# Instructions for Setting Up and Importing a New TTS Engine in AllTalk

Further details are provided within each script file and you can look at other model\_engine.py files for examples of code. This is a bit of a rough draft, though there is more detail added within the template files themselves.

## 1. Create Necessary Folders

Create Folders in `tts\_engines` and `models` Directories  
- Create a folder with the name of your TTS engine in the `tts\_engines` directory.  
- Similarly, create a corresponding folder in the `models` directory.

## 2. Copy Template Documents

Copy Template Documents  
- Copy over the provided template documents to the newly created folder in the `tts\_engines` directory.

## 3. Edit `model\_settings.json`

Set Features and Base Settings  
- Open the `model\_settings.json` file in the `tts\_engines/<your\_tts\_engine>` folder.  
- Update the `manufacturer\_name`, `manufacturer\_website`, and `model\_description` under `model\_details`.  
- Configure the capabilities under `model\_capabilities` according to your TTS engine's features.  
- Adjust the `settings` section to match your engine's default settings.  
- Add any necessary voices in the `openai\_voices` section.  
  
Example from `model\_settings.json`:  
  
{  
 "model\_details": {  
 "manufacturer\_name": "MyTTSEngine",  
 "manufacturer\_website": "https://github.com/MyTTSEngine",  
 "model\_description": "A description and detail about the TTS Engine"  
 },  
 "model\_capabilties": {  
 "audio\_format": "wav",  
 "deepspeed\_capable": false,  
 "generationspeed\_capable": false,  
 "languages\_capable": false,  
 "lowvram\_capable": false,  
 "multimodel\_capable": false,  
 "repetitionpenalty\_capable": false,  
 "streaming\_capable": false,  
 "temperature\_capable": false,  
 "multivoice\_capable": false,  
 "pitch\_capable": false,  
 "windows\_capable": true,  
 "linux\_capable": true,  
 "mac\_capable": true  
 },  
 "settings": {  
 "def\_character\_voice": "defaultvoice.wav",  
 "def\_narrator\_voice": "defaultvoice.wav",  
 "deepspeed\_enabled": false,  
 "engine\_installed": true,  
 "generationspeed\_set": 1,  
 "lowvram\_enabled": false,  
 "pitch\_set": 0,  
 "repetitionpenalty\_set": 10,  
 "temperature\_set": 0.75  
 },  
 "openai\_voices": {  
 "alloy": "defaultvoice.wav",  
 "echo": "defaultvoice.wav",  
 "fable": "defaultvoice.wav",  
 "nova": "defaultvoice.wav",  
 "onyx": "defaultvoice.wav",  
 "shimmer": "defaultvoice.wav"  
 }  
}

## 4. Edit `modelname\_settings\_page.py`

Update Function Names and Setup  
- Change the function names to match your TTS engine.  
- Implement the function to gather available voices for your model.  
- Add help guidance into the markdown area.  
- Create a model/voice downloader script that works with the `available\_models.json` file to manage downloads.

## 5. Configure `available\_models.json`

Manage Available Models and Default Downloads  
- Open the `available\_models.json` file.  
- Add your model to the `models` list, including necessary download links for each voice model.  
- Set a default first\_start\_model download.  
- Its up to you to figure out how to populate a list of downloadable models and handle the download process. Take a look at other engines for how this has been handled.

Example from `available\_models.json`:  
  
{  
 "first\_start\_model": "en\_GB-jenny\_dioco-medium",  
 "models": [  
 {  
 "model\_name": "ar\_JO-kareem-low",  
 "files\_to\_download": [  
 "https://example.com/path/to/ar\_JO-kareem-low.onnx",  
 "https://example.com/path/to/ar\_JO-kareem-low.onnx.json"  
 ]  
 },  
 {  
 "model\_name": "ar\_JO-kareem-medium",  
 "files\_to\_download": [  
 "https://example.com/path/to/ar\_JO-kareem-medium.onnx",  
 "https://example.com/path/to/ar\_JO-kareem-medium.onnx.json"  
 ]  
 }  
 ]  
}

## 6. Edit `model\_engine.py`

Populate Script with Relevant Code  
- Review and populate all relevant sections in the `model\_engine.py` script.  
- Use examples from other installed TTS engines for guidance on how to set up different parts of the script according to your engine's requirements.  
- For the most part you will be able to follow the manufacturers guidelines on generating TTS with Python scripts. Though you will need to figure out how to populate the voices names and model names.

## 7. Install Requirements

Add Dependencies  
- Determine the requirements for installing your TTS engine into Python.  
- Choose whether to include these requirements in the main `requirements.txt` file or the `atsetup.bat` or `atsetup.sh` files.  
- Consider keeping these separate to avoid conflicts with other engine requirements.

## 8. Update `new\_engines.json`

Register New Engine  
- Open the `new\_engines.json` file.  
- Add an entry for your new engine with its name and selected model.  
  
Example from `new\_engines.json`:

{

    "engines\_available": [

        {

            "name": "parler",

            "selected\_model": "these are examples of how you add new engines into the list here. You would put a model name in this line"

        },

        {

            "name": "xtts",

            "selected\_model": "xtts - xttsv2\_2.0.3"

        }

    ]

}

You would add a new engine into this list. When AllTalk starts up, as long as it notices an engine name that **doesn’t exist** in the main tts\_engines.json file **AND** also **the folder** for the new engine is the **same name** and **exists**, it will add the new engine into tts\_engines.json.

{

    "engines\_available": [

        {

            "name": "parler",

            "selected\_model": "these are examples of how you add new engines into the list here. You would put a model name in this line"

        },

        {

            "name": "mynewengine",

            "selected\_model": "mynewengine - mymodelthatitloads"

        },

        {

            "name": "xtts",

            "selected\_model": "xtts - xttsv2\_2.0.3"

        }

    ]

}