the first example the user is told explicitly what they can say, but the prompt is unnatural, we don't typically suggest responses to a conversation partner in real life. These long-winded prompts can become tiresome if they are used repeatedly throughout the skill. Remember the phone system example, typically poorly designed automated phone systems inundate the user with many options and additional instructions at every step of the interaction.

In the second example we see a better prompt, although the user gives a response that is easy for a human to understand it is more difficult to design a skill to understand. Instead, the skill designer can apply a re-prompt strategy that steers the user back on track and doesn't require them to start the interaction over. In this case the re-prompt changes the wording slightly to make it clear

that the user needs to say a number. The next time the user interacts with this Skill, they will likely say a number in their first interaction.

Determining whether to respond to the user with a statement or a prompt can be a bit tricky. When the user is somewhat vague it is tempting to assume the users intent in order to speed along the interaction. Sometimes this is the right approach and sometimes it is not. If you are very confident in the users input then it is alright to assume, and respond with a statement. For example in the Wikipedia Skill the Wikipedia API is used to confidently select the best response.

### Wikipedia Skill

```
| Speaker   |
|
|:-----|:-----|
|-----|
|-----|
|-----|
|-----|
|
| User   | _Tell me about Abraham Lincoln_
| | Mycroft | _Abraham Lincoln was an American statesman and lawyer who served as
the 16th president of the UnitedStates from 1861 to 1865. Lincoln led the nation
through its greatest moral, constitutional, and political crisis in the American
Civil War_. |
```

In contrast, lets look at an example of where a follow-up prompt is a better approach. Youre working on a skill that gives important information about birds, such as wingspan, laden and unladen airspeed, etc..

### Avoid

```
| Speaker   | \\\
|
|:-----|:-----|
|-----|
|
| User   | _what is the airspeed velocity of an unladen swallow?_
| | Mycroft | _The unladen airspeed of the european swallow is 20.1 miles per
hour._ |
```

### Better

```
| Speaker   |
|
|:-----|:-----|
```

```

-----|
| User | _What is the airspeed velocity of an unladen Swallow?_
| | Mycroft | _What do you mean, African or European Swallow?_
| | User | European Swallow.
| | Mycroft | _The unladen airspeed of the European Swallow is 20.1 miles per
hour._ |

```

In the first example Mycroft assumes the user is referring to the European Swallow, however there is nothing to give confidence to that assumption. In this case its dangerous to assume the User meant the European Swallow since it is just as likely they meant the African Swallow. When there is no confidence in one answer versus another, OVOS should follow up with a prompt as demonstrated in the second example.

Another approach to solving this problem is to offer the most likely answer and inform the user that there are other answers available with a statement.

## Confirmations

Confirmation approaches can also be defined by Statements or Prompts , but when we talk about them in the context of confirmations we call them Implicit and Explicit.

## Implicit Confirmation

This type of confirmation is also a statement. The idea is to parrot the information back to the user to confirm that it was correct, but not require additional input from the user. The implicit confirmation can be used in a majority of situations. For example Mycrofts Timer Skill uses implicit confirmation when creating new timers.

### Timer Skill

```

| Speaker | |
|:-----|:-----| | User | _Hey
Mycroft set a timer for 5 minutes_ | | Mycroft | _Alright, Im starting a 5
minute timer_ |

```

### Avoid

```

| Speaker | |
|:-----|:-----| | User | _Hey
Mycroft, add bananas to my shopping list_ | | Mycroft | _done_

```

|

### Better

| Speaker |  
|:-----|:-----| | **User** | \_Hey  
Mycroft, add bananas to my shopping list\_ | | **Mycroft** | \_Adding bananas\_  
|

## Explicit Confirmation

This type of confirmation requires an input from the user to verify everything is correct. Only use explicit confirmation when needed as it can be tedious for the user to always confirm their input. This strategy is best when there are multiple steps, and it would be difficult to fix a misstep without starting over, or when deleting or modifying an existing multistep input.

### Avoid

| **Speaker** |  
|:-----|:-----|  
| **User** | \_Hey Mycroft, set an alarm for 9:00 am tomorrow\_ ||  
**Mycroft** | \_Ok, Ive set an alarm for 9:00 am tomorrow, is that correct?\_ ||  
**User** | \_Yes\_ ||  
**Mycroft** | \_9:00 am alarm set\_ |

### Better

| Speaker |  
|:-----|:-----| |  
**User** | \_Hey Mycroft, set an alarm for 9:00 am tomorrow\_ ||  
**Mycroft** | I've set an alarm for tomorrow at 9:00 am, 16 hours from now |

The first example requires the user to confirm the alarm. Since the interaction is quite short it isnt necessary to use explicit confirmation.

### Avoid

| **Speaker** |  
|  
|:-----|:-----|  
-----|  
| **User** | \_Hey Mycroft, set a calendar event for a meeting with Chris Stevens\_  
| | **Mycroft** | \_When is the event?\_  
| | **User** | \_Tomorrow at 11:00 am\_

```

|| Mycroft | _should I send an invite to Chris Stevens?_
|| User   | _Yes_
|| Mycroft | _Alright, I've created a meeting for Tomorrow_
|| Mycroft | _Alright, Ive created a meeting for Tomorrow at 11:00 am with
Chris Stevens_ || User   | _Cancel my 11:00 am meeting tomorrow_
|| Mycroft | _Done_
|

```

## Better

```

| Speaker   |
|
|:-----|:-----|
-----|
|| User   | _Hey Mycroft, set a calendar event for a meeting with Chris Stevens_
|| Mycroft | _When is the event?_
|| User   | _Tomorrow at 11:00 am_
|| Mycroft | _should I send an invite to Chris Stevens?_
|| User   | Yes
|| Mycroft | _Alright, Ive created a meeting for Tomorrow at 11:00 am with
Chris Stevens_ || User   | _Cancel my 11:00 am meeting tomorrow_
|| Mycroft | _Are you sure you want to delete your meeting tomorrow at 11:00
am?_          || User   | _Yes_
|

```

Both examples use explicit and implicit confirmation correctly until the end of the interaction when the user wants to cancel the event. Unlike setting a simple timer or an alarm creating this calendar event requires multiple steps and would require reasonable effort to recreate. There may also be significant consequences if the request was misinterpreted and the meeting was cancelled unintentionally.

## Conversations

Any time the user needs to input a lot of information or the user needs to sort through a variety of options a conversation will be needed. Users may be used to systems that require them to separate input into different chunks. For example, many personal navigation devices \(\text{GPS units}\) require users to input the street number, street name, city, and state separately, like this:

```

| Speaker           |
|:-----|:-----| | Navigation Device |

```

```

_What is the street number_ || User | _123_ |
| Navigation Device | _What is the street name_ || User | _Main
Street_ || Navigation Device | _What is the city_ ||
User | _Springfield_ || Navigation Device | _Oregon_
|

```

This was a limitation of the technology, both the STT and intent parser were not capable of handling an entire address. With OVOS you do not need to separate the address into multiple inputs. Its also easy to think like a traditional GUI and translate what are typically separate fields into prompts for the user, but that will only slow down the interaction.

The best solution is to allow the user to input information both ways, all at once, or in pieces. For example the Alarm skill allows the user to set a complicated alarm all in one go, or in steps.

### In Steps:

```

| Speaker | |
|:-----|:-----| || User |
_Hey Mycroft, set a recurring alarm_ || Mycroft | _For what day?_
|| User | _weekdays_ || Mycroft | _For
what time?_ || User | _7 am_
|| Mycroft | I've set a recurring Alarm for 7 am on weekdays |

```

### All at once:

```

| Speaker | |
|:-----|:-----| || User |
Hey Mycroft, set a recurring alarm for weekdays at 7 am || Mycroft | I've set a
recurring Alarm for 7 am on weekdays |

```

## Context

Conversational context is a given when speaking to another person, but not always when interacting with a computer based system. Many skills are designed with only a quick back and forth interaction in mind, and can operate well without conversational context, but conversational context can be a powerful addition to a skill and can make the interaction much more life like. Here are some examples of how to use context in conversation.

```

| Speaker | |
|
|:-----|:-----|

```

```

-----|
-----|
| User | _Who is Shaquille O'Neal_
| | Mycroft | _Shaquille O'Neal is an American former professional basketball
player who is a sports analyst on the television program Inside the NBA on TNT.
He is considered one of the greatest players in National Basketball Association
history_ | | User | _How tall is he?_
| | Mycroft | _seven foot 1 inches tall_
| | User | _when did he retire_
| | Mycroft | _2011_
|

```

## Error Handling

Inevitably, the user will say something that your skill cant handle. Its best not to think of these as errors on the part of the user, remember there arent really errors in conversations. Conversations are naturally cooperative with both parties seeking to continue to a successful outcome. Keeping that in mind you should do your best to avoid Error Handling that is a dead-end.

### Avoid

```

| Speaker |
|:-----|:-----| | User | _Timer_
| | Mycroft | _I'm sorry I can't help you with that_ |

```

### Better

```

| Speaker | |:-----|:-----| |
User | _Timer_ | | Mycroft | _A timer for how long?_ | | User
| _5 minutes_ |

```

In the first example OVOS does not give the user any options to finish the job they set out to do. Try to avoid situations where the user has to start the conversation over.

## docs/401-skill\_structure.md

## Anatomy of a Skill



## vocab, dialog, and locale directories

The `dialog`, `vocab`, and `locale` directories contain subdirectories for each spoken language the skill supports. The subdirectories are named using the [IETF language tag](#) for the language. For example, Brazilian Portuguese is 'pt-br', German is 'de-de', and Australian English is 'en-au'.

`dialog` and `vocab` have been **deprecated**, they are still supported, but we strongly recommend you use `locale` for new skills

inside the `locale` folder you will find subfolders for each language (e.g. `en-us`), often all you need to do in order to translate a skill is adding a new folder for your language here

each language folder can have the structure it wants, you may see files grouped by type in subfolder or all in the base folder

You will find several unfamiliar file extensions in this folder, but these are simple text files

- `.dialog` files used for defining speech responses

- `.intent` files used for defining Padatious Intents

- `.voc` files define keywords primarily used in Adapt Intents

- `.entity` files define a named entity primarily used in Padatious Intents

## \_\_init\_\_.py

The `__init__.py` file is where most of the Skill is defined using Python code.

## Importing libraries

```
from ovos_workshop.intents import IntentBuilder
from ovos_workshop.decorators import intent_handler
from ovos_workshop.skills import OVOSkill
```

This section of code imports the required `_libraries_`. Some libraries will be required on every Skill, and your skill may need to import additional libraries.

## Class definition

The class definition extends the `OVOSSkill` class:

```
class HelloWorldSkill(OVOSSkill):
```

The class should be named logically, for example "TimeSkill", "WeatherSkill", "NewsSkill", "IPAddressSkill". If you would like guidance on what to call your Skill, please join the [skills Channel on OVOS Chat](#).

Inside the class, methods are then defined.

## `__init__()`

This method is the `_constructor_`. It is called when the Skill is first constructed. It is often used to declare state variables or perform setup actions, however it cannot fully utilise `OVOSSkill` methods as the skill is not fully initialized yet at this point.

**You usually don't have to include the constructor.**

An example `__init__` method might be:

```
def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)
    self.already_said_hello = False
    self.be_friendly = True
```

`__init__` method must accept at least `skill_id` and `bus` kwargs and pass them to `super()`, we recommend passing `*args, **kwargs` like in example above instead

**NOTE:** `self.skill_id`, `self.filesystem`, `self.settings`, `self.bus` are only available after the call to `super()`, if you need them consider using `initialize` instead

## `initialize()`

This method is called during `__init__`, if you implemented `__init__` in your skill it will be called during `super()`

Perform any final setup needed for the skill here. This function is invoked after the skill is fully constructed and registered with the system. Intents will be registered and Skill settings will be available.

If you need to access `self.skill_id`, `self.bus`, `self.settings` or `self.filesystem` you must do it here instead of `__init__`

```
def initialize(self):
    my_setting = self.settings.get('my_setting')
```

## @intent\_handler

We can use the `initialize` function to manually register intents, however the `@intent_handler` decorator is a cleaner way to achieve this. We will learn all about the different Intents shortly.

In skills we can see two different intent styles.

1. An Adapt handler, triggered by a keyword defined in a `ThankYouKeyword.voc` file.

```
@intent_handler(IntentBuilder('ThankYouIntent').require('ThankYouKeyword'))
def handle_thank_you_intent(self, message):
    self.speak_dialog("welcome")
```

1. A Padatious intent handler, triggered using a list of sample phrases.

```
@intent_handler('HowAreYou.intent')
def handle_how_are_you_intent(self, message):
    self.speak_dialog("how.are.you")
```

In both cases, the function receives two `_parameters_`:

`self` - a reference to the `HelloWorldSkill` object itself

`message` - an incoming message from the `messagebus`.

Both intents call the `self.speak_dialog()` method, passing the name of a dialog file to it. In this case `welcome.dialog` and `how.are.you.dialog`.

## stop()

You will usually also have a `stop()` method.

The `stop` method is called anytime a User says "Stop" or a similar command. It is useful for stopping any output or process that a User might want to end without needing to issue a Skill specific utterance such as media playback or an expired

alarm notification.

In the following example, we call a method `stop_beeping` to end a notification that our Skill has created.

If the skill "consumed" the stop signal it should return `True`, else return `False`.

```
def stop(self):
    if self.beeping:
        self.stop_beeping()
        return True
    return False
```

If a Skill has any active functionality, the `stop()` method should terminate the functionality, leaving the Skill in a known good state.

When the skill returns `True` no other skill will be stopped, when it returns `False` the next active skill will attempt to stop and so on until something consumes the stop signal

## shutdown()

The `shutdown` method is called during the Skill process termination. It is used to perform any final actions to ensure all processes and operations in execution are stopped safely. This might be particularly useful for Skills that have scheduled future events, may be writing to a file or database, or that have initiated new processes.

In the following example we cancel a scheduled event and call a method in our Skill to stop a subprocess we initiated.

```
def shutdown(self):
    self.cancel_scheduled_event('my_event')
    self.stop_my_subprocess()
```

## settingsmeta.yaml

This file defines the settings UI that will be available to a User through a backend or companion app

Jump to Skill Settings for more information on this file and handling of Skill settings.

# setup.py

This file allows a skill to be installed just like any other python package. This means you can publish your skill on pypi or favorite package manager and use it as a dependency

A typical setup.py file looks like this 'python #!/usr/bin/env python3 from setuptools import setup import os from os import walk, path

# TODO update this info!

# Define package information SKILL\_CLAZZ = "MySkill" # Make sure it matches

\_\_init\_\_.py class name VERSION = "0.0.1" URL =

"https://github.com/authorName/ovos-skill-name" AUTHOR = "authorName" EMAIL = ""

LICENSE = "Apache2.0" DESCRIPTION = "a skill for OVOS"

PYPI\_NAME = URL.split("/)[-1] # pip install PYPI\_NAME

# Construct entry point for plugin SKILL\_ID =

f"{PYPI\_NAME.lower()}.{AUTHOR.lower()}" SKILL\_PKG =

PYPI\_NAME.lower().replace('-', '\_') PLUGIN\_ENTRY\_POINT =

f"{SKILL\_ID}={SKILL\_PKG}:{SKILL\_CLAZZ}"

def get\_requirements(requirements\_filename: str): """ Parse requirements from a file.

Args: requirements\_filename (str, optional): The filename of the requirements file. Defaults to "requirements.txt".

Returns: List[str]: A list of parsed requirements.

Notes: If the environment variable MYCROFT\_LOOSE\_REQUIREMENTS is set, this function will modify the parsed requirements to use loose version requirements, replacing '==' with '>=' and '~=' with '>='.

""" requirements\_file = path.join(path.abspath(path.dirname(\_\_file\_\_)), requirements\_filename) with open(requirements\_file, 'r', encoding='utf-8') as r: requirements = r.readlines() requirements = [r.strip() for r in requirements if r.strip() and not r.strip().startswith("#")] if 'MYCROFT\_LOOSE\_REQUIREMENTS' in os.environ: print('USING LOOSE REQUIREMENTS!') requirements = [r.replace('==', '>=').replace('~=', '>=') for r in requirements] return requirements

def find\_resource\_files(): """ensure all non-code resource files are included in the package""" # add any folder with files your skill uses here!

resource\_base\_dirs = ("locale", "ui", "vocab", "dialog", "regex") base\_dir =

path.dirname(\_\_file\_\_) package\_data = [".json"] for res in resource\_base\_dirs: if

```
path.isdir(path.join(base_dir, res)): for (directory, _, files) in
walk(path.join(base_dir, res)): if files: package_data.append(
path.join(directory.replace(base_dir, "").lstrip('/'), *)) return package_data

# Setup configuration setup( name=PYPI_NAME, version=VERSION,
description=DESCRIPTION, url=URL, author=AUTHOR, author_email=EMAIL,
license=LICENSE, package_dir={SKILL_PKG: ""}, package_data={SKILL_PKG:
find_resource_files()}, packages=[SKILL_PKG], include_package_data=True,
install_requires=get_requirements("requirements.txt"), keywords='ovos skill
plugin', entry_points={'ovos.plugin.skill': PLUGIN_ENTRY_POINT} ) ‘
```

## docs/402-statements.md

# Statements

## Speaking a statement

One of OVOS's most important core capabilities is to convert text to speech, that is, to speak a statement.

Within a Skill's Intent handler, you may pass a string of text to OVOS and OVOS will speak it. For example: `self.speak('this is my statement')`. That's cool and fun to experiment with, but passing strings of text to Mycroft doesn't help to make Mycroft a multilingual product. Rather than hard-coded strings of text, OVOS has a design pattern for multilingualism.

## Multilingualism

To support multilingualism, the text that OVOS speaks must come from a file. That file is called a dialog file. The dialog file contains statements (lines of text) that a listener in a particular language would consider to be equivalent. For instance, in USA English, the statements "I am okay" and "I am fine" are equivalent, and both of these statements might appear in a dialog file used for responding to the USA English question: "How are you?".

By convention, the dialog filename is formed by `_dot connected_ _words_` and must end with `".dialog"`. The dialog filename should be descriptive of the contents as a whole. Sometimes, the filename describes the question being answered, and

other times, the filename describes the answer itself. For the example above, the dialog filename might be: **how.are.you.dialog** or **i.am.fine.dialog**.

Multilingualism is accomplished by translating the dialog files into other languages, and storing them in their own directory named for the country and language. The filenames remain the same. Using the same filenames in separate language dependent directories allows the Skills to be language agnostic; no hard-coded text strings. Adjust the language setting for your Device and OVOS uses the corresponding set of dialog files. If the desired file does not exist in the directory for that language, Mycroft will use the file from the USA English directory.

As an example of the concept, the contents of **how.are.you.dialog** in the directory for the French language in France (fr-fr) might include the statement: "Je vais bien".

## The Tomato Skill Revisited

To demonstrate the multilingualism design pattern, we examine the usage of the `speak_dialog()` method in the Tomato Skill.

The Tomato Skill has two Intents: one demonstrates simple, straightforward statements, and the other demonstrates the use of variables within a statement.

### Simple statement

The first Intent within the Tomato Skill, **what.is.a.tomato.intent**, handles inquiries about tomatoes, and the dialog file, **tomato.description.dialog**, provides the statements for OVOS to speak in reply to that inquiry.

Sample contents of the Intent and dialog files:

```
what.is.a.tomato.intent  what is a tomato what would you say a tomato
is describe a tomato what defines a tomato
```

```
tomato.description.dialog  The tomato is a fruit of the nightshade
family A tomato is an edible berry of the plant Solanum lycopersicum A
tomato is a fruit but nutritionists consider it a vegetable
```

Observe the statements in the `tomato.description.dialog` file. They are all acceptable answers to the question: "What is a tomato?" Providing more than one statement in a dialog file is one way to make OVOS to seem less robotic, more natural. OVOS will randomly select one of the statements.

The Tomato Skill code snippet:

```
@intent_handler('what.is.a.tomato.intent')
def handle_what_is(self, message):
    """Speaks a statement from the dialog file."""
    self.speak_dialog('tomato.description')
```

With the Tomato Skill installed, if the User utters "Hey Mycroft, what is a tomato?", the Intent handler method `handle_what_is()` will be called.

Inside `handle_what_is()`, we find: `self.speak_dialog('tomato.description')`

As you can probably guess, the parameter `'tomato.description'` is the dialog filename without the `".dialog"` extension. Calling this method opens the dialog file, selects one of the statements, and converts that text to speech. OVOS will speak a statement from the dialog file. In this example, OVOS might say "The tomato is a fruit of the nightshade family".

Remember, OVOS has a language setting that determines from which directory to find the dialog file.

## File locations

The [Skill Structure](#) section describes where to place the Intent file and dialog file. Basically, there are two choices:

1. Put both files in `locale/en-us`
2. Put the dialog file in `dialog/en-us`, and put the Intent file in `vocab/en-us`

## Statements with variables

The second Padatious Intent, **do.you.like.intent**, demonstrates the use of variables in the Intent file and in one of the dialog files:

```
do.you.like.intent do you like tomatoes do you like {type} tomatoes
```

```
like.tomato.type.dialog I do like {type} tomatoes {type} tomatoes are my favorite
```

```
like.tomato.generic.dialog I do like tomatoes tomatoes are my favorite
```

Compare these two dialog files. The **like.tomato.generic.dialog** file contains



only simple statements. The statements in the **like.tomato.type.dialog** file include a variable named `type`. The variable is a placeholder in the statement specifying where text may be inserted. The `speak_dialog()` method accepts a dictionary as an optional parameter. If that dictionary contains an entry for a variable named in the statement, then the value from the dictionary will be inserted at the placeholder's location.

Dialog file variables are formed by surrounding the variable's name with curly braces. In OVOS parlance, curly braces are known as a `_mustache_`.

For multi-line dialog files, be sure to include the **same** variable on **all** lines.

The Tomato Skill code snippet:

```
@intent_handler('do.you.like.intent')
def handle_do_you_like(self, message):
    tomato_type = message.data.get('type')
    if tomato_type is not None:
        self.speak_dialog('like.tomato.type',
                          {'type': tomato_type})
    else:
        self.speak_dialog('like.tomato.generic')
```

When the User utters "Hey Mycroft, do you like RED tomatoes?", the second of the two Intent lines "do you like {type} tomatoes" is recognized by Mycroft, and the value 'RED' is returned in the message dictionary assigned to the 'type' entry when `handle_do_you_like()` is called. &#x20;

The line `tomato_type = message.data.get('type')` extracts the value from the dictionary for the entry 'type'. In this case, the variable `tomato_type` will receive the value 'RED', and `speak_dialog()` will be called with the 'like.tomato.type' dialog file, and a dictionary with 'RED' assigned to 'type'. The statement "I do like {type} tomatoes" might be randomly selected, and after insertion of the value 'RED' for the placeholder variable {type}, OVOS would say: "I do like RED tomatoes".

Should the User utter "Hey Mycroft, do you like tomatoes?", the first line in the Intent file "do you like tomatoes" is recognized. There is no variable in this line, and when `handle_do_you_like()` is called, the dictionary in the message is empty. This means `tomato_type` is `None`, `speak_dialog('like.tomato.generic')` would be called, and Mycroft might reply with "Yes, I do like tomatoes".

## Waiting for speech

By default, the `speak_dialog()` method is non-blocking. That is any code following the call to `speak_dialog()` will execute whilst OVOS is talking. This is useful to allow your Skill to perform actions while it is speaking.

Rather than telling the User that we are fetching some data, then going out to fetch it, we can do the two things simultaneously providing a better experience.

However, there are times when we need to wait until the statement has been spoken before doing something else. We have two options for this.

## Wait Parameter

We can pass a `wait=True` parameter to our `speak_dialog()` method. This makes the method blocking and no other code will execute until the statement has been spoken.

```
@intent_handler('what.is.a.tomato.intent')
def handle_what_is(self, message):
    """Speaks a statement from the dialog file.
    Waits (i.e. blocks) within speak_dialog() until
    the speaking has completed. """
    self.speak_dialog('tomato.description', wait=True)
    self.log.info("I waited for you")
```

## Using translatable resources

There may be a situation where the dialog file and the `speak_dialog()` method do not give the Skill enough flexibility. For instance, there may be a need to manipulate the statement from the dialog file before having it spoken by OVOS.

The `OVOSSkill` class provides four multilingual methods to address these needs. Each method uses a file, and multilingualism is accomplished using the country/language directory system.

The `translate()` method returns a random string from a ".dialog" file (modified by a data dictionary).

The `translate_list()` method returns a list of strings from a ".list" file (each modified by the data dictionary). Same as `translate\_template()` just with a

different file extension.

The `translate_namedvalue()` method returns a dictionary formed from CSV entries in a ".value" file.

The `translate_template()` method returns a list of strings from a ".template" file (each modified by the data dictionary). Same as `translate\_list()` just with a different file extension.

## docs/403-intents.md

# Intent Design

A user can accomplish the same task by expressing their intent in multiple ways. The role of the intent parser is to extract from the user's speech key data elements that specify their intent in more detail. This data can then be passed to other services, such as Skills to help the user accomplish their intended task.

Example: Julie wants to know about today's weather in her current location, which is Melbourne, Australia.

> "hey mycroft, what's today's weather like?" > > "hey mycroft, what's the weather like in Melbourne?" > > "hey mycroft, weather"

Even though these are three different expressions, for most of us they probably have roughly the same meaning. In each case we would assume the user expects OVOS to respond with today's weather for their current location. The role of an intent parser is to determine what this intent is.

In the example above, we might extract data elements like:

**weather** - we know that Julie wants to know about the weather, but she has not been specific about the type of

weather, such as `_wind_`, `_precipitation_`, `_snowfall_` or the risk of `_fire danger_` from bushfires. Melbourne, Australia rarely experiences snowfall, but falls under bushfire risk every summer.

**location** - Julie has stipulated her location as Melbourne, but she does not state that she means Melbourne,

Australia. How do we distinguish this from Melbourne, Florida, United States?

**date** - Julie has been specific about the `_timeframe_` she wants weather data for - today. But how do we know what

today means in Julie's timezone. Melbourne, Australia is between 14-18 hours ahead of the United States. We don't want to give Julie yesterday's weather, particularly as Melbourne is renowned for having changeable weather.

OVOS has two separate Intent parsing engines each with their own strengths. Each of these can be used in most situations, however they will process the utterance in different ways.

**Example based** intents are trained on whole phrases. These intents are generally more accurate however require you to include sample phrases that cover the breadth of ways that a User may ask about something.

**Keyword / Rule based** these intents look for specific required keywords. They are more flexible, but since these are essentially rule based this can result in a lot of false matches. A badly designed intent may totally throw the intent parser off guard. The main advantage of keyword based intents is the integration with [conversational context](#), they facilitate continuous dialogs

OVOS is moving towards a plugin system for intent engines, currently only the default MycroftAI intent parsers are supported

**Padatious** is a light-weight neural network that is trained on whole phrases. You can find the official documentation [here](#)

**Adapt** is a keyword based parser. You can find the official documentation [here](#)

> NOTE: Padatious doesn't handle numbers well, internally sees all digits as "#". If you need to use digits in your intents, it is recommended you use Adapt instead.

We will now look at each in more detail, including how to use them in a Skill.

## Keyword Intents

Keyword based intent parsers determine user intent based on a list of keywords or entities contained within a user's utterance.

## Defining keywords and entities

### Vocab (.voc) Files

Vocab files define keywords that the intent parser will look for in a Users utterance to determine their intent.

These files can be located in either the `vocab/lang-code/` or `locale/lang-code/` directories of a Skill. They can have one or more lines to list synonyms or terms that have the same meaning in the context of this Skill. OVOS will match `_any_` of these keywords with the Intent.

Consider a simple `Potato.voc`. Within this file we might include:

```
potato
potatoes
spud
```

If the User speaks `_either_`:

> potato

or

> potatoes

or

> spud

OVOS will match this to any Keyword Intents that are using the `Potato` keyword.

## Regular Expression (.rx) Files

Regular expressions (or regex) allow us to capture entities based on the structure of an utterance.

We strongly recommend you avoid using regex, it is very hard to make portable across languages, hard to translate and the reported confidence of the intents is not great.

We suggest using example based intents instead if you find yourself needing regex

These files can be located in either the `regex/lang-code/` or `locale/lang-code/` directories of a Skill. They can have one or more lines to provide different ways that an entity may be referenced. OVOS will execute these lines in the order they appear and return the first result as an entity to the Intent Handler.

Let's consider a `type.rx` file to extract the type of potato we are interested in. Within this file we might include:

```
. * about (?P<Type>.* ) potatoes
. * (make|like) (?P<Type>.* ) potato
```

**What is this regex doing?** . \* matches zero, one or more of any single character. ( ?P<Type>.\* ) is known as a Named Capturing Group. The variable name is defined between the "<>", and what is captured is defined after this name. In this case we use . \* to capture anything.

Learn more about Regular Expressions.

So our first line would match an utterance such as:

> Tell me about \_sweet potatoes\_

Whilst the second line will match either:

> Do you like \_deep fried potato\_

or

> How do I make \_mashed potato\_

From these three utterances, what will the extracted Type be:\ 1\. sweet\ 2\  
deep fried\ 3\. mashed

This Type will be available to use in your Skill's Intent Handler on the message object. We can access this using:

```
message.data.get( 'Type' )
```

## Using Keyword Intents in a Skill

Now that we have a Vocab and Regular Expression defined, let's look at how to use these in a simple Skill.

For the following example we will use the two files we outlined above:

```
Potato.voc
```

```
Type.rx
```

We will also add some new .voc files:

```
Like.voc - containing a single line "like"
```

```
You.voc - containing a single line "you"
```

```
I.voc - containing a single line "I"
```

## Creating the Intent Handler

To construct a Keyword Intent, we use the `intent_handler()` \\_decorator\_ and pass in the `IntentBuilder` helper class.

[Learn more about \\_decorators\\_ in Python.](#)

Both of these must be imported before we can use them:

```
from ovos_workshop.intents import IntentBuilder
from ovos_workshop.decorators import intent_handler
```

The `IntentBuilder` is then passed the name of the Intent as a string, followed by one or more parameters that correspond with one of our `.voc` or `.rx` files.

```
@intent_handler(IntentBuilder('IntentName'))
    .require('Potato')
    .require('Like')
    .optionally('Type')
    .one_of('You', 'I'))
```

In this example:

the `Potato` and `Like` keywords are required. It must be present for the intent to match.

the `Type` entity is optional. A stronger match will be made if this is found, but it is not required.

we require at least one of the `You` or `I` keywords.

What are some utterances that would match this intent?

> Do you like potato? Do you like fried potato? Will I like mashed potato? Do you think I would like potato?

What are some utterances that would \_not\_ match the intent?

> How do I make mashed potato?

\_The required `Like` keyword is not found.\_

> Is it like a potato?

\_Neither the `You` nor `I` keyword is found.\_

## Including it in a Skill

Now we can create our Potato Skill:

```
from ovos_workshop.intents import IntentBuilder
from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class PotatoSkill(OVOSSkill):

    @intent_handler(IntentBuilder('WhatIsPotato').require('What')
                    .require('Potato'))
    def handle_what_is(self, message):
        self.speak_dialog('potato.description')

    @intent_handler(IntentBuilder('DoYouLikePotato').require('Potato')
                    .require('Like').optionally('Type').one_of('You', 'I'))
    def handle_do_you_like(self, message):
        potato_type = message.data.get('Type')
        if potato_type is not None:
            self.speak_dialog('like.potato.type',
                             {'type': potato_type})
        else:
            self.speak_dialog('like.potato.generic')
```

You can [download this entire Potato Skill from Github](#), or see another Keyword Intent handler example in the [Hello World Skill](#)

## Common Problems

### More vocab!

One of the most common mistakes when getting started with Skills is that the vocab file doesn't include all the keywords or terms that a User might use to trigger the intent. It is important to map out your Skill and test the interactions with others to see how they might ask questions differently.

**I have added new phrases in the .voc file, but**



## Mycroft isn't recognizing them

1. Compound words like "don't", "won't", "shouldn't" etc. are normalized by OVOS - so they become "do not", "will not", "should not". You should use the normalized words in your .voc files. Similarly, definite articles like the word "the" are removed in the normalization process, so avoid using them in your .voc or .rx files as well.
2. Tab != 4 Spaces, sometimes your text editor or IDE automatically replaces tabs with spaces or vice versa. This may lead to an indentation error. So make sure there's no extra tabs and that your editor doesn't replace your spaces!
3. Wrong order of files directories is a very common mistake. You have to make a language sub-folder inside the dialog, vocab or locale folders such as skill-dir/locale/en-us/somefile.dialog. So make sure that your .voc files and .dialog files inside a language subfolder.

## I am unable to match against the utterance string

The utterance string received from the speech-to-text engine is received all lowercase. As such any string matching you are trying to do should also be converted to lowercase. For example:

```
@intent_handler(IntentBuilder('Example').require('Example').require('Intent'))
def handle_example(self, message):
    utterance = message.data.get('utterance')
    if 'Proper Noun'.lower() in utterance:
        self.speak('Found it')
```

## Example based Intents

Example based parsers have a number of key benefits over other intent parsing technologies.

Intents are easy to create

You can easily extract entities and then use these in Skills. For example,  
"Find the nearest gas station" -> { "place": "gas station" }

Disambiguation between intents is easier

Harder to create a bad intent that throws the intent parser off

## Creating Intents

Most example based intent parsers use a series of example sentences to train a machine learning model to identify an intent. Regex can also be used behind the scenes for example to extract entities

The examples are stored in a Skill's `vocab/lang` or `local/lang` directory, in files ending in the file extension `.intent`. For example, if you were to create a `_tomato_` Skill to respond to questions about a `_tomato_`, you would create the file

`vocab/en-us/what.is.a.tomato.intent`

This file would contain examples of questions asking what a `_tomato_` is.

```
what would you say a tomato is
what is a tomato
describe a tomato
what defines a tomato
```

These sample phrases do not require punctuation like a question mark. We can also leave out contractions such as "what's", as this will be automatically expanded to "what is" by OVOS before the utterance is parsed.

Each file should contain at least 4 examples for good modeling.

The above example allows us to map many phrases to a single intent, however often we need to extract specific data from an utterance. This might be a date, location, category, or some other `entity`.

## Defining entities

Let's now find out OVOS's opinion on different types of tomatoes. To do this we will create a new intent file: `vocab/en-us/do.you.like.intent`

with examples of questions about mycroft's opinion about tomatoes:

```
are you fond of tomatoes
do you like tomatoes
what are your thoughts on tomatoes
are you fond of {type} tomatoes
do you like {type} tomatoes
what are your thoughts on {type} tomatoes
```

Note the `{type}` in the above examples. These are wild-cards where matching content is forwarded to the skill's intent handler.

> **WARNING:** digits are not allowed for the entity name inside the `{ }`, **do NOT** use `{room1}`, use `{room_one}`.

## Specific Entities

In the above example, `{type}` will match anything. While this makes the intent flexible, it will also match if we say something like Do you like eating tomatoes?. It would think the type of tomato is "eating" which doesn't make much sense. Instead, we can specify what type of things the `{type}` of tomato should be. We do this by defining the type entity file here:

```
vocab/en-us/type.entity
```

which might contain something like:

```
red
reddish
green
greenish
yellow
yellowish
ripe
unripe
pale
```

This must be registered in the Skill before use - most commonly in the `initialize()` method:

```

from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class TomatoSkill(OVOSSkill):
    def initialize(self):
        self.register_entity_file('type.entity')

```

Now, we can say things like "do you like greenish tomatoes?" and it will tag type as: "greenish". However, if we say "do you like eating tomatoes?" - the phrase will not match as "eating" is not included in our `type.entity` file.

## Number matching

Let's say you are writing an Intent to call a phone number. You can make it only match specific formats of numbers by writing out possible arrangements using # where a number would go. For example, with the following intent:

```

Call {number}.
Call the phone number {number}.

```

the `number.entity` could be written as:

```

+### (###) ###-####
+## (###) ###-####
+# (###) ###-####
(###) ###-####
###-####
###-###-####
###.###.###
### ## ##
#####

```

## Entities with unknown tokens

Let's say you wanted to create an intent to match places:

```
Directions to {place}.
Navigate me to {place}.
Open maps to {place}.
Show me how to get to {place}.
How do I get to {place}?
```

This alone will work, but it will still get a high confidence with a phrase like "How do I get to the boss in my game?". We can try creating a `.entity` file with things like:

```
New York City
#### Georgia Street
San Francisco
```

The problem is, now anything that is not specifically a mix of New York City, San Francisco, or something on Georgia Street won't match. Instead, we can specify an unknown word with `:0`. This would be written as:

```
:0 :0 City
#### :0 Street
:0 :0
```

Now, while this will still match quite a lot, it will match things like "Directions to Baldwin City" more than "How do I get to the boss in my game?"

\_NOTE: Currently, the number of `:0` words is not fully taken into consideration so the above might match quite liberally, but this will change in the future.\_

## Parentheses Expansion

Sometimes you might find yourself writing a lot of variations of the same thing. For example, to write a skill that orders food, you might write the following intent:

```
Order some {food}.
Order some {food} from {place}.
Grab some {food}.
Grab some {food} from {place}.
```

Rather than writing out all combinations of possibilities, you can embed them into one or more lines by writing each possible option inside parentheses with `\|`

in between each part. For example, that same intent above could be written as:

```
(Order | Grab) some {food}
(Order | Grab) some {food} from {place}
```

or even on a single-line:

```
(Order | Grab) some {food} (from {place} | )
```

Nested parentheses are supported to create even more complex combinations, such as the following:

```
(Look (at | for) | Find) {object}.
```

Which would expand to:

```
Look at {object}
Look for {object}
Find {object}
```

There is no performance benefit to using parentheses expansion. When used appropriately, this syntax can be much clearer to read. However, more complex structures should be broken down into multiple lines to aid readability and reduce false utterances being included in the model. Overuse can even result in the model training timing out, rendering the Skill unusable.

## Using it in a Skill

The `intent_handler() _decorator_` can be used to create an examples based intent handler by passing in the filename of the `.intent` file as a string.

You may also see the `@intent_file_handler` decorator used in Skills. This has been deprecated and you can now replace any instance of this with the simpler `@intent_handler` decorator.

From our first example above, we created a file `vocab/en-us/what.is.a.tomato.intent`. To register an intent using this file we can use:

```
@intent_handler('what.is.a.tomato.intent')
```

This `_decorator_` must be imported before it is used:

```
from ovos_workshop.decorators import intent_handler
```

[Learn more about `\_decorators\_` in Python.](#)

Now we can create our Tomato Skill:

```
from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class TomatoSkill(OVOSSkill):

    def initialize(self):
        self.register_entity_file('type.entity')

    @intent_handler('what.is.a.tomato.intent')
    def handle_what_is(self, message):
        self.speak_dialog('tomato.description')

    @intent_handler('do.you.like.intent')
    def handle_do_you_like(self, message):
        tomato_type = message.data.get('type')
        if tomato_type is not None:
            self.speak_dialog('like.tomato.type',
                              {'type': tomato_type})
        else:
            self.speak_dialog('like.tomato.generic')
```

See a Padatious intent handler example in the [Hello World Skill](#)

## Common Problems

### I am unable to match against the utterance string

The utterance string received from the speech-to-text engine is received all lowercase. As such any string matching you are trying to do should also be converted to lowercase. For example:

```
@intent_handler('example.intent')
def handle_example(self, message):
    utterance = message.data.get('utterance')
    if 'Proper Noun'.lower() in utterance:
        self.speak('Found it')
```

## docs/405-customization.md

# Customization

## Resource Files

Resource files are essential components of OVOS skills, containing data such as dialogs, intents, vocabularies, regular expressions, templates, and more.

These files define how a skill interacts with the user and responds to queries.

> **RECAP:** the skill contains a `locale` folder with subfolders for each lang, eg `en-us`, learn more in [skill structure docs](#)

## Customizing Dialogs

Users can personalize the behavior of skills by customizing dialogues to better suit their preferences.

To give a unique twist and personality to your assistant you don't need to fork existing skills only to change dialogs

Here's a step-by-step guide on how to replace the dialog of an existing skill:

### Identify the Skill and Resource to Replace:

Determine the ID of the skill whose dialog you want to replace. In this example, let's assume the skill ID is `skill-ovos-date-time.openvoiceos`.

Identify the specific dialog file you want to replace. For this example, let's say you want to replace the `time.current.dialog` file located in the `locale/en-us/dialog` directory of the skill.

### Create the Replacement Dialog File:



Create a new dialog file with the same name (`time.current.dialog`) as the original file.

Customize the content of the dialog file according to your preferences. You can modify the existing dialogues, add new ones, or remove any that you don't want to use.

### **Locate the User-Specific Resource Directory:**

Use the provided skill ID (`skill-ovos-date-time.openvoiceos`) to locate the user-specific resource directory.

The user-specific resource directory is located within the XDG data directory. It follows the path

`XDG_DATA_HOME/mycroft/resources/skill-ovos-date-time.openvoiceos`  
(where `XDG_DATA_HOME` is the user's data directory, usually `~/.local/share` on Linux).

If it does not exist, create it. This can be done using file manager tools or command-line utilities such as `mkdir` on Unix-like systems.

### **Copy the Replacement Dialog File to the User-Specific Directory:**

Copy or move the replacement dialog file (`time.current.dialog`) to the appropriate directory within the user-specific resource directory.

Place the file in the `locale/en-us/dialog` directory within the user-specific resource directory. This mirrors the directory structure of the original skill.

In this example the final path of the file would be

`~/.local/share/mycroft/resources/skill-ovos-date-time.openvoiceos/locale/en-us/dialog/time.current.dialog`

### **Verify the Replacement:**

Test the skill to ensure that the modified dialogues are being used instead of the original ones.

Customizing dialogues offers users flexibility in tailoring the behavior of skills to their specific needs and preferences.

## **Local Language support**

Adding support for additional languages to existing skills enables users to interact with OVOS in their preferred language.

While developing or waiting for skills to support your language you might want to add it locally

Users can add language support for a skill by creating a new language folder in the user resources directory and copying the necessary files over:

### **Identify the Skill and Language to Add:**

Determine the ID of the skill for which you want to add language support.

Let's continue using the skill ID `skill-ovos-date-time.openvoiceos`.

Identify the language you want to add support for. For this example, let's say you want to add support for Spanish (language code: `es-es`).

### **Create the New Language Folder:**

Create a new directory with the name of the language code (`es-es` for Spanish) within the `locale` directory of the skill.

This can be done using file manager tools or command-line utilities such as `mkdir` on Unix-like systems.

Using the previous example, we would create

```
~/ .local/share/mycroft/resources/skill-ovos-date-time.openvoiceos/locale/es-es/
```

### **Copy the Required Files to the New Language Folder:**

Copy all the necessary resource files from an existing language folder (e.g., `en-us`) to the newly created language folder (`es-es`).

This includes files such as dialogues, vocabularies, regex patterns, etc., depending on the resources used by the skill.

Ensure that all files are placed in the corresponding directories within the new language folder to maintain the directory structure of the original skill.

### **Verify the Language Addition:**

Once the files are copied over, verify that the new language is supported by the skill.

Restart OpenVoiceOS to allow the skill to recognize the newly added language

resources.

Test the skill using the newly added language to ensure that it functions correctly and uses the appropriate language-specific resources.

By following these steps, users can add support for additional languages to existing skills by creating new language folders and copying the required resource files.

> **NEXT STEPS:** consider sending a Pull Request to the skill to directly add language support!

This allows users to extend the language capabilities of skills beyond the languages provided by default.

## docs/406-messagebus.md

### OVOSSkill Bus Interaction

The base [OVOSSkill API](#) handles most of the Messagebus usage automatically.

For example, the `mycroft.stop` message is caught by the skill framework, invoking an overridden `OVOSSkills.stop()` method within a **Skill**.

Similarly, the `OVOSSkill.speak()` and `OVOSSkill.speak_dialog()` methods generate `speak` messages to be conveyed to the text-to-speech (TTS) and audio systems.

You will really only need to know about the Mycroft Messagebus if you are developing advanced **Skills**.

The `OVOSSkill.add_event()` method allows you to attach a handler which will be triggered when the message is seen on the Messagebus.

### Connecting Message handlers

```

class ListenForMessageSkill(OVOSSkill):
    def initialize(self):
        self.add_event('recognizer_loop:record_begin',
                        self.handle_listener_started)
        self.add_event('recognizer_loop:record_end',
                        self.handle_listener_ended)

    def handle_listener_started(self, message):
        # code to execute when active listening begins...

    def handle_listener_ended(self, message):
        # code to execute when active listening begins...

```

## Generating Messages

```

from ovos_bus_client import Message

class GenerateMessageSkill(OVOSSkill):
    def some_method(self):
        self.bus.emit(Message("recognizer_loop:utterance",
                               {'utterances': ["the injected utterance"],
                                'lang': 'en-us'}))

```

## docs/407-skill\_filesystem.md

### Filesystem access

Many Skills may want access to parts of the filesystem. To account for the many different platforms that can run OVOS there are three locations that a Skill can utilize.

- Persistent filesystem

- Temporary cache

Skill's own root directory

## Persistent Files

When your Skill needs to store some data that will persist over time and cannot easily be rebuilt, there is a persistent filesystem namespaced to your Skill.

## Reading and writing to files

This uses the standard Python `open()` method to read and write files. It takes two parameters:

`file_name` `(str)` - a path relative to the namespace. subdirs not currently supported.

`mode` `(str)` a file handle mode `[r, r+, w, w+, rb, rb+, wb+, a, ab, a+, ab+, x]`

Example:

```
def write_line_to_file(self, file_name, line):
    """Write a single line to a file in the Skills persistent filesystem."""
    with self.file_system.open(file_name, "w") as my_file:
        my_file.write(line)

def read_file(self, file_name):
    """Read the contents of a file in the Skills persistent filesystem."""
    with self.file_system.open(file_name, "r") as my_file:
        return my_file.read()
```

## Check if a file exists

Quick method to see if some file exists in the namespaced directory.

Example:

```
file_name = "example.txt"
with self.file_system.open(file_name, "w") as my_file:
    my_file.write("Hello world")
self.log.info(self.file_system.exists(file_name))
# True
self.log.info(self.file_system.exists("new.txt"))
# False
```

## Get the path of the namespaced directory.

`self.file_system.path` is a member value containing the root path of the namespace. However, it is recommended that you use the `self.file_system.open()` method to read and write files.

Example:

```
from ovos_workshop.skills import OVOSkill

class FileSystemSkill(OVOSkill):

    def initialize(self):
        """Log the path of this Skills persistent namespace."""
        self.log.info(self.file_system.path)
```

## Create subdirectories

Now that we have the path of our namespaced filesystem, we can organize our files however we like within that directory.

In this example, we create a subdirectory called "cache", then write to a text file inside of it.

```
from os import mkdir
from os.path import join

from ovos_workshop.skills import OVOSSkill

class FileSystemSkill(OVOSSkill):

    def initialize(self):
        """Create a cache subdirectory and write to a file inside it"""
        cache_dir = "cache"
        file_name = "example.txt"
        if not self.file_system.exists(cache_dir):
            mkdir(join(self.file_system.path, cache_dir))
        with self.file_system.open(join(cache_dir, file_name), "w") as my_file:
            my_file.write('hello')
```

## Example Skill

```

from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class FileSystemSkill(OVOSSkill):

    def initialize(self):
        """Perform initial setup for the Skill.

        For this example we do four things:
        1. Log the path of this directory.
        2. Write to a file in the directory.
        3. Check that our file exists.
        4. Read the contents of our file from disk.
        """
        file_name = "example.txt"
        self.log.info(self.file_system.path)
        self.write_line_to_file(file_name, "hello world")
        self.log.info(self.file_system.exists(file_name))
        self.log.info(self.read_file(file_name))

    def write_line_to_file(self, file_name, line):
        """Write a single line to a file in the Skills persistent filesystem."""
        with self.file_system.open(file_name, "w") as my_file:
            my_file.write(line)

    def read_file(self, file_name):
        """Read the contents of a file in the Skills persistent filesystem."""
        with self.file_system.open(file_name, "r") as my_file:
            return my_file.read()

```

## Temporary Cache

Skills can create a directory for caching temporary data to speed up performance.

This directory will likely be part of a small RAM disk and may be cleared at any time. So code that uses these cached files must be able to fall back and regenerate the file.

## Example Skill



```

from os.path import join
from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler
from ovos_utils.file_utils import get_cache_directory

class CachingSkill(OVOSSkill):

    def initialize(self):
        """Perform initial setup for the Skill.

        For this example we do four things:
        1. Get a cache directory namespaced for our Skill.
        2. Define a file path for the cache_file.
        3. Write some data to the cache_file
        4. Log the path of the cache_file
        4. Log the contents of the cache_file.
        """
        cache_dir = get_cache_directory('CachingSkill')
        self.cache_file = join(cache_dir, "myfile.txt")
        self.cache_data()
        self.log.info(self.cache_file)
        self.log.info(self.read_cached_data())

    def cache_data(self):
        with open(self.cache_file, "w") as cache_file:
            cache_file.write("Some cached data")

    def read_cached_data(self):
        with open(self.cache_file, "r") as cache_file:
            return cache_file.read()

```

## Skill Root Directory

```
self.root_dir
```

This member variable contains the absolute path of a Skills root directory e.g.  
`~.local/share/mycroft/skills/my-skill.me/.`

Generally Skills should not modify anything within this directory. Modifying anything in the Skill directory will reload the Skill. It is also not guaranteed that the Skill will have permission to write to this directory.

## docs/408-skill\_settings.md

### Skill Settings

Skill settings in OVOS allow users to configure and personalize the behavior of Skillseither through the command line, configuration files, or a web-based interface. This enables advanced customization and support for external integrations, while remaining completely optional for basic usage.

### Common Use Cases

Changing default behaviors (e.g. alarm sounds, display preferences)

Authenticating with third-party services (e.g. Spotify)

Entering longer or complex data (e.g. IP addresses, API keys)

---

### Using Skill Settings in Your Skill

Settings are managed through a dictionary-like object available on the `OVOSkill` base class. They are persisted to disk and can be updated locally by the Skill or remotely by the user via a frontend.

Settings are stored in your Skill's configuration directory, usually:

```
~/ .config/mycroft/skills/<your_skill_id>/settings.json
```

### Accessing Settings

You can read settings like a standard Python dictionary, but it's recommended to use `.get()` to avoid `KeyError` exceptions:

```
# Safely read the 'show_time' setting with a default fallback
show_time = self.settings.get("show_time", False)
```

> Avoid using `self.settings['show_time']` directly, as it will raise a `KeyError` if the setting is not defined.

Also, do not access `self.settings` in your `__init__()` method wait until the `initialize()` method to ensure the settings are fully loaded.

---

## Handling Settings Updates

OVOS automatically checks for setting changes, either locally or from a remote backend. You can register a callback to react when settings change:

```
def initialize(self):
    self.settings_change_callback = self.on_settings_changed
    self.on_settings_changed() # Also run immediately on start

def on_settings_changed(self):
    show_time = self.settings.get('show_time', False)
    self.trigger_time_display(show_time)
```

This ensures your Skill responds to user configuration changes dynamically.

---

## Writing to Settings

You can update and persist values to the settings file simply by assigning them:

```
self.settings['show_time'] = True
```

Changes will persist across restarts unless overridden remotely via a backend or web interface.

---

## Web-Based Skill Settings (Optional UI)

A **community-built** web interface, [OVOS Skill Config Tool](#), provides a modern,

user-friendly way to configure OVOS skills.

## Features

Clean UI for managing skill-specific settings

Grouping and organization of Skills

Dark mode support

Built-in Basic Authentication for security

[!Skill Config Interface](#)

## Installation

Install via pip:

```
pip install ovos-skill-config-tool
```

Run the tool:

```
ovos-skill-config-tool
```

Access it in your browser at <http://0.0.0.0:8000>

## Default Login

**Username:** ovos

**Password:** ovos

To customize credentials:

```
export OVOS_CONFIG_USERNAME=myuser
export OVOS_CONFIG_PASSWORD=mypassword
ovos-skill-config-tool
```

---

## Tips

Always use `.get(key, default)` for safe reads.

Use `initialize()` instead of `__init__()` for anything that depends on settings.

Use settings callbacks to keep your Skill reactive to user changes.

---

## See Also

[OVOS Skill Config Tool on GitHub](#)

# docs/408-skill\_settings\_meta.md

## settingsmeta.json

### Define settings UI for a Skill

To define our Skills settings UI we use a `settingsmeta.json` or `settingsmeta.yaml` file. This file must be in the root directory of the Skill and must follow a specific structure.

Once settings have been defined using a `settingsmeta` file, they will be presented to the user in the configured backend or helper application

### Example settingsmeta file

To see it in action, lets look at a simple example from the [Date-Time Skill](#). First using the JSON syntax as a `settingsmeta.json` file:

```
{
  "skillMetadata": {
    "sections": [
      {
        "name": "Display",
        "fields": [
          {
            "name": "show_time",
            "type": "checkbox",
            "label": "Show digital clock when idle",
            "value": "true"
          }
        ]
      }
    ]
  }
}
```

Now, here is the same settings, as it would be defined with YAML in a `settingsmeta.yaml` file:

```
skillMetadata:
  sections:
    - name: Display
      fields:
        - name: show_time
          type: checkbox
          label: Show digital clock when idle
          value: "true"
```

Notice that the value of `false` is surrounded by "quotation marks". This is because OVOS expects a string of `"true"` or `"false"` rather than a Boolean.

Both of these files would result in the same settings block.

!

It is up to your personal preference which syntax you choose.

## Structure of the settingsmeta file

Whilst the syntax differs, the structure of these two filetypes is the same. This

starts at the top level of the file by defining a `skillMetadata` object. This object must contain one or more `sections` elements.

## Sections

Each section represents a group of settings that logically sit together. This enables us to display the settings more clearly in the web interface for users.

In the simple example above we have just one section. However, the [Spotify Skill settings](#) contains two sections. The first is for Spotify Account authentication, and the second section contains settings to define your default playback device.

Each section must contain a `name` attribute that is used as the heading for that section, and an Array of `fields`.

## Fields

Each section has one or more `fields`. Each field is a setting available to the user. Each field takes four properties:

`name` (String)

The `name` of the `field` is used by the Skill to get and set the value of the `field`. It will not usually be displayed to the user, unless the `label` property has not been set.

`type` (Enum)

The data type of this field. The supported types are:

`text`: any kind of text

`email`: text validated as an email address

`checkbox`: boolean, True or False

`number`: text validated as a number

`password`: text hidden from view by default

`select`: a drop-down menu of options

`label`: special field to display text for information purposes only. No `name` or `value` is required for a `label` field.

`label` (String)

The text to be displayed above the setting field.

value (String)

The initial value of the field.

Examples for each type of field are provided in JSON and YAML at the end of this page.

## SettingsMeta Examples

### Label Field

```
skillMetadata:
  sections:
    - name: Label Field Example
      fields:
        - type: label
          label: This is descriptive text.
```

### Text Field

```
skillMetadata:
  sections:
    - name: Text Field Example
      fields:
        - name: my_string
          type: text
          label: Enter any text
          value:
```

### Email



```
skillMetadata:
  sections:
    - name: Email Field Example
      fields:
        - name: my_email_address
          type: email
          label: Enter your email address
          value:
```

## Checkbox

```
skillMetadata:
  sections:
    - name: Checkbox Field Example
      fields:
        - name: my_boolean
          type: checkbox
          label: This is an example checkbox. It creates a Boolean value.
          value: "false"
```

## Number

```
skillMetadata:
  sections:
    - name: Number Field Example
      fields:
        - name: my_number
          type: number
          label: Enter any number
          value: 7
```

## Password

```
skillMetadata:
  sections:
    - name: Password Field Example
      fields:
        - name: my_password
          type: password
          label: Enter your password
          value:
```

## Select

```
skillMetadata:
  sections:
    - name: Select Field Example
      fields:
        - name: my_selected_option
          type: select
          label: Select an option
          options: Option 1|option_one;Option 2|option_two;Option 3|option_three
          value: option_one
```

## docs/409-skill\_api.md

## Skill API

The Skill API uses the Message Bus to communicate between Skills and wraps the interaction in simple Python objects making them easy to use.

## Making a method available through the Skill API

A method can be tagged with the `skill_api_method` decorator. This will handle all the basics of making the method available to other Skills over the Message Bus.

```

@skill_api_method
def my_exported_method(self, my_arg, my_other_arg):
    """My skill api method documentation
    """

```

The decorator will generate everything needed for accessing the method over the Message Bus and extract the associated docstring.

## Limitations

The Skill API works over the Message Bus. This requires that the return values are json serializable. All common Python builtin types (such as List, String, None, etc.) work well, however custom classes are not currently supported.

## Example

```

from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler, skill_api_method

class RobberSkill(OVOSSkill):
    @skill_api_method
    def robber_lang(self, sentence):
        """Encode a sentence to "Rovarspraket".

        Each consonant gets converted to consonant + "o" + consonant,
        vowels are left as is.

        Returns: (str) sentence in the robber language.
        """
        vowels = "aeiouyaaoo"
        tokens = []
        for char in sentence.lower() and char.isalpha():
            if char not in vowels:
                tokens.append(char + 'o' + char)
            else:
                tokens.append(char)
        return ' '.join(tokens)

```

## Using another Skill's API

If you want to make use of exported functionality from another Skill, you must fetch that Skill's `SkillApi`. This will give you a small class with the target Skill's exported methods. These methods are nothing special and can be called like any other class's methods.

To access the `robber_lang()` method we created above, we could write:

```
from ovos_workshop.skills.api import SkillApi

class NewRobberSkill(OVOSSkill):
    def initialize(self):
        self.robber = SkillApi.get('robber-skill.forslund')
        self.speak(self.robber.robber_lang('hello world'))
```

When the `NewRobberSkill` is initialized, it will assign the API from the Skill `robber-skill.forslund` to `self.robber`. We then run the exported method `robber_lang()` passing the argument `'hello world'`.

Our `NewRobberSkill` will therefore speak something like "hoh e lol lol o wow o ror lol dod".

## docs/410-skill\_runtime\_requirements.md

### Runtime Requirements in OVOS

OVOS (Open Voice OS) introduces advanced runtime management to ensure skills are only loaded and active when the system is ready. This improves performance, avoids premature skill activation, and enables greater flexibility across different system setups (offline, headless, GUI-enabled, etc.).

This guide covers how to control when OVOS declares readiness, how dynamic skill loading works, and how developers can use `RuntimeRequirements` to specify resource dependencies for their skills.

---

# Usage Guide

## Step 1: Customize ready\_settings (in your configuration)

You can specify what the system must wait for before emitting the `mycroft.ready` message:

```
{
  "ready_settings": [
    "skills",
    "network_skills",
    "internet_skills",
    "audio",
    "speech"
  ]
}
```

In this example, the `ready_settings` are configured to wait for network and internet connectivity before emitting the `'mycroft.ready'` message. Each setup can customize these settings based on their needs, a offline install won't want internet skills, a server wont want a audio stack etc.

## Step 2: Define RuntimeRequirements in your skill

Use the `runtime_requirements` class property to control when and how your skill should load based on system resources like internet, network, or GUI.

Example: `'python from ovos_utils import classproperty from ovos_workshop.skills import OVOSSkill from ovos_utils.process_utils import RuntimeRequirements`

```
class MySkill(OVOSSkill):
```

```
    @classproperty def runtime_requirements(self): return RuntimeRequirements(
        requires_internet=True ) '
```

```
---
```

## Technical Explanation

## ready\_settings

The `ready_settings` config controls when OVOS emits `mycroft.ready`, which signals that the system is ready for use. Each entry in this list waits for a different component:

**"skills"** Waits for offline skills to load.

**"network\_skills"** Waits for the system to detect a network connection (`mycroft.network.connected`).

**"internet\_skills"** Waits for an internet connection (`mycroft.internet.connected`).

**"setup"** Waits for an external setup process (e.g., pairing or configuration).

**"audio"** Waits for the audio playback and capture systems to be initialized.

**"speech"** Waits for the STT (speech-to-text) engine to be ready.

**{skill\_id}** - Waits for a specific skill to be available

> **Note:** By default, OVOS only waits for offline skills. Unlike Mycroft-core, OVOS supports dynamic loading, so timing can impact skills that depend on the `mycroft.ready` message.

---

## Dynamic Loading and Unloading

Introduced in `ovos-core 0.0.8`, dynamic skill management improves system performance and reliability by:

**Only loading skills when their requirements are met.**

**Unloading skills when they become unusable due to lost resources.**

## Benefits:

Reduces memory and CPU usage.

Avoids unnecessary skill activations.

Simplifies skill logic (e.g., no need to check for connectivity manually).

Skills are loaded only when their specific requirements are met. This optimization prevents unnecessary loading, conserving system resources and ensuring a more efficient skill environment.

### Reducing Unintended Activations

Dynamic unloading of skills based on specific conditions significantly reduces the chance of unintentional activations. In scenarios where required conditions are not met, skills are gracefully unloaded, enhancing the user experience by avoiding unintended skill triggers.

This approach aligns with resource-conscious design, providing a more responsive and reliable voice assistant environment. Developers can focus on skill functionality, knowing that OVOS efficiently manages skill loading and unloading based on runtime requirements.

---

## RuntimeRequirements (@classproperty)

Also introduced in `ovos-core 0.0.8`, the `RuntimeRequirements` class property allows skill developers to declare when a skill should be loaded or unloaded based on runtime conditions.

> Replaces the deprecated `"priority_skills"` config.

### Key fields:

| Field                             | Description  |
|-----------------------------------|--|
| <code>internet_before_load</code> | Wait for internet before loading                     |
| <code>requires_internet</code>    | Unload if internet is lost (unless fallback enabled) |
| <code>no_internet_fallback</code> | If true, do <b>not</b> unload when internet is lost  |
| <code>network_before_load</code>  | Wait for network before loading                      |
| <code>requires_network</code>     | Unload if network is lost (unless fallback enabled)  |
| <code>gui_before_load</code>      | Wait for GUI before loading                          |
| <code>requires_gui</code>         | Unload if GUI is lost (unless fallback enabled)      |
| <code>no_gui_fallback</code>      | If true, do <b>not</b> unload when GUI is lost       |

> Uses `@classproperty` so the system can evaluate the requirements without loading the skill.

---

# Examples

## 1. Fully Offline Skill

In this example, a fully offline skill is defined. The skill does not require internet or network connectivity during loading or runtime. If the network or internet is unavailable, the skill can still operate.

Defining this will ensure your skill loads as soon as possible; otherwise, the `SkillManager` will wait for internet before loading the skill.

```
from ovos_utils import classproperty
from ovos_workshop.skills import OVOSSkill
from ovos_utils.process_utils import RuntimeRequirements

class MyOfflineSkill(OVOSSkill):

    @classproperty
    def runtime_requirements(self):
        return RuntimeRequirements(internet_before_load=False,
                                   network_before_load=False,
                                   requires_internet=False,
                                   requires_network=False,
                                   no_internet_fallback=True,
                                   no_network_fallback=True)
```

Loads immediately, runs without internet or network.

---

## 2. Internet-Dependent Skill (with fallback)

In this example, an online search skill with a local cache is defined. The skill requires internet connectivity during both loading and runtime. If the internet is not available, the skill won't load. Once loaded, the skill continues to require internet connectivity.

However, our skill keeps a cache of previous results; therefore, it declares it



can handle internet outages and will not be unloaded when the internet goes down.

```
from ovos_utils import classproperty
from ovos_workshop.skills import OVOSSkill
from ovos_utils.process_utils import RuntimeRequirements

class MyInternetSkill(OVOSSkill):

    @classproperty
    def runtime_requirements(self):
        # our skill can answer cached results when the internet goes down
        return RuntimeRequirements(
            internet_before_load=True, # only load once we have internet
            requires_internet=True, # indicate we need internet to work
            no_internet_fallback=True # do NOT unload if internet goes down
        )

    def initialize(self):
        ... # do something that requires internet connectivity
```

Loads only when internet is available. Stays loaded even if internet is lost, using a cached fallback.

---

### 3. LAN-Controlled IOT Skill

Consider a skill that should only load once we have a network connection. By specifying that requirement, we can ensure that the skill is only loaded once the requirements are met, and it is safe to utilize network resources on initialization.

In this example, an IOT skill controlling devices via LAN is defined. The skill requires network connectivity during loading, and if the network is not available, it won't load.

Once loaded, the skill continues to require network connectivity and will unload if the network is lost.

```

from ovos_utils import classproperty
from ovos_workshop.skills import OVOSSkill
from ovos_utils.process_utils import RuntimeRequirements

class MyIOTSkill(OVOSSkill):

    @classproperty
    def runtime_requirements(self):
        return RuntimeRequirements(
            network_before_load=True, # only load once network available
            requires_network=True, # we need network to work
            no_network_fallback=False # unload if network goes down
        )

    def initialize(self):
        ... # do something that needs LAN connectivity

```

Loads when the local network is connected. Unloads if the network is lost.

---

## 4. GUI + Internet Skill (Unloads without GUI)

Consider a skill with both graphical user interface (GUI) and internet dependencies is defined.

The skill requires both GUI availability and internet connectivity during loading.

If either the GUI or the internet is not available, the skill won't load. Once loaded, the skill continues to require both GUI availability, but internet connectivity is optional.

If the user asks "show me the picture of the day" and we have both internet and a GUI, our skill will match the intent.

If we do not have internet but have a GUI, the skill will still operate, using a cached picture. If no GUI is available then the skill will unload regardless of internet status

```

from ovos_utils import classproperty
from ovos_workshop.skills import OVOSSkill
from ovos_utils.process_utils import RuntimeRequirements

class MyGUIAndInternetSkill(OVOSSkill):

    @classproperty
    def runtime_requirements(self):
        return RuntimeRequirements(
            gui_before_load=True, # only load if GUI is available
            requires_gui=True, # continue requiring GUI once loaded
            internet_before_load=True, # only load if internet is available
            requires_internet=True, # continue requiring internet once loaded
            no_gui_fallback=False, # unload if GUI becomes unavailable
            no_internet_fallback=True # do NOT unload if internet becomes unavailable, us
        )

    def initialize(self):
        ... # do something that requires both GUI and internet connectivity

```

Requires GUI and internet to load. Will stay loaded if internet is lost (e.g., to show a cached picture), but unloads if GUI becomes unavailable.

---

## Tips and Caveats

If `runtime_requirements` is not defined, OVOS assumes **internet is required** but **GUI is optional**.

You can combine different requirements to handle a wide range of usage patterns (e.g., headless servers, embedded devices, smart displays).

Consider defining graceful fallbacks to avoid unnecessary unloading.

## docs/411-skill\_json.md

# Skill Metadata File

The `skill.json` file is an optional but powerful way to describe your Open Voice OS (OVOS) skill. It provides metadata used for installation, discovery, and display in GUIs or app stores.

## Purpose

Helps OVOS identify and install your skill.

Enhances GUI experiences with visuals and usage examples.

Lays the foundation for future help dialogs and skill documentation features.

---

## Usage Guide

1. Create a `skill.json` file inside your skill's `locale/<language-code>` folder.
2. Fill in the metadata fields as needed (see below).
3. If your skill supports multiple languages, include a separate `skill.json` in each corresponding `locale` subfolder.

> **Avoid using old 'skill.json' formats** found in some legacy skills where the file exists at the root level. These are deprecated.

---

## Example skill.json

```
{
  "skill_id": "skill-xxx.exampleauthor",
  "source": "https://github.com/ExampleAuthor/skill-xxx",
  "package_name": "ovos-skill-xxx",
  "pip_spec": "git+https://github.com/ExampleAuthor/skill-xxx@main",
  "license": "Apache-2.0",
  "author": "ExampleAuthor",
  "extra_plugins": {
    "core": ["ovos-utterance-transformer-xxx"],
    "PHAL": ["ovos-PHAL-xxx"],
    "listener": ["ovos-audio-transformer-xxx", "ovos-ww-plugin-xxx", "ovos-vad-plugin-xxx"],
    "audio": ["ovos-dialog-transformer-xxx", "ovos-tts-transformer-xxx", "ovos-tts-plugin-xxx"],
    "media": ["ovos-ocp-xxx", "ovos-media-xxx"],
    "gui": ["ovos-gui-extension-xxx"]
  },
  "icon": "http://example.com/icon.svg",
  "images": ["http://example.com/logo.png", "http://example.com/screenshot.png"],
  "name": "My Skill",
  "description": "Does awesome skill stuff!",
  "examples": [
    "do the thing",
    "say this to use the skill"
  ],
  "tags": ["productivity", "entertainment", "aliens"]
}
```

---

## Field Reference

| Field         | Type   | Required | Description   |
|---------------|--------|----------|---|
| skill_id      | string | Yes      | Unique ID, typically repo.author style (lowercase). |
| source        | string | Optional | Git URL to install from source.                     |
| package_name  | string | Yes      | Python package name (e.g., for PyPI installs).      |
| pip_spec      | string | Optional | PEP 508 install spec.                               |
| license       | string | Optional | License ID (see <a href="#">SPDX list</a> ).        |
| author        | string | Optional | Display name of the skill author.                   |
| extra_plugins | object | Optional | Dependencies to be installed in other               |

OVOS services (not this skill). || `icon` | string | Optional | URL  
to a skill icon (SVG recommended). || `images` | list | Optional |  
Screenshots or promotional images. || `name` | string | Optional |  
User-facing skill name. || `description` | string | Optional | Short,  
one-line summary of the skill. || `examples` | list | Optional |  
Example utterances your skill handles. || `tags` | list | Optional |  
Keywords for searchability. |

---

## Language Support

To support multiple languages, place a `skill.json` file in each corresponding `locale/<lang>` folder. Fields like `name`, `description`, `examples`, and `tags` can be translated for that locale.

---

## Installation Behavior

When installing a skill, OVOS will try the following methods in order:

1. `pip_spec` (if present)
2. `package_name` (from PyPI)
3. `source` (from Git)

At least **one valid installation path is required**.

---

## Tips & Caveats

This metadata format is **experimental** and may evolvecheck for updates regularly.

`extra_plugins` allows for declaring companion plugins your skill may require, but that aren't direct Python dependencies.

The [Skill store](#) and GUI tools like `ovos-shell` use `icon`, `images`, `examples`, and `description` to present the skill visually.

---

## See Also

[PEP 508 Dependency specification](#)

[SPDX License List](#)

## docs/420-ssml.md

# SSMLBuilder

## What is SSML?

Speech Synthesis Markup Language (SSML) is a markup language used to enhance synthesized speech output. It provides developers with a way to control various aspects of speech synthesis, such as pronunciation, intonation, volume, and speed, by using predefined tags and attributes.

SSML allows developers to create more natural and expressive speech output, making interactions with voice-based applications more engaging and user-friendly.

These use cases demonstrate how SSML can be applied in various contexts to improve the quality, expressiveness, and accessibility of synthesized speech output, ultimately enhancing the overall user experience.

- 1. Narration with Emphasis:** In storytelling applications or audiobooks, developers can use SSML to emphasize specific words or phrases to convey emotions or highlight key points in the narrative. For example, during a suspenseful moment in a story, the narrator's voice could be slowed down for dramatic effect using SSML.
- 1. Interactive Voice Responses:** In voice-based applications such as virtual assistants or customer service bots, SSML can be used to provide more natural and engaging interactions with users. For instance, developers can use SSML to insert pauses between sentences to mimic natural speech patterns or adjust the pitch and volume of the voice to convey empathy or urgency.

1. **Educational Content:** SSML can be valuable in educational applications where synthesized speech is used to deliver instructional content or quizzes. Developers can use SSML to modify the speaking rate to accommodate different learning paces or employ phonetic pronunciation tags to ensure correct pronunciation of specialized terms or foreign words.
1. **Accessibility Features:** For applications designed to assist users with visual impairments or reading difficulties, SSML can play a crucial role in enhancing accessibility. Developers can use SSML to provide auditory cues, such as tone changes or speech emphasis, to indicate important information or user interface elements.

## SSMLBuilder

The `SSMLBuilder` class simplifies the creation of SSML strings by providing intuitive methods to control various aspects of speech synthesis. It offers a range of methods for manipulating text, adjusting timing and prosody, specifying voice and phoneme characteristics, and more.

```
from ovos_utils.ssml import SSMLBuilder

class MySkill:

    def handle_intent(self, message):
        # Create an instance of SSMLBuilder
        ssml_builder = SSMLBuilder()

        # Generate SSML
        ssml_text = ssml_builder.sentence("Hello, world!").pause(500, "ms").say_slow("How

        # Output:
        # '<speak>\n<s>Hello, world!</s> <break time=500ms/><prosody rate='0.4'>How are yo

        # Speak the SSML text
        self.speak(ssml_text)
```

## Text Manipulation

1. `sub(alias, word)`: Replaces a word with a specified alias.



2. `emphasis(level, word)`: Emphasizes a word with a specified level.
3. `parts_of_speech(word, role)`: Specifies the usage or role of a word.
4. `pause_by_strength(strength)`: Inserts a pause with a specified strength.
5. `sentence(text)`: Wraps text with `<s>` tags to denote a sentence.
6. `say_emphasis(text)`: Emphasizes the text strongly.
7. `say_strong(text)`: Modifies the vocal-tract length to increase speech intensity.
8. `say_weak(text)`: Modifies the vocal-tract length to decrease speech intensity.
9. `say_softly(text)`: Modifies the phonation to produce softer speech.
10. `say_auto-breaths(text)`: Adds automatic breaths to the speech.
11. `paragraph(text)`: Wraps text with `<p>` tags to denote a paragraph.
12. `audio(audio_file, text)`: Embeds audio with specified text.

## Timing and Prosody

1. `pause(time, unit)`: Inserts a pause for a specified duration.
2. `prosody(attribute, text)`: Modifies prosodic attributes of the text such as pitch, rate, or volume.
3. `pitch(pitch, text)`: Changes the pitch of the text.
4. `volume(volume, text)`: Modifies the volume of the text.
5. `rate(rate, text)`: Adjusts the speaking rate of the text.

## Voice and Phoneme

1. `say(text)`: Adds normal speed text to SSML.
2. `say_loud(text)`: Increases the volume of the text.

3. `say_slow(text)`: Slows down the speaking rate of the text.
4. `say_fast(text)`: Speeds up the speaking rate of the text.
5. `say_low_pitch(text)`: Lowers the pitch of the text.
6. `say_high_pitch(text)`: Raises the pitch of the text.
7. `say_whispered(text)`: Converts text into whispered speech.
8. `phoneme(ph, text)`: Specifies the phonetic pronunciation of the text.
9. `voice(voice, text)`: Specifies the voice to use for the text.
10. `whisper(text)`: Converts text into whispered speech.

## Build and Utility

1. `build()`: Constructs the final SSML string.
2. `remove_ssml(text)`: Removes SSML tags from the given text.
3. `extract_ssml_tags(text)`: Extracts SSML tags from the given text.

## SSML Support in TTS Plugins

OVOS TTS plugins implement support for SSML, ensuring that SSML content is processed accurately during speech synthesis. Let's take a closer look at how SSML handling works within the `TTS` abstract class:

**SSML Validation:** The `validate_ssml()` method checks if the TTS engine supports SSML. Unsupported or invalid SSML tags are removed from the input text to ensure proper processing.

**SSML Tag Handling:** Supported SSML tags are processed by the TTS engine during synthesis. Unsupported tags are removed, while supported tags are modified or retained based on the implementation of the `modify_tag()` method.

```

        utterance = utterance.replace(tag, "")

# default handling of ssml, advanced plugins may override this method
# return text with supported ssml tags only
def modify_tag(self, tag):
    return utterance.replace(" ", " ")
    """Override to modify each supported ssml tag.

Arguments:
    tag (str): SSML tag to check and possibly transform.
    """
    return tag

def validate_ssml(self, utterance):
    """Check if engine supports ssml, if not remove all tags.

Remove unsupported / invalid tags

Arguments:
    utterance (str): Sentence to validate

Returns:
    str: validated_sentence
    """

    # Validate speak tags
    if not self.ssml_tags or "speak" not in self.ssml_tags:
        self.format_speak_tags(utterance, False)
    elif self.ssml_tags and "speak" in self.ssml_tags:
        self.format_speak_tags(utterance)

    # if ssml is not supported by TTS engine remove all tags
    if not self.ssml_tags:
        return self.remove_ssml(utterance)

    # find ssml tags in string
    tags = SSML_TAGS.findall(utterance)

    for tag in tags:
        if any(supported in tag for supported in self.ssml_tags):
            utterance = utterance.replace(tag, self.modify_tag(tag))
        else:
            # remove unsupported tag

```

## Platform-Specific SSML Handling

Some TTS plugins, like the `PollyTTS` plugin, may support platform-specific SSML tags that are not part of the standard specification. For example, Amazon Polly supports additional SSML tags specific to Amazon's speech synthesis service.

Let's take a closer look at how SSML support is implemented in the `PollyTTS` plugin:

```
class PollyTTS(TTS):
    def __init__(self, *args, **kwargs):
        ssml_tags = ["speak", "say-as", "voice", "prosody", "break",
                     "emphasis", "sub", "lang", "phoneme", "w", "whisper",
                     "amazon:auto-breaths", "p", "s", "amazon:effect", "mark"]
        super().__init__(*args, **kwargs, audio_ext="mp3",
                        ssml_tags=ssml_tags, validator=PollyTTSValidator(self))

    def get_tts(self, sentence, wav_file, lang=None, voice=None):
        # SSML handling specific to PollyTTS
        # Replace custom SSML tags for Amazon Polly
        sentence = sentence.replace("\whispered", "/amazon:effect") \
            .replace("\\whispered", "/amazon:effect") \
            .replace("whispered", "amazon:effect name=\"whispered\"")

        # alternatively the plugin could override self.modify_tag method instead

        # Synthesize speech using Amazon Polly API
        # Write audio stream to WAV file
        return wav_file, None
```

In this example, the `PollyTTS` plugin defines a list of supported SSML tags, including both standard and Amazon-specific tags. During initialization, the plugin sets up SSML support by providing the list of tags to the `TTS` superclass.

When synthesizing speech using Amazon Polly, the plugin translates platform-specific SSML tags like `amazon:effect` to ensure compatibility with

Amazon's speech synthesis service.

## **Behavior with Plugins That Do Not Support SSML**

When SSML text is sent to a TTS plugin that does not support SSML, the plugin will typically ignore the SSML tags and process the text as regular speech.

This means that any SSML-specific effects, such as pauses, emphasis, or prosody modifications, will be dropped, and the synthesized speech will be generated without considering the SSML markup.

It's important to ensure compatibility between the SSML content and the capabilities of the TTS plugin being used. If SSML-specific effects are essential for the intended speech output, it's recommended to verify that the selected TTS plugin supports SSML or consider using a different plugin that provides SSML support.

## **docs/430-skill\_dev\_faq.md**

### **Developer FAQ**

> This list is a work in progress, [Suggestions and Pull Requests welcome!](#)

### **How do I know what is currently happening in the GUI?**

```

        print("device has screen:", self.tracker.can_display())
from ovos_utils.gui import GUITracker
        print("mycroft-gui installed:", self.tracker.is_gui_installed())
from ovos_workshop.skills import OVOSSkill
        print("gui connected:", self.tracker.is_gui_connected())
from ovos_workshop.decorators import intent_handler
        # TODO - speak or something

    @intent_handler("list.idle.screens.intent")
class MyGUIEventTracker(GUITracker):
    def handle_idle_screens_intent(self, message):
        # GUI event handlers
        # check registered idle screens
        # skill can/should subclass this
        print("Registered idle screens:")

        for name in self.tracker.idle_screens:
            def on_idle(self, namespace):
                skill_id = self.tracker.idle_screens[name]
                print("IDLE", namespace)
                print("name", name, ":", skill_id)
                timestamp = self.idle_ts
                # TODO - speak or something

            def on_active(self, namespace):
                # NOTE: page has not been loaded yet
                # event will fire right after this one
                print("ACTIVE", namespace)
                # check namespace values, they should all be set before this event
                values = self.gui_values[namespace]

            def on_new_page(self, page, namespace, index):
                print("NEW PAGE", namespace, index, namespace)
                # check all loaded pages
                for n in self.gui_pages: # list of named tuples
                    nspace = n.name # namespace / skill_id
                    pages = n.pages # ordered list of page uris

            def on_gui_value(self, namespace, key, value):
                # WARNING this will pollute logs quite a lot, and you will get
                # duplicates, better to check values on a different event,
                # demonstrated in on_active
                print("VALUE", namespace, key, value)

class MySkill(OVOSSkill):
    def initialize(self):
        self.tracker = MyGUIEventTracker(bus=self.bus)

    @intent_handler("gui.status.intent")
    def handle_status_intent(self, message):

```

## How do I stop an intent mid execution?

Sometimes you want to abort a running intent immediately, the stop method may not be enough in some circumstances we provide a `killable_intent` decorator in `ovos_workshop` that can be used to abort a running intent immediately

a common use case is for GUI interfaces where the same action may be done by voice or clicking buttons, in this case you may need to abort a running `get_response` loop

```

        @killable_intent(msg="my.own.abort.msg", callback=handle_intent_aborted)
from ovos_workshop.skills import OVOSkill
        @intent_handler("test3.intent")
from ovos_workshop.decorators import killable_intent, intent_handler
        def handle_test_msg_intent(self, message):
from time import sleep
            if self.my_special_var != "default":
                self.speak("someone forgot to cleanup")
        while True:
class Test(OVOSkill):
    """
        sleep(1)
        self.speak("you can't abort me")
    send "mycroft.skills.abort_question" and confirm only get_response is aborted
    send "mycroft.skills.abort_execution" and confirm the full intent is aborted, except i
    send "my.own.abort.msg" and confirm intent3 is aborted
    say "stop" and confirm all intents are aborted
    """
    def __init__(self):
        super(Test, self).__init__("KillableSkill")
        self.my_special_var = "default"

    def handle_intent_aborted(self):
        self.speak("I am dead")
        # handle any cleanup the skill might need, since intent was killed
        # at an arbitrary place of code execution some variables etc. might
        # end up in unexpected states
        self.my_special_var = "default"

    @killable_intent(callback=handle_intent_aborted)
    @intent_handler("test.intent")
    def handle_test_abort_intent(self, message):
        self.my_special_var = "changed"
        while True:
            sleep(1)
            self.speak("still here")

    @intent_handler("test2.intent")
    @killable_intent(callback=handle_intent_aborted)
    def handle_test_get_response_intent(self, message):
        self.my_special_var = "CHANGED"
        ans = self.get_response("question", num_retries=99999)
        self.log.debug("get_response returned: " + str(ans))
        if ans is None:
            self.speak("question aborted")

```



## How do I send files over the bus?

Sometimes you may want to send files or binary data over the messagebus, `ovos_utils` provides some tools to make this easy

Sending a file ‘python from ovos\_utils.messagebus import  
send\_binary\_file\_message, decode\_binary\_message from ovos\_workshop.skills import  
OVOSSkill

```
class MySkill(OVOSSkill):
    def initialize(self):
        self.add_event("mycroft.binary.file", self.receive_file)

    def receive_file(self, message):
        print("Receiving file")
        path = message.data["path"] # file path, extract filename if needed
        binary_data = decode_binary_message(message) # TODO process data somehow

    def send_file(self, my_file_path):
        send_binary_file_message(my_file_path) ‘
```

Sending binary data directly ‘python from ovos\_utils.messagebus import  
send\_binary\_data\_message, decode\_binary\_message from ovos\_workshop.skills import  
OVOSSkill

```
class MySkill(OVOSSkill):
    def initialize(self):
        self.add_event("mycroft.binary.data", self.receive_binary)

    def send_data(self, binary_data):
        send_binary_data_message(binary_data)

    def receive_binary(self, message):
        print("Receiving binary data")
        binary_data = decode_binary_message(message) # TODO process data somehow ‘
```

## docs/50-ovos\_installer.md

# How to Install Open Voice OS with the ovos-installer

Welcome to the quick-start guide for installing Open Voice OS (OVOS) using the official `ovos-installer`! This guide is suitable for **Raspberry Pi** and **desktop/server** Linux environments. Whether you're running this on a headless Raspberry Pi or your everyday laptop, the steps are mostly the same; only the way you connect to the device differs.

> Note: Some exotic hardware (like ReSpeaker microphones or certain audio HATs) may require extra configuration. The installer aims for wide compatibility, but specialized setups might need some manual intervention.

Looking for a pre-built raspberry pi image instead? check out [raspOVOS](#) and the companion [tutorial](#)

---

## Step-by-step Installation

### 1. Connect to Your Device (if remote)

If you're installing on a headless device (like a Raspberry Pi), connect via SSH:

```
ssh -l your-username <your-device-ip>
```

---

### 2. Update Package Metadata

Make sure your package manager is up to date:

```
sudo apt update
```

---

### 3. Install Prerequisites

Install `git` and `curl`; these are required to run the installer:

```
sudo apt install -y git curl
```

---

## 4. Run the OVOS Installer

Now you're ready to kick off the installation process:

```
sudo sh -c "$(curl -fsSL https://raw.githubusercontent.com/OpenVoiceOS/ovos-installer/main
```

[!image](#)

---

## What Happens Next?

Once you run the script, the installer will:

- Perform system checks

- Install dependencies (Python, Ansible, etc.)

- Launch a **text-based user interface (TUI)** to guide you through the setup

This can take anywhere from **5 to 20 minutes**, depending on your hardware, internet speed, and storage performance. Now lets walk through the installer screens!

---

## The Installer Wizard

Navigation:

- navigation is done via arrow keys

- pressing space selects options in the lists

- eg. when selecting `virtualenv` or `containers`

- pressing tab will switch between the options and the `<next>/<back>` buttons

- pressing enter will execute the highlighted `<next>/<back>` option

---

## Language Selection

The first screen lets you select your preferred language. Just follow the on-screen instructions.

[!image](#)

---

## Environment Summary

You'll be shown a summary of the detected environment; no action needed here. It's just informative.

[!image](#)

---

## Choose Installation Method

You have two choices:

**Virtualenv:** Recommended for most users. Easier to understand and manage.

**Containers:** For advanced users familiar with Docker or Podman.

[!image](#)

---

## Choose Channel

Select the **development** channel. Once OVOS is production-ready, a stable channel will also be available.

[!image](#)

---

## Choose Profile

Pick the `ovos` profile. This is the classic, all-in-one Open Voice OS experience with all the necessary components running locally.

!image

---

## Feature Selection

Choose what features you'd like to install.

!image

> Note: Some features (like the GUI) may be unavailable on lower-end hardware like the Raspberry Pi 3B+.

---

## Raspberry Pi Tuning (if applicable)

On Raspberry Pi boards, you'll be offered system tweaks to improve performance. It's highly recommended to enable this!

!image

---

## Summary

Before the installation begins, you'll see a summary of your selected options. This is your last chance to cancel the process.

!image

---

## Anonymous Telemetry

You'll be asked whether to share **anonymous usage data** to help improve Open Voice OS. Please consider opting in!

!image

The data collection only happens during the installation process, nothing else will be collected once the installation is over.

**The installer will ask you if you want to share or not the data.**

Below is a list of the collected data \_(please have a look to the [Ansible tempalte](#) used to publish the data)\_.

| Data  | Description  |
|---|--|
| architecture                                  |  |
| CPU architecture where OVOS was installed     | channel  |
| stable or development version of OVOS         | container  |
| OVOS installed into containers                | country  |
| Country where OVOS has been installed         | cpu_capable  |
| Is the CPU supports AVX2 or SIMD instructions | display_server   |
| Is X or Wayland are used as display server    |  |
| extra_skills_feature                          | Extra OVOS's skills enabled during the installation      |
| gui_feature                                   | GUI enabled during the installation                      |
| hardware                                      | Is the device a Mark 1, Mark II or DevKit                |
| installed_at                                  | Date when OVOS has been installed                        |
| os_kernel                                     | Kernel version of the host where OVOS is running         |
| os_name                                       | OS name of the host where OVOS is running                |
| os_type                                       | OS type of the host where OVOS is running                |
| os_version                                    | OS version of the host where OVOS is running             |
| profile                                       | Which profile has been used during the OVOS installation |
| python_version                                | What Python version was running on the host              |
| raspberry_pi                                  | Does OVOS has been installed on Raspberry Pi             |
| skills_feature                                | Default OVOS's skills enabled during the installation    |
| sound_server                                  | What PulseAudio or PipeWire used                         |
| tuning_enabled                                | Did the Raspberry Pi tuning feature was used             |
| venv  | OVOS installed into a Python virtual environment         |

---

## Sit Back and Relax

The installation begins! This can take some time, so why not grab a coffee (or maybe a cupcake)?

Here is a demo of how the process should go if everything works as intended

[!asciicast\]\(https://asciinema.org/a/710286\)](https://asciinema.org/a/710286)

---

## Installation Complete!

Youve done it! OVOS is now installed and ready to serve you. Try saying things like:

Whats the weather?

Tell me a joke.

Set a timer for 5 minutes.

[!image](#)

Youre officially part of the Open Voice OS community!

---

## Additional Configuration and Known Issues

Depending on your language you probably want to change the default plugins, the ovos-installer is not perfect and might not always select the best defaults

It is recommend that you run `ovos-config autoconfigure --help` after the initial install

[!\[asciicast\]\(https://asciinema.org/a/710295\)](https://asciinema.org/a/710295)

---

## Troubleshooting

> Something went wrong?

Dont panic! If the installer fails, it will generate a log file and upload it to <https://dpaste.com>. Please share that link with the community so we can help you out.

OVOS is a community-driven project, maintained by passionate volunteers. Your feedback, bug reports, and patience are truly appreciated.

## docs/500-prompts.md

## Prompting the User for Responses

# in OVOS Skills

OVOS provides several built-in methods for engaging users in interactive conversations. These include asking open-ended questions, confirming yes/no responses, and offering multiple-choice selections all handled in a natural, voice-first way.

Here we look at how to implement the most common types of prompts. For more information on conversation design see the [Voice User Interface Design Guidelines](#).

.  
---

## Usage Guide

Here's how to use different types of prompts in your OVOS skills:

### 1. Open-Ended Questions

Let the user respond freely, either to trigger another skill or to handle the response with a custom intent.

```
from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler
import random

class AskMeSkill(OVOSSkill):
    @intent_handler('ask_me_something.intent')
    def handle_set_favorite(self):
        question = random.choice(self.question_list)
        self.speak(question, expect_response=True)
```

> `expect_response=True` keeps the mic open after speaking, so the response can be handled by OVOS's intent pipeline.

---

### 2. Request Extra Information with `get_response()`

Use this to ask a specific question and directly capture the user's reply.



```

from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class IceCreamSkill(OVOSSkill):
    @intent_handler('set.favorite.intent')
    def handle_set_favorite(self):
        favorite_flavor = self.get_response('what.is.your.favorite.flavor')
        self.speak_dialog('confirm.favorite.flavor', {'flavor': favorite_flavor})

```

### Optional 'get\_response()' arguments:

data: Dictionary to format the dialog file

validator: A function to check if the user response is valid

on\_fail: A fallback string to say if validation fails

num\_retries: How many times to retry if the response isnt valid

---

## 3. Yes/No Questions with ask\_yesno()

Detects affirmations or negations from user responses.

```

from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class IceCreamSkill(OVOSSkill):
    @intent_handler('do.you.like.intent')
    def handle_do_you_like(self):
        likes_ice_cream = self.ask_yesno('do.you.like.ice.cream')
        if likes_ice_cream == 'yes':
            self.speak_dialog('does.like')
        elif likes_ice_cream == 'no':
            self.speak_dialog('does.not.like')
        else:
            self.speak_dialog('could.not.understand')

```

### Behavior:

Returns "yes" or "no" for matching phrases.

Returns the full utterance if unclear.

Returns `None` if no valid response is detected.

> uses [ovos-solver-YesNo-plugin](#) to understand complex affirmations and denials even double negations.

Example mappings:

| User Says  | Detected As |
|--|-------------|
| ----- -----  | "yes"       |
| yes     "no"   no     "don't                             |             |
| think so"   no     "that's affirmative"                  |             |
| yes     "no, but actually, yes"   yes     "yes, but      |             |
| actually, no"   no     "yes, yes, yes, but actually, no" |             |
| "no"     "please"   "yes"                                |             |
| "please don't"   "no"     "no! please! I beg you"        |             |
| "no"     "yes, i don't want it for sure"   "no"          |             |
| "please! I beg you"   "yes"     "i want it for sure"     |             |
| "yes"     "obviously"   "yes"                            |             |
| "indeed"   "yes"     "no, I obviously hate it"           |             |
| "no"     "that's certainly undesirable"   "no"           |             |
| "yes, it's a lie"   "yes"     "no, it's a lie"           |             |
| "no"     "he is lying"   "no"                            |             |
| "correct, he is lying"   "yes"     "it's a lie"          |             |
| "no"     "you are mistaken"   "no"                       |             |
| "that's a mistake"   "no"     "wrong answer"             |             |
| "no"     "it's not a lie"   "yes"     "he                |             |
| is not lying"   "yes"     "you are not mistaken"         |             |
| "yes"     "you are not wrong"   "yes"                    |             |
| "beans"   None   |             |

---

## 4. Multiple-Choice Prompts with `ask_selection()`

Let users choose from a list of options, by name or number.

```

from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class IceCreamSkill(OVOSSkill):
    def initialize(self):
        self.flavors = ['vanilla', 'chocolate', 'mint']

    @intent_handler('request.icecream.intent')
    def handle_request_icecream(self):
        self.speak_dialog('welcome')
        selection = self.ask_selection(self.flavors, 'what.flavor')
        self.speak_dialog('coming.right_up', {'flavor': selection})

```

### Optional arguments:

`min_conf` (float): Minimum confidence threshold for fuzzy matching

`numeric` (bool): If True, speak the list with numbered options

User responses like "chocolate", "the second one", or "option three" are all supported.

---

## Technical Notes

All methods handle microphone activation and parsing behind the scenes.

OVOS automatically integrates with the intent engine to resolve follow-up responses.

These prompts are designed to support natural dialogue flows, validating and re-prompting as needed.

---

## Tips

Always confirm user input when using `get_response()` or `ask_selection()` for clarity.

Use `validator` with `get_response()` to catch unclear or unwanted input.

Use `ask_yesno()` for quick binary decisions, but gracefully handle unexpected answers.

## docs/501-context.md

### Follow up questions

Conversational context in Open Voice OS (OVOS) allows voice interactions to feel more natural by remembering parts of a conversation, like the subject being discussed. This is especially useful for follow-up questions where repeating context (like a person's name) would otherwise be necessary.

Currently, conversational context is only supported with the [Adapt Intent Parser](#), not [Padatious](#).

---

### Keyword Contexts

> How tall is John Cleese?

"John Cleese is 196 centimeters"

> Where's he from?

"He's from England"

Context is added manually by the **Skill** creator using either the `self.set_context()` method or the `@adds_context()` decorator.

Consider the following intent handlers:

```
@intent_handler(IntentBuilder().require('PythonPerson').require('Length'))
def handle_length(self, message):
    python = message.data.get('PythonPerson')
    self.speak(f'{python} is {length_dict[python]} cm tall')

@intent_handler(IntentBuilder().require('PythonPerson').require('WhereFrom'))
def handle_from(self, message):
    python = message.data.get('PythonPerson')
    self.speak(f'{python} is from {from_dict[python]}')
```

To interact with the above handlers the user would need to say

```
User: How tall is John Cleese?
Mycroft: John Cleese is 196 centimeters
User: Where is John Cleese from?
Mycroft: He's from England
```

To get a more natural response the functions can be changed to let OVOS know which `PythonPerson` we're talking about by using the `self.set_context()` method to give context:

```
@intent_handler(IntentBuilder().require('PythonPerson').require('Length'))
def handle_length(self, message):
    # PythonPerson can be any of the Monty Python members
    python = message.data.get('PythonPerson')
    self.speak(f'{python} is {length_dict[python]} cm tall')
    self.set_context('PythonPerson', python)

@intent_handler(IntentBuilder().require('PythonPerson').require('WhereFrom'))
def handle_from(self, message):
    # PythonPerson can be any of the Monty Python members
    python = message.data.get('PythonPerson')
    self.speak(f'He is from {from_dict[python]}')
    self.set_context('PythonPerson', python)
```

When either of the methods are called the `PythonPerson` keyword is added to OVOS's context, which means that if there is a match with `Length` but `PythonPerson` is missing OVOS will assume the last mention of that keyword. The interaction can now become the one described at the top of the page.

> User: How tall is John Cleese?

OVOS detects the `Length` keyword and the `PythonPerson` keyword

> OVOS: 196 centimeters

John Cleese is added to the current context

> User: Where's he from?

OVOS detects the `WhereFrom` keyword but not any `PythonPerson` keyword. The Context Manager is activated and returns the latest entry of `PythonPerson` which is `_John Cleese_`

> OVOS: He's from England

# Cross Skill Context

The context is limited by the keywords provided by the **current** Skill.

But we can use context across skills via `self.set_cross_skill_context` to enable conversations with **other** Skills as well.

```
@intent_handler(IntentBuilder().require(PythonPerson).require(WhereFrom))
def handle_from(self, message):
    # PythonPerson can be any of the Monty Python members
    python = message.data.get('PythonPerson')
    self.speak(f'He is from {from_dict[python]}')
    self.set_context('PythonPerson', python) # context for this skill only

    self.set_cross_skill_context('Location', from_dict[python]) # context for ALL skills
```

In this example Location keyword is shared with the WeatherSkill

```
User: Where is John Cleese from?
Mycroft: He's from England
User: What's the weather like over there?
Mycroft: Raining and 14 degrees...
```

## Hint Keyword contexts

Context do not need to have a value, their presence can be used to simply indicate a previous interaction happened

In this case Context can also be implemented by using decorators instead of calling `self.set_context`

```

from ovos_workshop.decorators import adds_context, removes_context

class TeaSkill(OVOSSkill):
    @intent_handler(IntentBuilder('TeaIntent').require("TeaKeyword"))
    @adds_context('MilkContext')
    def handle_tea_intent(self, message):
        self.milk = False
        self.speak('Of course, would you like Milk with that?',
                    expect_response=True)

    @intent_handler(IntentBuilder('NoMilkIntent').require("NoKeyword").
                    require('MilkContext').build())
    @removes_context('MilkContext')
    @adds_context('HoneyContext')
    def handle_no_milk_intent(self, message):
        self.speak('all right, any Honey?', expect_response=True)

```

> **NOTE:** cross skill context is not yet exposed via decorators

## Using context to enable Intents

To make sure certain **Intents** can't be triggered unless some previous stage in a conversation has occurred. Context can be used to create "bubbles" of available intent handlers.

```

User: Hey Mycroft, bring me some Tea
Mycroft: Of course, would you like Milk with that?
User: No
Mycroft: How about some Honey?
User: All right then
Mycroft: Here you go, here's your Tea with Honey

```

```

        else:
from ovos_workshop.decorators import adds_context, removes_context
        self.speak('Heres your Tea with Honey')

class TeaSkill(OVOSkill):
    @intent_handler(IntentBuilder('TeaIntent').require("TeaKeyword"))
    @adds_context('MilkContext')
    def handle_tea_intent(self, message):
        self.milk = False
        self.speak('Of course, would you like Milk with that?',
                    expect_response=True)

    @intent_handler(IntentBuilder('NoMilkIntent').require("NoKeyword").
                    require('MilkContext').build())
    @removes_context('MilkContext')
    @adds_context('HoneyContext')
    def handle_no_milk_intent(self, message):
        self.speak('all right, any Honey?', expect_response=True)

    @intent_handler(IntentBuilder('YesMilkIntent').require("YesKeyword").
                    require('MilkContext').build())
    @removes_context('MilkContext')
    @adds_context('HoneyContext')
    def handle_yes_milk_intent(self, message):
        self.milk = True
        self.speak('What about Honey?', expect_response=True)

    @intent_handler(IntentBuilder('NoHoneyIntent').require("NoKeyword").
                    require('HoneyContext').build())
    @removes_context('HoneyContext')
    def handle_no_honey_intent(self, message):
        if self.milk:
            self.speak('Heres your Tea with a dash of Milk')
        else:
            self.speak('Heres your Tea, straight up')

    @intent_handler(IntentBuilder('YesHoneyIntent').require("YesKeyword").
                    require('HoneyContext').build())
    @removes_context('HoneyContext')
    def handle_yes_honey_intent(self, message):
        if self.milk:
            self.speak('Heres your Tea with Milk and Honey')

```



When starting up only the `TeaIntent` will be available. When that has been triggered and `_MilkContext_` is added the `MilkYesIntent` and `MilkNoIntent` are available since the `_MilkContext_` is set. when a `_yes_` or `_no_` is received the `_MilkContext_` is removed and can't be accessed. In it's place the `_HoneyContext_` is added making the `YesHoneyIntent` and `NoHoneyIntent` available.

You can find an example [Tea Skill using conversational context on Github](#).

As you can see, Conversational Context lends itself well to implementing a [dialog tree or conversation tree](#).

## docs/502-converse.md

### Converse

Each Skill may define a `converse()` method. This method will be called anytime the Skill has been recently active and a new utterance is processed.

The converse method expects a single argument which is a standard Mycroft Message object. This is the same object an intent handler receives.

Converse methods must return a Boolean value. True if an utterance was handled, otherwise False.

### Basic usage

Let's use a version of the Ice Cream Skill we've been building up and add a converse method to catch any brief statements of thanks that might directly follow an order.

```

from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class IceCreamSkill(OVOSSkill):
    def initialize(self):
        self.flavors = ['vanilla', 'chocolate', 'mint']

    @intent_handler('request.icecream.intent')
    def handle_request_icecream(self):
        self.speak_dialog('welcome')
        selection = self.ask_selection(self.flavors, 'what.flavor')
        self.speak_dialog('coming-right-up', {'flavor': selection})

    def converse(self, message):
        if self.voc_match(message.data['utterances'][0], 'Thankyou'):
            self.speak_dialog("you-are-welcome")
            return True

```

In this example:

1. A User might request an ice cream which is handled by `handle_request_icecream()`;
  2. The Skill would be added to the system Active Skill list for up to 5 minutes.
  3. Any utterance received by OVOS would trigger this Skills converse system whilst it is considered active.
  4. If the User followed up with a pleasantry such as "Hey Mycroft, thanks" - the converse method would match this vocab against the `Thankyou.voc` file in the Skill and speak the contents of the `you-are-welcome.dialog` file. The method would return `True` and the utterance would be consumed meaning the intent parsing service would never be triggered.
  5. Any utterance that did not match would be silently ignored and allowed to continue on to other converse methods and finally to the intent parsing service.
- > **WARNING** skills that are not [Session](#) aware may behave weirdly with voice satellites, see the [parrot skill](#) for an example.

# Active Skill List

A Skill is considered active if it has been called in the last 5 minutes.

Skills are called in order of when they were last active. For example, if a user spoke the following commands:

> Hey Mycroft, set a timer for 10 minutes > > Hey Mycroft, what's the weather

Then the utterance "what's the weather" would first be sent to the Timer Skill's `converse()` method, then to the intent service for normal handling where the Weather Skill would be called.

As the Weather Skill was called it has now been added to the front of the Active Skills List. Hence, the next utterance received will be directed to:

1. `WeatherSkill.converse()`
2. `TimerSkill.converse()`
3. Normal intent parsing service

When does a skill become active?

1. **before** an intent is called the skill is **activated**
2. if a fallback **returns True** (to consume the utterance) the skill is **activated** right **after** the fallback
3. if `converse` **returns True** (to consume the utterance) the skill is **reactivated** right **after** `converse`
4. a skill can activate/deactivate itself at any time

## Making a Skill Active

There are occasions where a Skill has not been triggered by the User, but it should still be considered "Active".

In the case of our Ice Cream Skill - we might have a function that will execute when the customers order is ready. At this point, we also want to be responsive to the customers thanks, so we call `self.activate()` to manually add our Skill to the front of the Active Skills List.

```

from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler

class IceCreamSkill(OVOSSkill):
    def on_order_ready(self, message):
        self.activate()

    def handle_activate(self, message: Message):
        """
        Called when this skill is considered active by the intent service;
        converse method will be called with every utterance.
        Override this method to do any optional preparation.
        @param message: `{self.skill_id}.activate` Message
        """
        LOG.info("Skill has been activated")

```

## Deactivating a Skill

The active skill list will be pruned by `ovos-core`, any skills that have not been interacted with for longer than 5 minutes will be deactivated

Individual Skills may react to this event, to clean up state or, in some rare cases, to reactivate themselves

```

from ovos_workshop.skills import OVOSSkill

class AlwaysActiveSkill(OVOSSkill):

    def handle_deactivate(self, message: Message):
        """
        Called when this skill is no longer considered active by the intent
        service; converse method will not be called until skill is active again.
        Override this method to do any optional cleanup.
        @param message: `{self.skill_id}.deactivate` Message
        """
        self.activate()

```

A skill can also deactivate itself at any time

```
from ovos_workshop.skills import OVOSSkill
```

```
class LazySkill(OVOSSkill):
```

```
    def handle_intent(self, message: Message):
        self.speak("leave me alone")
        self.deactivate()
```

## Conversational Intents

**NEW** in ovos-core version **0.0.8**

Skills can have extra intents valid while they are active, those are internal and not part of the main intent system, instead each skill checks them BEFORE calling `converse`

the `@conversational_intent` decorator can be used to define converse intent handlers

these intents only trigger after an initial interaction, essentially they are only follow up questions

```
class DogFactsSkill(OVOSSkill):
```

```
    @intent_handler("dog_facts.intent")
```

```
    def handle_intent(self, message):
```

```
        fact = "Dogs sense of smell is estimated to be 100,000 times more sensitive than h
        self.speak(fact)
```

```
    @conversational_intent("another_one.intent")
```

```
    def handle_followup_question(self, message):
```

```
        fact2 = "Dogs have a unique nose print, making each one distinct and identifiable
        self.speak(fact2)
```

> **NOTE:** Only works with `.intent` files, Adapt/Keyword intents are NOT supported

A more complex example, a game skill that allows saving/exiting the game only during playback

```

class MyGameSkill(OVOSSkill):

    @intent_handler("play.intent")
    def handle_play(self, message):
        self.start_game(load_save=True)

    @conversational_intent("exit.intent")
    def handle_exit(self, message):
        self.exit_game()

    @conversational_intent("save.intent")
    def handle_save(self, message):
        self.save_game()

    def handle_deactivate(self, message):
        self.game_over() # user abandoned interaction

    def converse(self, message):
        if self.playing:
            # do some game stuff with the utterance
            return True
        return False

```

> **NOTE:** if these intents trigger, they are called **INSTEAD** of converse

## docs/503-layers.md

### Intent Layers

> **WARNING:** Skills using these features might not play well with [HiveMind](#) due to shared state across satellites

### Managing Intents

Sometimes you might want to manually enable or disable an intent, in OVOSSkills you can do this explicitly to create stateful interactions

```

class RotatingIntentsSkill(OVOSSkill):

    def initialize(self):
        # NOTE: this must be done in initialize, not in __init__
        self.disable_intent("B.intent")
        self.disable_intent("C.intent")

    @intent_handler("A.intent")
    def handle_A_intent(self, message):
        # do stuff
        self.enable_intent("B.intent")
        self.disable_intent("A.intent")

    @intent_handler("B.intent")
    def handle_B_intent(self, message):
        # do stuff
        self.enable_intent("C.intent")
        self.disable_intent("B.intent")

    @intent_handler("C.intent")
    def handle_C_intent(self, message):
        # do stuff
        self.enable_intent("A.intent")
        self.disable_intent("C.intent")

```

> **NOTE:** Intent states are currently shared across Sessions

## State Machines

Another utils provided by `ovos-workshop` is `IntentLayers`, to manage groups of intent together

`IntentLayers` lend themselves well to implement state machines.

## The Manual way

In this example we implement the [Konami Code](#), doing everything the manual way instead of using decorators

[!imagem](#)

```

def handle_intent_layers(message):
class KonamiCodeSkill(OVOSSkill):
    def initialize(self):
        self.intent_layers.deactivate_layer("up1")
        self.register_intent_layers.activate_layer("up2")
        self.counter = 0
    else:
        self.top_fails = 3
    def stop(self):
        self.intent_layers.activate_layer("down1")
        if self.intent_layers.deactivate_layer("up2")
        up_intent = IntentBuilder('KonamiUpIntent').require("KonamiUpKeyword").build()
        self.acknowledge()
        down_intent = IntentBuilder('KonamiDownIntent').require("KonamiDownKeyword").build()
        left_intent = IntentBuilder('KonamiLeftIntent').require("KonamiLeftKeyword").build()
    def handle_down_intent_layers(message):
        right_intent = IntentBuilder('KonamiRightIntent').require("KonamiRightKeyword").build()
        if self.intent_layers.is_active("down1"):
            b_intent = IntentBuilder('KonamiBIntent').require("KonamiBKeyword").build()
            self.intent_layers.deactivate_layer("down1")
            a_intent = IntentBuilder('KonamiAIntent').require("KonamiAKeyword").build()
            self.intent_layers.activate_layer("down2")
            "KonamiDownKeyword",
        else:
            self.register_intent(up_intent, self.handle_up_intent)
            self.intent_layers.activate_layer("left1")
            self.register_intent(down_intent, self.handle_down_intent)
            self.intent_layers.deactivate_layer("down2")
            self.register_intent(left_intent, self.handle_left_intent)
            self.acknowledge()
            self.register_intent(right_intent, self.handle_right_intent)
            self.counter += 1
            self.register_intent(b_intent, self.handle_b_intent)
    def handle_left_intent_layers(message):
        self.register_intent(a_intent, self.handle_a_intent)
        if self.intent_layers.is_active("left1"):
            self.intent_layers.deactivate_layer("left1")
    def build_intent_layers(self):
        self.intent_layers.activate_layer("right1")
        self.intent_layers.update_layer("up1", ["KonamiUpIntent"])
        else:
            self.speak("Wrong! Try again")
            self.intent_layers.update_layer("up2", ["KonamiUpIntent"])
            self.intent_layers.deactivate_layer("left2")
            self.intent_layers.update_layer("down1", ["KonamiDownIntent"])
            self.intent_layers.activate_layer("right2")
            self.intent_layers.update_layer("down2", ["KonamiDownIntent"])
            self.acknowledge()
            self.intent_layers.update_layer("left1", ["KonamiLeftIntent"])
            self.intent_layers.update_layer("right1", ["KonamiRightIntent"])
    def handle_right_intent(self, message):
        self.intent_layers.update_layer("left2", ["KonamiLeftIntent"])
        if self.intent_layers.is_active("right1"):
            self.intent_layers.update_layer("right2", ["KonamiRightIntent"])
            self.intent_layers.deactivate_layer("right1")
            self.intent_layers.update_layer("B", ["KonamiBIntent"])
            self.intent_layers.activate_layer("left2")
            self.intent_layers.update_layer("A", ["KonamiAIntent"])
        else:
            self.intent_layers.activate_layer("B")
            self.intent_layers.activate_layer("up1")
            self.intent_layers.deactivate_layer("right2")
        self.acknowledge()
    def reset(self):
        self.active = False
    def handle_b_intent(self, message):
        self.counter = 0
        self.intent_layers.activate_layer("A")
        self.intent_layers.disable()
        self.intent_layers.deactivate_layer("B")
        self.intent_layers.activate_layer("up1")
        self.acknowledge()
    def handle_up_intent(self, message):

```





## Decorators

When you have many complex chained intents `IntentLayers` often makes your life easier, a layer is a named group of intents that you can manage at once.

Slightly more complex than the previous example, we may want to offer several "forks" on the intent execution, enabling different intent groups depending on previous interactions

[skill-moon-game](#) is an example full voice game implemented this way

An excerpt from the game to illustrate usage of `IntentLayer` decorators

> **NOTE:** `IntentLayers` do not yet support `Session`, in this example all [voice satellites](#) would join the game

```

        self.speak_dialog("guard_yes")
from ovos_workshop.skills.decorators import layer_intent, enables_layer, \
    self.briefing_question1()
    disables_layer, resets_layers

    @layer_intent(IntentBuilder("No1Apollo11Intent").require("noKeyword"),
                  layer_name="guard")
class Apollo11GameSkill(OVOSSkill):
    @enables_layer(layer_name="guard2")
    @disables_layer(layer_name="guard")
    def initialize(self):
    def handle_no1(self, message=None):
        # start with all game states disabled
        self.speak_dialog("guard_no")
        self.intent_layers.disable()
        self.speak_dialog("present_id", expect_response=True)

    @intent_handler(IntentBuilder("StartApollo11Intent"). \
    # (...) more intent layers
        optionally("startKeyword"). \
            require("MoonGameKeyword"))
    def converse(self, message):
    def handle_start_intent(self, message=None):
        if not self.playing:
        if not self.playing:
            return False
            self.playing = True
        # (..)
        self.speak_dialog("start.game")
        # take corrective action when no intent matched
        self.handle_intro()
        if self.intent_layers.is_active("guard") or \
        else:
            self.intent_layers.is_active("guard2"):
            self.speak_dialog("already.started")
            self.speak_dialog("guard_dead")

        self.handle_game_over()
    @layer_intent(IntentBuilder("StopApollo11Intent"). \
    # (...)
        require("stopKeyword"). \
            else:
                optionally("MoonGameKeyword"),
                self.speak_dialog("invalid.command", expect_response=True)
                layer_name="stop_game")
        return True
    @resets_layers()
    def handle_game_over(self, message=None):
        if self.playing:
            self.speak_dialog("stop.game")

    @enables_layer(layer_name="guard")
    @enables_layer(layer_name="stop_game")
    def handle_intro(self):
        self.speak_dialog("reach_gate")
        self.speak_dialog("guard")
        self.speak_dialog("present_id", expect_response=True)

    @layer_intent(IntentBuilder("Yes1Apollo11Intent").require("yesKeyword"),
                  layer_name="guard")
    def handle_yes1(self, message=None):

```

**docs/504-session.md**

## **Session Aware Skills**

**NEW** ovos-core version **0.0.8**

If you want your skills to handle simultaneous users you need to make them [Session](#) aware

Each remote client, usually a [voice satellite](#), will send a `Session` with the `Message`

Your skill should keep track of any Session specific state separately, eg, a chat history

> **WARNING:** Stateful Skills need to be Session Aware to play well with [HiveMind](#)

## SessionManager

You can access the Session in a Message object via the SessionManager class

```
from ovos_bus_client.session import SessionManager, Session

class MySkill(OVOSkill):

    def on_something(self, message):
        sess = SessionManager.get(message)
        print(sess.session_id)
```

If the message originated in the device itself, the `session_id` is always equal to "default", if it comes from an external client then it will be a unique uuid

## Magic Properties

Skills have some "magic properties", these will always reflect the value in the current Session

```

    Get a SkillResources object for the current language. Objects are
    # magic properties -> depend on message.context / Session
    initialized for the current Session language as needed.
@property
    """
def lang(self) -> str:
    """
    Get the current language as a BCP-47 language code. This will consider
    current session data if available, else Configuration.
    """

@property
def location(self) -> dict:
    """
    Get the JSON data struction holding location information.
    This info can come from Session
    """

@property
def location_pretty(self) -> Optional[str]:
    """
    Get a speakable city from the location config if available
    This info can come from Session
    """

@property
def location_timezone(self) -> Optional[str]:
    """
    Get the timezone code, such as 'America/Los_Angeles'
    This info can come from Session
    """

@property
def dialog_renderer(self) -> Optional[MustacheDialogRenderer]:
    """
    Get a dialog renderer for this skill. Language will be determined by
    message context to match the language associated with the current
    session or else from Configuration.
    """

@property
def resources(self) -> SkillResources:
    """

```

## Per User Interactions

Let's consider a skill that keeps track of a chat history, how would such a skill keep track of `Sessions`?

```

from ovos_bus_client.session import SessionManager, Session
from ovos_workshop.decorators import intent_handler
from ovos_workshop.skills import OVOSkill

class UtteranceRepeaterSkill(OVOSkill):

    def initialize(self):
        self.chat_sessions = {}
        self.add_event('recognizer_loop:utterance', self.on_utterance)

    # keep chat history per session
    def on_utterance(self, message):
        utt = message.data['utterances'][0]
        sess = SessionManager.get(message)
        if sess.session_id not in self.chat_sessions:
            self.chat_sessions[sess.session_id] = {"current_stt": ""}
        self.chat_sessions[sess.session_id]["prev_stt"] = self.chat_sessions[sess.session_id]["current_stt"]
        self.chat_sessions[sess.session_id]["current_stt"] = utt

    # retrieve previous STT per session
    @intent_handler('repeat.stt.intent')
    def handle_repeat_stt(self, message):
        sess = SessionManager.get(message)
        if sess.session_id not in self.chat_sessions:
            utt = self.translate('nothing')
        else:
            utt = self.chat_sessions[sess.session_id]["prev_stt"]
        self.speak_dialog('repeat.stt', {"stt": utt})

    # session specific stop event
    # if this method returns True then self.stop will NOT be called
    def stop_session(self, session: Session):
        if session.session_id in self.chat_sessions:
            self.chat_sessions.pop(session.session_id)
            return True
        return False

```

A full example can be found in the [parrot skill](#)



**docs/51-install\_raspovos.md**

# **RaspOVOS: A Beginner's Guide to Setting Up Your Raspberry Pi with OVOS**

This tutorial is designed for users new to Raspberry Pi and RaspOVOS. Follow these steps to set up and optimize your device for the best experience.

---

## **Step 1: Prepare Your Hardware**

### **Raspberry Pi Model Recommendations**

**Recommended:** Raspberry Pi 4 or 5.

For offline STT (speech-to-text), the **Raspberry Pi 5** offers significant performance improvements.

**Minimum Requirement:** Raspberry Pi 3.

**Note:** The Raspberry Pi 3 will work but may be **extremely slow** compared to newer models.

### **Storage Options**

**SD Card or USB Storage:**

You can use either a microSD card or a USB drive.

**Recommended:** USB SSD Drive for maximum speed and performance.

Connect the USB drive to the **blue USB 3.0 port** for optimal performance.

### **Power Supply Considerations**

Raspberry Pi boards are notoriously **picky about power supplies**. Insufficient power can lead to performance issues, random reboots, or the appearance of the **undervoltage detected** warning (a lightning bolt symbol in the top-right corner of the screen).

#### **Recommended Power Supplies:**

Raspberry Pi 4: 5V 3A USB-C power adapter.

Raspberry Pi 5: Official Raspberry Pi 5 USB-C power adapter or equivalent high-quality adapter with sufficient current capacity.

#### **Common Issues:**

Using cheap or low-quality chargers or cables may result in voltage drops.

Long or thin USB cables can cause resistance, reducing the power delivered to the board.

#### **How to Fix:**

Always use the official power adapter or a trusted brand with a stable 5V output.

If you see the "**undervoltage detected**" warning, consider replacing your power supply or cable.

---

## **Step 2: Install RaspOVOS Image**

### **1. Download and Install Raspberry Pi Imager**

2. Visit [Raspberry Pi Imager](#) and download the appropriate version for your OS.
3. Install and launch the imager.

### **1. Flash the Image to Storage**

2. Insert your SD card or USB drive into your computer.
3. In the Raspberry Pi Imager:
4. **Choose OS:** Select "Use custom" and locate the RaspOVOS image file.

5. **Choose Storage:** Select your SD card or USB drive.

!image

!image

!image

### 1. **Advanced Configuration Options**

2. Click **Next** and select **Edit Settings** to customize settings, including:

3. **Password:** Change the default password.

4. **Hostname:** Set a custom hostname for your device.

5. **Wi-Fi Credentials:** Enter your Wi-Fi network name and password.

6. **Keyboard Layout:** Configure the correct layout for your region.

**Important: Do NOT change the default username** (ovos), as it is required for the system to function properly.

!image

!image

### 1. **Write the Image**

2. Click **Save** and then **Yes** to flash the image onto your storage device.

3. Once complete, safely remove the SD card or USB drive from your computer.

---

## Step 3: Initial Setup and First Boot

### Connect and Power On

Insert the SD card or connect the USB drive to your Raspberry Pi.

Plug in the power supply and connect an HDMI monitor to observe the boot process.

### First Boot Process

### 1. Initialization:

2. The system will expand the filesystem, generate SSH keys, and perform other setups.

### 3. Reboots:

4. The device will reboot **up to three times** during this process.

### 5. Autologin:

6. The `ovos` user will automatically log in to the terminal after boot.

### 1. Check System Status:

2. Use the `ologs` command to monitor logs and confirm that the system has fully initialized.

---

## Step 4: Setting Up Wi-Fi

### Option 1: Configure Wi-Fi Using Raspberry Pi Imager

The most straightforward method is to set up Wi-Fi during the imaging process.

1. Open Raspberry Pi Imager and select Edit Settings Option.
2. Enter your **SSID (Wi-Fi network name)** and **password** in the Wi-Fi configuration fields.
3. Write the image to your SD card or USB drive, and your Wi-Fi will be pre-configured.

### Option 2: Use Audio-Based Wi-Fi Setup (ggwave)

1. Open [ggwave Wi-Fi setup](#) on a device with speakers.
2. Enter your **SSID** and **password** and transmit the data as sound.
3. Place the transmitting device near the Raspberry Pi microphone.

4. If successful, you'll hear an acknowledgment tone.

5. If decoding fails or credentials are incorrect, you'll hear an error tone.

**Note:** ggwave is a **work-in-progress** feature and does not have any dialogs or provide on-screen feedback.

[!image](#)

---

## Step 5: Running OVOS

### OVOS First Launch

On the first run, OVOS may take longer to initialize.

When ready, OVOS will say: "**I am ready**" (requires an Internet connection).

---

## Step 6: Using OVOS Commands

### Helpful Commands

Once the terminal appears, you'll see a guide with OVOS commands. Some key commands include:

#### Configuration:

`ovos-config` Manage configuration files.

#### Voice Commands:

`ovos-listen` Activate the microphone for commands.

`ovos-speak <phrase>` Make OVOS speak a specific phrase.

#### Skill Management:

`ovos-install [PACKAGE_NAME]` Install OVOS packages.

`ovos-update` Update all OVOS and skill packages.

### Logs and Status:

`o logs` View logs in real-time.

`ovos-status` Check the status of OVOS-related services.

You use the command `ovos-help` to print the message with all commands again at any point

## Check Logs in Real-Time

Use the `o logs` command to monitor logs live on your screen.

If you're unsure whether the system has finished booting, check logs using this command.

---

Enjoy your journey with RaspOVOS! With your Raspberry Pi set up, you can start exploring all the features of OpenVoiceOS.

## docs/54-skill-examples.md

## Default Skills overview

A non-exhaustive list of skills available for OpenVoiceOS, these might be available out of the box or not depending on how you installed OVOS

---

## ovos-skill-alerts.openvoiceos

A skill to manage alarms, timers, reminders, events and todos and optionally sync them with a CalDAV service.

### Usage examples:

What are my reminders?

Cancel all reminders.

When is my next alarm?

Schedule a tennis event for 2 PM on friday spanning 2 hours.

What did I miss?

remind me to take out the trash every Thursday and Sunday at 7 PM.

Start a bread timer for 30 minutes.

Did I miss anything?

Set an alarm for 8 AM.

Set a daily alarm for 8 AM.

-----

## ovos-skill-cmd.forslund

No description available

### Usage examples:

run script \_\_\_\_

launch command \_\_\_\_

-----

## ovos-skill-confucius-quotes.openvoiceos

Quotes from Confucius

### Usage examples:

Quote from Confucius

When did Confucius die

When was Confucius born

Who is Confucius

-----

## ovos-skill-days-in-history.openvoiceos

Provides historical events for today or any other calendar day using information pulled from Wikipedia.

### Usage examples:

who died today in history?

who was born today in history?

What historical events happened on June 16th?

Tell me about events in history on December 12th

What happened today in history?

-----

## ovos-skill-dictation.openvoiceos

continuously transcribes user speech to text file while enabled

### Usage examples:

start dictation

end dictation

-----

## ovos-skill-ip.openvoiceos

Network connection information

### Usage examples:

What's your IP address?

What's your network address?

Tell me your network address

What network are you connected to?

Tell me your IP address



-----

## ovos-skill-iss-location.openvoiceos

Track the location of the ISS

### Usage examples:

When is the ISS passing over

Where is the ISS

Tell me about the IS

how many persons on board of the space station

Who is on board of the space station?

-----

## ovos-skill-moviemaster.openvoiceos

Find information about movies, actors, and production details. Easily find information about a movie with your voice.

### Usage examples:

What are popular movies playing now?

Tell me about the movie \_\_\_\_\_

What genres does the flick \_\_\_\_\_ belong to?

Who plays in the movie \_\_\_\_\_?

How long is the movie \_\_\_\_\_?

Look for information on the movie \_\_\_\_\_.

Do you have info on the film \_\_\_\_\_?

What is the movie \_\_\_\_\_ about?

What are the highest rated movies out?

When was the movie \_\_\_\_\_ made?

-----

## ovos-skill-number-facts.openvoiceos

Facts about numbers

### Usage examples:

random number trivia

trivia about next week

trivia about tomorrow

fact about number 666

fact about yesterday

curiosity about year 1992

math fact about number 7

-----

## ovos-skill-personal.openvoiceos

Learn history and personality of the assistant. Ask about the 'birth' and parentage of the voice assistant and get a taste of the community who is fostering this open source artificial intelligence.

### Usage examples:

Where were you born?

What are you?

When were you created?

Who made you?

-----

## ovos-skill-pyrradios.openvoiceos

a client for the client for the Radio Browser API

**Usage examples:**

play tsf jazz on pyradios

play tsf jazz radio

-----

## ovos-skill-speedtest.openvoiceos

runs an internet bandwidth test using speedtest.net

**Usage examples:**

run a speedtest

-----

## ovos-skill-wikihow.openvoiceos

How to do nearly everything.

**Usage examples:**

how do i get my dog to stop barking

how to boil an egg

-----

## skill-ovos-audio-recording.openvoiceos

No description available

**Usage examples:**

new recording named {name}

start recording

start a recording called {name}

start a new audio recording called {name}

begin recording

-----

## **skill-ovos-boot-finished.openvoiceos**

The Finished Booting skill provides notifications when OpenVoiceOS has fully started and all core services are ready

### **Usage examples:**

Disable ready notifications.

Is the system ready?

Enable ready notifications.

-----

## **skill-ovos-date-time.openvoiceos**

Get the time, date, day of the week

### **Usage examples:**

What time is it?

Tell me the day of the week

What day is Memorial Day 2020?

What's the date?

Show me the time

How many days until July 4th

What time is it in Paris?

-----

## **skill-ovos-ddg.openvoiceos**

Use DuckDuckGo to answer questions.

### **Usage examples:**

ask the duck about the big bang

when was stephen hawking born

who is elon musk

-----

## **skill-ovos-hello-world.openvoiceos**

Introductory Skill so that Skill Authors can see how an OVOS Skill is put together

### **Usage examples:**

Hello world

Thank you

How are you?

-----

## **skill-ovos-icanhazdadjokes.openvoiceos**

Brighten your day with dad humor. Laughter is not guaranteed, but eye rolls are likely.

### **Usage examples:**

Can you tell jokes?

Make me laugh.

Do you know any Chuck Norris jokes?

Tell me a joke about dentists.

Say a joke.

Tell me a joke.

Do you know any jokes?

-----

# **skill-ovos-local-media.openvoiceos**

Local Media File Browser For Open Voice OS

## **Usage examples:**

open my file browser

show my file browser

show my usb drive

start usb browser app

show my usb

show file browser app

show file browser

open usb

start usb browser

open my usb

-----

# **skill-ovos-naptime.openvoiceos**

Put the assistant to sleep when you don't want to be disturbed.

## **Usage examples:**

Nap time

Wake up

Go to sleep

-----

# **skill-ovos-news.openvoiceos**

News streams from around the globe.

**Usage examples:**

play npr news

play news in spanish

play euronews

play the news

play portuguese news

play catalan news

-----

## **skill-ovos-parrot.openvoiceos**

Turn OpenVoiceOS into a echoing parrot! Make OVOS repeat whatever you want.

**Usage examples:**

Tell me what I just said.

say Goodnight, Gracie

speak I can say anything you'd like!

start parrot

repeat Once upon a midnight dreary, while I pondered, weak and weary, Over  
many a quaint and curious volume of forgotten lore

Repeat what you just said

What did I just say?

Can you repeat that?

stop parrot

Repeat that

-----

## **skill-ovos-somafm.openvoiceos**

No description available

### **Usage examples:**

play soma fm radio

play metal detector

play secret agent

-----

## **skill-ovos-spelling.openvoiceos**

No description available

### **Usage examples:**

How do you spell bureacracy?

How do you spell aardvark?

Spell omnipotence

Spell succotash

-----

## **skill-ovos-volume.openvoiceos**

Control the volume of OVOS with verbal commands

### **Usage examples:**

unmute volume

volume low

mute audio

volume to high level

reset volume



volume to high

volume level low

toggle audio

low volume

set volume to maximum

-----

## **skill-ovos-weather.openvoiceos**

Get weather conditions, forecasts, expected precipitation and more! You can also ask for other cities around the world. Current conditions and weather forecasts come from OpenMeteo

### **Usage examples:**

What's the temperature in Paris tomorrow in Celsius?

When will it rain next?

What's the high temperature tomorrow

Is it going to snow in Baltimore?

what is the weather like?

How windy is it?

What is the weather this weekend?

What is the weather in Houston?

Will it be cold on Tuesday

What's the temperature?

-----

## **skill-ovos-wikipedia.openvoiceos**

Query Wikipedia for answers to all your questions. Get just a summary, or ask for

more to get in-depth information.

**Usage examples:**

Search for chocolate

More information

Tell me about beans

Tell me More

Tell me about the Pembroke Welsh Corgi

Check Wikipedia for beans

Tell me about Elon Musk

-----

## **skill-ovos-wolfie.openvoiceos**

Use Wolfram Alpha for general knowledge questions.

**Usage examples:**

How tall is Mount Everest?

What's 18 times 4?

How many inches in a meter?

What is Madonna's real name?

When was The Rocky Horror Picture Show released?

ask the wolf what is the speed of light

-----

## **skill-ovos-wordnet.openvoiceos**

Use Wordnet to answer dictionary-like questions.

**Usage examples:**

what is the definition of ...

what is the antonym of ...

-----

# docs/55-raspovos\_troubleshooting.m d

## RaspOVOS Troubleshooting

> This guide applies to raspOVOS and may assume some raspOVOS exclusive utilities are available, if you are not using raspOVOS [some command line utilities](#) will not be available

---

## Undervoltage Detected Warning

If you see an **undervoltage detected** warning:

Check your power adapter and cable.

Ensure the adapter can supply enough current (e.g., 5A for Raspberry Pi 5).

Replace long or thin cables with shorter, thicker ones for better power delivery.

---

## System Boot Issues

If the device does not complete its boot sequence:

1. Ensure the power supply is stable and sufficient for your Raspberry Pi model.
2. If the OS boots but OVOS doesn't work:
3. See if all OVOS services started up correctly with `ovos-status` command
4. Check log files in `~/.local/state/mycroft/` for OVOS error messages.

5. Re-flash the image if necessary, ensuring all configuration options are set correctly.

---

## OVOS Fails to Speak "I am Ready"

Confirm the device has a working Internet connection. otherwise OVOS won't consider itself ready

---

## How to debug intent matching

To easily debug intent parsing open a terminal and run `oologs | grep intent`, this will show you live logs related only to intent parsing

then in another terminal send commands with `ovos-say-to "sentence to test"` (or use your voice)

```
(ovos) ovos@raspOVOS:~ $ oologs | grep intent
2025-01-23 16:29:54.299 - skills - ovos_core.intent_services:handle_utterance:416 - INFO -
2025-01-23 16:29:54.300 - skills - ovos_core.intent_services:handle_utterance:436 - DEBUG
2025-01-23 16:34:07.672 - skills - ovos_core.intent_services:handle_utterance:399 - INFO -
2025-01-23 16:34:07.675 - skills - ovos_core.intent_services:get_pipeline:234 - DEBUG - Se
2025-01-23 16:34:07.678 - skills - ovos_core.intent_services:handle_utterance:430 - DEBUG
2025-01-23 16:34:07.686 - skills - ovos_core.intent_services:handle_utterance:430 - DEBUG
2025-01-23 16:34:07.691 - skills - ovos_core.intent_services:handle_utterance:430 - DEBUG
2025-01-23 16:34:07.696 - skills - ovos_core.intent_services:handle_utterance:416 - INFO -
2025-01-23 16:34:07.698 - skills - ovos_core.intent_services:handle_utterance:436 - DEBUG
```

## How to check installed skills

use the `ls-skills` command

```
'skill-ovos-news.openvoiceos'
(ovos) ovos@raspOVOS:~ $ ls-skills
'skill-ovos-wolfie.openvoiceos'
[INFO] Listing installed skills for OpenVoiceOS...
'ovos-skill-fuster-quotes.openvoiceos']
[WARNING] Scanning for installed skills. This may take a few moments, depending on the num
[SUCCESS] Skill listing completed.
```

The following skills are installed:

```
['skill-ovos-weather.openvoiceos',
 'ovos-skill-dictation.openvoiceos',
 'skill-ovos-parrot.openvoiceos',
 'ovos-skill-speedtest.openvoiceos',
 'ovos-skill-ip.openvoiceos',
 'skill-ovos-spelling.openvoiceos',
 'ovos-skill-iss-location.openvoiceos',
 'skill-ovos-audio-recording.openvoiceos',
 'skill-ovos-wordnet.openvoiceos',
 'ovos-skill-days-in-history.openvoiceos',
 'ovos-skill-confucius-quotes.openvoiceos',
 'skill-ovos-fallback-chatgpt.openvoiceos',
 'ovos-skill-alerts.openvoiceos',
 'skill-ovos-local-media.openvoiceos',
 'skill-ovos-volume.openvoiceos',
 'ovos-skill-wikihow.openvoiceos',
 'ovos-skill-personal.OpenVoiceOS',
 'ovos-skill-number-facts.openvoiceos',
 'skill-ovos-hello-world.openvoiceos',
 'ovos-skill-moviemaster.openvoiceos',
 'skill-ovos-date-time.openvoiceos',
 'skill-ovos-fallback-unknown.openvoiceos',
 'ovos-skill-pyradios.openvoiceos',
 'skill-ovos-icanhazdadjokes.openvoiceos',
 'ovos-skill-cmd.forslund',
 'ovos-skill-spotify.openvoiceos',
 'skill-ovos-randomness.openvoiceos',
 'skill-ovos-naptime.openvoiceos',
 'skill-ovos-wikipedia.openvoiceos',
 'skill-ovos-boot-finished.openvoiceos',
 'ovos-skill-camera.openvoiceos',
 'skill-ovos-ddg.openvoiceos',
 'ovos-skill-laugh.openvoiceos',
 'skill-ovos-somafm.openvoiceos',
```

## How to check available intents

Skills can optionally provide metadata, if they do instructions will be available under `ovos-commands`

```

32) - ovos-skill-laugh.openvoiceos
(ovos) ovos@raspOVOS:~$ ovos-commands
33) - skill-ovos-somafm.openvoiceos
#####
34) - skill-ovos-news.openvoiceos
OpenVoiceOS - Skills help
35) - skill-ovos-wolfie.openvoiceos
#####
36) - ovos-skill-fuster-quotes.openvoiceos

Select skill number: 36
Scanning skills...

Found 37 installed skills
Skill name: ovos-skill-fuster-quotes.openvoiceos
Skill ids:
Description: La cita del dia de Fuster
0) - skill-ovos-weather.openvoiceos
Usage examples:
1) - ovos-skill-dictation.openvoiceos
2) - La frase del Fuster del dia
   - skill-ovos-parrot.openvoiceos
3) - Necesito alguna idea fusteriana
   - ovos-skill-speedtest.openvoiceos
4) - Algun pensament fusteria?
   - ovos-skill-ip.openvoiceos
5) - Diguem un aforisme del Fuster
   - skill-ovos-spelling.openvoiceos
6) - Que diria Joan Fuster, aqui?
   - ovos-skill-iss-location.openvoiceos
7) - Vull sentir un aforisme fusteria
   - skill-ovos-audio-recording.openvoiceos
8) - Que diu en Fuster?
   - skill-ovos-wordnet.openvoiceos
9) - Que pensen els fusterians?
   - ovos-skill-days-in-history.openvoiceos
10) - Diques-me alguna cosa fusteriana
    - ovos-skill-confucius-quotes.openvoiceos
11) - skill-ovos-fallback-chatgpt.openvoiceos
12) - ovos-skill-alerts.openvoiceos
13) - skill-ovos-local-media.openvoiceos
14) - skill-ovos-volume.openvoiceos
15) - ovos-skill-wikihow.openvoiceos
16) - ovos-skill-personal.OpenVoiceOS
17) - ovos-skill-number-facts.openvoiceos
18) - skill-ovos-hello-world.openvoiceos
19) - ovos-skill-moviemaster.openvoiceos
20) - skill-ovos-date-time.openvoiceos
21) - skill-ovos-fallback-unknown.openvoiceos
22) - ovos-skill-pyradios.openvoiceos
23) - skill-ovos-icanhazdadjokes.openvoiceos
24) - ovos-skill-cmd.forslund
25) - ovos-skill-spotify.openvoiceos
26) - skill-ovos-randomness.openvoiceos
27) - skill-ovos-naptime.openvoiceos
28) - skill-ovos-wikipedia.openvoiceos
29) - skill-ovos-boot-finished.openvoiceos
30) - ovos-skill-camera.openvoiceos
31) - skill-ovos-ddg.openvoiceos

```

---

## How to remove all skills

If you want to revert OVOS to a blank state you can use `ovos-reset-brain` to remove ALL skills



```

--skillwolfee==0.3.0
(ovos) ovos@raspOVOS:~$ ovos-reset-brain
--skillwoodnet==0.1.1
[INFO] Starting OpenVoiceOS skill uninstallation process...
[SUCCESS] All tasks have been uninstalled successfully.
WARNING: This will uninstall all installed skills. Do you want to continue? (y/n): y
WARNING: Note: This operation only deletes the skills. Configuration files and pipelinep
Using Python 3.11.2 environment at: .venvs/ovos
Uninstalled 37 packages in 513ms
[INFO] The following skills will be uninstalled:
- ovos-skill-alerts==0.1.15
- ovos-skill-alerts
- ovos-skill-audio-recording==0.2.5a5
- ovos-skill-audio-recording
- ovos-skill-boot-finished==0.4.9
- ovos-skill-boot-finished
- ovos-skill-camera==1.0.3a4
- ovos-skill-camera
- ovos-skill-cmd==0.2.8
- ovos-skill-cmd
- ovos-skill-confucius-quotes==0.1.11a1
- ovos-skill-confucius-quotes
- ovos-skill-date-time==0.4.6
- ovos-skill-date-time
- ovos-skill-days-in-history==0.3.9
- ovos-skill-days-in-history
- ovos-skill-dictation==0.2.10
- ovos-skill-dictation
- ovos-skill-fallback-unknown==0.1.6a2
- ovos-skill-fallback-unknown
- ovos-skill-fuster-quotes==0.0.1
- ovos-skill-fuster-quotes
- ovos-skill-hello-world==0.1.11a4
- ovos-skill-hello-world
- ovos-skill-icanhazdadjokes==0.3.2
- ovos-skill-icanhazdadjokes
- ovos-skill-ip==0.2.7a1
- ovos-skill-ip
- ovos-skill-iss-location==0.2.10
- ovos-skill-iss-location
- ovos-skill-laugh==0.2.1a3
- ovos-skill-laugh
- ovos-skill-local-media==0.2.9
- ovos-skill-local-media
- ovos-skill-moviemaster==0.0.8a4
- ovos-skill-moviemaster
- ovos-skill-naptime==0.3.12a1
- ovos-skill-naptime
- ovos-skill-number-facts==0.1.10
- ovos-skill-number-facts
- ovos-skill-parrot==0.1.14
- ovos-skill-parrot
- ovos-skill-personal==0.1.9
- ovos-skill-personal
- ovos-skill-pyradios==0.1.5a1
- ovos-skill-pyradios
- ovos-skill-randomness==0.1.2a1
- ovos-skill-randomness
- ovos-skill-somafm==0.1.5
- ovos-skill-somafm
- ovos-skill-speedtest==0.3.3a4
- ovos-skill-speedtest
- ovos-skill-spelling==0.2.6a3
- ovos-skill-spelling
- ovos-skill-spotify==0.1.9
- ovos-skill-spotify
- ovos-skill-volume==0.1.13a2
- ovos-skill-volume
- ovos-skill-weather==0.1.14
- ovos-skill-weather
- ovos-skill-wikihow==0.2.14
- ovos-skill-wikihow
- ovos-skill-wikipedia==0.6.0a1
- ovos-skill-wikipedia
- skill-ddg==0.1.15
- skill-ddg
- skill-news==0.1.12
- skill-news
- skill-ovos-fallback-chatgpt==0.1.12
- skill-ovos-fallback-chatgpt

```



# Wake Word Issues

Wake word detection in raspOVOS offers several options, each with its advantages and limitations. Understanding these can help resolve potential issues and improve performance.

By default, raspOVOS uses the `precise-lite` model with the wake word "hey mycroft." This model was trained by MycroftAI for their Mark2 device. However, there are a few things to consider:

**Microphone Compatibility:** The performance of precise models can be impacted if the specific properties of your

microphone (e.g., sensitivity, frequency response) do not match the data used to train the model. While the default `precise-lite` model was trained with a balanced dataset from a variety of Mycroft users, there is no guarantee it will work optimally with your microphone.

**Speaker Demographics:** Precise models, including `precise-lite`, are often trained with datasets predominantly

featuring adult male voices. As a result, the model may perform poorly with voices that are outside this demographic, such as children's or women's voices. This is a common issue also seen in Speech-to-Text (STT) models.

## Custom Models

If the default model is not working well for you, consider training your own precise model. Here are some helpful resources for creating a more tailored solution:

[Helpful Wake Word Datasets on Hugging Face](#)

[Data Collection](#)

[Wake Word Trainer](#)

[precise-lite-trainer Code](#)

## Alternative Wake Word: Vosk Plugin

If you're looking for an alternative to the precise model, the Vosk wake word plugin is another option.

[Vosk Wake Word Plugin GitHub](#)

One of the main advantages of using the **Vosk Wake Word Plugin** is that it does **not require a training step**.

Instead, it uses Kaldi with a limited language model, which means it can work out-of-the-box with certain wake words without needing to collect and train custom data.

The performance of Vosk may vary depending on the wake word you choose. Some wake words may work better than others, so it's essential to test and evaluate the plugin with your chosen word.

Some wake words are hard to trigger, especially if missing from the language model vocabulary

> e.g. hey mycroft is usually transcribed as hey microsoft,  
example for "hey computer"

```
"listener": {
  "wake_word": "hey_computer"
},
"hotwords": {
  "hey_computer": {
    "module": "ovos-ww-plugin-vosk",
    "lang": "en",
    "listen": true,
    "debug": true,
    "samples": ["hey computer", "a computer", "hey computed"],
    "rule": "equals",
    "full_vocab": false,
  }
}
```

lang - lang code for model, optional, will use global value if not set. only

used to download models

`debug` - if true will print extra info, like the transcription contents, useful for adjusting "samples"

`rule` - how to process the transcript for detections

`contains` - if the transcript contains any of provided samples

`equals` - if the transcript exactly matches any of provided samples

`starts` - if the transcript starts with any of provided samples

`ends` - if the transcript ends with any of provided samples

`fuzzy` - fuzzy match transcript against samples

`samples` - list of samples to match the rules against, optional, by default uses keyword name

`full_vocab` - use the full language model vocabulary for transcriptions, if false (default) will run in keyword mode

> `"lang"` does not need to match the main language, if there is no vosk model for your language you can try faking it with similar sounding words from a different one

## Tips for Choosing a Good Wake Word

Selecting a wake word is crucial to improving the accuracy and responsiveness of your system. Here are some tips for choosing a wake word that will work well in various environments:

**3 or 4 Syllables:** Wake words that are 3 or 4 syllables long tend to perform better because they are more distinct

and less likely to be confused with common words in everyday speech. For example:

**Bad Example:** "Bob" (short, common name)

**Less Bad Example:** "Computer" (common word)

**Good Example:** "Ziggy" (uncommon)

**Better Example:** "Hey Ziggy" (3 syllables, longer)

**Uncommon Words:** Choose a wake word that is not often used in regular conversation. This reduces the chance of

false triggers when other words sound similar to your wake word. Unique and uncommon names, phrases, or combinations of sounds work best.

**Clear Pronunciation:** Make sure the wake word has a clear and easy-to-pronounce structure. Words with ambiguous or

difficult-to-articulate syllables may cause detection issues, especially in noisy environments.

**Avoid Overused Words:** Stay away from wake words like "hey" or "hello," as they are often used in daily speech and

can trigger false positives. Try combining a less common word with a familiar greeting for better results.

---

## Audio Issues

**Run Diagnostics script:**

raspOVOS includes a helper script `ovos-audio-diagnostics` that will print basic info about your sound system

```
(ovos) ovos@raspOVOS:~ $ ovos-audio-diagnostics
=====
raspOVOS Audio Diagnostics
=====

# Detected sound server:
pipewire

# Available audio outputs:
36 - Built-in Audio Stereo [vol: 0.40]
45 - Built-in Audio Stereo [vol: 0.85]
46 - Built-in Audio Digital Stereo (HDMI) [vol: 0.40]

# Default audio output:
ID: 36
NAME: WM8731 HiFi wm8731-hifi-0
CARD NUMBER: 2
CARD NAME: snd_rpi_proto
```

### Check Input Devices:

Run `arecord -l` to list all detected audio capture devices (microphones).

```
**** List of CAPTURE Hardware Devices ****
card 2: sndrpiproto [snd_rpi_proto], device 0: WM8731 HiFi wm8731-hifi-0 [WM8731 HiFi wm8731-hifi-0]
Subdevices: 0/1
Subdevice #0: subdevice #0
card 3: Device [USB Audio Device], device 0: USB Audio [USB Audio]
Subdevices: 1/1
Subdevice #0: subdevice #0
```

### Check Output Devices:

Run `aplay -l` to list all detected audio playback devices (speakers). Verify your card is being detected

correctly

```
**** List of PLAYBACK Hardware Devices ****
card 0: Headphones [bcm2835 Headphones], device 0: bcm2835 Headphones [bcm2835 Headphones]
Subdevices: 7/8
Subdevice #0: subdevice #0
Subdevice #1: subdevice #1
Subdevice #2: subdevice #2
Subdevice #3: subdevice #3
Subdevice #4: subdevice #4
Subdevice #5: subdevice #5
Subdevice #6: subdevice #6
Subdevice #7: subdevice #7
card 1: vc4hdmi [vc4-hdmi], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 2: sndrpiproto [snd_rpi_proto], device 0: WM8731 HiFi wm8731-hifi-0 [WM8731 HiFi wm8731-hifi-0]
Subdevices: 1/1
Subdevice #0: subdevice #0
```

### **Verify Volume and Mute status:**

Run alsamixer and verify that volume isn't too low or audio muted.

### **Check audio setup logs:**

During boot the audio setup generates logs, which are saved to the /tmp directory:

/tmp/autosoundcard.log (for soundcard autoconfiguration)



```

==> /tmp/autosoundcard.log <==
Fri 17 Jan 11:42:46 WET 2025 - **** List of PLAYBACK Hardware Devices ****
card 0: Headphones [bcm2835 Headphones], device 0: bcm2835 Headphones [bcm2835 Headphones]
Subdevices: 8/8
Subdevice #0: subdevice #0
Subdevice #1: subdevice #1
Subdevice #2: subdevice #2
Subdevice #3: subdevice #3
Subdevice #4: subdevice #4
Subdevice #5: subdevice #5
Subdevice #6: subdevice #6
Subdevice #7: subdevice #7
card 1: Device [USB Audio Device], device 0: USB Audio [USB Audio]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 2: vc4hdmi [vc4-hdmi], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 3: sndrpiproto [snd_rpi_proto], device 0: WM8731 HiFi wm8731-hifi-0 [WM8731 HiFi wm8731-hifi-0]
Subdevices: 0/1
Subdevice #0: subdevice #0
Fri 17 Jan 11:42:48 WET 2025 - Mark 1 soundcard detected by ovos-i2csound.
Fri 17 Jan 11:42:48 WET 2025 - Detected CARD_NUMBER for Mark 1 soundcard: 3
Fri 17 Jan 11:42:48 WET 2025 - Configuring ALSA default card
Fri 17 Jan 11:42:48 WET 2025 - Running as user, modifying ~/.asoundrc
Fri 17 Jan 11:42:48 WET 2025 - ALSA default card set to: 3

```

### Confirm available audio sinks:

Run `wpctl status` to check the available outputs as seen by pipewire.

The default sinks will be marked with \*

You can inspect a sink by its number with `wpctl inspect $SINK_ID`

```

id 36, type PipeWire:Interface:Node
(ovos) ovos@raspOVOS:~ $ wpctl status
    alsa.card = "2"
PipeWire 'pipewire-0' [1.2.4, ovos@raspOVOS, cookie:3349583741]
    alsa.card_name = "snd_rpi_proto"
Clients:
    alsa.class = "generic"
    33. WirePlumber [1.2.4, ovos@raspOVOS, pid:695]
    alsa.device = "0"
    34. WirePlumber [export] [1.2.4, ovos@raspOVOS, pid:695]
    alsa.driver_name = "snd_soc_rpi_proto"
    47. PipeWire ALSA [librespot] [1.2.4, ovos@raspOVOS, pid:702]
    alsa.id = "sndrpiproto"
    67. PipeWire ALSA [python3.11] [1.2.4, ovos@raspOVOS, pid:691]
    alsa.long_card_name = "snd_rpi_proto"
    75. PipeWire ALSA [python3.11] [1.2.4, ovos@raspOVOS, pid:699]
    alsa.name = "WM8731 HiFi wm8731-hifi-0"
    83. PipeWire ALSA [python3.11] [1.2.4, ovos@raspOVOS, pid:700]
    alsa.resolution_bits = "16"
    84. wpctl [1.2.4, ovos@raspOVOS, pid:1710]
    alsa.subclass = "generic-mix"

    alsa.subdevice = "0"
Audio
    alsa.subdevice_name = "subdevice #0"
Devices:
    42. Built-in Audio [alsa]
    43. Built-in Audio [alsa]
    44. Built-in Audio [alsa]

Sinks:
*   36. Built-in Audio Stereo [vol: 0.40]
    45. Built-in Audio Stereo [vol: 0.85]
    46. Built-in Audio Digital Stereo (HDMI) [vol: 0.40]

Sink endpoints:

Sources:
*   37. Built-in Audio Stereo [vol: 1.00]

Source endpoints:

Streams:
    48. PipeWire ALSA [librespot]
        63. output_FL > WM8731 HiFi wm8731-hifi-0:playback_FL [active]
        64. output_FR > WM8731 HiFi wm8731-hifi-0:playback_FR [active]
    68. PipeWire ALSA [python3.11]
        69. input_FL < WM8731 HiFi wm8731-hifi-0:capture_FL [active]
        70. monitor_FL
        71. input_FR < WM8731 HiFi wm8731-hifi-0:capture_FR [active]
        72. monitor_FR

```

(ovos) ovos@raspOVOS:~ \$ wpctl inspect 36

### Test Audio:

Record a short test file with `arecord -f test.wav`.

Play it back with `aplay test.wav`.

---

## STT tips and tricks

### Saving Transcriptions

You can enable saving of recordings to file, this should be your first step to diagnose problems, is the audio intelligible? is it being cropped? too noisy? low volume?

```
> set "save_utterances": true in > your listener config, > recordings will  
be saved to ~/.local/share/mycroft/listener/utterances
```

If the recorded audio looks good to you, maybe you need to use a different STT plugin, maybe the one you are using does not like your microphone, or just isn't very good for your language

### Wrong Transcriptions

If you consistently get specific words or utterances transcribed wrong, you can

remedy around this to some extent by using the [ovos-utterance-corrections-plugin](#)

> You can define replacements at word level

```
~/.local/share/mycroft/word_corrections.json
```

for example whisper STT often gets artist names wrong, this allows you to correct them

```
{
  "Jimmy Hendricks": "Jimi Hendrix",
  "Eric Klapptern": "Eric Clapton",
  "Eric Klappton": "Eric Clapton"
}
```

## Silence Removal

By default OVOS applies VAD (Voice Activity Detection) to crop silence from the audio sent to STT, this helps in performance and in accuracy (reduces hallucinations in plugins like FasterWhisper)

Depending on your microphone/VAD plugin, this might be removing too much audio

> set "remove\_silence": false in > your [listener config](#), this > will send the full audio recording to STT

## Listen Sound

does your listen sound contain speech? some users replace the "ding" sound with words such as "yes?"

In this case the listen sound will be sent to STT and might negatively affect the transcription

> set "instant\_listen": false in your [listener config](#), this > will drop the listen sound audio from the STT audio buffer. You will need to wait for the listen sound to finish before speaking your command in this case

---

## docs/590-voice\_apps.md

# Standalone Apps

Standalone applications can be made for OVOS, these applications are not skills, instead they are launched by the user

The main use case for these applications is in desktops, when you want to add a voice interface to a regular application

## OVOSAbstractApplication

`ovos-workshop` provides the `OVOSAbstractApplication` class, you can use all methods and decorators from regular Skills in applications built from this

```

from ovos_workshop.app import OVOSAbstractApplication

class MyApplication(OVOSAbstractApplication):
    def __init__(self, skill_id: str = "my_app",
                  bus: Optional[MessageBusClient] = None,
                  resources_dir: Optional[str] = None,
                  gui: Optional[GUIInterface] = None,
                  **kwargs):
        """
        Create an Application. An application is essentially a skill, but
        designed such that it may be run without an intent service.
        @param skill_id: Unique ID for this application
        @param bus: MessageBusClient to bind to application
        @param resources_dir: optional root resource directory (else defaults to
            application `root_dir`
        @param gui: GUIInterface to bind (if `None`, one is created)
        """
        super().__init__(skill_id, bus, resources_dir, gui=gui, **kwargs)

    def do_app_stuff(self):
        pass

    @intent_handler("app_action.intent")
    def do_intent_stuff_in_app(self, message):
        pass

if __name__ == "__main__":
    # launch your application
    from ovos_utils import wait_for_exit_signal

    app = MyApplication()

    # wait for user to exit
    wait_for_exit_signal()

```

> **NOTE:** from OVOS perspective Voice Apps are just like skills and need to have a unique skill\_id

# docs/600-fallbacks.md

## Fallback Skill

### Order of precedence

The Fallback **Skills** all have a priority and will be checked in order from low priority value to high priority value. If a Fallback **Skill** can handle the **Utterance** it will create a response and return `True`.

After this no other Fallback **Skills** are tried. This means the priority for Fallbacks that can handle a `_broad_` range of queries should be `_high_` `\(80-100\)` and Fallbacks that only responds to a very specific range of queries should be higher `\(20-80\)`. The more specific, the lower the priority value.

---

### Fallback Handlers

Import the `FallbackSkill` base class, create a derived class and register the handler with the fallback system

Implement the fallback handler `\(the method that will be called to potentially handle the Utterance\)`.

The method implements logic to determine if the **Utterance** can be handled and shall output speech if it can handle the query.

It shall return Boolean `True` if the **Utterance** was handled and Boolean `False` if not.

```

from ovos_workshop.skills.fallback import FallbackSkill

class MeaningFallback(FallbackSkill):
    """
        A Fallback skill to answer the question about the
        meaning of life, the universe and everything.
    """
    def initialize(self):
        """
            Registers the fallback handler
        """
        self.register_fallback(self.handle_fallback, 10)
        # Any other initialize code you like can be placed here

    def handle_fallback(self, message):
        """
            Answers question about the meaning of life, the universe
            and everything.
        """
        utterance = message.data.get("utterance")
        if 'what' in utterance
            and 'meaning' in utterance
            and ('life' in utterance
                or 'universe' in utterance
                or 'everything' in utterance):
            self.speak('42')
            return True
        else:
            return False

```

> **NOTE:** a `FallbackSkill` can register any number of fallback handlers

The above example can be found [here](#).

---

## Decorators

Alternatively, you can use decorators



```

from ovos_workshop.decorators.fallback_handler import fallback_handler

class MeaningFallback(FallbackSkill):
    """
        A Fallback skill to answer the question about the
        meaning of life, the universe and everything.
    """

    @fallback_handler(priority=10)
    def handle_fallback(self, message):
        """
            Answers question about the meaning of life, the universe
            and everything.
        """
        utterance = message.data.get("utterance")
        if 'what' in utterance
            and 'meaning' in utterance
            and ('life' in utterance
                or 'universe' in utterance
                or 'everything' in utterance):
            self.speak('42')
            return True
        else:
            return False

```

---

## Check utterances

Fallback skills should report if they are able to answer a question, without actually executing any action.

Besides providing performance improvements this allows other OVOS components to check how a utterance will be handled without side effects

```
def can_answer(self, utterances: List[str], lang: str) -> bool:
    """
    Check if the skill can answer the particular question. Override this
    method to validate whether a query can possibly be handled. By default,
    assumes a skill can answer if it has any registered handlers
    @param utterances: list of possible transcriptions to parse
    @param lang: BCP-47 language code associated with utterances
    @return: True if skill can handle the query
    """
    return len(self._fallback_handlers) > 0
```

## docs/610-common\_query.md

# Common Query Framework

The Common Query Framework handles the common use case of "general information" or question answering. Many Skills may implement handlers for "what is X" or "when did Y", the Common Query Framework allows all these Skills be queried and a single "best" answer to be selected. This is similar to the Common Play Framework that handles the common use of "playing" music or other media.

The Common Query Skill System is led by the Common Query Pipeline. The pipeline handles queries matching a question pattern such as "What is the height of the Eiffle Tower" and "When is lunch".

A matched question will be sent to all Skills based upon the `CommonQuerySkill` base class. The Skills will return whether they can answer the query along with an answer when applicable. The "best" match will be selected and spoken to the user.

## CommonQuerySkill

A Skill interfacing with the Common Query Framework inherits from the `CommonQuerySkill` and needs to define a method `CQS_match_query_phrase()` taking an utterance as argument.

The general structure is:

```

from ovos_workshop.skills.common_query_skill import CommonQuerySkill, CQSMatchLevel

class MyCommonQuerySkill(CommonQuerySkill):
    def CQS_match_query_phrase(self, utt):
        # Parsing implementation
        # [...]
        return (utt, CQSMatchLevel.LEVEL, answer_string)

```

The `CQS_match_query_phrase()` method will parse the utterance and determine if it can handle the query. if it can't answer it will return `None` and if it `_can_` answer it will return a data tuple with the format

```
((str)Input Query, CQSMatchLevel, (str)Answer Text)
```

The input query is returned to map the query to the answer.

`CQSMatchLevel` is an Enum with the possible values

`CQSMatchLevel.EXACT`: The Skill is very confident that it has the precise answer the user is looking for. There was a category match and a known entity is referenced.

`CQSMatchLevel.CATEGORY`: The Skill could determine that the type of question matches a category that the Skill is good at finding.

`CQSMatchLevel.GENERAL`: This Skill tries to answer all questions and found an answer.

To show visuals or take some other action in response to being selected, see the `'CQS_action()'` method below.

## An Example

Let's make a simple Skill that tells us the age of the various Monty Python members. A quick draft looks like this. (You can find the complete code [here](#))

```

from ovos_workshop.skills.common_query_skill import CommonQuerySkill, CQSMatchLevel

    {'person': python, 'age': age})

def CQS_match_query_phrase(self, utt):
# Dict mapping python members to their age and whether they're alive or dead
PYTHONS = {"Check the utterance if it is a question we can answer.
    'eric idle': (77, 'alive'),
    Arguments:
    'michael palin': (77, 'alive'),
    utt: The question
    'john cleese': (80, 'alive'),
    'graham chapman': (48, 'dead'),
    Returns: tuple (input utterance, match level, response sentence, extra)
    'terry gilliam': (79, 'alive'),
    """
    'terry jones': (77, 'dead')
    # Check if this is an age query
}
    age_query = self.voc_match(utt, 'age')

# Check if a monty python member is mentioned
def python_in_utt(utterance):
    python = full_python_in_utt(utt)
    """Find a monty python member in the utterance.
    Arguments:
    # If this is an age query and a monty python member is mentioned the
    utterance (str): Sentence to check for Monty Python members
    # skill can answer this
    Returns:
    if age_query and python:
    (str) name of Monty Python member or None
    # return high confidence
    """
    return (utt, CQSMatchLevel.CATEGORY, self.format_answer(python))
for key in PYTHONS:
    else:
    if key in utterance.lower():
    return None
    # Return the found python
    return key

# No python found
return None

class PythonAgeSkill(CommonQuerySkill):
    """A Skill for checking the age of the python crew."""

    def format_answer(self, python):
        """Create string with answer for the specified "python" person."""
        age, status = PYTHONS[python]
        if status == 'alive':
            return self.dialog_renderer.render('age_alive',
                                                {'person': python, 'age': age})
        else:
            return self.dialog_renderer.render('age_dead',

```

As seen above the `CQS_match_query_phrase()` checks if this is an age related utterance and if the utterance contains the name of a Monty Python member. If both criteria are met it returns a match with a `CQSMatchLevel.CATEGORY` confidence together with a rendered dialog containing the answer.

If both criteria are not fulfilled the method will return `None` indicating that it can't answer the query.

This will be able to provide answers to queries such as

> "how old is Graham Chapman" > > "what's Eric Idle's age"

To make this more exact we can add support for checking for the words "monty python", and if present return the highest confidence.

The method for parsing the example is quite simplistic but there are many different toolkits out there for doing the question parsing. [Adapt](#), [little questions](#), [padaos](#) and many more!

# Match Confidence

If we want to make sure this Skill is used when the user explicitly states it's the age of a Monty Python member, a slight modification to the Skill can be made:

We'll change the end of the `CQS_match_query_phrase()` method to

```
def CQS_match_query_phrase(self, utt):
    # (...)
    if 'monty python' in utt.lower():
        confidence = CQSMatchLevel.EXACT
    else:
        confidence = CQSMatchLevel.CATEGORY
    # return high confidence
    return (utt, confidence, self.format_answer(python))
```

So if the utterance contains the phrase "monty python" the confidence will be set to `CQSMatchLevel.EXACT` making the Skill very very likely to be chosen to answer the query.

## CQS\\_action()

In some cases the Skill should do additional operations when selected as the best match. It could be prepared for follow-up questions or show an image on the screen. The `CQS_action()` method allows for this, when a Skill is selected this method will be called.

Let's make our Python Age Skill gloat that it was selected by adding a `CQS_action()` method like this:

where `phrase` is the same phrase that were sent to `CQS_match_query_phrase()` and `data` is optional additional data from the query matching method.

```
def CQS_action(self, utt, data):
    self.log.info('I got selected! What you say about that Wolfram Alpha Skill!?!?!')
```

Now each time the Skill is called the above message will be added to the log! Not very useful you say? Hmm, yes... let's add something useful, like show the age on the Mark-1 display.

To accomplish this we need to get the age into the `CQS_action()` method in some way. we could store last age in as an internal variable but the more elegant way

is to send data as part of the match tuple. To do this we must extend the returned match tuple from `CQS_match_query_phrase()` with a data entry. So the return statement becomes

```
def CQS_match_query_phrase(self, utt):
    # (...)
    data = {'age': PYTHON[python], 'python': python}
    return (utt, confidence, self.format_answer(python), data)
```

The data structure declared here will be sent to `CQS_Action()` and we can update the method to

```
def CQS_action(self, utt, data):
    self.log.info('I got selected! What you say about that Wolfram Alpha Skill!?!?')
    age = data.get('age')
    if age:
        self.log.info(f'Showing the age {age}')
        self.enclosure.mouth_text(str(age))
```

## docs/620-universal\_skills.md

# UniversalSkill

The `UniversalSkill` class is designed to facilitate automatic translation of input and output messages between different languages.

This skill is particularly useful when native language support is not feasible, providing a convenient way to handle multilingual interactions.

> **NEW** - ovos-core version **0.0.8**

## Overview

This skill ensures that intent handlers receive utterances in the skill's internal language and are expected to produce responses in the same internal language.

The `speak` method, used for generating spoken responses, automatically translates utterances from the internal language to the original query language.

> **NOTE:** The `self.lang` attribute reflects the original query language, while received utterances are always in `self.internal_language`.

## Language Plugins

To run `UniversalSkills` you need to configure Translation plugins in `mycroft.conf`

```
// Translation plugins
"language": {
  // by default uses public servers
  // https://github.com/OpenVoiceOS/ovos-translate-server
  "detection_module": "ovos-lang-detector-plugin-server",
  "translation_module": "ovos-translate-plugin-server"
},
```

## Usage

## Initialization



```

# Example initialization
from ovos_workshop.skills.auto_translatable import UniversalSkill

class MyMultilingualSkill(UniversalSkill):
    """
    Skill that auto translates input/output from any language

    This skill is designed to automatically translate input and output messages
    between different languages. The intent handlers are ensured to receive
    utterances in the skill's internal language, and they are expected to produce
    utterances in the same internal language.

    The 'speak' method will always translate utterances from the internal language
    to the original query language ('self.lang').

    NOTE: 'self.lang' reflects the original query language, but received utterances
    are always in 'self.internal_language'.
    """
    def __init__(self, *args, **kwargs):
        """
        Initialize the UniversalSkill.

        Parameters for super():
        - internal_language (str): The language in which the skill internally operates.
        - translate_tags (bool): Whether to translate the private __tags__ value (adapt en
        - autodetect (bool): If True, the skill will detect the language of the utterance
            and ignore self.lang / Session.lang.
        - translate_keys (list): default ["utterance", "utterances"]
            Keys added here will have values translated in message.data

        """
        # skill hardcoded in portuguese
        super().__init__(internal_language="pt-pt", translate_tags=translate_tags,
            autodetect=autodetect, translate_keys=translate_keys, *args, **kwargs)

```

## Intents and Utterances

Use the `register_intent` and `register_intent_file` methods to register intents with universal intent handlers. The usual decorators also work

The `speak` method is used to generate spoken responses. It automatically

translates utterances if the output language is different from the skill's internal language or autodetection is enabled.

```
# Example speaking utterance, hardcoded to self.internal_language
self.speak("Hello, how are you?")
```

## Universal Intent Handler

> **NOTE** Users should NOT use the `create_universal_handler` method manually in skill intents; it is automatically utilized by `self.register_intent`.

The following example demonstrates its usage with `self.add_event`.

```
# Example universal handler creation
def my_event_handler(message):
    # Your event handling logic here
    pass

# Manual usage with self.add_event
my_handler = self.create_universal_handler(my_event_handler)
self.add_event("my_event", my_handler)
```

## EnglishCatFacts Skill Example

Let's create a simple tutorial skill that interacts with an API to fetch cat facts in English.

We'll use the `UniversalSkill` class to support translations for other languages.

```

from ovos_workshop.skills.auto_translatable import UniversalSkill

class EnglishCatFactsSkill(UniversalSkill):
    def __init__(self, *args, **kwargs):
        """
        This skill is hardcoded in english, indicated by internal_language
        """
        super().__init__(internal_language="en-us", *args, **kwargs)

    def fetch_cat_fact(self):
        # Your logic to fetch a cat fact from an API
        cat_fact = "Cats have five toes on their front paws but only four on their back paws"
        return cat_fact

    @intent_handler("cat_fact.intent")
    def handle_cat_fact_request(self, message):
        # Fetch a cat fact in self.internal_language
        cat_fact = self.fetch_cat_fact()
        # Speak the cat fact, it will be translated to self.lang if needed
        self.speak(cat_fact)

```

In this example, the `CatFactsSkill` class extends `UniversalSkill`, allowing it to seamlessly translate cat facts into the user's preferred language.

## SpanishDatabase Skill Example

A more advanced example, let's consider a skill that listens to bus messages.

Our skill listens for messages containing a "phrase" payload in `message.data` that can be in any language, and it saves this phrase *in spanish* to a database. Then it speaks a hardcoded spanish utterance, and it gets translated into the language of the bus message `Session`

```

from ovos_workshop.skills.auto_translatable import UniversalSkill

class SpanishDatabaseSkill(UniversalSkill):
    def __init__(self, *args, **kwargs):
        """
        This skill is hardcoded in spanish, indicated by internal_language
        """
        translate_keys=["phrase"] # translate "phrase" in message.data
        super().__init__(internal_language="es-es",
                        translate_keys=translate_keys,
                        *args, **kwargs)

    def initialize(self):
        # wrap the event into a auto translation layer
        handler = self.create_universal_handler(self.handle_entry)
        self.add_event("skill.database.add", handler)

    def handle_entry(self, message: Message):
        phrase = message.data["phrase"] # assured to be in self.internal_language

        # Your logic to save phrase to a database

        self.speak("agregado a la base de datos") # will be spoken in self.lang

```

## docs/630-OCP\_skills.md

### OCP Skills

OCP skills are built from the OVOSCommonPlaybackSkill class

These skills work as media providers, they return results for OCP to playback

The actual voice interaction is handled by OCP, skills only implement the returning of results

### Search Results

Search results are returned as a list of dicts, skills can also use iterators to

yield results 1 at a time as they become available

Mandatory fields are `python uri: str # URL/URI of media, OCP will handle formatting and file handling title: str media_type: MediaType playback: PlaybackType match_confidence: int # 0-100`

Other optional metadata includes artists, album, length and images for the GUI

`python artist: str album: str image: str # uri/file path bg_image: str # uri/file path skill_icon: str # uri/file path length: int # seconds, -1 for live streams`

[!imagem](#)

## OCP Skill

General Steps to create a skill

subclass your skill from `OVOSCommonPlaybackSkill`

In the `__init__` method indicate [the media types you want to handle](#)

`self.voc_match(phrase, "skill_name")` to handle specific requests for your skill

`self.remove_voc(phrase, "skill_name")` to remove matched phrases from the search request

Implement the `ocp_search` decorator, as many as you want (they run in parallel)

The decorated method can return a list or be an iterator of `result_dict` (track or playlist)

The search function can be entirely inline or call another Python library, like [pandorinha](#) or [plexapi](#)

`self.extend_timeout()` to delay OCP from selecting a result, requesting more time to perform the search

Implement a confidence score formula

[Values are between 0 and 100](#)

High confidence scores cancel other OCP skill searches

`ocp_featured_media`, return a playlist for the OCP menu if selected from GUI  
(optional)

Create a `requirements.txt` file with third-party package requirements

```

        if media_type == MediaType.RADIO:
from os.path import join, dirname
            base_score += 20

        else:
import radiosoma
            base_score -= 30

from ovos_utils import classproperty
        if self.voc_match(phrase, "radio"):
from ovos_utils.ocp import MediaType, PlaybackType
            base_score += 10
from ovos_utils.parse import fuzzy_match
        phrase = self.remove_voc(phrase, "radio")
from ovos_workshop.decorators.ocp import ocp_search, ocp_featured_media
from ovos_workshop.skills.common_play import OVOSCommonPlaybackSkill
        if self.voc_match(phrase, "somafm"):
            base_score += 30 # explicit request
        phrase = self.remove_voc(phrase, "somafm")
class SomaFMSkill(OVOSCommonPlaybackSkill):
    def __init__(self, *args, **kwargs):
        for ch in radiosoma.get_stations():
            score = round(base_score + fuzzy_match(ch.title.lower(),
            # media type this skill can handle
            phrase.lower()) * 100)
            self.supported_media = [MediaType.MUSIC, MediaType.RADIO]
            if score < 50:
                self.skill_icon = join(dirname(__file__), "ui", "somafm.png")
            super().__init__(*args, **kwargs)
            yield {
                "match_confidence": min(100, score),
                "media_type": MediaType.RADIO,
            }
        @ocp_featured_media()
        def featured_media(self):
            # uri: ch.direct_stream
            # playlist when selected from OCP skills menu
            return [{"playback": PlaybackType.AUDIO,
                "image": ch.image,
                "match_confidence": 90,
                "bg_image": ch.image,
                "media_type": MediaType.RADIO,
                "skill_icon": self.skill_icon,
                "uri": ch.direct_stream,
                "title": ch.title,
                "playback": PlaybackType.AUDIO,
                "artist": "SomaFM",
                "image": ch.image,
                "length": 0,
                "bg_image": ch.image,
            }, {"skill_icon": self.skill_icon,
                "title": ch.title,
                "author": "SomaFM",
                "length": 0
            }
            ]
        } for ch in radiosoma.get_stations()]

    @ocp_search()
    def search_somafm(self, phrase, media_type):
        # check if user asked for a known radio station
        base_score = 0

```

## OCP Keywords

OCP skills often need to match hundreds or thousands of strings against the query string, `self.voc_match` can quickly become impractical to use in this scenario

To help with this the OCP skill class provides efficient keyword matching



```

def register_ocp_keyword(self, label: str, samples: List, langs: List[str] = None):
    """ register strings as native OCP keywords (eg, movie_name, artist_name ...)
    ocp keywords can be efficiently matched with self.ocp_match helper method
    that uses AhoCorasick algorithm
    """

def load_ocp_keyword_from_csv(self, csv_path: str, lang: str):
    """ load entities from a .csv file for usage with self.ocp_voc_match
    see the ocp_entities.csv datasets for example files built from wikidata SPARQL queries

    examples contents of csv file

        label,entity
        film_genre,swashbuckler film
        film_genre,neo-noir
        film_genre,actual play film
        film_genre,alternate history film
        film_genre,spy film
        ...
    """

```

## OCP Voc match

uses [AhoCorasick algorithm](#) to match OCP keywords

this efficiently matches many keywords against an utterance

OCP keywords are registered via `self.register_ocp_keyword`

wordlists can also be loaded from a .csv file, see [the OCP dataset](#) for a list of keywords gathered from wikidata with SPARQL queries

## OCP Database Skill

[illegible]

```
s = HorrorBabbleSkill(bus=FakeBus(), skill_id="demo.fake")

entities = s.ocp_voc_match("read The Call of Cthulhu by Lovecraft")
# {'book_author': 'Lovecraft', 'book_name': 'The Call of Cthulhu'}
print(entities)

entities = s.ocp_voc_match("play HorrorBabble")
# {'audiobook_streaming_provider': 'HorrorBabble'}
print(entities)
```

## Playlist Results

Results can also be playlists, not only single tracks, for instance full albums or a full season for a series

When a playlist is selected from Search Results, it will replace the Now Playing list

Playlist results look exactly the same as regular results, but instead of a `uri` they provide a `playlist`

```
playlist: list # list of dicts, each dict is a regular search result
title: str
media_type: MediaType
playback: PlaybackType
match_confidence: int # 0-100
```

> NOTE: nested playlists are a work in progress and not guaranteed to be functional, ie, the "playlist" dict key should not include other playlists

## Playlist Skill

```

class MyJamsSkill(OVOSCommonPlaybackSkill):

    def __init__(self, *args, **kwargs):
        self.supported_media = [MediaType.MUSIC]
        self.skill_icon = join(dirname(__file__), "ui", "myjams.png")
        super().__init__(*args, **kwargs)

    @ocp_search()
    def search_my_jams(self, phrase, media_type):
        if self.voc_match(...):
            results = [...] # regular result dicts, as in examples above
            score = 70 # TODO

            yield {
                "match_confidence": min(100, score),
                "media_type": MediaType.MUSIC,
                "playlist": results, # replaces "uri"
                "playback": PlaybackType.AUDIO,
                "image": self.image,
                "bg_image": self.image,
                "skill_icon": self.skill_icon,
                "title": "MyJams",
                "length": sum([r["length"] for r in results]) # total playlist duration
            }

```

## docs/631-OCP\_pipeline.md

### OCP Pipeline

The **OCP (OVOS Common Playback)** Pipeline Plugin integrates seamlessly into the OVOS intent processing framework, enabling intelligent handling of media-related voice commands. By leveraging classifiers, skill-registered catalogs, and playback-specific filters, OCP facilitates accurate recognition and execution of user requests such as "play music," "pause video," or "next song."

---

# Pipeline Components

The OCP Pipeline Plugin registers three components within the OVOS intent pipeline, each corresponding to different confidence levels in interpreting media-related intents:

| Pipeline ID | Description                            | Recommended Use   |
|-------------|--|---|
| ocp_high    | High-confidence media intent matches   | Primary media commands  |
| ocp_medium  | Medium-confidence media intent matches | Ambiguous media queries   |
| ocp_low     | Low-confidence media intent matches    | Broad keyword matches. Only use if the device is exclusively for media playback |

These components should be ordered in the pipeline to prioritize higher-confidence matches

---

# Intent Classification

OCP employs a combination of techniques to classify and handle media-related intents:

**Keyword-Based Matching:** Identifies explicit media-related terms in user utterances.

**Skill-Registered Keywords:** Utilizes media keywords registered by OCP-aware skills (e.g., artist names, show titles) to enhance intent recognition.

**Media Type Classification:** Assigns a media type (e.g., music, podcast, movie) to the query based on keywords or an optional experimental classifier.

> The ocp\_low component relies on skill-registered keywords and may trigger on queries that include known media > terms, even if the user's intent is not to initiate playback.

---

# Media Type Handling

OCP supports various media types, including:

music

podcast

movie

radio

audiobook

news

Media type classification is primarily based on keywords within the user's query. For example, a query containing "play the latest news" would be classified under the `news` media type. An experimental classifier can also be enabled to predict media types based on the full query context.

---

## Result Filtering

After gathering potential media results from OCP-enabled skills, the plugin applies several filters to ensure relevance and playability:

**Confidence Threshold:** Results with a `match_confidence` below the configured `min_score` are discarded.

**Media Type Consistency:** If a media type has been classified, results of differing types are removed.

**Plugin Availability:** Results requiring unavailable playback plugins (e.g., `spotify://` URIs without the Spotify

plugin) are excluded.

**Playback Mode Preference:** Respects user or system preferences for audio-only or video-only playback, filtering out incompatible results.

---

# Playback Management

OCP delegates the actual media playback to the appropriate plugin managed by `ovos-audio`. Skills act solely as media catalogs, providing search results without handling playback directly. This separation ensures a consistent and centralized playback experience across different media types and sources.

the OCP Pipeline keeps track of media player status across `Sessions`, this is taken into account during the intent matching process

eg. if no media player is active, then "next song" will not trigger

---

# Configuration Options

OCP behavior can be customized via the `mycroft.conf` file under the `intents` section:

```
{
  "intents": {
    "OCP": {
      "experimental_media_classifier": false,
      "experimental_binary_classifier": false,
      "legacy": false,
      "classifier_threshold": 0.4,
      "min_score": 40,
      "filter_media": true,
      "filter_SEI": true,
      "playback_mode": 0,
      "search_fallback": true
    }
  }
}
```

| Key                           | Type | Default | Description           |
|-------------------------------|------|---------|-----------------------|
| experimental_media_classifier | bool | false   | Enable ML-based media |

```

type classification (English only).      ||
experimental_binary_classifier | bool | false | Enable ML-based media
detection for ocp_medium. (English only). || legacy
| bool | false | Use legacy audio service API instead of OCP (not
recommended).      || classifier_threshold      | float | 0.4 |
Minimum confidence for trusting classifier results (0.01.0).      ||
min_score      | int | 40 | Minimum match confidence to
retain a skill result (0100).      || filter_media      |
bool | true | Enable media type-based result filtering.
|| filter_SEI      | bool | true | Filter out results
requiring unavailable plugins (Stream Extractors). || playback_mode
| int | 0 | Playback preference: 0 = auto, 10 = audio-only, 20 =
video-only. || search_fallback      | bool | true | Perform a
generic media search if no type-specific results are found. |
---

```

## docs/700-homescreen.md

# OpenVoiceOS Home Screen

The home screen is the central place for all your tasks. It is what your device displays when it is idle

## Configuration

Select a homescreen in `mycroft.conf` or via `ovos-shell`

```

"gui": {
    "idle_display_skill": "skill-ovos-homescreen.openvoiceos"
}

```

## Resting Faces

The resting face API provides skill authors the ability to extend their skills to supply their own customized IDLE screens that will be displayed when there is no activity on the screen.



```

import requests
from ovos_workshop.skills import OVOSSkill
from ovos_workshop.decorators import intent_handler, resting_screen_handler

class CatSkill(OVOSSkill):
    def update_cat(self):
        r = requests.get('https://api.thecatapi.com/v1/images/search')
        return r.json()[0]['url']

    @resting_screen_handler("Cat Image")
    def idle(self, message):
        img = self.update_cat()
        self.gui.show_image(img)

    @intent_handler('show_cat.intent')
    def cat_handler(self, message):
        img = self.update_cat()
        self.gui.show_image(img)
        self.speak_dialog('mjau')

```

A more advanced example, refreshing a webpage on a timer

```

        self.display_homescreen
from ovos_workshop.skills import OVOSSkill
0
from ovos_workshop.decorators import intent_handler, resting_screen_handler
        self.settings.get("refresh_frequency", 600),
class WebpageHomescreen(OVOSSkill):
    name="refresh-webpage-homescreen",
    )

    def initialize(self):
    def shutdown(self):
        """Perform final setup of Skill."""
        """Actions to perform when Skill is shutting down."""
        # Disable manual refresh until this Homepage is made active.
        self.is_active = False
        self.is_active = False
        self.cancel_all_repeating_events()
        self.disable_intent("refresh-homepage.intent")

        self.settings_change_callback = self.refresh_homescreen

    def get_intro_message(self):
        """Provide instructions on first install."""
        self.speak_dialog("setting-url")
        self.speak_dialog("selecting-homescreen")

    @resting_screen_handler("Webpage Homescreen")
    def handle_request_to_use_homescreen(self, message: Message):
        """Handler for requests from GUI to use this Homescreen."""
        self.is_active = True
        self.display_homescreen()
        self.refresh_homescreen(message)
        self.enable_intent("refresh-homepage.intent")

    def display_homescreen(self):
        """Display the selected webpage as the Homescreen."""
        default_url = "https://openvoiceos.github.io/status"
        url = self.settings.get("homepage_url", default_url)
        self.gui.show_url(url)

    @intent_handler("refresh-homepage.intent")
    def refresh_homescreen(self, message: Message):
        """Update refresh rate of homescreen and refresh screen.

        Defaults to 600 seconds / 10 minutes.
        """
        self.cancel_scheduled_event("refresh-webpage-homescreen")
        if self.is_active:
            self.schedule_repeating_event(

```

# docs/701-gui\_protocol.md

## Protocol

The [gui service](#) in ovos-core will expose a websocket to the GUI clients following the protocol outlined in this page

The transport protocol works between gui service and the gui clients, OpenVoiceOS does not directly use the protocol but instead communicates with `ovos-gui` via the standard messagebus

The QT library which implements the protocol lives in the [mycroft-gui-qt5](#) repository.

[!imagem](#)

## Specification

This protocol defines how ovos-gui communicates with connected clients

CONNECTION - mycroft.gui.connected

NAMESPACES

Active Skills - mycroft.system.active\_skills

PAGES - mycroft.gui.list.xxx

Insert new page at position

Move pages within the list

Remove pages from the list

EVENTS - mycroft.events.triggered

SPECIAL EVENT: page\_gained\_focus

SKILL DATA - mycroft.session.xxx

Sets a new key/value in the sessionData dictionary

Deletes a key/value pair from the sessionData dictionary

Lists

Inserts new items at position

Updates item values starting at the given position, as many items as there are in the array

Move items within the list

Remove items from the list

## CONNECTION - mycroft.gui.connected

on connection gui clients announce themselves

This is an extension by OVOS to the original mycroft protocol

```
{
  "type": "mycroft.gui.connected",
  "gui_id": "unique_identifier_provided_by_client"
}
```

## NAMESPACES

ovos-gui maintains a list of namespaces with GUI data, namespaces usually correspond to a skill\_id

Every message in the gui protocol specifies a namespace it belongs to

gui clients usually display all namespaces, but can be requested to display a

single one,

eg, have a dedicated window to show a skill as a [traditional desktop app](#)

## Active Skills - mycroft.system.active\_skills

a reserved namespace is "mycroft.system.active\_skills", the data contained in this namespace defines the namespace display priority

Recent skills are ordered from the last used to the oldest, so the first item of the list will always be the the one showing any GUI page, if available.

see the section about lists if you need to modify active skills

## PAGES - mycroft.gui.list.xxx

Each active skill is associated with a list of uris to the QML files of all gui items that are supposed to be visible.

Non QT GUIs get sent other file extensions such as .jsx or .html using the same message format

## Insert new page at position

```
{
  "type": "mycroft.gui.list.insert",
  "namespace": "mycroft.weather"
  "position": 2
  "values": [{"url": "file://..../currentWeather.qml"}, ...] //values must always be in
}
```

## Move pages within the list

```
{
  "type": "mycroft.gui.list.move",
  "namespace": "mycroft.weather"
  "from": 2
  "to": 5
  "items_number": 2 //optional in case we want to move a big chunk of list at once
}
```

## Remove pages from the list

```
{
  "type": "mycroft.gui.list.remove",
  "namespace": "mycroft.weather"
  "position": 2
  "items_number": 5 //optional in case we want to get rid a big chunk of list at once
}
```

## EVENTS - mycroft.events.triggered

Events can either be emitted by a gui client (eg, some element clicked) or by the skill (eg, in response to a voice command)

```
{
  "type": "mycroft.events.triggered"
  "namespace": "my_skill_id"
  "event_name": "my.gui.event",
  "parameters": {"item": 3}
}
```

## SPECIAL EVENT: page\_gained\_focus

This event is used when the ovos-gui wants a page of a particular skill to gain user attention focus and become the current active view and "focus of attention" of the user.

when a GUI client receives it, it should render the requested GUI page

GUI clients can also emit this event, if a new page was rendered (eg, in response to a user swipping left)

NOTE: for responsiveness it is recommended this message is only emitted after the rendering has actually been done, skills may be waiting for this event to initiate some actions

```
{
  "type": "mycroft.events.triggered",
  "namespace": "mycroft.weather",
  "event_name": "page_gained_focus",
  "data": {"number": 0}
}
```

The parameter "number" is the position (starting from zero) of the page

## SKILL DATA - mycroft.session.xxx

At the center of data sharing there is a key/value dictionary that is kept synchronized between ovos-gui and the GUI client.

Values can either be simple strings, numbers and booleans or be more complicated data types

this event can be sent from gui clients (eg, in response to a dropdown selection) or from skills (eg, change weather data)

NOTE: Once a new gui client connects to ovos-gui, all existing session data is sent to the client, after that the client gets live updates via these events

## Sets a new key/value in the sessionData dictionary

Either sets a new key/value pair or replace an existing old value.

```
{
  "type": "mycroft.session.set",
  "namespace": "weather.mycroft"
  "data": {
    "temperature": "28",
    "icon": "cloudy",
    "forecast": [{...},...] //if it's a list see below for more message types
  }
}
```

## Deletes a key/value pair from the sessionData

## dictionary

```
{
  "type": "mycroft.session.delete",
  "namespace": "weather.mycroft"
  "property": "temperature"
}
```

## Lists

### Inserts new items at position

```
{
  "type": "mycroft.session.list.insert",
  "namespace": "weather.mycroft"
  "property": "forecast" //the key of the main data map this list in contained into
  "position": 2
  "values": [{"date": "tomorrow", "temperature" : 13, ...}, ...] //values must always be
```

### Updates item values starting at the given position, as many items as there are in the array

```
{
  "type": "mycroft.session.list.update",
  "namespace": "weather.mycroft"
  "property": "forecast"
  "position": 2
  "values": [{"date": "tomorrow", "temperature" : 13, ...}, ...] //values must always be
```

### Move items within the list



```
{
  "type": "mycroft.session.list.move",
  "namespace": "weather.mycroft"
  "property": "forecast"
  "from": 2
  "to": 5
  "items_number": 2 //optional in case we want to move a big chunk of list at once
}
```

## Remove items from the list

```
{
  "type": "mycroft.session.list.remove",
  "namespace": "weather.mycroft"
  "property": "forecast"
  "position": 2
  "items_number": 5 //optional in case we want to get rid a big chunk of list at once
}
```

# docs/702-ovos-shell.md

## OVOS Shell

[OVOS-shell](#) is the OpenVoiceOS client implementation of the [mycroft-gui-qt5](#) library used in our embedded device images

## Design Principles

The OpenVoiceOS Shell was designed with some simple principles in mind.

> **The visual interface is always secondary to the voice interface.**

Our goal is to make all interactions Voice First, meaning that the user could accomplish their tasks with just voice interaction.

> **Touchscreen menus should be kept to a minimum, this reinforces using the primary mode of interaction, voice.**

However, many important controls need to be implemented as multimodal such as the ability to return to the home screen, change the volume, change the brightness of the screen, control media playback, and other system settings.

OpenVoiceOS images ship with [ovos-homescreen](#) and `ovos-shell`, built on top of QT5, these components ensure the viability of the GUI in embedded devices running `ovos-shell` via EGLFS, without requiring a traditional desktop environment

## Companion Plugins

To unlock full functionality you also need to configure [ovos-gui-plugin-shell-companion](#) in `mycroft.conf`

This plugin integrates with `ovos-gui` to provide:

- color scheme manager
- notifications widgets
- configuration provider (settings UI)
- brightness control (night mode etc)

```
{
  "gui": {
    // Extensions provide additional GUI platform support for specific devices
    "extension": "ovos-gui-plugin-shell-companion",

    // homescreen skill to use
    "idle_display_skill": "skill-ovos-homescreen.openvoiceos"
  }
}
```

OVOS-shell is tightly coupled to PHAL, the following companion plugins should also be installed

[ovos-PHAL-plugin-network-manager](#)

[ovos-PHAL-plugin-gui-network-client](#)

[ovos-PHAL-plugin-wifi-setup](#)

[ovos-PHAL-plugin-alsa](#)

## Configuration

The Shell can be configured in a few ways.

### GUI

Display settings

!

Color Theme editor

!

### Shell Options

`~/.config/OpenvoiceOS/OvosShell.conf` can be edited to change shell options that may also be changed via UI. An example config would look like: `[General]`  
`fakeBrightness=1 menuLabels=true`

### Themes

Shell themes can be included in `/usr/share/OVOS/ColorSchemes/` or `~/.local/share/OVOS/ColorSchemes/` in json format. Note that colors should include an alpha value (usually `FF`).

```
{
  "name": "Neon Green",
  "primaryColor": "#FF072103",
  "secondaryColor": "#FF2C7909",
  "textColor": "#FFF1F1F1"
}
```

**docs/710-qt5-gui.md**

**Mycroft-GUI QT5**

> **NOTE** - Currently only a [QT5 gui-client](#) is available, help wanted to [migrate to QT6](#)!

## Introduction to QML

The reference GUI client implementation is based on the QML user interface markup language that gives you complete freedom to create in-depth innovative interactions without boundaries or provide you with simple templates within the GUI framework that allow minimalistic display of text and images based on your skill development specifics and preferences.

QML user interface markup language is a declarative language built on top of Qt's existing strengths designed to describe the user interface of a program: both what it looks like, and how it behaves. QML provides modules that consist of sophisticated set of graphical and behavioral building elements.

## Before Getting Started

A collection of resources to familiarize you with QML and Kirigami Framework.

[Introduction to QML](#)

[Introduction to Kirigami](#)

## Importing Modules

A QML module provides versioned types and JavaScript resources in a type namespace which may be used by clients who import the module. Modules make use of the QML versioning system which allows modules to be independently updated. More in-depth information about QML modules can be found here [Qt QML Modules Documentation](#)

In the code snippet example below we will look at importing some of the common modules that provide the components required to get started with our Visual User Interface.

```
import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft
import org.kde.lottie 1.0
```

### **QTQuick Module:**

Qt Quick module is the standard library for writing QML applications, the module provides a visual canvas and includes types for creating and animating visual components, receiving user input, creating data models and views and delayed object instantiation. In-depth information about QtQuick can be found at [Qt Quick Documentation](#)

### **QTQuick.Controls Module:**

The QtQuick Controls module provides a set of controls that can be used to build complete interfaces in Qt Quick. Some of the controls provided are button controls, container controls, delegate controls, indicator controls, input controls, navigation controls and more, for a complete list of controls and components provided by QtQuick Controls you can refer to [QtQuick Controls 2 Guidelines](#)

### **QtQuick.Layouts Module:**

QtQuick Layouts are a set of QML types used to arrange items in a user interface. Some of the layouts provided by QtQuick Layouts are Column Layout, Grid Layout, Row Layout and more, for a complete list of layouts you can refer to [QtQuick Layouts Documentation](#)

### **Kirigami Module:**

[Kirigami](#) is a set of QtQuick components for mobile and convergent applications. [Kirigami](#) is a set of high level components to make the creation of applications that look and feel great on mobile as well as desktop devices and follow the [Kirigami Human Interface Guidelines](#)

### **Mycroft Module:**

Mycroft GUI frameworks provides a set of high level components and events system for aiding in the development of Mycroft visual skills. One of the controls provided by Mycroft GUI frameworks are Mycroft-GUI Framework Base Delegates [Mycroft-GUI Framework Base Delegates Documentation](#)

### **QML Lottie Module:**

This provides a QML `Item` to render Adobe After Effects™ animations exported as JSON with Bodymovin using the Lottie Web library. For list of all properties supported refer [Lottie QML](#)

## Mycroft-GUI Framework Base Delegates

When you design your skill with QML, Mycroft-GUI frameworks provides you with some base delegates you should use when designing your GUI skill. The base delegates provide you with a basic presentation layer for your skill with some property assignments that can help you setup background images, background dim, timeout and grace time properties to give you the control you need for rendering an experience. In your GUI Skill you can use:

**Mycroft.Delegate:** A basic and simple page based on Kirigami.Page

Simple display Image and Text Example using Mycroft.Delegate

```
import Mycroft 1.0 as Mycroft

Mycroft.Delegate {
    skillBackgroundSource: sessionData.exampleImage
    ColumnLayout {
        anchors.fill: parent
        Image {
            id: imageId
            Layout.fillWidth: true
            Layout.preferredHeight: Kirigami.Units.gridUnit * 2
            source: "https://source.unsplash.com/1920x1080/?+autumn"
        }
        Label {
            id: labelId
            Layout.fillWidth: true
            Layout.preferredHeight: Kirigami.Units.gridUnit * 4
            text: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod"
        }
    }
}
```

**Mycroft.ScrollableDelegate:** A delegate that displays skill visuals in a scroll enabled Kirigami Page.

Example of using Mycroft.ScrollableDelegate

```

import QtQuick 2.4          source: modelData.image
import QtQuick.Controls 2.2  Layout.fillWidth: true
import QtQuick.Layouts 1.4   Layout.preferredHeight: Kirigami.Units.gridUnit * 3
import org.kde.kirigami 2.4 as Kirigami fillMode: Image.PreserveAspectCrop
import Mycroft 1.0 as Mycroft
                                Item {
Mycroft.ScrollableDelegate {
    Layout.fillWidth: true
    id: root                    Layout.preferredHeight: Kirigami.Units.gridUnit * 1
    skillBackgroundSource: sessionData.background
    property var sampleModel: sessionData.sampleBlob
}

Kirigami.CardsListView {
}
    id: exampleListView
    Layout.fillWidth: true
    Layout.fillHeight: true
    model: sampleModel.lorem
    delegate: Kirigami.AbstractCard {
        id: rootCard
        implicitHeight: delegateItem.implicitHeight + Kirigami.Units.largeSpacing
        contentItem: Item {
            implicitWidth: parent.implicitWidth
            implicitHeight: parent.implicitHeight
            ColumnLayout {
                id: delegateItem
                anchors.left: parent.left
                anchors.right: parent.right
                anchors.top: parent.top
                spacing: Kirigami.Units.largeSpacing
                Kirigami.Heading {
                    id: restaurantNameLabel
                    Layout.fillWidth: true
                    text: modelData.text
                    level: 2
                    wrapMode: Text.WordWrap
                }
                Kirigami.Separator {
                    Layout.fillWidth: true
                }
                Image {
                    id: placeImage

```



# QML Design Guidelines

Before we dive deeper into the Design Guidelines, lets look at some concepts that a GUI developer should learn about:

## Units & Theming

### Units:

Mycroft.Units.GridUnit is the fundamental unit of space that should be used for all sizing inside the QML UI, expressed in pixels. Each GridUnit is predefined as

16 pixels // Usage in QML Components example width:

```
Mycroft.Units.gridUnit * 2 // 32px Wide height: Mycroft.Units.gridUnit  
// 16px Tall
```

### Theming:

OVOS Shell uses a custom Kirigami Platform Theme plugin to provide global theming to all our skills and user interfaces, which also allows our GUI's to be fully compatible with the system themes on platforms that are not running the OVOS Shell.

Kirigami Theme and Color Scheme guide is extensive and can be found [here](#)

OVOS GUI's developed to follow the color scheme depend on only a subset of

available colors, mainly:

1. Kirigami.Theme.backgroundColor = Primary Color (Background Color: This will always be a dark palette or light palette depending on the dark or light chosen color scheme)
1. Kirigami.Theme.highlightColor = Secondary Color (Accent Color: This will always be a standout palette that defines the themes dominating color and can be used for buttons, cards, borders, highlighted text etc.)
1. Kirigami.Theme.textColor = Text Color (This will always be an opposite palette to the selected primary color)

## QML Delegate Design Best Practise

\_\_Let's look at this image and qml example below, this is a representation of the Mycroft Delegate:\_\_ !

1. When designing your first QML file, it is important to note the red triangles in the above image, these triangles represent the margin from the screen edge the GUI needs to be designed within, these margins ensure your GUI content does not overlap with features like edge lighting and menus in the platforms that support it like OVOS-Shell
1. The content items and components all utilize the selected color scheme, where black is the primary background color, red is our accent color and white is our contrasting text color

\_\_Let's look at this in QML:\_\_ ' import ... import Mycroft 1.0 as Mycroft

```
Mycroft.Delegate { skillBackgroundSource: sessionData.exampleImage leftPadding: 0  
rightPadding: 0 topPadding: 0 bottomPadding: 0
```

```
Rectangle { anchors.fill: parent // Setting margins that need to be left for the  
screen edges anchors.margins: Mycroft.Units.gridUnit * 2
```

```
//Setting a background dim using our primary theme / background color on top of  
our skillBackgroundSource image for better readability and contrast color:  
Qt.rgb(Kirigami.Theme.backgroundColor.r, Kirigami.Theme.backgroundColor.g,  
Kirigami.Theme.backgroundColor.b, 0.3)
```

```
Kirigami.Heading { level: 2 text: "An Example Pie Chart" anchors.top: parent.top  
anchors.left: parent.left anchors.right: parent.right height:  
Mycroft.Units.gridUnit * 3 // Setting the text color to always follow the color
```

```
scheme for this item displayed on the screen color: Kirigami.Theme.textColor }
PieChart { anchors.centerIn: parent pieColorMinor: Kirigami.Theme.backgroundColor
// As in the image above the minor area of the pie chart uses our primary color
pieColorMid: Kirigami.Theme.highlightColor // As in the image above the middle
area is assigned the highlight or our accent color pieColorMajor:
Kirigami.Theme.textColor // As in the image above the major area is assigned the
text color } } } '

```

## QML Delegate Multi Platform and Screen Guidelines

OVOS Skill GUIs are designed to be multi-platform and screen friendly, to support this we always try to support both Horizontal and Vertical display's. Let's look at an example and a general approach to writing multi resolution friendly UI's

\_\_Let's look at these images below that represent a Delegate as seen in a Horizontal screen:\_\_ !

\_\_Let's look at these images below that represent a Delegate as seen in a Vertical screen:\_\_ !

1. When designing for different screens it is preferred to utilize Grids, GridLayouts and GridViews this allows easier content placement as one can control the number of columns and rows displayed on the screen
2. It is also recommended to use Flickables when you believe your content is going to not fit on the screen, this allows for content to always be scrollable. To make it easier to design scrollable content, Mycroft GUI provides you with a ready to use Mycroft.ScrollableDelegate.
3. It is also preferred to use the width vs height comparison on the root delegate item to know when the screen should be using a vertical layout vs horizontal layout

\_\_Let's look at this in QML:\_\_ ' import ... import Mycroft 1.0 as Mycroft

```
Mycroft.Delegate { id: root skillBackgroundSource: sessionData.exampleImage
leftPadding: 0 rightPadding: 0 topPadding: 0 bottomPadding: 0 property bool
horizontalMode: width >= height ? 1 : 0 // Using a ternary operator to detect if
width of the delegate is greater than the height, which provides if the delegate
is in horizontalMode

```

```

Rectangle { anchors.fill: parent // Setting margins that need to be left for the
screen edges anchors.margins: Mycroft.Units.gridUnit * 2

//Setting a background dim using our primary theme / background color on top of
our skillBackgroundSource image for better readability and contrast color:
Qt.rgb(Kirigami.Theme.backgroundColor.r, Kirigami.Theme.backgroundColor.g,
Kirigami.Theme.backgroundColor.b, 0.3)

Kirigami.Heading { level: 2 text: "An Example Pie Chart" // Setting the text
color to always follow the color scheme color: Kirigami.Theme.textColor }

GridLayout { id: examplesGridView // Checking if we are in horizontal mode, we
should display two columns to display the items in the image above, or if we are
in vertical mode, we should display a single column only columns:
root.horizontalMode ? 2 : 1

Repeater { model: examplesModel delegates: ExamplesDelegate { ... } } } } '

```

## Advanced skill displays using QML

### Display Lottie Animations:

You can use the `LottieAnimation` item just like any other `QtQuick` element, such as an `Image` and place it in your scene any way you please.

### QML Example

```

import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft
import org.kde.lottie 1.0

Mycroft.Delegate {
    LottieAnimation {
        id: fancyAnimation
        anchors.fill: parent
        source: Qt.resolvedUrl("animations/fancy_animation.json")
        loops: Animation.Infinite
        fillMode: Image.PreserveAspectFit
        running: true
    }
}

```

## Display Sliding Images

Contains an image that will slowly scroll in order to be shown completely

### QML Example

```

import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft

Mycroft.Delegate {
    background: Mycroft.SlidingImage {
        source: "foo.jpg"
        running: bool //If true the sliding animation is active
        speed: 1 //Animation speed in Kirigami.Units.gridUnit / second
    }
}

```

## Display Paginated Text

Takes a long text and breaks it down into pages that can be horizontally swiped

### QML Example

```

import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft

Mycroft.Delegate {
    Mycroft.PaginatedText {
        text: string          //The text that should be displayed
        currentIndex: 0       //The currently visible page number (starting from 0)
    }
}

```

## Display A Vertical ListView With Information Cards

Kirigami CardsListView is a ListView which can have AbstractCard as its delegate: it will automatically assign the proper spacing and margins around the cards adhering to the design guidelines.

## Python Skill Example

```

...
def handle_food_places(self, message):
    ...
    self.gui["foodPlacesBlob"] = results.json
    self.gui.show_page("foodplaces.qml")
    ...

```

## QML Example

```

import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4
import Mycroft 1.0 as Mycroft

Image {
    id: placeImage
    source: modelData.image
    Layout.fillHeight: true
    as Kirigami
    Layout.preferredWidth: placeImage.implicitHeight + Kirigami.Units.largeSpacing
    fillMode: Image.PreserveAspectFit
}

Mycroft.Delegate {
    id: root
    Kirigami.Separator {
        id: separator
        Layout.fillHeight: true
        property var foodPlacesModel: sessionData.foodPlacesBlob
    }
    Kirigami.CardsListView {
        id: form
        Kirigami.FormLayout {
            id: restaurantsListView
            Layout.fillWidth: true
            Layout.minimumWidth: aCard.implicitWidth
            Layout.fillHeight: true
            Layout.alignment: Qt.AlignLeft | Qt.AlignBottom
            model: foodPlacesModel
            delegate: Kirigami.AbstractCard {
                id: rootCard
                Kirigami.FormData.label: "Description:"
                implicitHeight: delegateItem.implicitHeight + Kirigami.Units.largeSpacing
                contentItem: Item {
                    wrapMode: Text.WordWrap
                    implicitWidth: parent.implicitWidth
                    implicitHeight: parent.implicitHeight
                    text: modelData.restaurantDescription
                }
                ColumnLayout {
                    {
                        id: delegateItem
                        Kirigami.FormData.label: "Phone:"
                        anchors.left: parent.left
                        anchors.right: parent.right
                        Layout.fillWidth: true
                        wrapMode: Text.WordWrap
                        anchors.top: parent.top
                        spacing: Kirigami.Units.smallSpacing
                        text: modelData.phone
                    }
                    Kirigami.Heading {
                        id: restaurantNameLabel
                        Layout.fillWidth: true
                        text: modelData.name
                    }
                    level: 3
                }
                wrapMode: Text.WordWrap
            }
        }
    }
    Kirigami.Separator {
        id: separator2
        Layout.fillWidth: true
    }
    RowLayout {
        id: rowLayout
        Layout.fillWidth: true
        Layout.preferredHeight: form.implicitHeight
    }
}

```

## Using Proportional Delegate For Simple Display Skills & Auto Layout

**ProportionalDelegate** is a delegate which has proportional padding and a columnlayout as mainItem. The delegate supports a `proportionalGridUnit` which is



based upon its size and the contents are supposed to be scaled proportionally to the delegate size either directly or using the `proportionalGridUnit`.

**AutoFitLabel** is a label that will always scale its text size according to the item size rather than the other way around

### QML Example

```
import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft

Mycroft.ProportionalDelegate {
    id: root

    Mycroft.AutoFitLabel {
        id: monthLabel
        font.weight: Font.Bold
        Layout.fillWidth: true
        Layout.preferredHeight: proportionalGridUnit * 40
        text: sessionData.month
    }

    Mycroft.AutoFitLabel {
        id: dayLabel
        font.weight: Font.Bold
        Layout.fillWidth: true
        Layout.preferredHeight: proportionalGridUnit * 40
        text: sessionData.day
    }
}
```

### Using Slideshow Component To Show Cards Slideshow

Slideshow component lets you insert a slideshow with your custom delegate in any skill display which can be tuned to autoplay and loop and also scrolled or flicked manually by the user.

### QML Example

```

}import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft

Mycroft.Delegate {
    id: root

    Mycroft.SlideShow {
        id: simpleSlideShow
        model: sessionData.exampleModel // model with slideshow data
        anchors.fill: parent
        interval: 5000 // time to switch between slides
        running: true // can be set to false if one wants to swipe manually
        loop: true // can be set to play through continuously or just once
        delegate: Kirigami.AbstractCard {
            width: rootItem.width
            height: rootItem.height
            contentItem: ColumnLayout {
                anchors.fill: parent
                Kirigami.Heading {
                    Layout.fillWidth: true
                    wrapMode: Text.WordWrap
                    level: 3
                    text: modelData.Title
                }
                Kirigami.Separator {
                    Layout.fillWidth: true
                    Layout.preferredHeight: 1
                }
                Image {
                    Layout.fillWidth: true
                    Layout.preferredHeight: rootItem.height / 4
                    source: modelData.Image
                    fillMode: Image.PreserveAspectCrop
                }
            }
        }
    }
}

```

## Event Handling

Mycroft GUI API provides an Event Handling Protocol between the skill and QML display which allow Skill Authors to forward events in either direction to an event consumer. Skill Authors have the ability to create any amount of custom events. Event names that start with "system." are available to all skills, like previous/next/pick.

### Simple Event Trigger Example From QML Display To Skill

#### Python Skill Example

```
def initialize(self):
    # Initialize...
    self.gui.register_handler('skill.foo.event', self.handle_foo_event)
...
def handle_foo_event(self, message):
    self.speak(message.data["string"])
...
...
```

#### QML Example

```

import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft

Mycroft.Delegate {
    id: root

    Button {
        anchors.fill: parent
        text: "Click Me"
        onClicked: {
            triggerGuiEvent("skill.foo.event", {"string": "Lorem ipsum dolor sit amet"})
        }
    }
}

```

## Simple Event Trigger Example From Skill To QML Display

### Python Skill Example

```

...
def handle_foo_intent(self, message):
    self.gui['foobar'] = message.data.get("utterance")
    self.gui['color'] = "blue"
    self.gui.show_page("foo")
...
...

```

### QML Example

```

import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft

Mycroft.Delegate {
    id: root
    property var fooString: sessionData.foobar

    onFooStringChanged: {
        fooRect.color = sessionData.color
    }

    Rectangle {
        id: fooRect
        anchors.fill: parent
        color: "#fff"
    }
}

```

## Resting Faces

The resting face API provides skill authors the ability to extend their skills to supply their own customized IDLE screens that will be displayed when there is no activity on the screen.

### Simple Idle Screen Example

#### Python Skill Example

```

from ovos_workshop.decorators import resting_screen_handler
...
@resting_screen_handler('NameOfIdleScreen')
def handle_idle(self, message):
    self.gui.clear()
    self.log.info('Activating foo/bar resting page')
    self.gui["exampleText"] = "This Is A Idle Screen"
    self.gui.show_page('idle.qml')

```

#### QML Example

```

import QtQuick 2.4
import QtQuick.Controls 2.2
import QtQuick.Layouts 1.4
import org.kde.kirigami 2.4 as Kirigami
import Mycroft 1.0 as Mycroft

Mycroft.Delegate {
    id: root
    property var fooString: sessionData.exampleText

    Kirigami.Heading {
        id: headerExample
        anchors.centerIn: parent
        text: fooString
    }
}

```

## docs/711-qt\_voice\_apps.md

# QT Applications

> **NOTE:** Only QT5 is supported, help wanted to migrate to QT6!

You can build full standalone QT Voice Applications using [QML](#) with ovos-gui either via skills or OVOSAbstractApp

## Desktop Files

GUI clients are allowed to filter a namespace, providing a GUI skill in it's own dedicated window.

This is what powers [Plasma Bigscreen Voice Apps](#) via .desktop files

Desktop files are also parsed to populate the skills in the [homescreen](#) app drawer

!

> **NOTE:** be sure to have [mycroft-gui-qt5](#) installed

```
[Desktop Entry]
X-DBUS-StartupType=None
X-KDE-StartupNotify=false
Version=1.0
Terminal=false
Type=Application
Name=OCP
Exec=ovos-gui-app --hideTextInput --skill=ovos.common_play
Icon=OCP
Categories=VoiceApp
StartupNotify=false
```

> Learn more about the [Desktop Entry Specification](#)

## docs/720-skill\_gui.md

### GUIInterface

Any component wanting to implement a GUI for OpenVoiceOS can do so via the GUIInterface class from [ovos-bus-client](#)

Sending custom pages from skills requires skill to explicitly support a client platform

```
class GUIInterface:
    """
    Interface to the Graphical User Interface, allows interaction with
    the mycroft-gui from anywhere

    Values set in this class are synced to the GUI, accessible within QML
    via the built-in sessionData mechanism.  For example, in Python you can
    write in a skill:
        self.gui['temp'] = 33
        self.gui.show_page('Weather')

    Then in the Weather.qml you'd access the temp via code such as:
        text: sessionData.time
    """
```

in OVOS Skills `self.gui` provides a `GUIInterface` under `self.skill_id` namespace

## Page Templates

To have a unified look and feel, and to allow simple UIs to be integrated into skills without UI framework knowledge, the `GUIInterface` provides page templates

A page template is a ui file, like QML or html, that is used by gui clients to render the info provided by `ovos-gui`.

Skills may provide their own pages, for example for [QT Voice Apps](#), but is their responsibility to explicitly support individual gui client apps if not using a provided template

## Text

Display simple strings of text.

```
self.gui.show_text(self, text, title=None, override_idle=None, override_animations=False)
```

Arguments:

`text` `\(str\)`: Main text content. It will auto-paginate

`title` `\(str\)`: A title to display above the text content.

`override_idle` `\(boolean, int\)`:

True: Takes over the resting page indefinitely

`\(int\)`: Delays resting page for the specified number of seconds.

`override_animations` `\(boolean\)`:

True: Disables showing all platform skill animations.

False: 'Default' always show animations.

## Static Image

Display a static image such as a jpeg or png.



```
self.gui.show_image(self, url, caption=None, title=None, fill=None, override_idle=None, ov
```

Arguments:

url \(\str\): Pointer to the image

caption \(\str\): A caption to show under the image

title \(\str\): A title to display above the image content

fill \(\str\): Fill type - supports:

'PreserveAspectFit',

'PreserveAspectCrop',

'Stretch'

override\\_idle \(\boolean, int\):

True: Takes over the resting page indefinitely

\(int\): Delays resting page for the specified number of seconds.

override\\_animations \(\boolean\):

True: Disables showing all platform skill animations.

False: 'Default' always show animations.

## Animated Image

Display an animated image such as a gif.

```
self.gui.show_animated_image(self, url, caption=None, title=None, fill=None, override_idle
```

Arguments:

url \(\str\): Pointer to the .gif image

caption \(\str\): A caption to show under the image

title \(\str\): A title to display above the image content

fill \(\str\): Fill type - supports:

'PreserveAspectFit',

'PreserveAspectCrop',

'Stretch'

override\\_idle \ (boolean, int\):

True: Takes over the resting page indefinitely

\ (int\): Delays resting page for the specified number of seconds.

override\\_animations \ (boolean\):

True: Disables showing all platform skill animations.

False: 'Default' always show animations.

## HTML Page

Display a local HTML page.

```
self.gui.show_html(self, html, resource_url=None, override_idle=None, override_animations=
```

Arguments:

html \ (str\): HTML text to display

resource\\_url \ (str\): Pointer to HTML resources

override\\_idle \ (boolean, int\):

True: Takes over the resting page indefinitely

\ (int\): Delays resting page for the specified number of seconds.

override\\_animations \ (boolean\):

True: Disables showing all platform skill animations.

False: 'Default' always show animations.

## Remote URL

Display a webpage.

```
self.gui.show_url(self, url, override_idle=None, override_animations=False)
```

Arguments:

url \(\str\): URL to render

override\\_idle \(\boolean, int\):

True: Takes over the resting page indefinitely

\(int\): Delays resting page for the specified number of seconds.

override\\_animations \(\boolean\):

True: Disables showing all platform skill animations.

False: 'Default' always show animations.

## docs/800-mk1\_api.md

### Enclosure Api

The EnclosureApi is an abstraction over an hypothetical "body" housing OVOS

eg, The Mark 1 **Device** is housed in an **Enclosure**. The Enclosure is the shell that houses a **Device** that runs OVOS.

```
from ovos_bus_client.apis.enclosure import EnclosureApi
```

```
api = EnclosureApi(bus)
```

### The Mark 1 Enclosure capabilities

The Mark 1 `_mouth_` and `_eyes_` can be controlled by **Skills** using the `self.enclosure` object inherited from the `OVOSkill` base class. This object acts as an interface to the **Enclosure** and allows the Skill creator to draw to the `_mouth_` display.

This is how the `_mouth_` and `_eyes_` are made to change during operations such as audio playback.

Dedicated utils for fine grained control over the mark 1 can be found at [ovos-mark1-utils](#)

## Drawing to the `_mouth_` display

### Drawing text to the `_mouth_` display

Text can be sent to the display using the `mouth_text()` method of the `enclosure` object.

```
self.enclosure.mouth_text('The meaning of life, the universe and everything is 42')
```

If the text is too long to fit on the display, the text will scroll.

@TODO how many characters will fit on the display before it will scroll?

### Drawing images to the `_mouth_` display

#### Clearing an existing image from the mouth display

Before writing an image to the `_mouth_` display, you should clear any previous image.

```
self.enclosure.mouth_display(img_code="HIAAAAAAAAAAAAA", refresh=False)
self.enclosure.mouth_display(img_code="HIAAAAAAAAAAAAA", x=24, refresh=False)
```

### How images are drawn on the `_mouth_` display

The `_mouth_` display is a grid, 32 pixels wide and 8 pixels high.

There are two ways to draw an image on the `_mouth_` display.

#### Addressing each pixel using a string encoding

You can draw an image to the `_mouth_` display by binary encoding pixel information in a string.

The binary encoding is straightforward value substitution.

| Letter | Value | Pixel value |
|--------|-------|-------------|
| A      | 0     | 1           |
| B      | 1     | 2           |
| C      | 2     | 3           |
| D      | 3     | 4           |
| E      | 4     | 5           |
| F      | 5     | 6           |
| G      | 6     | 7           |
| H      | 7     | 8           |

and so on.

```
self.enclosure.mouth_display(img_code="HIAAAAAAAAAAAAA", refresh=False)
self.enclosure.mouth_display(img_code="HIAAAAAAAAAAAAA", x=24, refresh=False)
```

The code above clears the image by sending a string consisting of HI which stands for a Width of 7 and a height of 8 and each A stands for a segment of 4 pixels in the off state.

@TODO we really need a grid image here to show how it works - to make it easier to understand.

### **Sending a PNG image to the mouth display**

Another way to draw an image on the `_mouth_` display is to create a PNG-formatted image with a width of 32 pixels and a height of 8 pixels, then use the `mouth_display_png()` method of the `enclosure` object.

The image should be black and white, with white meaning a dark pixel, and black indicating an illuminated pixel.

`mouth_display_png()` expects the first argument to be the image absolute path. Optional arguments are

`threshold`: The value at which a pixel should be considered 'dark' or 'illuminated'

`invert`: Treat white in the image as illuminated pixels, and black as dark pixels

`x`: The x position \(\horizontal\) at which the image should be displayed, in pixels

`y`: The y position \(\vertical\) at which the image should be displayed, in pixels

`refresh`: clear the display before writing to it

@TODO all the above needs to be validated - the information is educated guesswork

```
self.mouth_display_png('/path/to/image.png', threshold=70, invert=False, x=0, y=0, refresh=True)
```

Example image: [!A note symbol](#)

### **Tools for converting PNG image representations to string representations**

If you don't want to convert PNG files at runtime \(\text{for example when creating simple animations}\) [this short python script](#) will convert PNG files to strings

compatible with the `img_code` of `self.enclosure.mouth_display()`.

## Resetting the display to the default state

When the **Skill** is finished, you should reset the **Enclosure** to the default state using

```
self.enclosure.reset()
```

This will clear the screen and blink the Mark 1's `_eyes_` once.

## docs/801-mk1\_utils.md

### Mark1 Utils

small library to interact with a [Mycroft Mark1 faceplate via the messagebus](#)

gives you full control of the faceplate and eyes pixel by pixel

!

### Animate the eyes

```
from ovos_mark1.eyes import Eyes
from ovos_bus_client.utils import get_mycroft_bus

bus = get_mycroft_bus("0.0.0.0")

eyes = Eyes(bus)
eyes.hue_spin()
```

### Faceplate Icons

```

from ovos_mark1.faceplate import BlackScreen

class MusicIcon(BlackScreen):
    str_grid = """
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX      XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX      XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX XXX XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX XXX XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX XX XXXXXXXXXXXXXXX
XXXXXXXXXXXXX   X   XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX XXX XXXXXXXXXXXXXXX
"""

icon = MusicIcon()
icon.print() # show in terminal
icon.display() # show in mark1

```

## Faceplate Animations

```

# it's snowing !
class FallingDots(FacePlateAnimation):
    def __init__(self, n=10, bus=None):
        super().__init__(bus=bus)
        self._create = True
        assert 0 < n < 32
        self.n = n

    @property
    def n_dots(self):
        n = 0
        for y in range(self.height):
            for x in range(self.width):
                if self.grid[y][x]:
                    n += 1
        return n

    def animate(self):
        self.move_down()
        if self._create:
            if random.choice([True, False]):
                self._create = False
                x = random.randint(0, self.width - 1)
                self.grid[0][x] = 1
        if self.n_dots < self.n:
            self._create = True

```

Pre made animations



```
from ovos_mark1.faceplate.animations import ParticleBox
from ovos_bus_client.utils import get_mycroft_bus
from time import sleep

bus = get_mycroft_bus("0.0.0.0")

for faceplate in ParticleBox(bus=bus):
    faceplate.display(invert=False)
    sleep(0.5)

from ovos_mark1.faceplate.cellular_automaton import Rule110

a = Rule110(bus=bus)

for grid in a:
    grid.print() # animate in terminal
    grid.display(invert=False)
    sleep(0.5)
```

## docs/890-date\_parser.md

### ovos-date-parser

`ovos-date-parser` is a comprehensive library for multilingual date and time parsing, extraction, and formatting, designed to handle a range of human-readable date, time, and duration expressions.

## Features

**Date and Time Extraction:** Extract specific dates and times from natural language phrases in various languages.

**Duration Parsing:** Parse phrases that indicate a span of time, such as "two hours and fifteen minutes."

**Friendly Time Formatting:** Format time for human-friendly output, supporting

both 12-hour and 24-hour formats.

**Relative Time Descriptions:** Generate relative descriptions (e.g., "tomorrow," "in three days") for given dates.

**Multilingual Support:** Includes extraction and formatting methods for multiple languages, such as English, Spanish, French, German, and more.

# Installation

```
pip install ovos-date-parser
```

# Languages Supported

ovos-date-parser supports a wide array of languages, each with its own set of methods for handling natural language time expressions.

- supported
- not supported
- imperfect placeholder, usually a language agnostic implementation or external library

## Parse

| Language              | extract_duration | extract_datetime |
|-----------------------|------------------|------------------|
| ----- ----- -----  az |                  |                  |
| ca                    |                  |                  |
| cs                    | da               |                  |
| de                    |                  |                  |
| en                    | es               |                  |
| gl                    | eu               |                  |
| fa                    |                  |                  |
| fr                    | hu               |                  |
| it                    | nl               |                  |
| pl                    |                  |                  |
| pt                    | ru               |                  |
| sv                    |                  |                  |

```
uk      |          |          |
```

> If a language is not implemented for `extract_datetime` then [dateparser](#) will be used as a fallback

## Format

```
| Language | nice_date<br>nice_date_time<br>nice_day <br>nice_weekday <br>nice_month <br>nice_year <br>get_date_strings | nice_time | nice_relative_time | nice_duration |
```

```
|-----|-----|
-----|-----|
-----|-----|
```

```
| az      |
|         |         |         | | ca      |
|         |         |         | | cs      |
|         |         |         | | da      |
|         |         |         | | de      |
|         |         |         | | en      |
|         |         |         | | es      |
|         |         |         | | gl      |
|         |         |         | | eu      |
|         |         |         | | fa      |
|         |         |         | | fr      |
|         |         |         | | hu      |
|         |         |         | | it      |
|         |         |         | | nl      |
|         |         |         | | pl      |
|         |         |         | | pt      |
|         |         |         | | ru      |
|         |         |         | | sv      |
|         |         |         | | sl      |
|         |         |         | | uk      |
|         |         |         |
```

# Usage

## Date and Time Extraction

Extract specific dates and times from a phrase. This function identifies

date-related terms in natural language and returns both the datetime object and any remaining text.

```
from ovos_date_parser import extract_datetime

result = extract_datetime("Meet me next Friday at 3pm", lang="en")
print(result)  # (datetime object, "at 3pm")
```

## Duration Extraction

Identify duration phrases in text and convert them into a `timedelta` object. This can parse common human-friendly duration expressions like "30 minutes" or "two and a half hours."

```
from ovos_date_parser import extract_duration

duration, remainder = extract_duration("It will take about 2 hours and 30 minutes", lang="en")
print(duration)  # timedelta object
print(remainder)  # "about"
```

## Formatting Time

Generate a natural-sounding time format suitable for voice or display in different languages, allowing customization for speech or written text.

```
from ovos_date_parser import nice_time
from datetime import datetime

dt = datetime.now()
formatted_time = nice_time(dt, lang="en", speech=True, use_24hour=False)
print(formatted_time)  # "three o'clock"
```

## Relative Time Descriptions

Create relative phrases for describing dates and times in relation to the current moment or a reference datetime.

```
from ovos_date_parser import nice_relative_time
from datetime import datetime, timedelta

relative_time = nice_relative_time(datetime.now() + timedelta(days=1), datetime.now(), language="en")
print(relative_time)  # "tomorrow"
```

## Related Projects

[ovos-number-parser](#) - for handling numbers

[ovos-lang-parser](#) - for handling languages

[ovos-color-parser](#) - for handling colors

## docs/891-number\_parser.md

# OVOS Number Parser

OVOS Number Parser is a tool for extracting, pronouncing, and detecting numbers from text across multiple languages. It supports functionalities like converting numbers to their spoken forms, extracting numbers from text, identifying fractional and ordinal numbers, and more.

## Features

**Pronounce Numbers:** Converts numerical values to their spoken forms.

**Pronounce Ordinals:** Converts numbers to their ordinal forms.

**Extract Numbers:** Extracts numbers from textual inputs.

**Detect Fractions:** Identifies fractional expressions.

**Detect Ordinals:** Checks if a text input contains an ordinal number.

## Supported Languages

- supported
- not supported
- imperfect placeholder, usually a language agnostic implementation or external library

| Language Code         | Pronounce Number | Pronounce Ordinal | Extract Number |
|-----------------------|------------------|-------------------|----------------|
| numbers_to_digits     |                  |                   |                |
| -----                 | -----            | -----             | -----          |
| -----                 |                  |                   |                |
| en (English)          |                  |                   |                |
| az (Azerbaijani)      |                  |                   |                |
| ca (Catalan)          |                  |                   |                |
| gl (Galician)         |                  |                   |                |
| cs (Czech)            |                  |                   |                |
| da (Danish)           |                  |                   |                |
| de (German)           |                  |                   |                |
| es (Spanish)          |                  |                   |                |
| eu (Euskara / Basque) |                  |                   |                |
| fa (Farsi / Persian)  |                  |                   |                |
| fr (French)           |                  |                   |                |
| hu (Hungarian)        |                  |                   |                |
| it (Italian)          |                  |                   |                |
| nl (Dutch)            |                  |                   |                |
| pl (Polish)           |                  |                   |                |
| pt (Portuguese)       |                  |                   |                |
| ru (Russian)          |                  |                   |                |
| sv (Swedish)          |                  |                   |                |
| sl (Slovenian)        |                  |                   |                |
| uk (Ukrainian)        |                  |                   |                |

> If a language is not implemented for `pronounce_number` or `pronounce_ordinal` then [unicode-rbnf](#) will be used as a fallback

## Installation

To install OVOS Number Parser, use:

```
pip install ovos-number-parser
```

## Usage

### Pronounce a Number

Convert a number to its spoken equivalent.

```
def pronounce_number(number: Union[int, float], lang: str, places: int = 2, short_scale: bool = True,
                    scientific: bool = False, ordinals: bool = False) -> str:
    """
    Convert a number to its spoken equivalent.

    Args:
        number: The number to pronounce.
        lang (str): A BCP-47 language code.
        places (int): Number of decimal places to express. Default is 2.
        short_scale (bool): Use short (True) or long scale (False) for large numbers.
        scientific (bool): Pronounce in scientific notation if True.
        ordinals (bool): Pronounce as an ordinal if True.

    Returns:
        str: The pronounced number.
    """
```

#### Example Usage:

```
from ovos_number_parser import pronounce_number

# Example
result = pronounce_number(123, "en")
print(result)  # "one hundred and twenty-three"
```

### Pronounce an Ordinal

Convert a number to its ordinal spoken equivalent.

```
def pronounce_ordinal(number: Union[int, float], lang: str, short_scale: bool = True) -> str:
    """
    Convert an ordinal number to its spoken equivalent.

    Args:
        number: The number to pronounce.
        lang (str): A BCP-47 language code.
        short_scale (bool): Use short (True) or long scale (False) for large numbers.

    Returns:
        str: The pronounced ordinal number.
    """
```

### Example Usage:

```
from ovos_number_parser import pronounce_ordinal

# Example
result = pronounce_ordinal(5, "en")
print(result)  # "fifth"
```

## Extract a Number

Extract a number from a given text string.

```
def extract_number(text: str, lang: str, short_scale: bool = True, ordinals: bool = False) -> Union[int, float, bool]:
    """
    Extract a number from text.

    Args:
        text (str): The string to extract a number from.
        lang (str): A BCP-47 language code.
        short_scale (bool): Use short scale if True, long scale if False.
        ordinals (bool): Consider ordinal numbers.

    Returns:
        int, float, or False: The extracted number, or False if no number found.
    """
```

### Example Usage:



```
from ovos_number_parser import extract_number

# Example
result = extract_number("I have twenty apples", "en")
print(result)  # 20
```

## Check for Fractional Numbers

Identify if the text contains a fractional number.

```
def is_fractional(input_str: str, lang: str, short_scale: bool = True) -> Union[bool, float]:
    """
    Check if the text is a fraction.

    Args:
        input_str (str): The string to check if fractional.
        lang (str): A BCP-47 language code.
        short_scale (bool): Use short scale if True, long scale if False.

    Returns:
        bool or float: False if not a fraction, otherwise the fraction as a float.
    """
```

### Example Usage:

```
from ovos_number_parser import is_fractional

# Example
result = is_fractional("half", "en")
print(result)  # 0.5
```

## Check for Ordinals

Determine if the text contains an ordinal number.

```
def is_ordinal(input_str: str, lang: str) -> Union[bool, float]:
    """
    Check if the text is an ordinal number.

    Args:
        input_str (str): The string to check if ordinal.
        lang (str): A BCP-47 language code.

    Returns:
        bool or float: False if not an ordinal, otherwise the ordinal as a float.
    """
```

### Example Usage:

```
from ovos_number_parser import is_ordinal

# Example
result = is_ordinal("third", "en")
print(result)  # 3
```

## Related Projects

[ovos-date-parser](#) - for handling dates and times

[ovos-lang-parser](#) - for handling languages

[ovos-color-parser](#) - for handling colors

## License

This project is licensed under the Apache License 2.0.

## docs/893-color\_parser.md

## OVOS Color Parser

> :warning: this package is a work in progress

# What does this have to do with voice?

"change the lamp color to moss green"

"make it darker"

"more saturated"

"a bit more yellowish"

"perfect"

> NOTE: physicists are huge nerds, so they might say something like "change the lamp wave length to X nanometers", this > is a terrible way to talk about color and inaccurate but we also added basic support for this

## Extracting a color from text

The parser will do its best to parse "color modifiers"

```
from ovos_color_parser import color_from_description

names = [
    "Bright, vibrant green",
    "Pale pink",
    "Muted, warm gray",
    "Dark, cool blue",
]

for n in names:
    c = color_from_description(n)
    print(c.hex_str)
    print(c)
```

[!image](#)

Color names are ambiguous, the same name sometimes refers to multiple colors.

When a color is matched by the parser it "averages all matched colors" 'python

```
from ovos_color_parser import color_from_description
```

```
color = color_from_description("Red") print(color.hex_str) #D21B1B print(color)
```

```
# sRGBColor(r=210, g=27, b=27, name='Red', description='Red') ' !image
```

We can tell the parser to always return a known/named color with

```
cast_to_palette=True, but this might not always return what you expect 'python  
from ovos_color_parser import color_from_description
```

```
color = color_from_description("Red", cast_to_palette=True) print(color.hex_str)  
#CE202B print(color)
```

```
# sRGBColor(r=206, g=32, b=43, name='Fire engine red', description='Red') ' !  
image
```

## Beware of impossible colors

Some colors are [impossible](#), but that doesn't stop text from describing them

"Reddish-green" doesn't make much sense as a description, unless you mean yellow or orange, which you don't, because you would have said yellow or orange. The same applies to "Yellowishblue"

> the Colour of Magic or the King Colour, was the eighth colour of the Discworld spectrum. Only visible to wizards and cats. It is described in "The Colour of Magic" as the colour of imagination and is a fluorescent greenish yellow-purple. The only time non-wizards can see it is when they close their eyes; the bursts of color are octarine.

Fluorescent greenish-yellow and purple are essentially opposite colors on the color wheel, with wavelengths that can't coexist in a single light wave in the visible spectrum. Here's why:

Color Wavelengths and Light: Greenish-yellow light falls in a wavelength range of about 560-590 nanometers, while purple is not a pure spectral color but a combination of blue (around 450-495 nm) and red (around 620-750 nm). Human eyes perceive purple as a combination of these two ends of the spectrum.

Color Opponency Theory: The human visual system relies on color opponency, where certain pairs of colors (like red-green and blue-yellow) are processed in opposing channels. Because of this, our brains can't interpret colors that simultaneously activate both ends of an opponent channel. This is why we don't perceive colors like reddish-green or yellowish-blue; our brains are simply wired to cancel out those combinations.

Perceptual Limits: Fluorescent colors are especially intense because they emit light in a narrow, concentrated wavelength range, making them appear very

saturated and bright. Attempting to mix fluorescent greenish-yellow with purple not only challenges the physiology of the eye but would also result in a muted brown or gray tone, as the colors cancel each other out.

In short, fluorescent greenish-yellow and purple light cant coexist in a way our eyes can interpret as a single, stable color because of the biological limits of human color perception.

```
from ovos_color_parser import color_from_description

# look! an impossible color
color = color_from_description("fluorescent greenish-yellow purple")
color.name = "Octarine"
print(color.hex_str) #76B11D
print(color)
# sRGBColor(r=118, g=177, b=29, name='Octarine', description='fluorescent greenish-yellow
```

the parser will gladly output something... it just might not make sense

in this case the parser focused on "greenish-yellow"

[!image](#)

but it could have focused on "purple"

[!image](#)

## Comparing color objects

compare color distances (smaller is better)

```
from ovos_color_parser import color_distance, color_from_description

color_a = color_from_description("green")
color_b = color_from_description("purple")
print(color_distance(color_a, color_b))
# 64.97192890677195

color_a = color_from_description("green")
color_b = color_from_description("yellow")
print(color_distance(color_a, color_b))
# 44.557493285361

color_a = color_from_description("yellow")
color_b = color_from_description("purple")
print(color_distance(color_a, color_b))
# 78.08287998809946
```

match a color object to a list of colors

```

from ovos_color_parser import sRGBAColor, sRGBAColorPalette, closest_color

# https://en.wikipedia.org/wiki/Blue-green
BlueGreenPalette = sRGBAColorPalette(colors=[
    sRGBAColor(r=0, g=128, b=128, name="Blue-green"),
    sRGBAColor(r=0, g=255, b=255, name="Cyan (Aqua)", description="Brilliant bluish green"),
    sRGBAColor(r=64, g=224, b=208, name="Turquoise", description="Brilliant bluish green"),
    sRGBAColor(r=17, g=100, b=180, name="Green-blue", description="Strong blue"),
    sRGBAColor(r=57, g=55, b=223, name="Bondi blue"),
    sRGBAColor(r=0, g=165, b=156, name="Blue green (Munsell)", description="Brilliant bluish green"),
    sRGBAColor(r=0, g=123, b=167, name="Cerulean", description="Strong greenish blue"),
    sRGBAColor(r=0, g=63, b=255, name="Cerulean (RGB)", description="Vivid blue"),
    sRGBAColor(r=0, g=128, b=128, name="Teal", description="Moderate bluish green"),
])

print(closest_color(sRGBAColor(r=0, g=0, b=255, name="Blue"),
    BlueGreenPalette.colors))

# sRGBAColor(r=0, g=63, b=255, name='Cerulean (RGB)', description='Vivid blue')
print(closest_color(sRGBAColor(r=0, g=255, b=0, name="Green"),
    BlueGreenPalette.colors))

# sRGBAColor(r=64, g=224, b=208, name='Turquoise', description='Brilliant bluish green')

---
```

## Language support

When describing color in natural language to approximate it in RGB, there are several keywords that can convey its properties effectively

```

# Parse complex color descriptions
color = color_from_description("very bright, slightly warm muted blue")
```

**Description:** Hue refers to the basic color family, such as red, blue, green, or yellow.

### Translation to RGB:

The hue determines which of the primary RGB channels (red, green, or blue) will be most prominent. For example,

red means a strong red channel with low green and blue, while blue means a

high blue channel with low red and green.

Hues like "yellow" indicate both red and green channels are high with blue low, while "purple" combines red and

blue with little green.

---

**Description:** Saturation, or chroma, is how pure or intense the color is.

Terms like vibrant, dull, or washed

out refer to saturation.

**Translation to RGB:**

High saturation (vibrant): Increase the difference between the dominant channel(s) and others. For example, making

the red channel much higher than green and blue for a vibrant red.

Low saturation (dull): Reduce the contrast between channels, creating a blend closer to grayscale. For instance,

balancing red, green, and blue channels to similar values lowers saturation.

---

**Description:** Brightness refers to how light or dark the color appears. Words like bright, dim, dark, or

pale are often used.

**Translation to RGB:**

High brightness (bright): Increase the values across all channels.

Low brightness (dark): Decrease values across channels while maintaining the hue's relative balance.

---

**Description:** Color temperature reflects whether a color feels warm or cool.

Terms like "warm red," "cool green,"

or "cold blue" apply here.

**Translation to RGB:**



Warm colors: Increase red or red and green channels.

Cool colors: Increase blue or decrease red.

---

**Description:** Opacity doesn't affect RGB but is relevant for color perception, especially in design. Terms like

translucent, opaque, or sheer describe it.

#### **Translation to RGB:**

Opacity affects the alpha channel (RGBA) rather than RGB values.

This approach, while interpretative, offers a structured way to translate natural language color descriptions into RGB approximations.

## **Color Keywords**

To categorize adjectives and keywords that describe color in ways that translate into RGB or color space adjustments the parser uses a `.json` file per language

Example JSON structure for English color keywords:

```
{
  "saturation": {
    "high": ["vibrant", "rich", "bold", "deep"],
    "low": ["dull", "muted", "washed-out", "faded"]
  },
  "brightness": {
    "high": ["bright", "light", "pale", "glowing"],
    "low": ["dim", "dark", "shadowy", "faint"]
  }
}
```

Color name lists in each language are also used to determine the **hue**.

> English has a word list of almost ~6000 color name mappings

Below are some examples of non-color-name keywords that define other qualities of a color

**Very High Saturation:** For colors that are extremely intense or vivid.

Keywords: neon, saturated, intense, brilliant, flamboyant

**High Saturation:** These adjectives indicate vibrant or intense colors where the hue is pronounced.

Keywords: vibrant, rich, bold, deep, vivid, intense, pure, electric

**Low Saturation:** These adjectives imply a muted or washed-out appearance, often making the color appear closer to grayscale.

Keywords: dull, muted, washed-out, faded, soft, pale, subdued, pastel

**Very Low Saturation:** For colors that are very desaturated, nearing grayscale.

Keywords: drab, grayed, washed-out, faded, subdued

---

**Very High Brightness:** Extremely bright colors, often implying high lightness or near-whiteness.

Keywords: blinding, radiant, glowing, white, light-filled

**High Brightness:** Bright colors, often indicating a lighter shade or close to white.

Keywords: bright, light, pale, glowing, luminous, brilliant, clear, radiant

**Low Brightness:** These terms describe darker or dimmer shades, closer to black.

Keywords: dim, dark, shadowy, faint, gloomy, subdued, deep, midnight

**Very Low Brightness:** Colors that are nearly black or very dark.

Keywords: pitch-dark, black, shadowed, deep, ink-like

---

**Very High Temperature (Very Warm):** Intense warm colors, strongly leaning toward red, orange, or intense yellow.

Keywords: fiery, lava-like, burning, blazing

**High Temperature (Warm Colors):** Warmer colors suggest a shift towards red or yellow tones, giving the color a warmer feel.

Keywords: warm, hot, fiery, sunny, toasty, scorching, amber, reddish

**Low Temperature (Cool Colors):** Cooler colors involve blue or green tones, giving the color a cooler or icy appearance.

Keywords: cool, cold, chilly, icy, frosty, crisp, bluish, aqua

**Very Low Temperature (Very Cool):** Extremely cool tones, verging on cold, icy blues or greens.

Keywords: icy, arctic, frigid, wintry, glacial

---

**Very High Opacity:** Extremely solid or dense colors.

Keywords: impenetrable, opaque, thick

**High Opacity:** Describes solid colors without transparency.

Keywords: opaque, solid, dense, thick, cloudy, impenetrable, strong

**Low Opacity:** Indicates transparency or translucency, where the background may show through.

Keywords: transparent, translucent, sheer, see-through, misty, delicate, airy

**Very Low Opacity:** Highly transparent or barely visible colors.

Keywords: ethereal, ghostly, barely-there, translucent

## Related Projects

[ovos-number-parser](#) - for handling numbers

[ovos-date-parser](#) - for handling dates and times

[ovos-lang-parser](#) - for handling languages

## docs/900-bus\_client.md

# MessageBus Client

The [OVOS MessageBus Client](#) is a Python module providing a simple interface for the OVOS MessageBus. It can be used to connect to OVOS, send messages, and react to messages sent by the OVOS system.

The module is available through [PyPI.org](#) or directly [on Github](#).

## MessageBusClient()

The `MessageBusClient()` object can be setup to connect to any host and port as well as any endpoint on that host. this makes it quite versatile and will work on the main bus as well as on a gui bus.

If no arguments are provided it will try to connect to a local instance of OVOS on the default endpoint and port.

> in skills and plugins `self.bus` provides a `MessageBusClient` connections out of the box, you don't usually need to initialize this yourself

## Message()

The `Message` object is a representation of the messagebus message, this will always contain a message type but can also contain data and context. Data is usually real information while the context typically contain information on where the message originated or who the intended recipient is.

```
Message('MESSAGE_TYPE', data={'meaning': 42}, context={'origin': 'A.Dent'})
```

## Sending a Message

In the following example we setup an instance of the `MessageBusClient` then emit a

send Message with a data payload. OVOS would consume this Message and speak "Hello World".

```
from ovos_bus_client import MessageBusClient, Message

print('Setting up client to connect to a local mycroft instance')
client = MessageBusClient()
client.run_in_thread()

print('Sending speak message...')
client.emit(Message('speak', data={'utterance': 'Hello World'}))
```

## Listening for a Message

In the following example we setup an instance of the MessageBusClient. We then define a function `print_utterance` that prints the utterance from a Message. This is registered as a handler for the `speak` Message. Finally we call the `run_forever()` method to keep the client running.

If this code had run before the example above, it would catch the `speak` Message we emitted and print: OVOS said "Hello World"

```
from ovos_bus_client import MessageBusClient, Message

print('Setting up client to connect to a local ovos instance')
client = MessageBusClient()

def print_utterance(message):
    print('OVOS said "{}"'.format(message.data.get('utterance'))))

print('Registering handler for speak message...')
client.on('speak', print_utterance)

client.run_forever()
```

# docs/910-quebra\_frases.md

# Quebra Frases

The `quebra_frases` package provides essential text processing tools for tokenization, chunking, and token analysis.

**No External Dependencies:** `quebra_frases` is designed to be lightweight and does not rely on external libraries other than `regex` for efficient text processing.

## Installation

You can install the `quebra_frases` package using `pip`:

```
pip install quebra_frases
```

## Overview

The `quebra_frases` package includes several modules and functionalities:

**Tokenization:** Text tokenization is the process of splitting text into meaningful units such as words, sentences, or paragraphs.

**Chunking:** Text chunking involves dividing text into smaller chunks based on specified delimiters or patterns.

**Token Analysis:** This package also provides methods to analyze tokens across multiple text samples, extracting common, uncommon, and exclusive tokens.

## Usage

### Tokenization

The `quebra_frases` package offers various tokenization methods:

`word_tokenize(input_string)`: Tokenizes an input string into words.

`sentence_tokenize(input_string)`: Splits an input string into sentences.

`paragraph_tokenize(input_string)`: Divides an input string into paragraphs.

# Chunking

Chunking is performed using the following functions:

`chunk(text, delimiters)`: Splits text into chunks based on specified delimiters.

`get_common_chunks(samples)`: Extracts common chunks from a list of text samples.

`get_uncommon_chunks(samples)`: Extracts uncommon chunks from text samples.

`get_exclusive_chunks(samples)`: Extracts exclusive chunks that are unique to each text sample.

# Token Analysis

Token analysis functions are available for text sample comparison:

`get_common_tokens(samples)`: Extracts tokens that are common across multiple text samples.

`get_uncommon_tokens(samples)`: Extracts tokens that are uncommon across multiple text samples.

`get_exclusive_tokens(samples)`: Extracts tokens that are exclusive to each individual text sample.

# Example Usage

Tokenization

```

#(23, 77, 'This is another one.\t\n\tUsing multiple lines\t \n \n'),
import quebra_frases
#(77, 102, '\tparagraph 3 says goodbye')]

```

```

sentence = "sometimes i develop stuff for mycroft, mycroft is FOSS!"
print(quebra_frases.word_tokenize(sentence))
# ['sometimes', 'i', 'develop', 'stuff', 'for', 'mycroft', ',', '!', 'mycroft', 'is', 'FOSS', '!']

```

```

print(quebra_frases.span_indexed_word_tokenize(sentence))
# [(0, 9, 'sometimes'), (10, 11, 'i'), (12, 19, 'develop'),
# (20, 25, 'stuff'), (26, 29, 'for'), (30, 37, 'mycroft'),
# (37, 38, ','), (39, 46, 'mycroft'), (47, 49, 'is'),
# (50, 54, 'FOSS'), (54, 55, '!')]

```

```

print(quebra_frases.sentence_tokenize(
    "Mr. Smith bought cheapsite.com for 1.5 million dollars, i.e. he paid a lot for it. Di
#['Mr. Smith bought cheapsite.com for 1.5 million dollars, i.e. he paid a lot for it.',
# 'Did he mind?',
#"Adam Jones Jr. thinks he didn't.",
#"In any case, this isn't true...",
#"Well, with a probability of .9 it isn't."])

```

```

print(quebra_frases.span_indexed_sentence_tokenize(
    "Mr. Smith bought cheapsite.com for 1.5 million dollars, i.e. he paid a lot for it. Di
#[(0, 82, 'Mr. Smith bought cheapsite.com for 1.5 million dollars, i.e. he paid a lot for
#(83, 95, 'Did he mind?'),
#(96, 128, "Adam Jones Jr. thinks he didn't."),
#(129, 160, "In any case, this isn't true..."),
#(161, 201, "Well, with a probability of .9 it isn't.")])

```

```

print(quebra_frases.paragraph_tokenize('This is a paragraph!\n\t\nThis is another '
                                         'one.\t\n\tUsing multiple lines\t \n '
                                         '\n\tparagraph 3 says goodbye'))
#['This is a paragraph!\n\t\n',
# 'This is another one.\t\n\tUsing multiple lines\t \n \n',
# '\tparagraph 3 says goodbye']

```

```

print(quebra_frases.span_indexed_paragraph_tokenize('This is a paragraph!\n\t\nThis is ano
                                         'one.\t\n\tUsing multiple lines\t \n
                                         '\n\tparagraph 3 says goodbye'))
#[(0, 23, 'This is a paragraph!\n\t\n'),

```



## chunking

```
import quebra_frases

delimiters = ["OpenVoiceOS"]
sentence = "sometimes i develop stuff for OpenVoiceOS, OpenVoiceOS is FOSS!"
print(quebra_frases.chunk(sentence, delimiters))
# ['sometimes i develop stuff for', 'OpenVoiceOS', ',', 'OpenVoiceOS', 'is FOSS!']
```

## token analysis

```
import quebra_frases

samples = ["tell me what do you dream about",
           "tell me what did you dream about",
           "tell me what are your dreams about",
           "tell me what were your dreams about"]
print(quebra_frases.get_common_chunks(samples))
# {'tell me what', 'about'}
print(quebra_frases.get_uncommon_chunks(samples))
# {'do you dream', 'did you dream', 'are your dreams', 'were your dreams'}
print(quebra_frases.get_exclusive_chunks(samples))
# {'do', 'did', 'are', 'were'}

samples = ["what is the speed of light",
           "what is the maximum speed of a firetruck",
           "why are fire trucks red"]
print(quebra_frases.get_exclusive_chunks(samples))
# {'light', 'maximum', 'a firetruck', 'why are fire trucks red'})
print(quebra_frases.get_exclusive_chunks(samples, squash=False))
#[['light'],
#[ 'maximum', 'a firetruck'],
#[ 'why are fire trucks red']]
```

**docs/920-padacioso.md**

# Padacioso

*A lightweight, dead-simple intent parser*

Built on top of [simplematch](#), inspired by [Padaos](#)

## Example

```

from padacioso import IntentContainer

container = IntentContainer()

## samples
container.add_intent('hello', ['hello', 'hi', 'how are you', "what's up"])

## "optionally" syntax
container.add_intent('hello world', ["hello [world]"])

## "one_of" syntax
container.add_intent('greeting', ["(hi|hey|hello)"])

## entity extraction
container.add_intent('buy', [
    'buy {item}', 'purchase {item}', 'get {item}', 'get {item} for me'
])
container.add_intent('search', [
    'search for {query} on {engine}', 'using {engine} (search|look) for {query}',
    'find {query} (with|using) {engine}'
])
container.add_entity('engine', ['abc', 'xyz'])
container.calc_intent('find cats using xyz')
# {'conf': 1.0, 'name': 'search', 'entities': {'query': 'cats', 'engine': 'xyz'}}

## wildcards syntax
container.add_intent('say', ["say *"])
container.calc_intent('say something, whatever')
# {'conf': 0.85, 'entities': {}, 'name': 'test'}

## typed entities syntax
container.add_intent('pick_number', ['* number {number:int}'])
container.calc_intent('i want number 3')
# {'conf': 0.85, 'entities': {'number': 3}, 'name': 'pick_number'})

```

## docs/99-architecture-overview.md

# Architecture Overview

!

## docs/990-eggscript.md

### Eggscript

Eggscript is a markup language that can be "compiled" into a valid OVOS Skill

> **EXPERIMENTAL** This is an experimental feature

It is intended as an easy way for user to create simple skills, while offering an easy transition to regular skills

It also helps getting a lot of the boilerplate done for you when getting started

You can find a developer preview of eggscript in [github](#)

### Crash Course

Example files written in eggscript

hello.eggscript

```
// this is a comment // all comments and blank lines are ignored // special
interperter variables can be set with @var syntax // - @name -> skill name // -
@author -> skill author // - @email -> author contact // - @license -> skill
license // - @interpreter -> supported interperter, eg, cli // - @compiler ->
supported compiler, eg, mycroft skill
```

```
@author jarbasai @email jarbasai@mailfence.com @license MIT @name hello world
@url https://github.com/author/repo @version 0.1.0
```

```
// this script can be used standalone in the cli @interpreter cli // a standalone
python file can be generated @compiler cli // a mycroft skill can be generated
@compiler mycroft
```

```
// intent definition # hello world
```

```
hello world
```

```
hello world
```

```
// you can define python code, executed after TTS  hello = "world" if hello
== "world": print("python code!")
```

dialogs.eggscript

```
// this is a comment // all comments and blank lines are ignored
```

```
// text after # is the intent name # hello world // text after + is the user
utterance
```

hello world

```
// text after - is mycroft's response
```

hello world

```
# weather in location // you can capture variables and use them using {var}
syntax
```

how is the weather in {location}

how am i supposed to know the weather in {location}

```
# weather // this will create a intent file with the 3 + utterances
```

what is the weather like

how is the weather

how does it look outside

```
// this will create a dialog file with the 2 - utterances
```

i do not know how to check the weather

stick your head out of the window and check for yourself

```
# count to 10
```

count to 10

```
// if ident level matches its an alternate dialog
```

i will only count to 5

i only know how to count to 5

```
// use tab for indentation // each ident level defines a new utterance to be
spoken
```

1

2

3

4

5

layers.eggscript

// this is a comment // all comments and blank lines are ignored // this sample  
scripts show intent layers usage

// the number of # in intent definition determines an intent layer # tell me  
about

tell me about {thing}

{thing} exists

// N times + will enable layer N // to enable layer 2 ++

// use N times # for layer N // this intent is in layer 2, enabled by previous  
intent ## tell me more

tell me more

continue

i do not know more

// N times - will disable layer N // to disable layer 2 --

## Interpreters

Can run a subset of eggscript directly, enough to test simple skills in the  
terminal

```
from eggscript import CliInterpreter
from os.path import dirname

c = CliInterpreter()
c.load_eggscript_file(f"{dirname(__file__)}/dialogs.eggscript")
c.run()
```

## Compilers

```
from eggscript import OVOSSkillCompiler
from os.path import dirname

c = OVOSSkillCompiler()
c.load_eggscript_file(f"{dirname(__file__)}/layers.eggscript")
c.export("myskill")
```

You can now continue extending your exported skill to add more advanced functionality

## docs/999-ovos\_bigscreen.md

# Plasma Bigscreen - OVOS Edition

**EXPERIMENTAL** - experimental repository

> **WARNING** - Not actively maintained, this fork is essentially a snapshot in time since Plasma Bigscreen dropped support for OVOS and moved to QT6

## Introduction

A big launcher giving you easy access to any installed apps and skills.  
Controllable via voice or TV remote.

This project is using various open-source components like Plasma Bigscreen, OpenVoiceOS and libcec.

[!ovos-bigscreen](#)

This is a fork from <https://invent.kde.org/plasma/plasma-bigscreen/>

Changes:

- moves from Mycroft to OVOS

- "mycroft" is no longer optional and it's integration is enabled by default

- Remove MycroftSkillInstaller (not OVOS compliant)

- Remove "Recent" section

- Remove generic "Applications" section

- Add "Media" section

- Add "Network" section

- Add "Graphics" section

## Voice Control

Bigscreen supports OpenVoiceOS, a free and open-source voice assistant that can be run completely decentralized on your own server.

Download new apps (aka skills) for your Bigscreen or add your own ones for others to enjoy.

## Remote control your TV via CEC

CEC (Consumer Electronics Control) is a standard to control devices over HDMI.

Use your normal TV remote control, or a RC with built-in microphone for voice control and optional mouse simulation.

## Application Launcher

Bigscreen replaces your DE, to stop an application from showing up you can edit

`/etc/xdg/applications-blacklistrc`

Adding new applications only requires a [.desktop file](#)

see [plasma-bigscreen/bigscreen-image-settings](#) for more settings you might want to tweak in a Bigscreen image



# Installing from source

```
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/usr -DCMAKE_BUILD_TYPE=Release -DKDE_INSTALL_LIBDIR=lib
make
sudo make install
```

## Running

To start the Bigscreen homescreen in a window, run:

```
QT_QPA_PLATFORM=wayland dbus-run-session kwin_wayland "plasmashell -p org.kde.plasma.mycroft"
```

you can also select [plasma-bigscreen-x11](#) on your login screen as DE

## Related repositories

Image Settings for Bigscreen

<https://invent.kde.org/plasma-bigscreen/bigscreen-image-settings>

Plasma Remote Controllers

<https://invent.kde.org/plasma-bigscreen/plasma-remotecontrollers>

ovos-gui-app - <https://github.com/OpenVoiceOS/mycroft-gui-qt5>

bigscreen gui extension

<https://github.com/OpenVoiceOS/ovos-gui-plugin-bigscreen>

## docs/adapt\_pipeline.md

## Adapt Pipeline Plugin

The **Adapt Pipeline Plugin** brings rule-based intent parsing to the **OVOS intent pipeline** using the Adapt parser. It supports *high*, *medium*, and *low* confidence intent detection and integrates seamlessly with OVOSs multi-stage pipeline.

While Adapt is powerful for **explicit, deterministic matching**, it has notable limitations in multilingual environments and complex skill ecosystems. **In general, Adapt is not recommended for broad deployments** it is best suited for **personal skills** where you control the full context and can craft precise intent rules.

---

## Pipeline Stages

This plugin registers three pipelines:

| Pipeline ID          | Description  | Recommended Use                      |
|----------------------|--------------|--------------------------------------|
| -----                | -----        |                                      |
| -----                | adapt_high   | High-confidence Adapt intent matches |
| Personal skills only | adapt_medium | Medium-confidence Adapt matches      |
| Use with caution     | adapt_low    | Low-confidence Adapt matches         |
| Not recommended      |              |                                      |

Each pipeline is scored by Adapt and routed according to configured confidence thresholds.

---

## Limitations

Adapt requires **hand-crafted rules** for every intent:

**Poor scalability** hard to manage with many skills

**Difficult to localize** rules rely on exact words and phrases

**Prone to conflicts** multiple skills defining overlapping rules can cause collisions or missed matches

As your skill library grows or if you operate in a multilingual setup, these problems increase.

### Recommendation:

> Use Adapt **only** in personal projects or controlled environments where you can fully define and test every possible phrase.

---

# Configuration

Adapt confidence thresholds can be set in `ovos.conf`:

```
"intents": {
  "adapt": {
    "conf_high": 0.65,
    "conf_med": 0.45,
    "conf_low": 0.25
  }
}
```

These thresholds control routing into `adapt_high`, `adapt_medium`, and `adapt_low`.

The plugin is included by default in OVOS.

---

## When to Use Adapt in OVOS

Use this plugin **only when**:

You are building **a personal or private skill**.

You need **strict, predictable matching** (e.g., command-and-control).

You are working in **a single language** and **control all skill interactions**.

Avoid using Adapt for public-facing or general-purpose assistant skills. Modern alternatives like **Padatious**, **LLM-based parsers**, or **neural fallback models** are more scalable and adaptable.

## docs/audio\_transformers.md

## Audio Transformers

**Audio Transformers** in OpenVoiceOS (OVOS) are plugins designed to process raw audio input before it reaches the Speech-to-Text (STT) engine. They enable

functionalities such as noise reduction, language detection, and data transmission over sound, thereby enhancing the accuracy and versatility of voice interactions.

---

## Processing Flow

The typical audio processing pipeline in OVOS is as follows:

1. **Audio Capture:** Microphone captures raw audio input.
2. **Audio Transformation:** Audio Transformers preprocess the raw audio.
3. **Speech-to-Text (STT):** Transformed audio is converted into text.
4. **Intent Recognition:** Text is analyzed to determine user intent.

Audio Transformers operate in step 2, allowing for enhancements and modifications to the audio signal before transcription.

---

## Configuration

To enable Audio Transformers, add them to your `mycroft.conf` under the `audio_transformers` section:

```
"audio_transformers": {  
  "plugin_name": {  
    // plugin-specific configuration  
  }  
}
```

Replace `"plugin_name"` with the identifier of the desired plugin and provide any necessary configuration parameters.

---

## Available Audio Transformer Plugins

### OVOS GGWave Audio Transformer

**Purpose:** Enables data transmission over sound using audio QR codes.

**Features:**

Transmit data such as Wi-Fi credentials, URLs, or commands via sound.

Integrates with the `ovos-skill-ggwave` for voice-controlled activation.

**Installation:**

```
pip install ovos-audio-transformer-plugin-ggwave
```

**Configuration Example:**

```
"audio_transformers": {  
  "ovos-audio-transformer-plugin-ggwave": {  
    "start_enabled": true  
  }  
}
```

For more information, visit the [GitHub repository](#).

## OVOS SpeechBrain Language Detection Transformer

**Purpose:** Automatically detects the language of spoken input to route it to the appropriate STT engine.

**Features:**

Utilizes SpeechBrain models for language identification.

Enhances multilingual support by dynamically selecting the correct language model.

**Installation:**

```
pip install ovos-audio-transformer-plugin-speechbrain-langdetect
```

**Configuration Example:**

```
"audio_transformers": {  
  "ovos-audio-transformer-plugin-speechbrain-langdetect": {}  
}
```

For more information, visit the [GitHub repository](#).

---

## Creating Custom Audio Transformers

To develop your own Audio Transformer plugin for OVOS, implement a class that extends the base `AudioTransformer` template.

This class allows you to process raw audio chunks at various stages before the Speech-to-Text (STT) engine processes the audio.

### Base Class Overview

Your custom transformer should subclass:

```

from ovos_plugin_manager.templates.transformers import AudioTransformer

class MyCustomAudioTransformer(AudioTransformer):
    def __init__(self, config=None):
        super().__init__("my-custom-audio-transformer", priority=10, config=config)

    def on_audio(self, audio_data):
        # Process non-speech audio chunks (e.g., noise)
        return audio_data

    def on_hotword(self, audio_data):
        # Process full hotword/wakeword audio chunks
        return audio_data

    def on_speech(self, audio_data):
        # Process speech audio chunks during recording (not full utterance)
        return audio_data

    def on_speech_end(self, audio_data):
        # Process full speech utterance audio chunk
        return audio_data

    def transform(self, audio_data):
        # Optionally perform final transformation before STT stage
        # Return tuple (transformed_audio_data, optional_message_context)
        return audio_data, {}

```

## Lifecycle & Methods

**Initialization:** Override `initialize()` for setup steps.

### Audio Feed Handlers:

`on_audio`: Handle background or non-speech chunks.

`on_hotword`: Handle wakeword/hotword chunks.

`on_speech`: Handle speech chunks during recording.

`on_speech_end`: Handle full utterance audio.

### Final Transformation:

`transform`: Return the final processed audio and optionally a dictionary of additional metadata/context that will be passed along with the `recognize_loop:utterance` message.

**Reset:** The `reset()` method clears internal audio buffers, called after STT completes.

## Plugin Registration

In your `setup.py`, register the plugin entry point:

```
entry_points={
    'ovos.plugin.audio_transformer': [
        'my-custom-audio-transformer = my_module:MyCustomAudioTransformer'
    ]
}
```

## Configuration Example

Add your transformer to `mycroft.conf`:

```
"audio_transformers": {
    "my-custom-audio-transformer": {
        // plugin-specific config options here
    }
}
```

## docs/converse\_pipeline.md

## Converse Pipeline

The **Converse Pipeline** in **OpenVoiceOS (OVOS)** manages active conversational contexts between the assistant and skills. It allows skills to keep handling user input across multiple turns, enabling more natural, stateful conversations.

---



# Purpose

The **Converse pipeline** enables **multi-turn conversations** by prioritizing which skills are given the opportunity to handle an utterance through their `converse()` method before normal intent parsing occurs.

Key purposes include:

- Preserve conversational context** across multiple turns.

- Prioritize recently used skills** for more natural interactions.

- Enable stateful behavior**, such as follow-up questions or corrections.

- Prevent unnecessary intent parsing** when a skill is already engaged.

- Support skill-defined session control** via manual activation/deactivation.

This allows OVOS to act more like a true conversational assistant rather than a single-turn command system.

---

# Active Skill List

A Skill is considered active if it has been called in the last 5 minutes.

Skills are called in order of when they were last active. For example, if a user spoke the following commands:

> Hey Mycroft, set a timer for 10 minutes > > Hey Mycroft, what's the weather

Then the utterance "what's the weather" would first be sent to the Timer Skill's `converse()` method, then to the intent service for normal handling where the Weather Skill would be called.

As the Weather Skill was called it has now been added to the front of the Active Skills List. Hence, the next utterance received will be directed to:

1. `WeatherSkill.converse()`
2. `TimerSkill.converse()`
3. Normal intent parsing service

When does a skill become active?

1. **before** an intent is called the skill is **activated**
2. if a fallback **returns True** (to consume the utterance) the skill is **activated** right **after** the fallback
3. if converse **returns True** (to consume the utterance) the skill is **reactivated** right **after** converse
4. a skill can activate/deactivate itself at any time

---

## Pipeline Stages

This plugin registers a single pipeline:

| Pipeline ID           | Description                  | Recommended Use  |
|-----------------------|------------------------------|--|
|                       |                              |  |
| -----                 | -----                        | -----  |
| -----                 |                              |  |
| <code>converse</code> | Continuous dialog for skills | Should always be present, do not remove unless you know what you are doing |

---

## How It Works

When a user speaks, the pipeline checks if any skill is actively conversing.

Active skills implement a `converse()` method that determines if they want to handle the utterance.

If no active skill accepts the input, the regular intent matching process continues.

Skills can automatically deactivate after a timeout or based on custom logic.

Only a limited number of skills can be active at any given time (defaults configurable).

---

# Skill Integration

Skills integrate with the converse pipeline by:

Implementing a `converse()` method that checks if the skill wants to handle an utterance.

Returning `True` if the utterance was handled, `False` otherwise.

Managing internal state to determine when to exit conversation mode.

This enables modular, stateful conversations without hardcoding turn-taking logic into the core assistant.

---

## Configuration

Customize the pipeline via `mycroft.conf`:

```
"skills": {
  "converse": {
    "cross_activation": true,
    "converse_activation": "accept_all",
    "converse_mode": "accept_all",
    "converse_blacklist": [],
    "converse_whitelist": [],
    "max_activations": 3,
    "skill_activations": {
      "skill-example": 5
    },
    "timeout": 300,
    "skill_timeouts": {
      "skill-example": 600
    },
    "max_skill_runtime": 10
  }
}
```

### Key Options

| Config Key | Description |
|------------|-------------|
|------------|-------------|

```

|| ----- |
----- ||
cross_activation | Allow skills to activate/deactivate other skills during a
conversation. || converse_mode | Global mode for allowing/disallowing
skills from converse participation. || converse_blacklist | Skills that are
not allowed to enter converse mode. || converse_whitelist
| Skills explicitly allowed to converse. ||
max_activations | Default number of times a skill can consecutively handle
turns. || skill_activations | Per-skill override of
max_activations. || timeout | Time
(in seconds) before an idle skill is removed from converse mode. ||
skill_timeouts | Per-skill override of timeout.
|| max_skill_runtime | Maximum time (in seconds) to wait for a skills
converse() response. |
---

```

## Converse Modes

```

| Mode | Description
|| ----- |
----- ||
accept_all | All skills are allowed to use converse mode (default).
|| whitelist | Only skills explicitly listed in converse_whitelist can use
converse mode. || blacklist | All skills can use converse mode except those in
converse_blacklist. |
---

```

## Security & Performance

A malicious or badly designed skill using the converse method can potentially hijack the whole conversation loop and render the skills service unusable

Because skills can "hijack" the conversation loop indefinitely, misbehaving or malicious skills can degrade UX. Protections include:

- Timeouts for inactivity and maximum runtime.

- `max_activations` limits per skill.

- Blacklist/whitelist enforcement to restrict which skills can enter converse

mode.

`cross_activation` can be disabled to prevent skill-to-skill manipulation.

The concept of "converse priority" is under active development

```

"skills": { // if true any skill can activate any other skill
  "cross_activation": true,
  // converse stage configuration
  "converse": {
    // if false only skills can deactivate themselves
    // the default number of seconds a skill remains active,
    // if true any skill can deactivate any other skill
    // if the user does not interact with the skill in this timespan it
    // NOTE: skill deactivation is not yet implemented
    // will be deactivated, default 5 minutes (same as mycroft)
    "cross_deactivation": true,
    "timeout": 300,

    // override of "skill_timeouts" per skill_id
    // you can add skill_id: priority to override the developer defined
    // you can configure specific skills to remain active longer
    // priority of those skills,
    "skill_timeouts": {},

    // converse priority is work in progress and not yet exposed to skills
    // conversational mode has 3 modes of operations:
    // - "accept_all" # default mycroft-core behavior
    // the only current source for converse priorities is this setting
    // - "whitelist" # only call converse for skills in "converse_whitelist"
    "converse_priorities": {
      // - "blacklist" # only call converse for skills NOT in "converse_blacklist"
      // "skill_id": 10
      "converse_mode": "accept_all",
    }
    "converse_whitelist": [],
  }
  "converse_blacklist": [],
},

// converse activation has 4 modes of operations:
// - "accept_all" # default mycroft-core behavior, any skill can
// # activate itself unconditionally
// - "priority" # skills can only activate themselves if no skill with
// # higher priority is active
// - "whitelist" # only skills in "converse_whitelist" can activate themselves
// - "blacklist" # only skills NOT in converse "converse_blacklist" can activate
// NOTE: this does not apply for regular skill activation, only to skill
// initiated activation requests, eg, self.make_active()
"converse_activation": "accept_all",

// number of consecutive times a skill is allowed to activate itself
// per minute, -1 for no limit (default), 0 to disable self-activation
"max_activations": -1,

// override of "max_activations" per skill_id
// you can configure specific skills to activate more/less often
"skill_activations": {},

// if false only skills can activate themselves

```

---

## Notes

The plugin **does not enforce a fallback behavior** if no skill accepts the input.

If no skill handles the utterance via converse, the pipeline falls back to normal intent matching or fallback skills.

This mechanism is ideal for multi-turn conversations like dialogs, games, or assistant flows that require memory of previous input.

**docs/cq\_pipeline.md**

# Common Query Pipeline

The **Common Query Pipeline Plugin** in OVOS is a specialized pipeline component designed **exclusively for handling general knowledge questions**. It processes utterances that resemble questions typically starting with interrogatives like *what*, *who*, *how*, *when*, etc. and queries a set of registered general knowledge skills to find the most accurate factual answer.

Unlike conversational or chit-chat pipelines, this plugin focuses strictly on **fact-based question answering**. It does **not** generate answers or perform retrieval-augmented generation (RAG). Instead, it relies on a **reranker module** to evaluate candidate answers from all queried skills and selects the most relevant and factually accurate response.

---

## Purpose

**Handle only question-like utterances** (e.g., What is the tallest mountain?, Who wrote Hamlet?).

Query multiple **general knowledge skills** to obtain candidate answers.

Use a **reranker mechanism** to evaluate and select the most confident and factually accurate response.

Provide a robust fallback for answering factual queries outside of high-confidence intent matches.

---

## Pipeline Stages

This plugin registers a single pipeline:

| Pipeline ID | Description          | Recommended Use                                     |
|-------------|----------------------|---|
|             |                      |   |
| -----       | -----                | -----   |
| -----       |                      |   |
| common_qa   | Common Query matches | Only as good as the common query skills you install |



---

## How It Works

1. **Question Detection:** The pipeline filters incoming utterances to only process those that appear to be questions, based on interrogative keywords.
2. **Parallel Skill Querying:** The plugin sends the utterance to all registered common query skills capable of answering general knowledge questions.
3. **Candidate Collection:** Each skill returns zero or more candidate answers along with confidence scores.
4. **Reranking:** A reranker component evaluates all candidate answers across skills to identify the best response, focusing on factual accuracy and confidence.
5. **Answer Delivery:** If a suitable answer is found, it is returned to the user; otherwise, the query is passed on or marked as unanswered.

---

## Installation

The Common Query Pipeline Plugin is included by default in `ovos-core`, but can also be installed independently:

```
pip install ovos-common-query-pipeline-plugin
```

---

## Configuration

```
"intents": {
  "common_query": {
    "min_self_confidence": 0.5,
    "min_reranker_score": 0.5,
    "reranker": "ovos-flashrank-reranker-plugin",
    "ovos-flashrank-reranker-plugin": {
      "model": "ms-marco-TinyBERT-L-2-v2"
    }
  }
}
```

**min\\_self\\_confidence:** Minimum confidence required from the skill answer itself before reranking.

**min\\_reranker\\_score:** Minimum reranker score threshold to accept an answer.

**reranker:** The reranker plugin to use (must be installed separately).

**Model:** Choose a suitable reranker model based on accuracy and device constraints.

---

## Performance Considerations

The plugins response time depends on the slowest queried skill the latency of installed common query skills affects overall speed.

Enabling rerankers, especially on resource-limited hardware (e.g., Raspberry Pi), may add noticeable latency.

Timeout (default 2 seconds) ensures responsiveness but might cause some slow skill answers to be discarded.

Tune confidence thresholds and reranker settings according to your hardware capabilities and user experience goals.

---

## Example Usage Scenario

User says: *"When was the Declaration of Independence signed?"*

The utterance is detected as a question.

The plugin queries `ovos-skill-wolfram-alpha` and `ovos-skill-wikipedia`.

Each skill returns candidate answers with confidence scores.

The reranker evaluates answers and selects the most reliable response.

The selected answer is delivered back to the user.

---

## Important Notes

**No generation or RAG:** The plugin only retrieves answers from skills; it does not generate or synthesize new content.

**No chit-chat:** This pipeline is strictly for general knowledge queries, **not for casual conversation or small talk**.

**Reranker-based selection:** The reranker improves the quality of responses by ranking answers from multiple sources.

**Skills required:** Ensure that relevant common query skills (e.g., knowledge bases, encyclopedias) are installed and enabled.

## docs/dialog\_transformers.md

## Dialog Transformers

**Dialog Transformers** in OpenVoiceOS (OVOS) are plugins that modify or enhance text responses just before they are sent to the Text-to-Speech (TTS) engine. This allows for dynamic adjustments to the assistant's speech, such as altering tone, simplifying language, or translating content, without requiring changes to individual skills.

---

# How They Work

1. **Intent Handling:** After a user's utterance is processed and an intent is matched, the corresponding skill generates a textual response.
2. **Transformation Phase:** Before this response is vocalized, it passes through any active dialog transformers.
3. **TTS Output:** The transformed text is then sent to the TTS engine for audio synthesis.

This pipeline ensures that all spoken responses can be uniformly modified according to the desired transformations.

---

## Configuration

To enable dialog transformers, add them to your `mycroft.conf` file under the `dialog_transformers` section:

```
"dialog_transformers": {  
  "plugin_name": {  
    // plugin-specific configuration  
  }  
}
```

Replace `"plugin_name"` with the identifier of the desired plugin and provide any necessary configuration parameters.

---

## Available Dialog Transformer Plugins

### OVOS Dialog Normalizer Plugin

**Purpose:** Prepares text for TTS by expanding contractions and converting digits to words, ensuring clearer pronunciation.

**Example:**

Input: "I'm 5 years old."

Output: "I am five years old."

**Installation:**

```
pip install ovos-dialog-normalizer-plugin
```

**Configuration:**

```
"dialog_transformers": {  
  "ovos-dialog-normalizer-plugin": {}  
}
```

**Source:** [GitHub Repository](#)

---

## OVOS OpenAI Dialog Transformer Plugin

**Purpose:** Utilizes OpenAI's API to rewrite responses based on a specified persona or tone.

**Example:**

Rewrite Prompt: "Explain like I'm five"

Input: "Quantum mechanics is a branch of physics that describes the behavior of particles at the smallest scales."

Output: "Quantum mechanics helps us understand really tiny things."

**Installation:**

```
pip install ovos-openai-plugin
```

**Configuration:**

```
"dialog_transformers": {
  "ovos-dialog-transformer-openai-plugin": {
    "rewrite_prompt": "Explain like I'm five"
  }
}
```

**Source:** [GitHub Repository](#)

---

## OVOS Bidirectional Translation Plugin

**Purpose:** Translates responses to match the user's language, enabling multilingual interactions.

**Features:**

Detects the language of the user's input.

Works together with a companion utterance transformer plugin

Translates the assistant's response back into the user's language.

**Installation:**

```
pip install ovos-bidirectional-translation-plugin
```

**Configuration:**

```
"dialog_transformers": {
  "ovos-bidirectional-dialog-transformer": {
    "bidirectional": true
  }
}
```

**Source:** [GitHub Repository](#)

---

## Creating Custom Dialog Transformers

To develop your own dialog transformer:

### Create a Python Class:

```
from ovos_plugin_manager.templates.transformers import DialogTransformer

class MyCustomTransformer(DialogTransformer):
    def __init__(self, config=None):
        super().__init__("my-custom-transformer", priority=10, config=config)

    def transform(self, dialog: str, context: dict = None) -> Tuple[str, dict]:
        """
        Optionally transform passed dialog and/or return additional context
        :param dialog: str utterance to mutate before TTS
        :returns: str mutated dialog
        """
        # Modify the dialog as needed
        return modified_dialog, context
```

**Register as a Plugin:** In your setup.py, include:

```
entry_points={
    'ovos.plugin.dialog_transformer': [
        'my-custom-transformer = my_module:MyCustomTransformer'
    ]
}
```

**Install and Configure:** After installation, add your transformer to the mycroft.conf:

```
"dialog_transformers": {
    "my-custom-transformer": {}
}
```

## docs/fallback\_pipeline.md

## Fallback Pipeline

The **Fallback Pipeline** in **OpenVoiceOS (OVOS)** manages how fallback skills are queried when no primary skill handles a users utterance. It coordinates multiple fallback handlers, ensuring the system gracefully attempts to respond even when regular intent matching fails.

---

## Pipeline Stages

| Pipeline ID                   | Priority Range                       | Description                     | Use Case |
|-------------------------------|--------------------------------------|---------------------------------|----------|
| -----                         | -----                                | -----                           | -----    |
| -----                         | fallback_high                        | 0 5                             |          |
| High-priority fallback skills | Critical fallback handlers           |                                 |          |
| fallback_medium               | 5 90                                 | Medium-priority fallback skills |          |
| General fallback skills       | fallback_low                         | 90 101                          |          |
| Low-priority fallback skills  | Catch-all or chatbot fallback skills |                                 |          |

Fallback skills register with a priority, allowing the pipeline to query them in order.

---

## How It Works

When no regular skill handles an utterance, the fallback pipeline queries registered fallback skills asynchronously.

Each fallback skill can decide whether to handle the utterance.

Fallback skills are tried by priority level (can be overridden by users)

If no fallback skill accepts the utterance, no fallback response is generated by the pipeline itself.

---

## Skill Integration

Skills integrate as fallbacks by:

Registering on the message bus with a fallback priority.



Listening for fallback queries carrying all utterance variations.

Responding with success/failure on whether they handled the fallback.

This enables modular and customizable fallback behavior depending on your skill ecosystem.

---

## Notes

The pipeline itself **does not define or enforce a default fallback response**

.

The default "I dont understand" reply is implemented in the separate `ovos-skill-fallback-unknown` skill.

This modular design allows developers to create custom fallback strategies or add fallback chatbot skills without modifying the core pipeline.

Fallback skills are expected to implement some dialog if they consume the utterance

---

## Security

Just like with converse a badly designed or malicious skill can hijack the fallback skill loop, while this is not as serious as with converse some protections are also provided

You can configure what skills are allowed to use the fallback mechanism, you can also modify the fallback priority to ensure skills behave well together.

Since priority is defined by developers sometimes the default value is not appropriate and does not fit well with the installed skills collection

```

"skills": {
  // fallback skill configuration
  "fallbacks": {
    // you can add skill_id: priority to override the developer defined
    // priority of those skills, this allows customization
    // of unknown intent handling for default_skills + user preferences
    "fallback_priorities": {
      // "skill_id": 10
    },
    // fallback skill handling has 3 modes of operations:
    // - "accept_all" # default mycroft-core behavior
    // - "whitelist" # only call fallback for skills in "fallback_whitelist"
    // - "blacklist" # only call fallback for skills NOT in "fallback_blacklist"
    "fallback_mode": "accept_all",
    "fallback_whitelist": [],
    "fallback_blacklist": []
  }
},

```

## docs/gitlocalize\_tutorial.md

### Contribute to Translations with GitLocalize!

Thank you for your interest in helping translate our project! Your contributions will help make our project accessible to more people around the world.

<iframe width="560" height="315" src="https://www.youtube.com/embed/2udvdIW9W2s" title="OVOS translation guide" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" referrerpolicy="strict-origin-when-cross-origin" allowfullscreen></iframe>

Weve made it easy for you to get started, even if youre not familiar with GitHub or coding. Follow the steps below to join our translation effort using GitLocalize.

### Step-by-Step Guide to Translating with

# GitLocalize

## Visit Our GitLocalize Project Page



1. **Click on the link** to our GitLocalize project:  
<https://gitlocalize.com/users/OpenVoiceOS>
2. You will see a list of OVOS repositories to translate, select one
3. You will see a list of languages and translation tasks available.



## Sign Up or Log In

1. If you dont have an account, **sign up** with your email or GitHub account (you dont need to know GitHub to do this!).
2. If you already have an account, simply **log in**.

## Choose a Language

> Adding new languages to the list is a manual process, if your language is unlisted let us know!

1. Select the language you want to translate into from the list of available languages.
2. You will see a list of files that need translation.
3. `dialogs.json` contains sentences that OVOS will speak
4. `intents.json` contains sentences that the user will speak to OVOS
5. `vocabs.json` similar to intents, but contain sentence fragments/keywords, not full utterances



## Start Translating

1. Click on a file that you want to translate.

2. The translation editor will open. Here, you'll see the original text on the left and a space to enter your translation on the right.
3. Begin translating the text. If you're unsure about any phrase, feel free to leave it and move on to the next one.

When you open a JSON file for translation in GitLocalize, you'll see two parts:

**Key:** This corresponds to a file name in the OVOS repository you selected.

**Value:** This is the sentence you need to translate.

## Variables

Variables are placeholders within sentences that represent changing content, such as names or numbers.

Original: My name is {var\_name}

Translation: Mi nombre es {var\_name}

[!image](#)

### Important Rules:

**Do not translate the variable names** (the text inside curly braces { }).

You can **rearrange the position** of variables in your translation, but do not create new variables.

Ensure that variables are not separated by only whitespace; there should be at least one word between them.

## Slots

Sometimes, the same file will appear several times, each with a different variation of the same sentence. These variations are called "slots".

[!image](#)

### Important Rules:

**Translate at least one slot** in each file.

If a slot is not needed in your language, enter [UNUSED]. This tells us that you reviewed the slot and marked it as translated.

If you run out of slots to fit all variations of a sentence, you can use newlines to add more translations.

[!image](#)

## Alternative/Optional words

You can use the "alternative word" syntax to provide options or optional words within a sentence.

Alternative words: I love (cats|dogs|birds) becomes Amo  
(gatos|perros|pajaros)

Optional words: I (really|) love (cats|dogs|birds) becomes Yo  
(realmente|) amo (gatos|perros|pajaros)

[!image](#)

## Tips for Effective Translation

**Consistency:** Try to use consistent terminology throughout the project.

**Context:** If a phrase seems unclear, consider the overall context of the project or reach out for clarification.

**Accuracy:** Aim to convey the meaning as accurately as possible, rather than a literal word-for-word translation.

## Key Take Aways

For each sentence (slot), enter your translation.

If a slot is not needed, enter [UNUSED].

Leave the variable names in curly braces {} unchanged.

Rearrange variables as needed but do not create new ones.

Provide multiple options using the syntax (option1|option2|option3).

Include optional words using the syntax (optional|).

If there are not enough slots, press `Enter` to add a new line and enter your

alternative translation on the new line.

## Review and Feedback

1. Once you've finished translating a file, you can **submit** it for review.
2. Your translations will be reviewed by other native speakers and project maintainers.
3. If any changes are needed, you might receive feedback. Simply log back in, review the comments, and make the necessary adjustments.

[!image](#)

## Need Help?

If you have any questions or need assistance at any point:

**Join our Matrix chat:** <https://matrix.to/#/#openvoiceos-languages:matrix.org>

**Email us:** [support@openvoiceos.org](mailto:support@openvoiceos.org)

## Thank You!

Your contributions are invaluable, and we appreciate your effort in helping us reach a global audience. Happy translating!

## docs/index.md

# The OpenVoiceOS Technical Manual

!

Welcome to the **Open Voice OS (OVOS)** developer documentation. This guide is your starting point for exploring, building, and contributing to OVOS – an open and modular platform for voice-enabled applications.

---

## What is OVOS?

**Open Voice OS (OVOS)** is a flexible voice platform that goes beyond traditional voice assistants. It provides the foundational tools and frameworks for integrating voice interaction into a wide range of projects.

While OVOS can power a Hey Mycroft...-style assistant, it is not limited to that use case. As a voice operating system, OVOS is highly customizable and has been used in:

Robots and automation systems

Smart furniture and mirrors

Cloud-based voice services

Embedded devices and smart TVs

OVOS is designed to work wherever voice interfaces are needed whether that's on a local device or in the cloud.

---

## OVOS Distro

If you'd rather not install and configure components manually, OVOS has several prebuilt distributions:

**`raspOVOS`**: A Raspberry Pi image with pre-installed OVOS services.

**`ovos-installer`**: Installs OVOS on top of supported Linux systems.

**`ovos-buildroot`**: A highly customizable buildroot-based image (in development).

These distros offer a faster setup experience with curated default skills and settings.

---

## What You'll Find in This Manual

This documentation includes:

**Architecture Overview** A breakdown of how OVOS components work together

**Plugin System** Details on STT, TTS, wake word engines, GUI backends, and

more

**Application Development** How to create and deploy custom voice-enabled applications

**Testing and Debugging** Tools and practices for developing with OVOS

---

## Related Projects and External Resources

**Docker Setup:** [OVOS Docker Documentation](#)

**Message Bus Reference:** [OVOS Message Spec](#)

**Remote Client Framework:** [HiveMind Documentation](#)

---

## Contributing to This Documentation

This manual is maintained by the OVOS community developers, users, and contributors who help shape the platform.

Whether you're spotting errors, clarifying instructions, or adding new information, contributions are always welcome.

To contribute, please [open an issue or pull request](#) on GitHub.

---

## Getting Started Tips

OVOS is modular you can run just one or two components to start.

Try a pre-built distro for a plug-and-play experience.

Check the message bus reference to see how OVOS components communicate.

Explore real-world examples from the community to see OVOS in action.

# docs/intent\_transformers.md



# Intent Transformers

**Intent Transformers** are a pluggable mechanism in OVOS that allow you to enrich or transform intent data **after** an intent is matched by an engine (Padatious, Adapt, etc.), but **before** it is passed to the skill handler.

This is useful for:

- Named Entity Recognition (NER)

- Keyword extraction

- Slot filling

- Contextual enrichment

Transformers operate on `IntentHandlerMatch` or `PipelineMatch` objects and are executed **in order of priority**. They enable complex processing pipelines without requiring every skill to reimplement entity logic.

---

## Default Transformers

In a standard OVOS installation, the following plugins are **installed and enabled by default**:

| Plugin   | Description   |
|--|---|
| Priority   |   |
| -----  |   |
| -----  |   |
| -----    ovos-keyword-template-matcher   | Extracts values from {placeholder}-style intent templates   1 |
| ovos-ahocorasick-ner-plugin  |   |
| Performs NER using Aho-Corasick keyword matching based on registered entities from skill templates   5 |   |

These are **not built into core**, but are bundled in standard OVOS setups and configured via `intent_transformers` in your configuration file.

---

## Configuration

To enable or disable specific transformers, modify your `mycroft.conf`:

```
"intent_transformers": {  
  "ovos-keyword-template-matcher": {  
    "active": true  
  },  
  "ovos-ahocorasick-ner-plugin": {  
    "active": false  
  }  
}
```

---

## How It Works

### Example Workflow

1. An utterance matches an intent via Padatious, Adapt, or another engine.
2. The matched intent is passed to the `IntentTransformersService`.
3. Each registered transformer plugin runs its `transform()` method.
4. Extracted entities are injected into the intents `match_data`.
5. The updated `match_data` is passed to the skill via the `Message` object.

### Skill Access

Entities extracted by transformers are made available to your skill in the `message.data` dictionary:

```
location = message.data.get("location")  
person = message.data.get("person")
```

---

## Default Plugins

## ovos-ahocorasick-ner-plugin

This plugin builds a per-skill Aho-Corasick automaton using keywords explicitly provided by the developer via registered entities.

> It will **only match keywords that the skill developer has accounted for**

It does **not** use external data or extract entities generically.

---

## ovos-keyword-template-matcher

This plugin parses registered intent templates like:

```
what's the weather in {location}
```

It uses the template structure to extract `{location}` directly from the utterance.

If the user says "what's the weather in Tokyo", the plugin will populate:

```
match_data = {  
    "location": "Tokyo"  
}
```

---

## Writing Your Own Intent Transformer

To create a custom transformer:

```
from ovos_plugin_manager.templates.transformers import IntentTransformer  
  
class MyCustomTransformer(IntentTransformer):  
    def __init__(self, config=None):  
        super().__init__("my-transformer", priority=10, config=config)  
  
    def transform(self, intent):  
        # Modify intent.match_data here  
        return intent
```

# docs/lang\_support.md

## Language Support in OpenVoiceOS

OpenVoiceOS (OVOS) aims to support multiple languages across its components, including intent recognition, speech-to-text (STT), text-to-speech (TTS), and skill dialogs. However, full language support requires more than translation of interface text. This document outlines the current state of language support, known limitations, and how contributors can help improve multilingual performance in OVOS.

---

While the OVOS installer allows users to select a preferred language, **selecting a language does not guarantee full support across all subsystems**. True multilingual support requires dedicated:

- Translations (intents, dialogs, settings, etc.)

- STT (Speech-to-Text) plugins trained on the target language

- TTS (Text-to-Speech) plugins capable of generating speech in the selected language

- Language-specific intent adaptation and fallback logic

Without these, many core features (e.g., voice commands, speech output, skill interactions) may not function as expected.

---

## Adding a New Language

Adding support for a new language in OVOS is a multi-step process requiring:

- Translations of assistant dialog and intent files

- A compatible STT plugin with reliable speech recognition

- A natural-sounding TTS voice

- Validation using real-world user data

We welcome and encourage community participation to improve language support. Every contribution helps make OVOS more accessible to speakers around the world.

---

## STT and TTS Requirements

For a language to function correctly in a voice assistant environment, it must have **dedicated STT and TTS plugins** that support the language reliably.

### STT (Speech-to-Text)

STT plugins must be able to recognize speech in the target language with high accuracy.

Some plugins are multilingual (e.g., Whisper, MMS), but accuracy varies across languages.

For production use, **language-specific tuning or models are recommended**.

### TTS (Text-to-Speech)

The TTS engine must generate clear, natural-sounding speech in the selected language.

Not all TTS plugins support all languages.

Quality varies significantly by model and backend.

A list of early TTS and STT plugins test with per-language accuracy benchmarks is available at:

[STT Bench](#)

[TTS Bench](#)

---

## Translation Coverage

OVOS uses [GitLocalize](#) for managing translation files across its repositories. This includes:

Skill dialog files

Intent files (used by Padatious/Adapt)

Configuration metadata

## Translation Progress

Translation progress is tracked at:

<https://openvoiceos.github.io/lang-support-tracker>

The tracker provides daily updates and displays all languages that have reached at least 25% translation coverage.

> If your language is missing from GitLocalize, [open an issue](#) to request it. Currently, languages must be added manually.

---

## Known Limitations

Selecting a language during installation only automatically configures a compatible STT/TTS plugin for **some languages**. Manual action might be required for full support

Many skills contain only partial translations or outdated strings.

Skills may be partially translated, with only a subset of intents available for your language

Skills may have translated intents but missing dialog translations. The assistant typically speaks the dialog filename if it is not translated

---

## How to Improve Language Support

### 1. Contribute Translations

Use GitLocalize to translate dialog and intent files:

[GitLocalize for OVOS](#)

## [Translation Tutorial](#)

Translation stats for each language are also available in:

Markdown summaries (e.g., 'translate\_status\_pt.md')

[JSON format](#) (e.g., 'pt-PT.json')

---

## 2. Test in Real-World Usage

Translation coverage alone does not ensure accuracy. Native speakers are encouraged to test OVOS with real speech input and report issues with:

Intent matching failures

Mispronunciations or robotic speech

Incorrect or unnatural translations

You can help by **enabling open data collection** in your OVOS instance:

```
"open_data": {  
  "intent_urls": [  
    "https://metrics.tigregotico.pt/intents"  
  ]  
}
```

> Alternatively, you may self-host the reporting server: [ovos-opendata-server](#) on [GitHub](#)

## Monitoring Tools

Live Data Dashboard: <https://opendata.tigregotico.pt>

Server Status: <https://metrics.tigregotico.pt/status>

---

## Benchmark Projects (Open Data)

Explore public benchmark tools for evaluating model performance:

| Project                           | Description  |
|-----------------------------------|--|
| <a href="#">GitLocalize Bench</a> | Evaluate intent translation coverage and performance   |
| <a href="#">STT Bench</a>         | Test STT plugin accuracy across datasets and languages |
| <a href="#">TTS Bench</a>         | Compare TTS output quality across plugins              |
| <a href="#">Meteocat</a>          | Catalan weather query benchmark                        |

---

## Tips for Contributors

**Translators:** Use GitLocalizes side-by-side editor to keep intent logic intact.

**Developers:** Review user-submitted errors on the dashboard to improve skill performance.

**Curious users:** Explore benchmark results to see how well OVOS handles your language.

## docs/m2v\_pipeline.md

### Model2Vec Intent Pipeline

The **Model2Vec Intent Pipeline** is an advanced plugin for OpenVoiceOS, designed to enhance intent classification using pretrained Model2Vec models. By leveraging vector-based representations of natural language, this pipeline offers improved accuracy over traditional deterministic engines, especially in scenarios where intent recognition is challenging.

---

### Features

**Model2Vec-Powered Classification:** Uses pretrained Model2Vec models for rich vector-based intent understanding.

**Seamless OVOS Integration:** Plug-and-play compatibility with existing OVOS



intent pipelines.

**Multilingual & Language-Specific Models:** Offers large multilingual models distilled from LaBSE and smaller, efficient language-specific models ideal for limited hardware (e.g., Raspberry Pi).

**Dynamic Intent Syncing:** Automatically synchronizes with Adapt and Padatious intents at runtime.

**Skill-Aware Matching:** Classifies *only* official OVOS skill intents, reducing false positives by ignoring unregistered or personal skill intents.

**Supports Partial Translations:** Multilingual models allow usage of partially translated skills, provided their **dialogs** are translated.

---

## Installation

Install the plugin via pip:

```
pip install ovos-m2v-pipeline
```

---

## Configuration

Configure the plugin in your `mycroft.conf` file:

```
{
  "intents": {
    "ovos-m2v-pipeline": {
      "model": "Jarbas/ovos-model2vec-intents-LaBSE",
      "conf_high": 0.7,
      "conf_medium": 0.5,
      "conf_low": 0.15,
      "ignore_intents": []
    },
    "pipeline": [
      "converse",
      "ovos-m2v-pipeline-high",
      "padatious_high",
      "fallback_low"
    ]
  }
}
```

### Parameters:

`model`: Path to the pretrained Model2Vec model or Hugging Face repository.

`conf_high`: Confidence threshold for high-confidence matches (default: 0.7).

`conf_medium`: Confidence threshold for medium-confidence matches (default: 0.5).

`conf_low`: Confidence threshold for low-confidence matches (default: 0.15).

`ignore_intents`: List of intent labels to ignore during matching.

> **Note:** Model2Vec models are pretrained and *do not* dynamically learn new skills at runtime.

---

## How It Works

1. Receives a user utterance as text input.
2. Predicts intent labels using the pretrained Model2Vec embedding model.

3. Filters out any intents *not* associated with currently loaded official OVOS skills.
4. Returns the highest-confidence matching intent.

This process enhances intent recognition, particularly in cases where traditional parsers like Adapt or Padatious may struggle.

---

## Models Overview

**Multilingual Model:** Over 500MB, distilled from LaBSE, supports many languages and partially translated skills.

**Language-Specific Models:** Smaller (~10x smaller than multilingual), highly efficient, almost as accurate ideal for devices with limited resources.

Models can be specified via local paths or Hugging Face repositories: [OVOS Model2Vec Models on Hugging Face](#)

---

## Training Data

The Model2Vec intent classifier is trained on a diverse, aggregated collection of intent examples from:

OVOS LLM Augment Intent Examples synthetic utterances generated by large language models for OVOS skills.

Music Query Templates focused on music-related intents.

Language-Specific Skill Intents extracted CSV files from OpenVoiceOS GitLocalize covering English, Portuguese, Basque, Spanish, Galician, Dutch, French, German, Catalan, Italian, and Danish.

Models are regularly updated with new data to improve performance and language coverage.

---

## Important Usage Notes

**Official OVOS Skills Only:** The Model2Vec pipeline classifies intents *only* from official OVOS skills. For personal or custom skills, you should continue to use Adapt and Padatious parsers alongside Model2Vec.

**Complementary Pipeline:** Model2Vec is designed to *augment* your intent pipeline, not replace Adapt or Padatious. Using all three together provides the best overall recognition.

**Padatious Intent Data & Training:** Padatious intent data and example utterances are available in [GitLocalize](#) for translations and new model training. The Model2Vec models are continuously updated with this data.

**Language Support:** The multilingual model (500MB+) supports many languages and works well with partially translated skills, as long as **dialogs** are localized.

**Optimization:** Language-specific models are on average 10x smaller and nearly as accurate as the multilingual model, making them ideal for constrained hardware or single-language setups.

## docs/padatious\_pipeline.md

### Padatious Pipeline

The **Padatious Pipeline Plugin** brings examples-based intent recognition to the **OpenVoiceOS (OVOS)** pipeline. It enables developers to define intents using example sentences, offering a simple and code-free way to create natural language interfaces for voice skills.

---

### Pipeline Stages

This plugin registers the following pipeline stages:

| Pipeline ID              | Description                         | Recommended Use |
|--------------------------|-------------------------------------|-----------------|
| -----                    | -----                               |                 |
| -----                    | padatious_high                      | High-confidence |
| Padatious intent matches | Primary stage for Padatious use     |                 |
| padatious_medium         | Medium-confidence Padatious matches | Backup if       |

confidence tuning allows || `padatious_low` | Low-confidence Padatious matches | Not recommended (often inaccurate) |

Each stage is triggered based on the confidence level of the parsed intent, as configured in your system.

---

## Configuration

Configure Padatious thresholds in your `ovos.conf`:

```
"intents": {
  "padatious": {
    "conf_high": 0.85,
    "conf_med": 0.65,
    "conf_low": 0.45
  }
}
```

These thresholds control which pipeline level receives a given intent result.

---

## Multilingual Support

Padatious is **excellent for multilingual environments** because intents are defined in plain text `.intent` files, not in code. This allows translators and non-developers to contribute new languages easily without touching Python.

To add another language, simply create a new `.intent` file in the relevant language folder, such as:

```
locale/pt-pt/weather.intent
locale/fr-fr/weather.intent
```

---

## Defining Intents

Intent examples are written line-by-line in `.intent` files:

```
what is the weather
tell me the weather
what's the weather like
```

In your skill:

```
from ovos_workshop.decorators import intent_handler

@intent_handler("weather.intent")
def handle_weather(self, message):
    # Your code here
    pass
```

---

## Limitations

Padatious is reliable in terms of **not misclassifying** it rarely picks the wrong intent. However, it has key limitations:

**Weak paraphrase handling:** If the user speaks a sentence that doesn't closely match an example, Padatious will often fail to match anything at all.

**Rigid phrasing required:** You may end up in a train the user to speak correctly scenario, instead of training the system to understand variations.

**Maintenance burden for sentence diversity:** Adding more phrasing requires adding more sentence examples per intent, increasing effort and clutter.

---

## When to Use

Padatious is a good choice in OVOS when:

You want **easy localization/multilingual support**.

You're creating **simple, personal, or demo skills**.

You can **control or guide user phrasing**, such as in kiosk or assistant environments.

Avoid Padatious for complex conversational use cases, skills with overlapping intents, or scenarios requiring broad paraphrasing support.

## docs/pipelines\_overview.md

# OVOS Intent Pipeline

The OpenVoiceOS (OVOS) Intent Pipeline is a modular and extensible system designed to interpret user utterances and map them to appropriate actions or responses.

It orchestrates various intent parsers and fallback mechanisms to ensure accurate and contextually relevant responses.

---

## What is an Intent Pipeline?

An intent pipeline in OVOS is a sequence of processing stages that analyze user input to determine the user's intent. Each stage employs different strategies, ranging from high-confidence intent parsers to fallback mechanisms, to interpret the input.

This layered approach ensures that OVOS can handle a wide range of user queries with varying degrees of specificity and complexity.

---

## Pipeline Structure

OVOS pipelines are structured to prioritize intent matching based on confidence levels:

**High Confidence:** Primary intent parsers that provide precise matches.

**Medium Confidence:** Secondary parsers that handle less specific queries.

**Low Confidence:** Fallback mechanisms for ambiguous or unrecognized inputs.

Each component in the pipeline is a plugin that can be enabled, disabled, or reordered according to user preferences. This flexibility allows for

customization based on specific use cases or device capabilities.

---

## Available Pipeline Components

Below is a list of available pipeline components, categorized by their confidence levels and functionalities:

### High Confidence Components

| Pipeline  | Description   |
|---|---|
| Notes   |   |
| ----- -----   |   |
| ----- -----   |   |
| -----   |   |
| stop_high   | Exact match for stop commands   |
| Replaces <a href="#">skill-ovos-stop</a>   converse | Continuous conversation interception for skills   |
| padatious_high                                      | High-confidence matches using Padatious   |
|   |   |
| adapt_high  | High-confidence matches using Adapt   |
|   |   |
| fallback_high                                       | High-priority fallback skill matches  |
|   |   |
| ocp_high  | High-confidence media-related queries   |
|   |   |
| ovos-persona-pipeline-plugin-high                   | Active persona conversation (e.g., LLM integration)   |
| ovos-m2v-pipeline-high                              | Multilingual intent classifier capable of handling paraphrasing   Only supports <b>default skills</b> , not dynamic |
|   |   |

### Medium Confidence Components

| Pipeline    | Description |
|-------------|-------------|
| Notes       |             |
| ----- ----- |             |
| ----- ----- |             |
| -----       |             |



|  |   |
|--|---|
| stop_medium                              | Medium-confidence stop command matches  |
| Replaces <a href="#">skill-ovos-stop</a> | padatious_medium   Medium-confidence matches using Padatious  |
| adapt_medium                             | Medium-confidence matches using Adapt   |
|  |   |
| ocp_medium                               | Medium-confidence media-related queries   |
|  |   |
| fallback_medium                          | Medium-priority fallback skill matches  |
|  |   |
| ovos-m2v-pipeline-medium                 | Multilingual intent classifier capable of handling paraphrasing   Only supports <b>default skills</b> , not dynamic |
|  |   |

## Low Confidence Components

|   |   |
|---|---|
| Pipeline                                      | Description   |
| Notes   |   |
| -----   | -----   |
| -----   | -----   |
| stop_low                                      | Low-confidence stop command matches   |
| Disabled by default                           | padatious_low   |
| Low-confidence matches using Padatious        | Often inaccurate; disabled by default   |
| adapt_low                                     |   |
| Low-confidence matches using Adapt            |   |
| ocp_low                                       | Low-confidence media-related queries  |
|   | fallback_low  |
| Low-priority fallback skill matches           |   |
| common_query                                  | Sends utterance to common_query skills  |
| Selects the best match among available skills |   |
| ovos-persona-pipeline-plugin-low              | Persona catch-all fallback (e.g., LLM integration)  |
|   |   |
| ovos-m2v-pipeline-low                         | Multilingual intent classifier capable of handling paraphrasing   Only supports <b>default skills</b> , not dynamic |
|   |   |

---

## Customizing the Pipeline

OVOS allows users to customize the intent pipeline through configuration files.

Users can enable or disable specific components, adjust their order, and set confidence thresholds to tailor the system's behavior to their needs. This customization ensures that OVOS can be optimized for various applications, from simple command recognition to complex conversational agents.

```
"intents": {
  "adapt": {
    "conf_high": 0.5,
    "conf_med": 0.3,
    "conf_low": 0.2
  },
  "persona": {
    "handle_fallback": true,
    "default_persona": "Remote Llama"
  },
  "pipeline": [
    "ovos-m2v-pipeline-high",
    "ocp_high",
    "stop_high",
    "converse",
    "padatious_high",
    "adapt_high",
    "stop_medium",
    "adapt_medium",
    "common_qa",
    "fallback_medium",
    "fallback_low"
  ]
},
```

---

By understanding and configuring the OVOS Intent Pipeline, developers and users can enhance the accuracy and responsiveness of their voice assistant applications.

## docs/stop\_pipeline.md

# Stop Pipeline

The **stop pipeline** is a core component of the Open Voice OS (OVOS) pipeline architecture. It defines the logic responsible for stopping ongoing interactions with active skills. This includes aborting responses, halting speech, and terminating background tasks that skills may be performing.

Because stopping is a **fundamental feature of a voice assistant**, it is implemented as a **dedicated pipeline plugin**, not just a fallback or intent handler.

---

## Purpose

A voice assistant must always be capable of responding to a "stop" command. Whether the user says *stop*, *cancel*, or another localized phrase, OVOS must quickly:

- Determine if a skill is actively responding

- Allow skills to confirm whether they can be stopped

- Abort conversations, questions, or spoken responses

The `stop` pipeline guarantees this behavior through a flexible plugin system and localized vocab matching.

---

## How it works

The stop pipeline activates based on **high-confidence** or **medium-confidence** utterance matches.

### High-confidence (`stop_high`)

This is triggered when a user says an exact match for a stop command, e.g.,:

- Stop

- Cancel

Parar (in Portuguese)

Stopp (in German)

The plugin:

1. Checks if any **active skills** can be stopped.
2. Pings active skills
3. Waits briefly (0.5s) for replies.
4. Calls stop on relevant skills.
5. If no skills are active, emits a **global stop**: `mycroft.stop`.

## Medium-confidence (stop\_medium)

Triggered for more complex phrases that include a stop command but are not exact matches, such as:

Can you stop now?

I dont want that anymore

Stop playing music please

This match falls back to fuzzy intent matching.

---

## Localization

The plugin supports stop commands in multiple languages using `.voc` files stored in:

```
locale/  
  en-us/  
    stop.voc  
    global_stop.voc  
  pt-pt/  
    stop.voc  
    global_stop.voc
```

You can help with language support via GitLocalize

---

## Session Integration

The stop plugin interfaces with the OVOS session system:

Skills that respond to `stop` will be removed from **active skill list**

Session blacklists are respected, blacklisted skills will not be pinged

Session state is updated after each successful stop

---

## Design Philosophy

**Low latency:** Matches and stops skills within 0.5 seconds

**Extensible:** Other plugins can extend or override this pipeline

**Localized:** All behavior is language-aware and configurable

**Resilient:** Falls back to global stop if skills are unresponsive

---

## Summary

The `stop` pipeline ensures that OVOS is always in control. Whether a user needs to quickly interrupt a skill, cancel a conversation, or shut down all interactions, the `StopService` plugin provides the robust, language-aware foundation to make that possible.

It is **not considered optional**, all OVOS installations should include this pipeline by default.

# docs/tts\_transformers.md

# TTS Transformers

**TTS Transformers** in OpenVoiceOS (OVOS) are plugins that process synthesized speech audio after the Text-to-Speech (TTS) engine generates it but before it's played back to the user.

They enable post-processing of audio to apply effects, enhance clarity, voice clone or tailor the output to specific needs.

---

## How They Work

The typical flow for speech output in OVOS is:

1. **Dialog Generation:** The assistant formulates a textual response.
2. **Dialog Transformation:** Optional plugins modify the text to adjust tone or style.
3. **Text-to-Speech (TTS):** The text is converted into speech audio.
4. **TTS Transformation:** Plugins apply audio effects or modifications to the speech.
5. **Playback:** The final audio is played back to the user.

TTS Transformers operate in step 4, allowing for dynamic audio enhancements without altering the original TTS output.

---

## Configuration

To enable TTS Transformers, add them to your `mycroft.conf` under the `tts_transformers` section:

```
"tts_transformers": {  
  "plugin_name": {  
    // plugin-specific configuration  
  }  
}
```

Replace "plugin\_name" with the identifier of the desired plugin and provide any necessary configuration parameters.

---

## Available TTS Transformer Plugins

### OVOS SoX TTS Transformer

**Purpose:** Applies various audio effects using SoX (Sound eXchange) to the TTS output.

**Features:**

Pitch shifting

Reverb

Tempo adjustment

Equalization

Noise reduction

And many more

**Installation:**

```
pip install ovos-tts-transformer-sox-plugin
```

**Configuration Example:**

```
"tts_transformers": {  
  "ovos-tts-transformer-sox-plugin": {  
    "effects": ["pitch 300", "reverb"]  
  }  
}
```

**Requirements:** Ensure SoX is installed and available in your system's PATH.

**Source:** [GitHub Repository](#)

---

## Creating Custom TTS Transformers

To develop your own TTS Transformer:

### Create a Python Class:

```
from ovos_plugin_manager.templates.transformers import TTSTransformer

class MyCustomTTSTransformer(TTSTransformer):
    def __init__(self, config=None):
        super().__init__("my-custom-tts-transformer", priority=10, config=config)

    def transform(self, wav_file: str, context: dict = None) -> Tuple[str, dict]:
        """Transform passed wav_file and return path to transformed file"""
        # Apply custom audio processing to wav_file
        return modified_wav_file, context
```

**Register as a Plugin:** In your `setup.py`, include:

```
entry_points={
    'ovos.plugin.tts_transformer': [
        'my-custom-tts-transformer = my_module:MyCustomTTSTransformer'
    ]
}
```

**Install and Configure:** After installation, add your transformer to the `mycroft.conf`:

```
"tts_transformers": {
    "my-custom-tts-transformer": {}
}
```

---

By leveraging TTS Transformers, you can enhance the auditory experience of your OVOS assistant, tailoring speech output to better suit your preferences or application requirements.([ovoshatchery.github.io][4])



# docs/utterance\_transformers.md

## Utterance Transformers

**Utterance Transformers** in OpenVoiceOS (OVOS) are plugins that process and modify user utterances immediately after speech-to-text (STT) conversion but before intent recognition. They serve to enhance the accuracy and flexibility of the assistant by correcting errors, normalizing input, and handling multilingual scenarios.

---

## How They Work

1. **Speech Recognition:** The user's spoken input is transcribed into text by the STT engine.
2. **Transformation Phase:** The transcribed text passes through any active utterance transformers.
3. **Intent Recognition:** The transformed text is then processed by the intent recognition system to determine the appropriate response.

This sequence ensures that any necessary preprocessing is applied to the user's input, improving the reliability of intent matching.

---

## Configuration

To enable utterance transformers, add them to your `mycroft.conf` file under the `utterance_transformers` section:

```
"utterance_transformers": {  
  "plugin_name": {  
    // plugin-specific configuration  
  }  
}
```

Replace "plugin\_name" with the identifier of the desired plugin and provide any necessary configuration parameters.

---

## Available Utterance Transformer Plugins

### OVOS Utterance Normalizer Plugin

**Purpose:** Standardizes user input by expanding contractions, converting numbers to words, and removing unnecessary punctuation.

**Example:**

Input: "I'm 5 years old."

Output: "I am five years old"

**Installation:**

```
pip install ovos-utterance-normalizer
```

**Configuration:**

```
"utterance_transformers": {  
  "ovos-utterance-normalizer": {}  
}
```

**Source:** [GitHub Repository](#)

---

### OVOS Utterance Corrections Plugin

**Purpose:** Applies predefined corrections to common misrecognitions or user-defined replacements to improve intent matching.

**Features:**

Full utterance replacements via `corrections.json`

Word-level replacements via `word_corrections.json`

Regex-based pattern replacements via `regex_corrections.json`

**Example:**

Input: "shalter is a switch"

Output: "schalter is a switch"

**Installation:**

```
pip install ovos-utterance-corrections-plugin
```

**Configuration:**

```
"utterance_transformers": {  
  "ovos-utterance-corrections-plugin": {}  
}
```

**Source:** [GitHub Repository](#)

---

## OVOS Utterance Cancel Plugin

**Purpose:** Detects phrases indicating the user wishes to cancel or ignore the current command and prevents further processing.

**Example:**

Input: "Hey Mycroft, can you tell me the... umm... oh, nevermind that"

Output: *Utterance is discarded; no action taken*

**Installation:**

```
pip install ovos-utterance-plugin-cancel
```

**Configuration:**

```
"utterance_transformers": {  
  "ovos-utterance-plugin-cancel": {}  
}
```

**Source:** [GitHub Repository](#)

---

## OVOS Bidirectional Translation Plugin

**Purpose:** Detects the language of the user's input and translates it to the assistant's primary language if necessary, enabling multilingual interactions.

### Features:

Language detection and translation to primary language

Optional translation of responses back to the user's language

### Example:

Input: "Cual es el clima hoy?" (Spanish)

Output: "What is the weather today?" (translated to English for processing)

### Installation:

```
pip install ovos-bidirectional-translation-plugin
```

### Configuration:

```
"utterance_transformers": {  
  "ovos-bidirectional-utterance-transformer": {  
    "verify_lang": true,  
    "ignore_invalid_langs": true  
  }  
}
```

**Source:** [GitHub Repository](#)

---

# Creating Custom Utterance Transformers

To develop your own utterance transformer:

## Create a Python Class:

```
from ovos_plugin_manager.templates.transformers import UtteranceTransformer

class MyCustomTransformer(UtteranceTransformer):
    def __init__(self, config=None):
        super().__init__("my-custom-transformer", priority=10, config=config)

    def transform(self, utterances, context):
        # Modify the utterances as needed
        return modified_utterances, context
```

**Register as a Plugin:** In your `setup.py`, include:

```
entry_points={
    'ovos.plugin.utterance_transformer': [
        'my-custom-transformer = my_module:MyCustomTransformer'
    ]
}
```

**Install and Configure:** After installation, add your transformer to the `mycroft.conf`:

```
"utterance_transformers": {
    "my-custom-transformer": {}
}
```

## it/core.md

## OVOS-CORE

[OpenVoiceOS](#) è una assistente vocale open source che permette di dialogare con i tuoi dispositivi.

OpenVoiceOS ha un design completamente modulare, tutti i suoi componenti sono sviluppati come plugin. Cio significa che non solo e un ottimo assistente vocale, ma e anche una bella libreria di funzioni vocali!

`ovos-core` e "il cervello" di OpenVoiceOS, tutti i suoi componenti e gli algoritmi di Natural Language Processing vengono gestiti qui.

## Gestione delle competenze

Il sistema di gestione delle competenze, **skills service** in inglese, e responsabile per la gestione delle skills e degli intenti.

Tutte le richieste all'assistente vocale sono gestite dal servizio di gestione delle skills, il cervello di OVOS

OVOS e davvero modulare. Ad esempio, tutti i plugins di Mycroft funzionano senza problemi fino alla versione **0.1.0** di `ovos-core`!

Al momento, tutte le skills sono semplici pacchetti Python. Una volta installate saranno caricati automaticamente da `ovos-core`

Importante! A partire da `ovos-core` **0.0.8** e possibile provare una skill senza dover installare tutto, ma semplicemente usando `ovos-workshop`, un sistema di contenitori che si basa su [ovos-docker](#)

Questa funzione si rivela essere utile soprattutto durante lo sviluppo delle skills, dove bisogna fare molte prove prima di creare un pacchetto.

```
ovos-skill-launcher {skill_id} [path/to/my/skill_id]
```

## Configurazione

```

"skills": {

    // blacklisted skills to not load
    // NB: This is the skill_id, usually the basename() of the directory where the skill lives
    // the skill you want to blacklist is in /usr/share/mycroft/skills/mycroft-alarm.mycroft
    // then you should write `["mycroft-alarm.mycroftai"]` below.
    "blacklisted_skills": [],

    // fallback skill configuration (see below)
    "fallbacks": {...},

    // converse stage configuration (see below)
    "converse": {...}

},

```

## Utterance transformers

**Introdotta da ovos-core v.0.0.8**

Originariamente sviluppato da [Neon.ai](#), la **utterance transformer** permette a ovos-core di ricevere una richiesta in linguaggio naturale o una utterance da un utilizzatore, e di inviarla a un plugin di "pre trattamento".

Il framework dei utterance transformers è costituito da un numero qualsiasi di plugin ordinati per priorità (definita dallo sviluppatore). utterance e messaggio.context vengono quindi inviati a tutti i plugin che li analizzeranno in modo sequenziale.

[!imagem](#)

Per attivare un nuovo utterance transformer è sufficiente configurarlo in `mycroft.conf` dopo averlo installato

```
// To enable a utterance transformer plugin just add it's name with any relevant config
// these plugins can mutate the utterance between STT and the Intent stage
// they may also modify message.context with metadata
// plugins only load if they are installed and enabled in this section
"utterance_transformers": {

  "ovos-utterance-normalizer": {},

  // cancel utterances mid command
  "ovos-utterance-plugin-cancel": {},

  // define utterance fixes via fuzzy match ~/.local/share/mycroft/corrections.json
  // define unconditional replacements at word level ~/.local/share/mycroft/word_correction
  "ovos-utterance-corrections-plugin": {},

  // translation plugin
  "ovos-utterance-translation-plugin": {
    "bidirectional": true,
    "verify_lang": false,
    "ignore_invalid": true,
    "translate_secondary_langs": false
  }
},
```

## Metadata Transformers

**Introdotta** da ovos-core **v.0.0.8**

Come gli **utterance transformers**, questi plugins trasformano solo il `message.context`.

```
// To enable a metadata transformer plugin just add it's name with any relevant config
// these plugins can mutate the message.context between STT and the Intent stage
"metadata_transformers": {},
```

## Pipelines

**Introdotta** da ovos-core **v.0.0.8**

Il concetto di pipeline configurabile è stato introdotto con



`ovos-core` **v.0.0.8** ed è pianificato per diventare completamente operativo con la versione **0.1.0**.

Le pipeline includono parser di intenti, framework di conversazione, di query comuni e di skills di fallback.

```
// Intent Pipeline / plugins config
"intents" : {
  // the pipeline is a ordered set of frameworks to send an utterance too
  // if one of the frameworks fails the next one is used, until an answer is found
  // NOTE: if padatious is not installed, it will be replaced with padacioso (much slower)
  // in the future these will become plugins, and new pipeline stages can be added by en
  "pipeline": [
    "stop_high",
    "converse",
    "padatious_high",
    "adapt_high",
    "fallback_high",
    "stop_medium",
    "padatious_medium",
    "adapt_medium",
    "adapt_low",
    "common_qa",
    "fallback_medium",
    "fallback_low"
  ]
},
```

La maggior parte dei componenti della pipeline potrà essere configurata. Per esempio:

pipeline | descrizione | note --- | --- | --- stop\_high | il comando di "stop" corrisponde esattamente | rimpiazza [OpenVoiceOS/skill-ovos-stop](#) converse | intercettazione continua delle conversazioni | padacioso\_high | fidarsi ciecamente di padacioso | estremamente lento!!! <br> disabilitato padatious\_high | fidarsi ciecamente di padatious | adapt\_high | fidarsi ciecamente di adapt | fallback\_high | alta priorita della skill di ripiego | stop\_medium | il comando di "stop" corrisponde all'incirca | rimpiazza [OpenVoiceOS/skill-ovos-stop](#) padacioso\_medium | fidarsi cosi cosi di padacioso | estremamente lento!!! <br> disabilitato padatious\_medium | fidarsi cosi cosi di padatious | adapt\_medium | fidarsi cosi cosi di adapt | adapt\_low | fidarsi appena di adapt | common\_query |

invia l'utterance alle skills di common\_query e seleziona la corrispondenza migliore | fallback\_medium | media prioritaria della skill di ripiego | stop\_low | fidarsi appena del comando "stop" captato | disabilitato padacioso\_low | fidarsi appena di padacioso | estremamente lento!!! <br> disabilitato padatious\_low | fidarsi appena di padatious | sbaglia quasi sempre<br> disabilitato fallback\_low | bassa prioritaria della skill di ripiego |

## it/index.md

# Manuale tecnico di OpenVoiceOS

!

Benvenuti nella documentazione per sviluppatori OVOS.

Se non stai cercando di sviluppare qualcosa per OpenVoiceOS, i [documenti della community](#) sono sicuramente una lettura più digesta.

## Prospettiva

OVOS aspira ad essere un sistema operativo completamente vocale, e come tale ha bisogno di molte applicazioni e framework. OVOS fornisce tutti gli strumenti necessari per integrare la voce nei tuoi progetti.

Il tipico caso d'uso di OVOS è come assistente vocale, "Hey Mycroft, cos'è . . . " , ma come sistema operativo completamente vocale OVOS può essere personalizzato per molti altri usi. La nostra comunità si concentra su progetti che mirano a integrare OVOS in robot, mobili, specchi intelligenti, applicazioni cloud, TV, IoT e altro ancora (sì, anche un manichino!)

I progetti qui elencati sono delle "distribuzioni OVOS", combinazioni di pacchetti OVOS create per fornire uno stack completo di assistenti vocali un po' per tutti i gusti.

[raspOVOS](#) - ottimizzato per Raspberry Pi, costruito sul [sistema operativo Raspberry Pi](#)

[ovos-buildroot](#) - Linux minimale per dispositivi embedded, basato su [ovos-docker](#) e [buildroot](#)

[ovos-installer](#) - configura OVOS su un sistema operativo esistente utilizzando un perfetto equilibrio di scripts e ansible.

In questi documenti troverai informazioni sull'architettura di OVOS, i diversi tipi di plugin disponibili e le linee guida per lo sviluppo di nuove applicazioni per OVOS.

## Altre risorse

La documentazione per utilizzatori finali è pubblicata automaticamente su <https://openvoiceos.github.io/community-docs>

La documentazione specifica di Docker è disponibile su <https://openvoiceos.github.io/ovos-docker>

Un indice di tutti i messaggi bus emessi o ascoltati da tutti i progetti nelle repository OVOS può essere trovato su [https://openvoiceos.github.io/message\\_spec](https://openvoiceos.github.io/message_spec)

Esiste anche il progetto HiveMind per creare clusters di assistenti vocali. La sua documentazione dedicata può essere trovata su <https://jarbashivemind.github.io/HiveMind-community-docs/>

## Contribuire alla documentazione

La documentazione per sviluppatori OVOS è scritta e gestita da utenti proprio come te!

Pensa a questi documenti come punto di partenza e non dimenticare di contribuire laddove trovi un errore o una mancanza.

Per favore, apri [Issues](#) e [Pull Requests](#) !

## it/why.md

## Informazioni su OpenVoiceOS

Presentazione di OpenVoiceOS: l'assistente vocale intelligente rispettoso della privacy e open source.

OpenVoiceOS è un nuovo attore nel mercato degli assistenti vocali, che offre un'alternativa potente e flessibile a soluzioni proprietarie come Amazon Echo e

Google Home.

Con OpenVoiceOS hai il controllo completo sui tuoi dati personali e puoi facilmente personalizzare e aggiungere nuove funzionalità al tuo assistente vocale.

Basato su software open source, OpenVoiceOS è progettato per fornire agli utenti un'interfaccia vocale semplice e intuitiva che permette di controllare i propri dispositivi domestici intelligenti, riprodurre musica, impostare promemoria, raccontare barzellette, chiacchierare con un'intelligenza artificiale e molto altro.

La piattaforma si basa su tecnologie all'avanguardia: machine learning, elaborazione del linguaggio naturale NLP e altre, configurate per offrire un'esperienza altamente reattiva e accurata su tutti i tipi di supporto.

Oltre alle sue funzionalità vocali, OpenVoiceOS presenta un'interfaccia touch-screen realizzata utilizzando QT5 e il framework KF5.

L'interfaccia è intuitiva e facile da usare e consente di accedere all'intera gamma di caratteristiche e funzionalità di OpenVoiceOS.

Che tu preferisca i comandi vocali o l'interfaccia touch più tradizionale, OpenVoiceOS è sicuramente quello che fa per te.

Uno dei principali vantaggi di OpenVoiceOS è la sua comunità open source, se anche tu hai delle competenze tecniche prova a contribuire alla piattaforma.

Che tu sia uno sviluppatore di software, un data scientist o semplicemente qualcuno con una passione per la tecnologia e le lingue, puoi partecipare e contribuire a costruire la prossima generazione di assistenti vocali intelligenti.

Con OpenVoiceOS hai la possibilità di far funzionare il tuo assistente completamente offline, in modo da avere il controllo completo sui tuoi dati e assicurandoti che le tue informazioni non vengano mai condivise con terze parti. E questo che fa di OpenVoiceOS la scelta perfetta per chiunque apprezzi la privacy e la sicurezza.

Quindi, se stai cercando un assistente vocale intelligente che ti dia la libertà e il controllo che meriti, prova OpenVoiceOS!

**Disclaimer** : questo post è stato scritto (e tradotto) in collaborazione con ChatGPT