TTS-o-matic by Evolvedlabs SAS

staff@noiseomatic.com

www.evolvedlabs.com

DOCUMENTATION 1.1.0

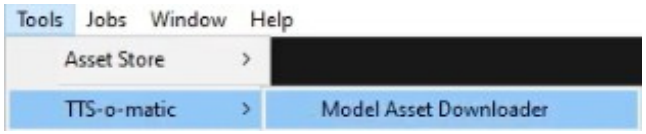
Table of Contents

Getting started

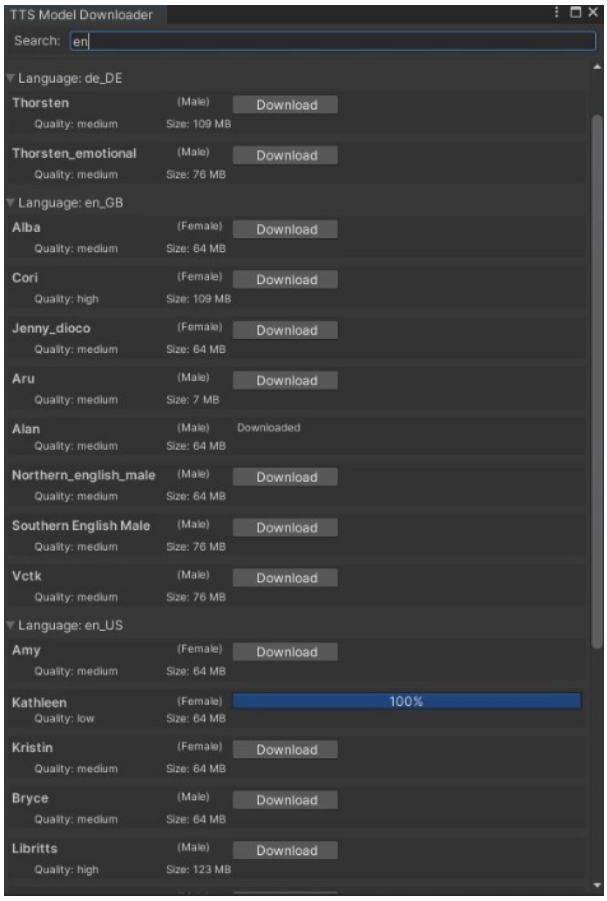
After importing TTS-o-matic in your project, you can find a demo scene available for testing, or you can directly start using it in your project. We'll go through the code later on.

In both cases however, we need to download some voice models in your project, too.

All voice models are compatible with the Sherpa/Icefall projects (see appendix on how to train your own voices) and a curated selection is directly downloadable from the Unity3D interface. From the "Tools" menu, pick "TTS-o-matic" and then launch the "Model Asset Downloader".



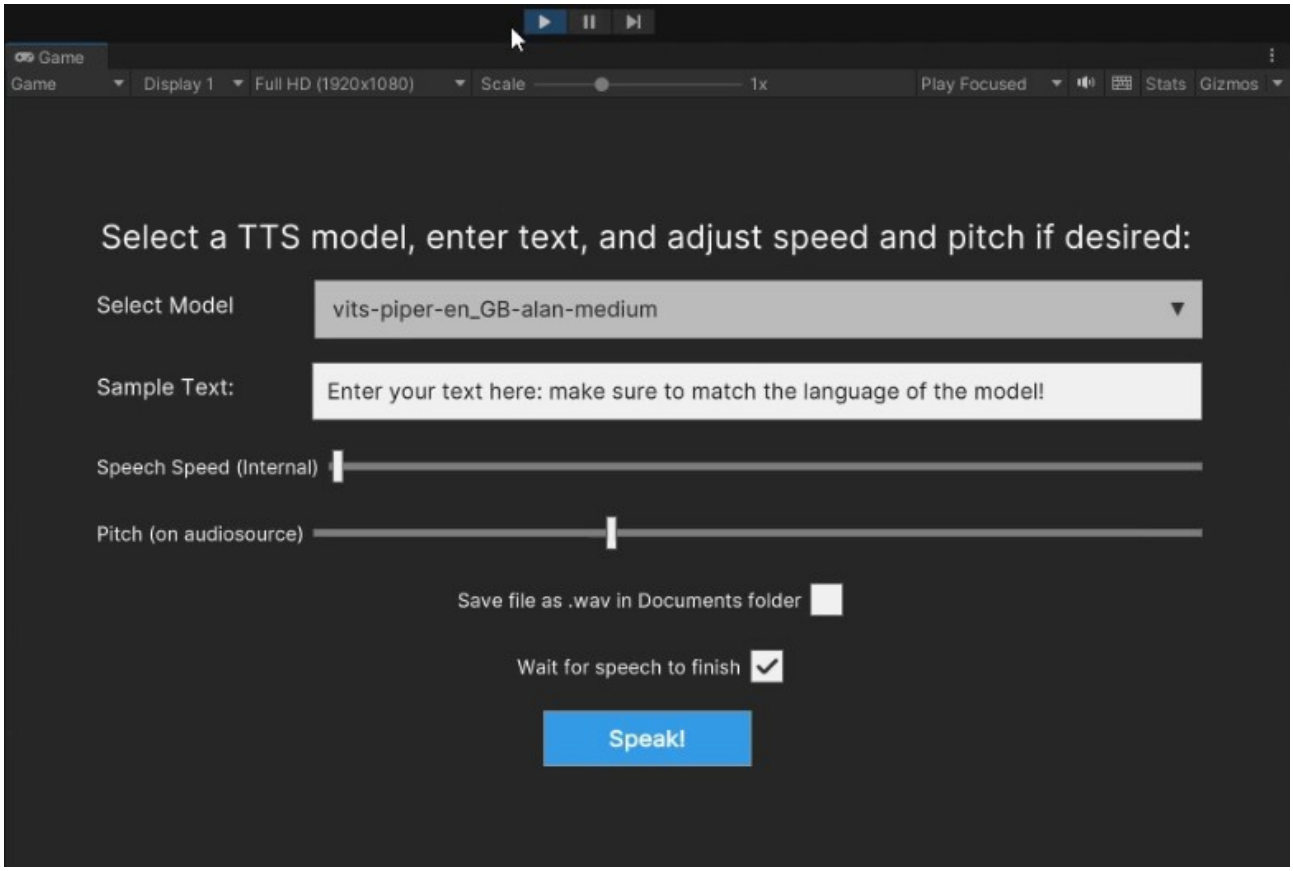
From this interface, click "download" on your desired voice: note how voices are grouped by language using the IETF language tags (The format is usually: language_COUNTRY, where language is a two-letter ISO 639-1 language code and COUNTRY is a two-letter ISO 3166-1 alpha-2 country code).



Once downloaded, your model will be automatically extracted in your "StreamingAssets" directory, creating a "ttsmodels" subdirectory.

Running the demo scene

Find and doubleclick the “TTS-o-matic DemoScene.unity” scene file, unpacked in the /Assets/Evolvedlabs SAS/TTS_o_matic/Scenes directory to open the demo scene in your editor.



This simple scene can be used as a playground to try TTS-o-matic.

Select Model: allows you to switch between the models currently found in the StreamingAssets\ttsmodels directory.

Sample Text: text that will be read out loud (and eventually saved to disk, see below)

Speech Speed: the speed of the speech sample itself. This is internal.

Pitch: The pitch of the sample. This is applied as an audio effect by the AudioSource component of Unity3D.

Save file as .wav: the generated audio will be saved as .wav file in the documents directory of the user running the application.

Wait for speech to finish: blocks the UI until the speech has finished playing.

Using TTS-o-matic in your code

Although we suggest looking at the demo scene code, you can simply deploy the **Evolvedlabs SAS/TTS_o_matic/Scripts/NomTTS.cs** file (at the very minimum) and eventually the WavUtility.cs file if you need to provide .wav saving capabilities in your application.

Before generating speech, you need to load the model that TTS-o-matic will use to fulfill your request: you do this via the InitModel function:

```
bool loaded = NomTTS.InitModel("vits-piper-en_GB-aru-medium");
```

All TTS generation function calls will use the latest loaded model. InitModel has only a mandatory argument, as its second and third argument have a default value as tokens.txt and espeak-ng-data.

After verifying that the model has been loaded correctly, you can generate any speech by calling either the SpeakToClip function or the SpeakToClipAsync functions.

While both functions do essentially the same work, SpeakToClip is blocking and thus we advise using the Async function. You should wrap it in a separate thread as long text (or even short text, when set to run very slow) can take a “long” time to generate. In Unity3D this is easily done inside a Coroutine.

```
var speakTask = NomTTS.SpeakToClipAsync(text, speed);

yield return new WaitUntil(() => speakTask.IsCompleted);

if (speakTask.Exception != null)
{
    Debug.LogError("[NOMtts] failed: " + speakTask.Exception);
    yield break;
}

AudioClip clip = speakTask.Result;
if (clip == null)
{
    Debug.LogError("[NOMtts] No AudioClip returned.");
    yield break;
}
```

These functions simply take two arguments: the text you want to run (string) and the speed. The normal speed is 1.0f. It is suggested to clamp the minimum speed to 0.1f.

You can see how to integrate this code directly in NOMttsDemo.cs – this example files run the code using parameters set in the game UI over at TTS_o_matic_SampleScene.cs (see the OnRun() function) and then calls the SpeakToClipAsync() functions accordingly. In order to immediately hear it, note that the NOMttsDemo behaviour is attached to a gameobject that also has attached an AudioSource component.

Speaker ID

Some TTS models provide multiple speakers within the same model file. When that is the case, it is possible to specify the speaker ID directly. The speaker ID is an index where the first element is 0 – of course – and so on, with an increasing value. By default, the speak function takes 0 as default value, as it is optional.

To know if the model supports multiple speakers, please refer to the model card of the model itself.

Use the **getNumSpeakers()** function in order to obtain the number of speaker the currently loaded model offers.

Simply provide the desired index in the **SpeakText()** function.

Note: the demo scene loads and creates procedurally the dropdown menu – the model loading function is blocking, but should be fast. However, you'll notice a slight slowdown when changing model – that is because the TTS_o_matic_SampleScene.cs will create runtime the dropdown menu in a rather unoptimized way.

Appendix A: FAQ & Troubleshooting

Q: Which operating systems are supported by TTS-o-matic?

TTS-o-matic requires Windows 10 or 11 (x86_64) (sherpatts.dll, onnxruntime.dll). While it *might* run on older version of windows, those are not supported.

Q: I get an error when loading the model, what do I do?

Make sure that **StreamingAssets** exists and that your models have been downloaded and extractly correctly. If you imported custom trained voices, make sure that the tokens path (generally tokens.txt) and configuration (generally espeak-ng-data) are passed correctly to the InitModel function.

Q: I used to be able to hear the voice being generated, now I hear the first syllable and then silence, what's the issue?

You may have downloaded a model, then deleted the directory in StreamingAssets. This might create issues. Restart your Unity3D editor.

Q: I can't save the Wav files to disk, what's the issue?

If you're using the demo scene you should be able as the demo is setup to save the file in your Documents directory. However, if you have changed it, make sure that the path exists and it is writable by the user.

Appendix B: training your own voice models

As TTS-o-matic is based on Sherpa/Icefall, you can train your voice models. This is a short guide to point you in the right direction, although please keep in mind we can not provide support for Sherpa or Icefall.

The Sherpa TTS engine is an open source software released with Apache 2.0 license. However, while the software itself is open source, not all the voice models it supports (or the datasets they have been trained on) might not be distributed with a license that allows redistribution.

In this guide, you'll see how to manually download and install additional models and some pointers on how to create your own.

TTS-o-matic is a commercial product, and thus, all models that are redistributed by the Sherpa project under a non-commercial license can not be directly shipped with TTS-o-matic. TTS-o-matic only ships those models that are made with public datasets and that are distributed with proper rights for commercial use in third party products such as TTS-o-matic.

If you're the author of a pre-trained model shipped with TTS-o-matic and would like to see your model not shipped with TTS-o-matic despite being allowed by the license, or if you encounter an error in our selection, please get in touch at staff@noiseomatic.com

To train your own model, you can find the icefall documentation here: https://k2-fsa.github.io/icefall/

You can train your own model or you can pick one from the Sherpa TTS project repository here https://github.com/k2-fsa/sherpa-onnx/releases/tag/tts-models to go ahead with this guide.

Once your trained model is ready (or if you downloaded a model, make sure that there is a .onnx file along with a espeak-ng-data directory - this means that the model is compatible with TTS-o-matic) simply put all the model files in the desired directory, so it would look like this

..\YourProject\Assets\StreamingAssets\ttsmodels\mynewmodel\

..\YourProject\Assets\StreamingAssets\ttsmodels\mynewmodel\model.onnx

..\YourProject\Assets\StreamingAssets\ttsmodels\mynewmodel\MODEL_CARD

..\YourProject\Assets\StreamingAssets\ttsmodels\mynewmodel\espeak-ng-data\

..\YourProject\Assets\StreamingAssets\ttsmodels\mynewmodel\espeak-ng-data\....

etc.

In short, simply extract the archive in the ttsmodels directory.

If everything went well, you should be to load your custom voice in TTS-o-matic like you would with any other model that has been downloaded.

Keep in mind you are entirely responsible for what you train and how you use every trained model, despite the license they are distributed with.

References:

Sherpa TTS source code on github: https://github.com/k2-fsa/sherpa-onnx

Sherpa TTS project page: https://k2-fsa.github.io/sherpa/onnx/tts/index.html

Sherpa TTS Models downloads: https://github.com/k2-fsa/sherpa-onnx/releases/tag/tts-models

Appendix C: license

Please refer to your purchase during checkout of TTS-o-matic at the asset store. For reference:

Asset Store EULA : https://unity.com/legal/as-terms
Asset Store FAQ : https://assetstore.unity.com/browse/eula-faq

TTS-o-matic uses open source components that allows commercial redistribution.

Sherpa TTS : https://www.noiseomatic.com/licenses/sherpa.txt

TTS-o-matic allows download of onnx models from the SherpaTTS repository when their license allows for it. Please refer to the license file in the downloaded model file for additional information.

Appendix D: support

You can reach support at staff@noiseomatic.com or you can join the Discord Server at http://noiseomatic.com/discord

We can help with:

- bugs
- crashes
- documentation clarification

Please note that we can not help you training your own models. See Appendix B.