CREATORS GUIDE

Table of Contents

Last updated March 20, 2025.

Getting Started

This manual serves as a reference guide for creating **Playback Engines** in NotePerformer.

A **Playback Engine** functions similarly to an **Expression Map** or **Sound Set**, but for NotePerformer. It replaces NotePerformer's internal sounds with those of a **VST3 instrument**.

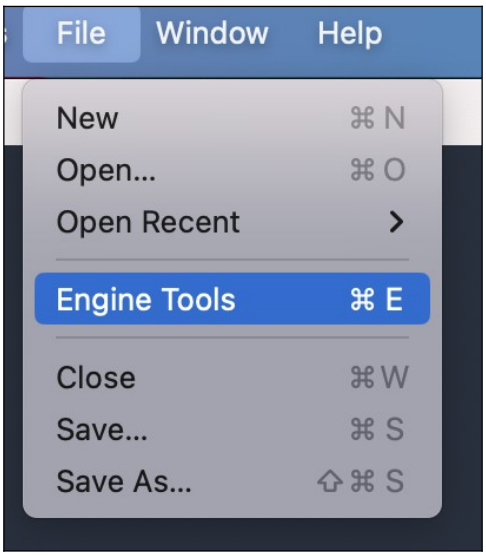
Creating a **Playback Engine** is an **advanced feature** intended for experienced users who are comfortable with **VST3 preset creation** and **basic scripting**.

Playback Engines are loaded through the standalone application **NotePerformer Playback Engines (NPPE)**, which is installed alongside NotePerformer.

- **Windows**: Available in the **Start Menu**
- **Mac**: Located in the **Applications folder**

Additionally, this guide frequently references **Engine Tools**, which can be accessed within NPPE by navigating to:

File > Engine Tools



File > Engine Tools...

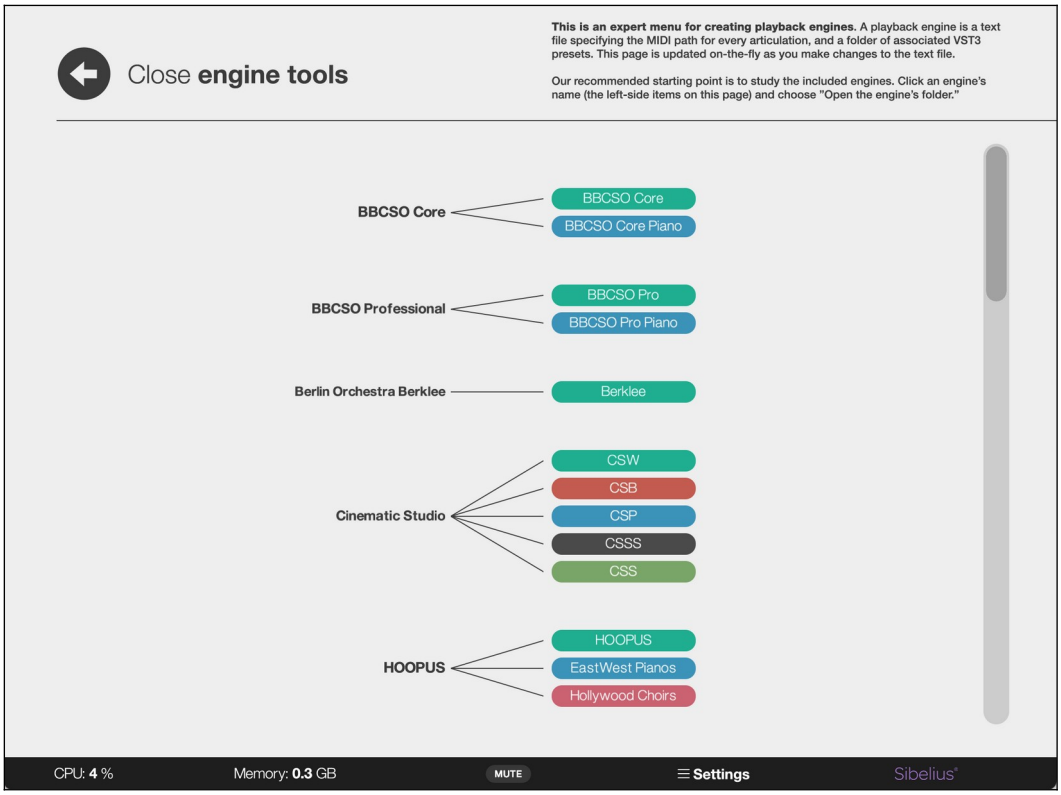
The **main difference** from an Expression Map is that the Playback Engine **preloads a subset of samples from the VST3 instrument in a lossy compression format**, effectively creating a **miniature sample library in memory** optimized for NotePerformer. This allows the Playback Engine to run with **minimal RAM and CPU usage**, making it ideal for **notation users**, especially those on **laptops or mainstream computers**. The **preloading process** is covered in more detail in a separate chapter.

While Playback Engines **don't replicate** the exact sound of a VST3 instrument, they function as a **hybrid between the sample library and NotePerformer**. The **timbre of the sample library is**

preserved, but the instrument is integrated into the NotePerformer system. **Users can freely adjust reverb levels**, even for wet sample libraries such as those from **Spitfire Audio or Orchestral Tools**, since NPPE **combines samples with signal processing** rather than relying on **extensive crossfading techniques**. This **hybrid approach** not only ensures stability and efficiency but also **enables users to create Playback Engines from virtually any sample library** with minimal effort.

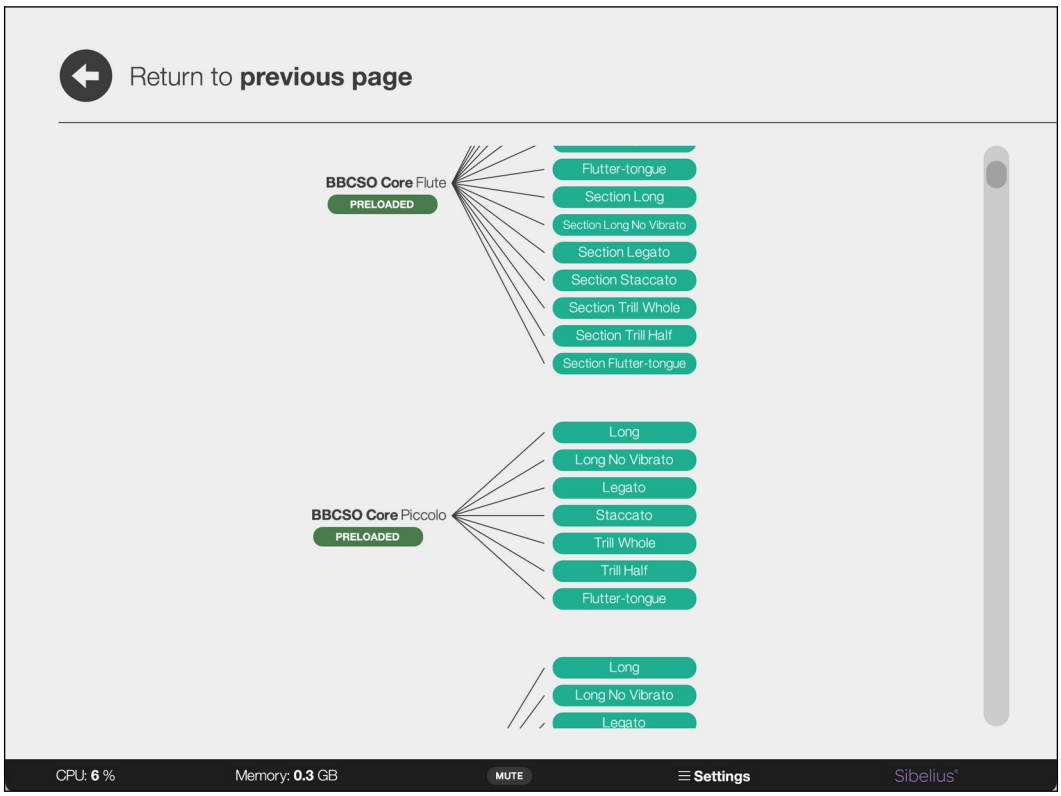
Engine Tools: At First Glance

The main Engine Tools screen shows all the Playback Engines **on your system**, and the Products they include.



Engine Tools: Main Screen

Clicking the **Playback Engine's name** (on the left) opens a popup menu. Clicking the **Product names** (on the right) navigates into that Product, showing its contents.



Engine Tools: Inside a Product

Inside the Product, you'll find a list of **Instruments** and their associated **Articulations**.

Clicking the **Instrument's name** (on the left) opens a popup menu for that Instrument. Clicking the **Articulation** (on the right) opens a popup menu for the selected Articulation.

From the **Articulation popup menu**, you can **access the VST3 plug-in and create VST3 presets**, which will be important later on.

File Structure

Playback Engines are stored in the "**/NotePerformer/Playback Engines**" folder within the user's application directory.

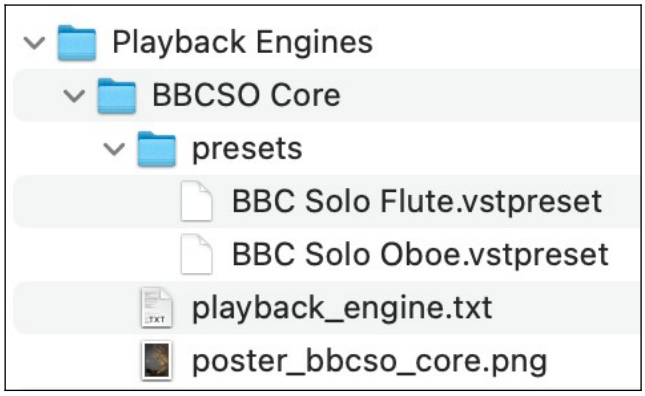
- **Windows**: Located in `%appdata%`
- **Mac**: Located in `~/Library/Application Support`

Each engine is identified by a **unique folder name** (e.g., `"BBCSO Core"`), which should be carefully chosen to avoid naming conflicts. Stick to letters, numbers, spaces, and hyphens (`-`), as special characters may be incompatible with filenames on an end-user's system.

Inside each **Playback Engine** folder, you will find three essential components:

1. **`presets` subfolder** – Stores all **VST3 presets** (`*.vstpreset`) associated with the engine.

2. **`playback_engine.txt` file** – A basic script that defines the engine's settings and behavior.
3. **`.png` image file** – The engine's front cover image, which must be exactly **372 x 526 pixels**.



Inside a Playback Engine folder.

Creating Playback Engine Files

- **VST3 Presets (`.vstpreset`)**
    - These can be saved from most **VST3-compatible DAWs**.
    - However, it is recommended to create them using **Engine Tools**, which includes built-in utilities for `.vstpreset` **creation** and **key-switch testing**.
- **Editing the `playback_engine.txt` File**
    - The `playback_engine.txt` file is a plain text script that defines all aspects of the Playback Engine.
    - It can be **edited in any text editor** while keeping **Engine Tools open** in the background.
    - Changes made to this file are **automatically reflected in Engine Tools** when saved.
- **Creating a `.png` Image File**
    - Create a `.png` front cover image using any image editor (e.g., Photoshop).
    - The image must be exactly **372 x 526 pixels**.

Workflow for Editing Playback Engines

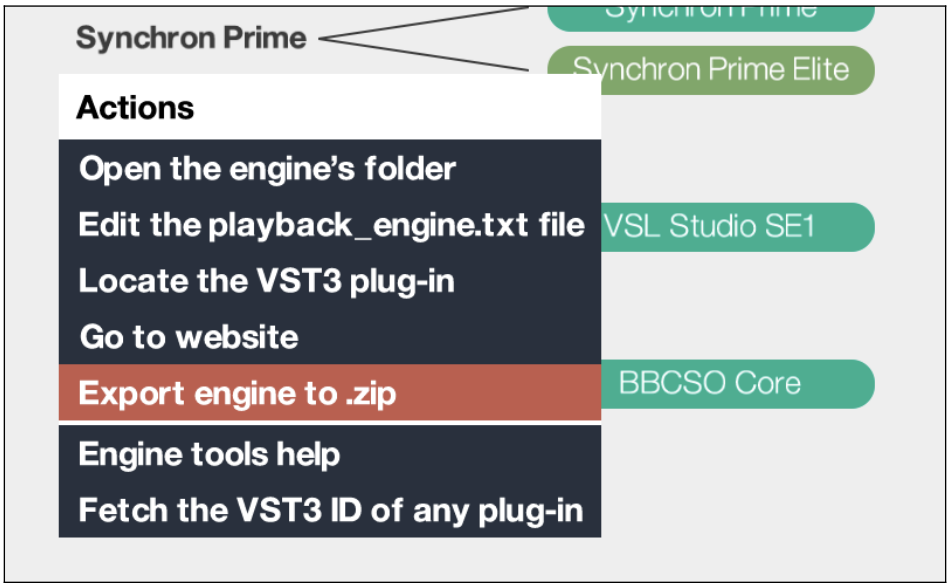
1. Open **Engine Tools** and your preferred **text editor**.
2. Edit the **`playback_engine.txt`** file, adding Instruments, Articulations, or other configurations.
3. Save the file—**Engine Tools will update automatically** to reflect your changes.
4. **Engine Tools instantly reflects your changes, giving you immediate feedback as you work.**
5. Create the associated **VST3 preset** in **Engine Tools** and **run tests** to confirm that all Articulations function as expected.

This workflow makes it easy to tweak and refine your Playback Engine.

Playback Engines are always visible in Engine Tools, but **only appear on the main screen if they have *no errors***. If your engine is missing from the main screen, Engine Tools will display the error.

Sharing a Playback Engine

Playback Engines can be easily shared with others. They are exported from **Engine Tools** as a `.zip` archive containing the `playback_engine.txt` file, the `.png` cover image file, and the VST3 presets.



To install the Playback Engine on another system, users can simply drag and drop the `.zip` archive into the *NPPE interface*. **Existing Playback Engines won't be overwritten**. Instead, the user must manually remove the old Playback Engine before installing a new one with the same base folder name.

A `playback_engine.txt` example

Below is a simple example of a **Playback Engine** with one Instrument, a flute. We'll use this to illustrate how **Playback Engines** work and how they are structured into **Products**, **Instruments**, and **Articulations**. Throughout this guide, we'll refer back to this example to explain key concepts in context.

The `playback_engine.txt` file is processed sequentially from top to bottom.

```
begin_playback_engine(BBC Symphony Orchestra Core)
set_recording_location(Maida Vale Studios)
set_developer(Spitfire Audio)
set_release_year(2020)
set_website(https://www.spitfireaudio.com/bbc-symphony-orchestra-core)
set_cover_art(poster_bbcso_core.png)
set_vst3_id(56535453616E746262632073796D7068)
set_vst3_name(BBC Symphony Orchestra)
add_recommended_software_version(BBC Symphony Orchestra 1.7.0)
set_microphone_configuration(Mix 1)



begin_product(BBCSO Core)
set_long_product_name(BBC Symphony Orchestra Core)



begin_instrument(My Flute)
set_noteperformer_override_sound(Flute)

set_vst3_preset(BBC Solo Flute.vstpreset)
add_sustain(NOTEON=1;CC21=127, CC1, 1)
add_legato(NOTEON=0;CC21=127, CC1, 126)
add_staccato(NOTEON=2, VEL, 1)
```

The `playback_engine.txt` file is not a high-level programming language, but commands must be entered exactly as specified. If a command is prefixed with a symbol such as `%` or `#`, the line is ignored. This can be useful for adding comments or temporarily editing a `playback_engine.txt` file.

```
%add_staccato(NOTEON=3, VEL, 1) - this line is ignored
#add_staccato(NOTEON=4, VEL, 1) - this line is also ignored
add_staccato(NOTEON=2, VEL, 1)
```

Playback Engines

Playback Engines follow this hierarchical structure:

Playback Engine > Products > Instruments > Articulations

- A **Playback Engine** contains one or more **Products**.
- Each **Product** can include multiple **Instruments**.
- Each **Instrument** consists of various **Articulations**.

The following sections explain these components and how they relate.

Defining the Playback Engine

A Playback Engine is created in the `playback_engine.txt` file using the following command:

```
begin_playback_engine(BBC Symphony Orchestra Core)
```

Here, **"BBC Symphony Orchestra Core"** is the engine's **display name** in NPPE.

Once the engine is created, additional details must be specified, such as the **release year, developer name, and cover art file**. These settings do not affect functionality but are displayed in NPPE to provide helpful information to users.

Example Engine Metadata:

```
set_recording_location(Maida Vale Studios)
set_developer(Spitfire Audio)
set_release_year(2020)
set_website(https://www.spitfireaudio.com/bbc-symphony-orchestra-core)
set_cover_art(poster_bbcso_core.png)
```

Software Version Compatibility

The software version specifies which **VST3 plug-in version** was used to create the VST3 presets. While this information does not affect the functionality of the Playback Engine, it helps users ensure they have the correct version installed. If a user tries to load an older version of the plug-in, the Playback Engine's **VST3 presets may not work properly**.

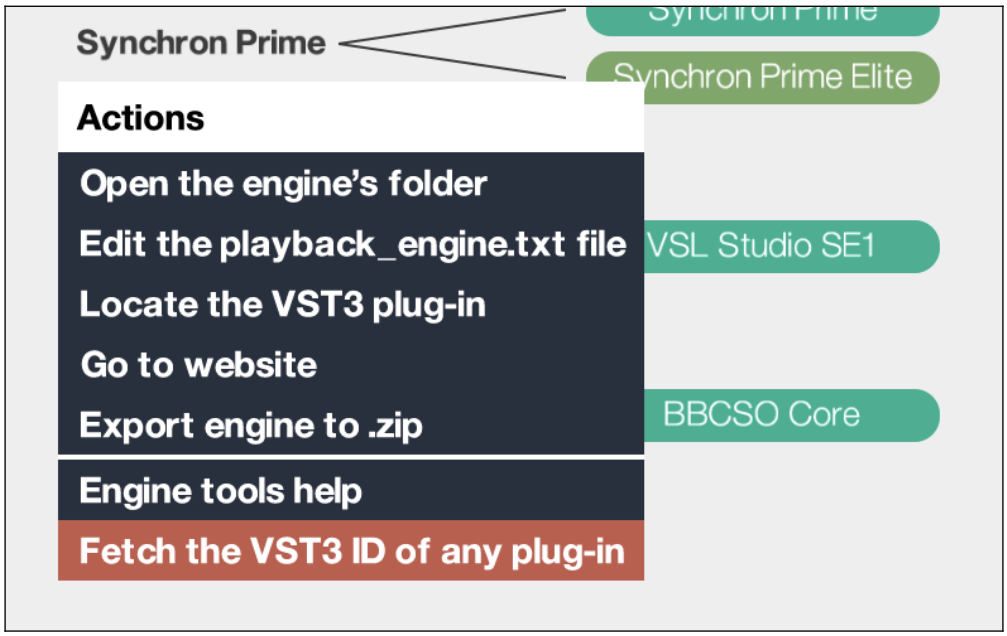
```
add_recommended_software_version(BBC Symphony Orchestra 1.7.0)
```

The `add_recommended_software_version` function can be called multiple times to list different relevant versions, such as a sample library version and a VST version. Here's an example from another Playback Engine:

```
add_recommended_software_version(Kontakt Player 7.6)
add_recommended_software_version(Cinematic Studio Brass 1.0.0)
add_recommended_software_version(Cinematic Studio Piano 1.0.0)
```

Specifying the VST3 Plug-in ID

Every **VST3 plug-in** has a **unique identifier code**, which must be set in the `playback_engine.txt` file to ensure compatibility with the correct plug-in on the end-user's system. You can fetch this identifier from any VST3 plugin through **Engine Tools**.



The **VST3 unique identifier** is added to the script as shown below. Additionally, we provide a **VST3 name**, which helps the end-user identify the required VST3 plugin.

```
set_vst3_id(56535453616E746262632073796D7068)
set_vst3_name(BBC Symphony Orchestra)
```

If a plug-in has multiple versions with different identifiers (e.g., **Kontakt 7 vs. Kontakt 8**), the engine can support both by listing multiple IDs.

```
set_vst3_id(5653544E694B376B6F6E74616B742037, 5653544E694B386B6F6E74616B742038)
```

Configuring Microphones

Some **VST3 plug-ins** include multiple microphone positions, allowing users to mix different mic signals for a more customized sound. The **Playback Engine** can take advantage of up to three microphones, but the stereo outputs of the VST3 plug-in **must match** the configuration specified in the `playback_engine.txt`.

If the **sample library has only one channel**, the microphone configuration is simply a **label** that appears in NPPE:

```
set_microphone_configuration(Mix 1)
```

For **libraries with multiple microphone positions**, the engine can be configured to support up to three microphones. The first three stereo outputs of the VST3 plug-in must be mapped accordingly.

Example Configuration for Three Microphones:

```
set_microphone_configuration(Tree, Close, Ambient)
set_default_microphone_balance(1.0, 0.5, 0.5)
```

- The **microphone names** (`Tree, Close, Ambient`) must match the order of the **actual outputs** of the VST3 plug-in.
- The **first microphone listed** will always correspond to the first stereo output, the second to the second output, and the third to the third output.
- The **default microphone balance** (`1.0, 0.5, 0.5`) sets the initial relative mix levels for each microphone in NPPE. These values define the proportions between microphones, but **they are always normalized** to ensure the overall balance remains consistent.
- Users can manually adjust the microphone balance later from the NPPE interface.
- Adding microphones proportionally increases the Playback Engine's preload size, RAM usage, and CPU load.

If the `set_microphone_configuration` command does not match the **actual microphone output mapping** of the VST3 plug-in, the preloading may fail, or the microphone labels in NPPE may appear out of order. Always ensure that the microphone setup in `playback_engine.txt` aligns with how the microphones are assigned to the stereo outputs of the plug-in.

Place the most important microphone first in your configuration. This allows users with basic computers to quickly reduce the Playback Engine's footprint by removing supplementary microphones from the `playback_engine.txt` file with a single edit.

```
#set_microphone_configuration(Tree, Close, Ambient)
set_microphone_configuration(Tree)
```

Products

A **Playback Engine** can contain multiple **Products**, which represent separate purchasable sample libraries. You should create multiple Products whenever the sounds are sold individually—for example, **"Cinematic Studio Strings"** and **"Cinematic Studio Brass"**.

Dividing an engine into Products ensures compatibility for end-users who only own some of the available sounds. When using the engine, end-users **manually select** which Products they have during the **preloading step**.

Defining a Product

In our example, the engine contains only **one Product**, which shares the same name as the engine. A Product must have both a **short name** and a **long name**, defined in `playback_engines.txt`. If a long name is not specified, **Engine Tools will display an error** and the engine won't show up on the main NPPE screen.

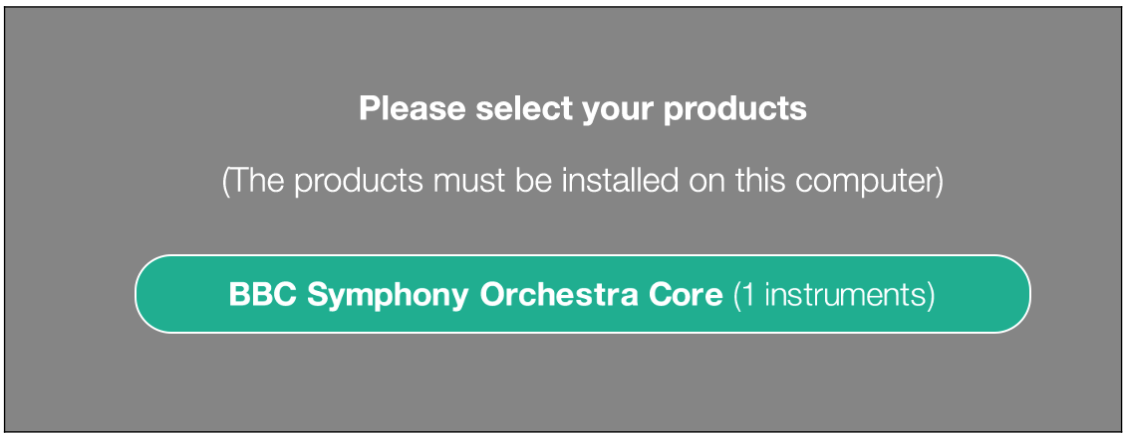
Example Product Definition:

```
begin_product(BBCSO Core)
set_long_product_name(BBC Symphony Orchestra Core)
```

How Product Names Are Displayed

- The **Short Product Name** is a prefix for Instrument names and appears in NPPE **Instrument Slots** and other places.

- The **Long Product Name** is displayed when users select Products during the **preloading step**, helping them identify their sample libraries:



This structure ensures that users can easily distinguish between different Products and **avoid preloading sounds they don't own**.

Instruments

Each **Product** contains one or more **Instruments**, which represent the sounds available to the user in NPPE. They appear as **Slots** in the NPPE interface.

In our example, the Product contains a single Instrument: "**My Flute**".

Defining an Instrument in `playback_engine.txt`

To add an Instrument, you must specify:

1. Run `begin_instrument` with a **custom name**.
2. Run `set_noteperformer_override_sound` to link the Instrument to a **predefined NotePerformer sound**.

```
begin_instrument(My Flute)
set_noteperformer_override_sound(Flute)
```

Why the Override Sound Matters

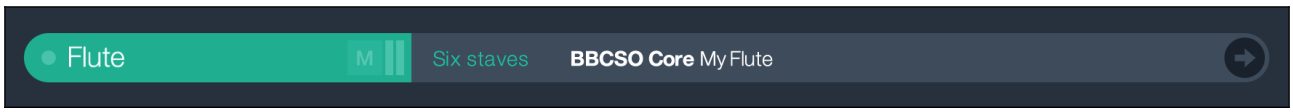
The end of this guide includes a **list of available Override Sounds**.

The choice of O**verride Sound** is critical because:

1. It determines how **NotePerformer routes programs to NPPE Slots**.
2. NPPE **automatically adjusts volume and dynamics** relative to the chosen sound, maintaining orchestral balance regardless of the VST3 plug-in used.
3. It may **affect Instrument behavior**, including whether NPPE should loop the samples.
4. If the Override Sound is misspelled, **Engine Tools will display an error**.

How Name and Override Appears in NPPE

The **custom Instrument name** appears in many places, such as the right side of the Instrument Slot, and is always prefixed by the Short Product Name.



Instrument Slot

The label on the left side of the Instrument Slot represents the *Override Sound*. If multiple Slots override the same NotePerformer sound, they are **numbered sequentially** (e.g., *Flute 1*, *Flute 2*, etc.).

Assigning a VST3 Preset

Before adding **Articulations**, a **VST3 preset** must be set. This VST3 preset will apply to all Articulations **until a new preset is specified**.

```
set_vst3_preset(BBC Solo Flute.vstpreset)
```

- A single **VST3 preset** can be assigned **multiple times** and shared between Instruments.
- This is particularly useful for **unpitched percussion**, where multiple sounds may be mapped across the MIDI keyboard while using the same preset.
- The preset does not need to exist yet—**Engine Tools can create it later**.
- **An Instrument can draw from multiple VST3 presets** by calling `set_vst3_preset` between Articulation definitions.

Articulations

An **Instrument** can include multiple **Articulations**, which NotePerformer uses to shape musical phrasing. Every Instrument must have at least a **"sustain" Articulation**, which serves as its default sound.

Articulations are added to `playback_engine.txt` using a **standardized format** that requires three parameters separated by commas (`,`):

```
add_sustain(NOTEON=1;CC21=127, CC1, 1)
add_legato(NOTEON=0;CC21=127, CC1, 126)
add_staccato(NOTEON=2, VEL, 1)
```

Each **Articulation definition** consists of the following three parameters:

1. **MIDI queue** – The sequence of MIDI events (such as key switches or CC changes) needed to trigger the Articulation. Events are separated by semicolons (`;`).
2. **Dynamic controller** – Specifies how dynamics are controlled (`VEL` for velocity or `CCx` for a MIDI CC, such as `CC1` for the Modulation Wheel or `CC11` for the Expression Pedal).
3. **Default velocity** – Defines the **attack velocity** (or **transition velocity** for legato/portamento) on a scale from **1 to 127**. This parameter is ignored for velocity-controlled Articulations.

Understanding the MIDI Queue (Parameter 1)

The **MIDI queue** defines how NPPE switches between Articulations in a VST3 preset. It consists of **one or more MIDI events**, separated by semicolons (`;`).

There are three types of MIDI events:

- **NOTEON** – A key switch (e.g., `NOTEON=1` for key switch 1).
- **CC (Control Change)** – Adjusts a MIDI controller (e.g., `CC21=127` sets CC21 to 127).
- **CHANNEL** – Sets the MIDI channel for multi-channel articulation switching (e.g., `CHANNEL=4` routes the articulation to MIDI channel 4)

It is essential to match the *MIDI queue* to the *VST3 preset* for Articulations to function correctly. Engine Tools provides **helper features** to test whether **the Articulation is triggered by your MIDI queue**.

The Dynamic Controller (Parameter 2)

The second parameter specifies **how the Articulation's dynamics are controlled**:

- **`VEL`** – Dynamics are controlled by **key velocity**.
- **`CCx`** – A specific **MIDI CC** controls dynamics (e.g., `CC1` for the Modulation Wheel or `CC11` for the Expression Pedal).

The Default Velocity (Parameter 3)

The third parameter defines **note velocity** for **CC-controlled Articulations**, with values ranging from **1 to 127**. Some sample libraries use note velocity to shape the **attack of the note**, making it crucial to select a velocity that produces the desired sound for the **Playback Engine**.

For **legato and portamento Articulations**, this parameter instead represents **transition velocity**, which may affect **legato transition speed** in some sample libraries.

If the **Articulation is velocity-controlled**, this parameter is **ignored**, as **dynamics** already determines key velocity.

Analyzing the "My Flute" Articulations

1. Sustain Articulation (Default Sound)

```
add_sustain(NOTEON=1;CC21=127, CC1, 1)
```

- **MIDI queue:** Triggers key switch `1` and sets `CC21` to `127` (activating vibrato in BBCSO Core Flute).
- **Dynamic control:** Uses `CC1` (Modulation Wheel).
- **Default velocity:** Set to `1` to avoid staccato overlays (though BBCSO Core Flute does not use them).

2. Legato Articulation (Legato Transition Samples)

```
add_legato(NOTEON=0;CC21=127, CC1, 126)
```

- **MIDI queue:** Uses key switch `0` and `CC21=127` (vibrato enabled).

- **Dynamic control:** Uses `CC1` (Modulation Wheel).
- **Default velocity:** Set to `126` (affecting transition speed if the library supports velocity-sensitive legato). BBCSO Core Flute **does not use velocity-sensitive legato**, so this setting is irrelevant.

Important Notes:

- NPPE always uses the **sustain** Articulation for the **first note** under a slur.
- The **legato** and **portamento** Articulations are only applied for **note transitions** and are relevant **only if the sample library includes true legato** (i.e., actual recorded transition samples).
- If the **library does not support true legato**, the **legato Articulation should not be added.** NPPE will fall back on **sustain samples** for slurred passages.

3. Staccato Articulation (Short Notes)

```
add_staccato(NOTEON=2, VEL, 1)
```

- **MIDI queue:** Uses key switch `2`, but **does not set CC21** (BBCSO Core Flute does not have vibrato control for staccato samples).
- **Dynamic control:** Uses `VEL` (velocity-controlled dynamics).
- **Default velocity:** Set to `1`, but since this Articulation is velocity-controlled, the value is ignored.

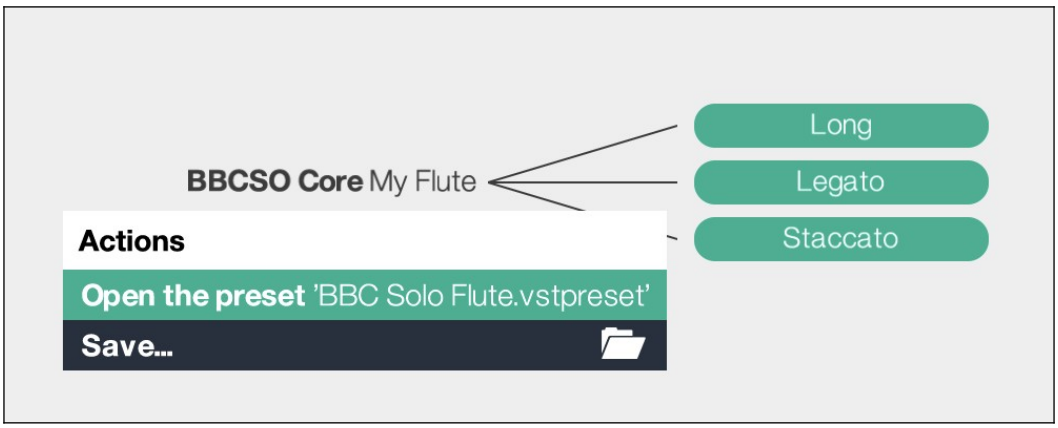
Important Note:

- The term "**staccato**" in NPPE does **not** strictly refer to staccato markings. This Articulation may be used for **any short note**, even if it's not explicitly marked as staccato in the score.

Creating a VST3 Preset

Open the VST3 preset in Engine Tools by **clicking on the Articulation** and selecting the relevant option from the popup menu.

If the VST3 preset does not exist yet, an empty plug-in instance will open, allowing you to create the preset.



Opening a VST3 plug-in and preset.

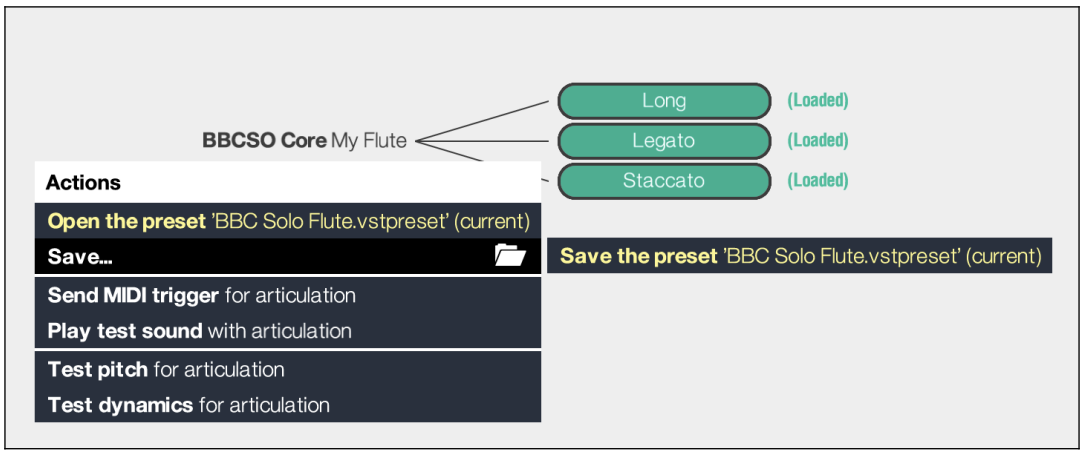
The VST3 plug-in opens in a new window.

When a VST3 window is open, it receives MIDI from all MIDI input devices. **On Mac, this includes *all* devices**, while **on Windows, it only applies to *free* devices** not in use by other applications. Since **Engine Tools** is just a development environment, this feature **cannot be turned off**.

You can only open a VST3 plug-in if your notation program is running **NotePerformer** in the background. The plug-in's audio output is routed through your notation program.

Saving a VST3 Preset

When you're done setting up the VST3 plug-in, you can **save the preset**.



Saving a VST3 preset

Engine Tools saves a **VST3 preset** with the correct file path, as defined in `playback_engines.txt`, making it more convenient to create presets through **Engine Tools** rather than a DAW.

When a **VST3 plug-in** is open, the **Articulation popup menu** provides several **helper features**:

- **Send MIDI trigger** – Sends the **MIDI queue** for the Articulation. This allows you to verify that the correct Articulation is triggered in the **VST3 plug-in's interface**.
- **Play test sound** – Sends the **MIDI queue** and plays a note, helping you confirm that the correct sound is used for this Articulation.
- **Test pitch** – Plays a **sawtooth wave** followed by the test sound. This helps verify that the sound is in the **correct octave** and has not been transposed.
- **Test dynamics** – Plays a series of test notes with different dynamics, allowing you to check if **dynamics change correctly** between notes.

The available helper features may vary depending on the selected Articulation and its settings.

Creating an Unpitched-Percussion Instrument

Unpitched percussion is created differently from pitched Instruments. Below is an example of a basic unpitched-percussion Instrument as it may appear in `playback_engines.txt`.

```
begin_percussion_instrument(Bass Drum Ludwig 40 inch)
set_vst3_preset(HOOPUS Bass Drum Ludwig 40in.vstpreset)
add_bass_drum(38, CHANNEL=1, VEL, 1)
add_bass_drum_roll(41, CHANNEL=1, CC1, 127)
add_bass_drum_short(36, CHANNEL=1, VEL, 1)
```

Let's highlight the differences from an ordinary Instrument:

- We call `begin_percussion_instrument` instead of `begin_instrument`.
- We don't need `set_noteperformer_override_sound`.
- We add Articulations by their *sound*, e.g., `bass_drum`.
- Articulations wants *an extra parameter*: the **MIDI key that triggers the associated sound in the VST3 preset**.

The unpitched-percussion Instrument may span multiple sounds, e.g., an NPPE tom-tom Instrument would often map different variants (`_low`, `_high`, `_medium_low`, etc.).

Theoretically, unrelated sounds (such as bass drum and cymbals) can share an Instrument, but we advise against it since it limits the NPPE end-user to using *both* sounds and may be confusing.

There are three variants for unpitched percussion sounds.

- The **standard hit** (no suffix).
- The **roll** variant (using the `_roll` suffix).
- The **short/damped/muted** variant(using the `_short` suffix).

NPPE automatically produces short notes and rolls. NPPE may trigger a roll if *the note is repeated*, or a short hit if *the note is sufficiently short*.

This guide includes a list of available unpitched-percussion commands.

Preloading

NPPE uses a preloading system to optimize playback performance. Instead of streaming samples in real time, NPPE processes each instrument in advance, building a **database with metadata and preloaded samples**. This allows NPPE Instruments to **load almost instantly** while maintaining **efficient CPU and RAM usage**.

Preloading is a **resource-intensive process** that occurs **locally on the end-user's computer** the first time they use a Playback Engine. Depending on the **Playback Engine's size** and the **VST3 instrument's performance**, it can take anywhere from **30 minutes to several hours**. During this process, NPPE scans the VST3 instrument, **extracts essential samples**, and generates metadata—such as **timing, tuning, and articulation information**. The processed data is then cached as **encrypted** `.np_preload` **files** in the Playback Engine's designated folder, with progress logged in a `/logs` subfolder.

Once preloading is complete, a **single-microphone orchestral library** typically requires **around 2 gigabytes of RAM**, while a **three-microphone orchestral library** uses **about 6 gigabytes**. Since all samples are **fully loaded into RAM**, no **disk streaming** is involved. The CPU load remains **comparable to baseline NotePerformer**, ensuring a **smooth and efficient playback experience**.

Another key advantage of preloading is that **the VST3 plugin is no longer needed for real-time playback**. This eliminates the risk of **crashes, latency spikes, or performance issues** that could arise from running a VST3 instrument live. Offloading the VST3 instrument also **reduces the maintenance burden on VST3 developers**, as they no longer need to optimize their software specifically for NPPE compatibility.

By having all essential samples **preloaded and analyzed in advance**, NotePerformer gains a **complete overview** of the available sounds, including their **tuning, duration, volume, and articulations**. This allows NPPE to **make informed musical decisions**, selecting the most suitable samples for a passage **before playback begins**. The result is a **more natural and expressive performance**, as NPPE **proactively chooses the best sounds** rather than reacting in real time with **limited or outdated information**.

For **security and licensing reasons**, `.np_preload` files **require access to the original VST3 library**, ensuring the end-user has a valid license. Additionally, these files are **encrypted and cannot be shared between systems**—they must be **generated locally** using a licensed **VST3 instrument**.

Command List

Playback Engine Commands

- `begin_playback_engine(DISPLAY_NAME)`
- `set_vst3_name(VST3_PLUGIN_NAME)`
- `set_vst3_id(VST3_ID_STRING)`
- `set_microphone_configuration(MIC1, [OPTIONAL_MIC2], [OPTIONAL_MIC3])`
- `set_default_microphone_balance(MIC1, MIC2, [OPTIONAL_MIC3])`
- `set_recording_location(STUDIO_NAME)`
- `set_developer(DEVELOPER_NAME)`
- `set_release_year(YEAR)`
- `set_website(URL)`
- `set_cover_art(PNG_FILENAME)`
- `add_recommended_software_version(SOFTWARE_VERSION_STRING)`

Product Commands

- `begin_product(SHORT_PRODUCT_NAME)`
- `set_long_product_name(LONG_PRODUCT_NAME)`

Instrument Commands

- `begin_instrument(INSTRUMENT_NAME)`
- `begin_percussion_instrument(INSTRUMENT_NAME)`
- `set_noteperformer_override_sound(NP_OVERRIDE_SOUND)`
- `set_vst3_preset(PRESET_FILENAME)`
- `set_layered_vst3_presets(PRESET_FILENAME1, PRESET_FILENAME2, ...)`
- `set_eq_low_shelf(DB_GAIN, FREQUENCY)`
- `set_eq_peak1(DB_GAIN, FREQUENCY, Q_VALUE)`
- `set_eq_peak2(DB_GAIN, FREQUENCY, Q_VALUE)`
- `set_eq_high_shelf(DB_GAIN, FREQUENCY)`
- `set_eq_high_pass(FREQUENCY)`
- `set_eq_saturation(PERCENTAGE)`
- `set_eq_output_gain(DB_GAIN)`
- `set_next_articulation_force_notes(NOTE1, NOTE2, NOTE3, ...)`
- `set_next_articulation_force_velocities(VELOCITY1, VELOCITY2, ...)`
- `set_next_articulation_transpose(NUM_SEMITONES)`
- `set_next_articulation_offset_velocity_layers_db(DB_OFFSET)`
- `set_next_articulation_limit_midi_keyboard_range(LOW_NOTE, HIGH_NOTE)`
- `set_next_articulation_limit_midi_dynamics_range(LOW_VEL, HIGH_VEL)`

Special Commands

- `set_default_eq_saturation(AMOUNT)`
- `preserve_tuning()`
- `preserve_timing()`
- `use_double_bitrate()`
- `disable_round_robins()`
- `disable_dynamic_crossfading()`
- `default_to_original_panning()`
- `trills_are_triggered_by_two_notes()`
- `force_full_length_shorts()`

Solo Articulations

- `add_col_legno(MIDI_QUEUE, DYN, [VEL])`
- `add_cuivre(MIDI_QUEUE, DYN, [VEL])`
- `add_flutter(MIDI_QUEUE, DYN, [VEL])`

- add_legato(MIDI_QUEUE, DYN, [VEL])
- add_long_accent(MIDI_QUEUE, DYN, [VEL])
- add_long_flutter(MIDI_QUEUE, DYN, [VEL])
- add_long_harmonics(MIDI_QUEUE, DYN, [VEL])
- add_long_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
- add_long_sul_tasto(MIDI_QUEUE, DYN, [VEL])
- add_molto_vib(MIDI_QUEUE, DYN, [VEL])
- add_non_vib(MIDI_QUEUE, DYN, [VEL])
- add_pizzicato(MIDI_QUEUE, DYN, [VEL])
- add_portamento(MIDI_QUEUE, DYN, [VEL])
- add_short_harmonics(MIDI_QUEUE, DYN, [VEL])
- add_short_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
- add_short_sul_tasto(MIDI_QUEUE, DYN, [VEL])
- add_snap_pizzicato(MIDI_QUEUE, DYN, [VEL])
- add_staccatissimo(MIDI_QUEUE, DYN, [VEL])
- add_staccato(MIDI_QUEUE, DYN, [VEL])
- add_sustain(MIDI_QUEUE, DYN, [VEL])
- add_tremolo(MIDI_QUEUE, DYN, [VEL])
- add_tremolo_harmonics(MIDI_QUEUE, DYN, [VEL])
- add_tremolo_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
- add_tremolo_sul_tasto(MIDI_QUEUE, DYN, [VEL])
- add_trill_half(MIDI_QUEUE, DYN, [VEL])
- add_trill_whole(MIDI_QUEUE, DYN, [VEL])

Section Articulations

- add_section_col_legno(MIDI_QUEUE, DYN, [VEL])
- add_section_cuivre(MIDI_QUEUE, DYN, [VEL])
- add_section_flutter(MIDI_QUEUE, DYN, [VEL])
- add_section_legato(MIDI_QUEUE, DYN, [VEL])
- add_section_long_accent(MIDI_QUEUE, DYN, [VEL])
- add_section_long_flutter(MIDI_QUEUE, DYN, [VEL])
- add_section_long_harmonics(MIDI_QUEUE, DYN, [VEL])
- add_section_long_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
- add_section_long_sul_tasto(MIDI_QUEUE, DYN, [VEL])
- add_section_molto_vib(MIDI_QUEUE, DYN, [VEL])
- add_section_non_vib(MIDI_QUEUE, DYN, [VEL])
- add_section_pizzicato(MIDI_QUEUE, DYN, [VEL])
- add_section_portamento(MIDI_QUEUE, DYN, [VEL])
- add_section_short_harmonics(MIDI_QUEUE, DYN, [VEL])
- add_section_short_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
- add_section_short_sul_tasto(MIDI_QUEUE, DYN, [VEL])
- add_section_snap_pizzicato(MIDI_QUEUE, DYN, [VEL])
- add_section_staccatissimo(MIDI_QUEUE, DYN, [VEL])
- add_section_staccato(MIDI_QUEUE, DYN, [VEL])
- add_section_sustain(MIDI_QUEUE, DYN, [VEL])
- add_section_tremolo(MIDI_QUEUE, DYN, [VEL])
- add_section_tremolo_harmonics(MIDI_QUEUE, DYN, [VEL])
- add_section_tremolo_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
- add_section_tremolo_sul_tasto(MIDI_QUEUE, DYN, [VEL])
- add_section_trill_half(MIDI_QUEUE, DYN, [VEL])
- add_section_trill_whole(MIDI_QUEUE, DYN, [VEL])

Unpitched-Percussion Articulations

- add_agogo_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_agogo_medium(KEY, MIDI_QUEUE, DYN, [VEL])
- add_anvil(KEY, MIDI_QUEUE, DYN, [VEL])
- add_bar_chimes(KEY, MIDI_QUEUE, DYN, [VEL])
- add_bass_drum(KEY, MIDI_QUEUE, DYN, [VEL])
- add_bell_tree(KEY, MIDI_QUEUE, DYN, [VEL])
- add_cabasa(KEY, MIDI_QUEUE, DYN, [VEL])
- add_castanets(KEY, MIDI_QUEUE, DYN, [VEL])
- add_china_cymbal(KEY, MIDI_QUEUE, DYN, [VEL])

- add_clash_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_clash_medium(KEY, MIDI_QUEUE, DYN, [VEL])
- add_claves(KEY, MIDI_QUEUE, DYN, [VEL])
- add_cowbell_high_muted(KEY, MIDI_QUEUE, DYN, [VEL])
- add_cowbell_high_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_cowbell_medium_muted(KEY, MIDI_QUEUE, DYN, [VEL])
- add_cowbell_medium_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_egg_shaker(KEY, MIDI_QUEUE, DYN, [VEL])
- add_finger_cymbals_closed(KEY, MIDI_QUEUE, DYN, [VEL])
- add_finger_cymbals_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_gong(KEY, MIDI_QUEUE, DYN, [VEL])
- add_guiro_long(KEY, MIDI_QUEUE, DYN, [VEL])
- add_guiro_short(KEY, MIDI_QUEUE, DYN, [VEL])
- add_hand_clap(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_bongo_bass_tone(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_bongo_finger_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_bongo_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_bongo_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_bongo_slap(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_bongo_slap_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_bass_tone(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_finger_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_finger_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_harmonics(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_mute(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_slap(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_slap_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_high_conga_slap_mute(KEY, MIDI_QUEUE, DYN, [VEL])
- add_hihat_closed(KEY, MIDI_QUEUE, DYN, [VEL])
- add_hihat_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_hihat_pedal(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_bongo_bass_tone(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_bongo_finger_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_bongo_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_bongo_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_bongo_slap(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_bongo_slap_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_bass_tone(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_finger_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_finger_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_harmonics(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_mute(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_slap(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_slap_muffled(KEY, MIDI_QUEUE, DYN, [VEL])
- add_low_conga_slap_mute(KEY, MIDI_QUEUE, DYN, [VEL])
- add_maracas_left(KEY, MIDI_QUEUE, DYN, [VEL])
- add_maracas_right(KEY, MIDI_QUEUE, DYN, [VEL])
- add_opera_gong_down(KEY, MIDI_QUEUE, DYN, [VEL])
- add_opera_gong_up(KEY, MIDI_QUEUE, DYN, [VEL])
- add_rain_stick(KEY, MIDI_QUEUE, DYN, [VEL])
- add_ratchet(KEY, MIDI_QUEUE, DYN, [VEL])
- add_ride_bell(KEY, MIDI_QUEUE, DYN, [VEL])
- add_ride_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_ride_medium(KEY, MIDI_QUEUE, DYN, [VEL])
- add_sleigh_bells(KEY, MIDI_QUEUE, DYN, [VEL])
- add_snare(KEY, MIDI_QUEUE, DYN, [VEL])
- add_snare_cross_stick(KEY, MIDI_QUEUE, DYN, [VEL])
- add_snare_rim_shot(KEY, MIDI_QUEUE, DYN, [VEL])
- add_snare_side_stick(KEY, MIDI_QUEUE, DYN, [VEL])
- add_snares_off(KEY, MIDI_QUEUE, DYN, [VEL])

- add_snares_off_rim_shot(KEY, MIDI_QUEUE, DYN, [VEL])
- add_snares_off_side_stick(KEY, MIDI_QUEUE, DYN, [VEL])
- add_splash_cymbal(KEY, MIDI_QUEUE, DYN, [VEL])
- add_sticks(KEY, MIDI_QUEUE, DYN, [VEL])
- add_suspended_cymbal(KEY, MIDI_QUEUE, DYN, [VEL])
- add_taiko_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_taiko_medium(KEY, MIDI_QUEUE, DYN, [VEL])
- add_taiko_very_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tam_tam_medium(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tambourine(KEY, MIDI_QUEUE, DYN, [VEL])
- add_temple_block_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_temple_block_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_temple_block_medium(KEY, MIDI_QUEUE, DYN, [VEL])
- add_temple_block_medium_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_temple_block_medium_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_timbale_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_timbale_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tom_tom_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tom_tom_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tom_tom_medium(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tom_tom_medium_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tom_tom_medium_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tom_tom_very_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_tom_tom_very_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_triangle_muted(KEY, MIDI_QUEUE, DYN, [VEL])
- add_triangle_open(KEY, MIDI_QUEUE, DYN, [VEL])
- add_vibraslap(KEY, MIDI_QUEUE, DYN, [VEL])
- add_whip(KEY, MIDI_QUEUE, DYN, [VEL])
- add_whistle(KEY, MIDI_QUEUE, DYN, [VEL])
- add_wind_gong(KEY, MIDI_QUEUE, DYN, [VEL])
- add_woodblock_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_woodblock_low(KEY, MIDI_QUEUE, DYN, [VEL])
- add_woodblock_medium(KEY, MIDI_QUEUE, DYN, [VEL])
- add_woodblock_medium_high(KEY, MIDI_QUEUE, DYN, [VEL])
- add_woodblock_medium_low(KEY, MIDI_QUEUE, DYN, [VEL])

Command Reference

`begin_playback_engine(DISPLAY_NAME)` *Required*

Defines the Playback Engine and sets its name as visible to end-users in NPPE.

Example: `begin_playback_engine(BBC Symphony Orchestra Core)`

`set_vst3_name(VST3_PLUGIN_NAME)` *Required*

Describes the required VST3 plugin's name as visible to end-users in NPPE.

Example 1: `set_vst3_name(Kontakt 7 Player)`
Example 2: `set_vst3_name(BBC Symphony Orchestra)`
Example 3: `set_vst3_name(Opus)`

`set_vst3_id(VST3_ID_STRING)` *Required*

Specifies the unique identifier for the VST3 plugin. The identifier can be fetched from any `.vst3` file using Engine Tools.

If applicable, this command can take multiple identifiers. For example, Kontakt 7 and 8 have different VST3 identifiers but both loads Kontakt 7-saved VST3 presets.

Example 1: `set_vst3_id(5653544E694B376B6F6E74616B742037)`
Example 2: `set_vst3_id(5653544E694B376B6F6E74616B742037, [ID2])`

`set_cover_art(PNG_FILENAME)` *Required*

Sets the `.png` filename for the Playback Engine's front cover image.

The `.png` file must be **exactly 372 x 526 pixels** and in the same folder as the `playback_engine.txt` file.

Example: `set_cover_art(poster_bbcso_core.png)`

`set_microphone_configuration(MIC1, [MIC2], [MIC3])` *Required*

This command determines if 1, 2, or 3 stereo outputs should be used from the VST3. The end-user may graphically adjust the microphone balance in NPPE, using a slider for 2-channel Instruments and a triangle for 3-channel Instruments.

The parameters become labels for the outputs, as visible to NPPE end-users.

Your VST3 preset's mixer levels are irrelevant since the volume is adaptive, being automatically balanced by NPPE.

Two Instruments in the same Playback Engine may have different microphone configurations by running this command between Instruments in `playback_engine.txt`.

Example 1: `set_microphone_configuration(Mix 1)`
Example 2: `set_microphone_configuration(Tree, Close)`
Example 3: `set_microphone_configuration(Tree, Close, Ambient)`

`set_default_microphone_balance(MIC1, [MIC2], [MIC3])` *Optional* *Advanced*

Sets the default microphone balance, as presented in the slider or triangle editor for the NPPE end-user when Slots are added.

These values define the proportions between microphones, but they are always normalized to ensure the overall balance remains consistent.

Example 1: `set_default_microphone_balance(1.0, 0.0)`
Example 2: `set_default_microphone_balance(1.0, 0.5, 0.5)`

`set_recording_location(STUDIO_NAME)` *Optional*

Describes the recording location as visible to end-users in NPPE (if applicable.)

Example: `set_recording_location(Maida Vale Studios)`

`set_developer(DEVELOPER_NAME)` *Required*

Describes the VST3 developer as visible to end-users in NPPE.

Example: `set_developer(Spitfire Audio)`

`set_release_year(YEAR)` *Required*

Describes the VST3's release year as visible to end-users in NPPE.

Example: `set_release_year(2020)`

`set_website(URL)` *Required*

Describes the VST3's release year as visible to end-users in NPPE.

Example: `set_website(https://www.spitfireaudio.com/bbc-symphony-orchestra-core)`

`add_recommended_software_version(SOFTWARE_VERSION_STRING)` *Required*

Describes a recommended software version as visible to end-users in NPPE.

This command can be called multiple times to list different relevant versions or products, such as a sample library version and a VST3 version.

Example:

```
add_recommended_software_version(Kontakt Player 7.6)
add_recommended_software_version(Cinematic Studio Brass 1.0.0)
add_recommended_software_version(Cinematic Studio Brass 1.0.0)
```

Product Command Reference

`begin_product(SHORT_PRODUCT_NAME)` *Required*

Defines a Product and sets its short name/prefix as visible to end-users in NPPE.

Example: `begin_product(BBCSO Core)`

`set_long_product_name(LONG_PRODUCT_NAME)` *Required*

Defines a Long Product Name as visible to end-users in NPPE.

Example: `set_long_product_name(BBC Symphony Orchestra Core)`

Instrument Command Reference

`begin_instrument(INSTRUMENT_NAME)` *Required*

Defines an Instrument and sets name as visible to NPPE end-users.

Example: `begin_instrument(My Flute)`

`begin_percussion_instrument(INSTRUMENT_NAME)` *Optional*

Defines an unpitched percussion Instrument and sets name as visible to NPPE end-users.

Example: `begin_percussion_instrument(My Snare Drum)`

`set_noteperformer_override_sound(NP_OVERRIDE_SOUND)` *Required*

This command isn't required for unpitched percussion Instruments.

The choice of Override Sound is critical because it determines how NotePerformer routes programs to NPPE Slots, the orchestral balance, and the expected sample properties such as looping vs. fading out (e.g., don't override NotePerformer's *Flute* program with piano sounds.)

This guide includes a list of available Override Sounds. If the Override Sound is misspelled, Engine Tools will display an error.

Example: `set_noteperformer_override_sound(Flute)`

`set_vst3_preset(PRESET_FILENAME)` *Required*

Sets a VST3 preset which is used for subsequently added Articulations.

The VST3 preset is remembered until this command is run again. The same VST3 preset can be re-used across multiple Articulations or Instruments. The VST3 preset can also change between Articulations, e.g., drawing legato and staccato notes from different VST3 presets.

Example: `set_vst3_preset(BBC Solo Flute.vstpreset)`

`set_layered_vst3_presets(PRESET1, PRESET2, ...)` *Optional* *Advanced*

Layers VST3 presets by opening multiple plug-in instances.

NPPE follows a universal layout, distinguishing between solo and section sounds rather than using multiple solo instruments. This command can be used to generate section sounds by layering multiple solo instruments, if the library provides multiple soloists instead of section samples.

Another use for this command is layering solo and section sounds in sample libraries where the two differ too much. In such cases, layering a soloist with the section can create a more uniform sound, ensuring that the section remains cohesive while NPPE alternates between solo and section sounds within a phrase.

For this command to work correctly, all layered presets must have an identical MIDI mapping and be balanced in volume to taste.

Example: `set_layered_vst3_presets(Solo Flute 1.vstpreset, Solo Flute 2.vstpreset)`

Equalizer Commands:

```
set_eq_low_shelf(DB_GAIN, FREQUENCY)
set_eq_peak1(DB_GAIN, FREQUENCY, Q_VALUE)
set_eq_peak2(DB_GAIN, FREQUENCY, Q_VALUE)
set_eq_high_shelf(DB_GAIN, FREQUENCY)
set_eq_high_pass(FREQUENCY)
set_eq_saturation(PERCENTAGE)
set_eq_output_gain(DB_GAIN)
```

Optional *Advanced*

These commands override the Instrument EQ's default settings.

EQ settings may be *copied* from the NPPE interface by selecting **Copy equalizer to clipboard** and pasted as text into `playback_engine.txt`. It's more convenient than entering these values by hand.

Example:

```
set_eq_peak1(-2.5, 4200.0, 0.5)
set_eq_peak2(4.7, 9750.0, 1.0)
set_eq_saturation(5.0)
```

`set_next_articulation_force_notes(NOTE1, NOTE2, NOTE3, …)`
Optional *Advanced*

Force notes (MIDI pitches) to be preloaded by the upcoming Articulation.

This is often redundant. NPPE automatically scans the VST3 instrument for relevant notes and velocity levels to include with the Articulation's mapping.

This command may be used to exclude unwanted samples from a VST3 instrument when there are quality problems. NPPE will play any note by pitch-shifting the nearest sample.

Example: `set_next_articulation_force_notes(48, 52, 57, 61, 65, 67, 69, 74, 79, 81)`

`set_next_articulation_force_velocities(VEL1, VEL2, VEL3, …)`
Optional *Advanced*

Force velocity levels (or CC levels) to be preloaded by the upcoming Articulation.

This is often redundant. NPPE automatically scans the VST3 instrument for relevant notes and velocity levels to include with the Articulation's mapping.

This command can be used to force a specific set of dynamic levels.

Example: `set_next_articulation_force_velocities(20, 85, 127)`

`set_next_articulation_transpose(NUM_SEMITONES)` *Optional* *Advanced*

Transpose the upcoming Articulation.

Some VST3 instruments play in the wrong octave and must be transposed to the correct octave before preloading. This is critical, since preloading involves pitch-sensitive signal-processing technologies such as pitch detection.

Use the **Test pitch** feature in Engine Tools to compare the VST3's pitch against a sawtooth wave. If necessary, use this command to transpose the Articulation until it matches the sawtooth wave.

Example: `set_next_articulation_transpose(12)`

`set_next_articulation_offset_velocity_layers_db(DB_OFFSET)`
Optional *Advanced*

Offsets the dynamic layers for the next Articulation.

This is often redundant. NPPE analyzes the VST3 instrument's dynamics and automatically adapts the velocity layers to NotePerformer. We advise against using this command unless you know exactly what you're doing.

Suppose that your VST3 instrument has a problematic dynamic range, which results in the Articulation playing the fortissimo timbre at a much too high or low dynamic for your tastes. This command may offset the velocity level boundaries up or down.

This function is measured in decibels (dB) for internal reasons. In practice, it can only be calibrated by ear. Play a note at a fixed dynamic repeatedly while adjusting the setting until it selects the desired velocity layer.

Note: The volume balance remains unchanged; only the sample selection boundaries are adjusted. As a result, changes are inaudible unless a velocity layer boundary is crossed.

Example 1: `set_next_articulation_offset_velocity_layers_db(-3.0)`
Example 2: `set_next_articulation_offset_velocity_layers_db(7.5)`

`set_next_articulation_limit_midi_keyboard_range(LOW_NOTE, HIGH_NOTE)`
Optional *Advanced*

Limits the keyboard range preloaded by the next Articulation.

Some VST3 instruments are mapped in non-standard ways, such as timpani *hits* and *rolls* sharing the same program but in different octaves. This command can exclude the *roll* samples from your *hits* Articulation, and vice versa.

Example: `set_next_articulation_limit_midi_keyboard_range(48, 72)`

`set_next_articulation_limit_midi_dynamics_range(LOW_DYN, HIGH_DYN)`
Optional *Advanced*

Limits the dynamic range preloaded by the next Articulation.

Some VST3 instruments are mapped in non-standard ways, such as *pizzicato* using velocities 1-126 and *snap pizzicato* using velocity 127. This command can exclude the *snap pizzicato* samples from your *pizzicato* Articulation, and vice versa.

Example: `set_next_articulation_limit_midi_dynamics_range(1, 126)`

Special Command Reference

`set_default_eq_saturation(AMOUNT)` *Optional* *Advanced*

Sets the default equalizer saturation amount (on a scale of 0 to 100).

The flag applies to all subsequent Instruments.

Example: `set_default_eq_saturation(5.0)`

`preserve_tuning()` *Optional* *Advanced*

Turns off NPPE's automatic tuning correction and uses the VST3 instrument's tuning.

The flag applies to all subsequent Instruments.

Example: `preserve_tuning()`

`preserve_timing()` *Optional* *Advanced*

Turns off NPPE's automatic timing correction and uses the VST3 instrument's note timing.

The flag applies to all subsequent Instruments.

Example: `preserve_timing()`

`use_double_bitrate()` *Optional* *Advanced*

Uses a twice-as-high bitrate for preloaded sounds.

This is often redundant, but the flag may resolve quality problems with a Playback Engine, at the expense of doubling RAM use, CPU use, and loading time for the end-user.

The flag applies to all subsequent Instruments.

Example: `use_double_bitrate()`

`disable_round_robins()` *Optional* *Advanced*

Disables round-robin samples.

By default, NPPE may strategically employ round-robin samples if the VST3 instrument includes it, to avoid the machine-gun effect with repeated notes. If there are quality problems, you can disable round-robins with this command. NPPE will then use the same sample for repeated notes.

The flag applies to all subsequent Instruments.

Example: `disable_round_robins()`

`disable_dynamic_crossfading()` *Optional* *Advanced*

Disables dynamic crossfading between samples.

By default, NPPE handles hairpins by strategically crossfading between dynamic layers if the VST3 instrument includes it. If there are quality problems, you can turn off dynamic crossfading with this command. NPPE will then use the same sample throughout a note and only respond to hairpins by changing the volume.

The flag applies to all subsequent Instruments.

Example: `disable_dynamic_crossfading()`

`default_to_original_panning()` *Optional* *Advanced*

Sets the Instrument's default panning mode to **Original panning from recording**.

By default, an NPPE's default panning mode is **Natural panning (with HAAS delay)**, a hybrid-mode that allow natural-sounding left/right panning, even if the VST3 sample library was recorded *in situ* (natural orchestral seating).

This command sets the default mode to **Original panning from recording**; a panning mode that preserves the VST3's *in situ* panning. The end-user is limited to placing an Instrument *either* left or right, since NPPE will only flip the left and right channels for end-users wanting the Instrument on the opposite side.

This is just a convenience command, since the Instrument's panning mode can also be changed from the NPPE interface.

The flag applies to all subsequent Instruments.

Example: `default_to_original_panning()`

`trills_are_triggered_by_two_notes()` *Optional* *Advanced*

Specifies that trills are triggered by holding an interval of two notes.

Some sample libraries, e.g., the *Cinematic Studio Series*, uses the interval of two MIDI notes to trigger trills (e.g., whole-tone trills are triggered by holding a whole-tone interval and half-tone trills by a half-tone interval.)

The flag applies to all subsequent Instruments.

Example: `trills_are_triggered_by_two_notes()`

`force_full_length_shorts()` *Optional* *Advanced*

Forces short notes (e.g., staccato) to play their full-duration samples, including any recorded ambiance.

NotePerformer doesn't rely on "release samples" but may truncate samples early to ensure musical cohesion between articulations. Any sample library will sound dry in NotePerformer until reverb is added. Users who prefer to hear short notes in their full, unaltered form can use this option to force full-duration shorts. It can be activated on the fly to A/B test the difference during playback.

The flag applies to all subsequent Instruments.

Example: `force_full_length_shorts()`

Articulation Command Reference

Long Articulations

`add_sustain(MIDI_QUEUE, DYN, [VEL])` *Required*

```
add_non_vib(MIDI_QUEUE, DYN, [VEL])
add_molto_vib(MIDI_QUEUE, DYN, [VEL])
add_tremolo(MIDI_QUEUE, DYN, [VEL])
add_long_flutter(MIDI_QUEUE, DYN, [VEL])
add_trill_half(MIDI_QUEUE, DYN, [VEL])
add_trill_whole(MIDI_QUEUE, DYN, [VEL])
add_long_harmonics(MIDI_QUEUE, DYN, [VEL])
add_long_accent(MIDI_QUEUE, DYN, [VEL])
add_long_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
add_long_sul_tasto(MIDI_QUEUE, DYN, [VEL])
add_tremolo_harmonics(MIDI_QUEUE, DYN, [VEL])
add_tremolo_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
add_tremolo_sul_tasto(MIDI_QUEUE, DYN, [VEL])
```

NPPE automatically picks the best available Articulation and falls back on other sounds when it must.

The sustain articulation is also the default for fading sounds, such as a piano.

`add_long_flutter` is only available for brass and woodwinds. The equivalent for strings and unpitched percussion is `add_tremolo`.

Con-sordino techniques *cannot be mapped* since NPPE generates these sounds (for any VST3 instrument) using signal processing with the standard Articulations.

Example: `add_sustain(CHANNEL=3;NOTE_ON=4;CC19=1, CC1, 127)`

Interval Articulations *Optional*

```
add_legato(MIDI_QUEUE, DYN, TRANSITION_VELOCITY)
add_portamento(MIDI_QUEUE, DYN, TRANSITION_VELOCITY])
```

Interval Articulations should only be added if the library supports true legato (i.e., actual recorded transition samples).

The first note of a slurred passage uses the standard `sustain` Articulation. Interval Articulations are only used for interval samples.

NPPE crossfades between interval and long samples as it sees fit, incorporating various techniques such as `non_vib` or `molto_vib` with slurred passages.

Example: `add_legato(CHANNEL=1;NOTE_ON=3, CC11, 64)`

Special Cuivre Articulation *Optional*

```
add_cuivre(MIDI_QUEUE, DYN, TRANSITION_VELOCITY)
```

Some sample libraries, e.g., Spitfire Audio BBCSO, have a cuivre layer for brass in a separate program.

This special Articulation increases the dynamic range of the sustain and legato Articulations by incorporating cuivre samples. NPPE will automatically use cuivre samples for dynamics above a certain point (which is determined internally.)

Example: `add_cuivre(NOTE_ON=1, CC1, 127)`

Short Articulations *Optional*

```
add_staccato(MIDI_QUEUE, DYN, [VEL])
add_staccatissimo(MIDI_QUEUE, DYN, [VEL])
add_pizzicato(MIDI_QUEUE, DYN, [VEL])
add_col_legno(MIDI_QUEUE, DYN, [VEL])
add_snap_pizzicato(MIDI_QUEUE, DYN, [VEL])
add_short_harmonics(MIDI_QUEUE, DYN, [VEL])
```

```
add_short_sul_ponticello(MIDI_QUEUE, DYN, [VEL])
add_short_sul_tasto(MIDI_QUEUE, DYN, [VEL])
```

NPPE automatically uses short samples for short notes.

NPPE automatically picks the best available Articulation and falls back on other sounds when it must. E.g., if `col_legno` is missing, NPPE will simulate it with `pizzicato` at an appropriate volume.

Staccato is used for *short* notes, while staccatissimo is used for *very short* notes.

Example: `add_staccato(NOTE_ON=2, VEL, 127)`

Section Articulations *Optional*

```
add_section_staccato(MIDI_QUEUE, DYN, [VEL])
add_section_sustain(MIDI_QUEUE, DYN, [VEL])
add_section_legato(MIDI_QUEUE, DYN, [VEL])
...
```

For brass and woodwinds, all Articulations have the equivalent *section* technique.

NPPE selectively uses section and solo samples if provided. The section timre is more chorusing and used for unison notes. NPPE continuously adapts the volume for the written number of players for the right orchestral balance.

You don't use the section Articulations for string instruments, but NPPE provides different *Override Sounds* for section and solo strings.

Example: `add_section_sustain(NOTE_ON=0, CC1, 64)`

Unpitched Percussion Command Reference

Unpitched-Percussion Hits *Required*

```
add_agogo_high(KEY, MIDI_QUEUE, DYN, [VEL])
add_agogo_medium(KEY, MIDI_QUEUE, DYN, [VEL])
add_anvil(KEY, MIDI_QUEUE, DYN, [VEL])
add_bar_chimes(KEY, MIDI_QUEUE, DYN, [VEL])
add_bass_drum(KEY, MIDI_QUEUE, DYN, [VEL])
...
```

NotePerformer's unpitched-percussion sounds may all be replaced by an NPPE Instrument.

Unpitched-percussion Articulations takes an extra command which is the *MIDI key* that's mapped to the associated sound in the VST3. You can use Engine Tools's testing features to ensure the Articulation plays the correct sound.

Example: `add_snare(48, NOTE_ON=2, VEL, 127)`

Unpitched-Percussion Rolls *Optional*

```
add_agogo_high_roll(KEY, MIDI_QUEUE, DYN, [VEL])
add_agogo_medium_roll(KEY, MIDI_QUEUE, DYN, [VEL])
add_anvil_roll(KEY, MIDI_QUEUE, DYN, [VEL])
add_bar_chimes_roll(KEY, MIDI_QUEUE, DYN, [VEL])
add_bass_drum_roll(KEY, MIDI_QUEUE, DYN, [VEL])
...
```

The unpitched-percussion Articulations are also available in **roll** variants, e.g., `add_bass_drum` for a regular hit vs. `add_bass_drum_roll` for tremolo/roll.

Example: `add_snare_roll(49, NOTE_ON=2, CC1, 127)`

Unpitched-Percussion Short/Muted/Damped Hits *Optional*

```
add_agogo_high_short(KEY, MIDI_QUEUE, DYN, [VEL])
add_agogo_medium_short(KEY, MIDI_QUEUE, DYN, [VEL])
add_anvil_short(KEY, MIDI_QUEUE, DYN, [VEL])
add_bar_chimes_short(KEY, MIDI_QUEUE, DYN, [VEL])
add_bass_drum_short(KEY, MIDI_QUEUE, DYN, [VEL])
...
```

The unpitched-percussion Articulations are also available in **short** variants, e.g., `add_bass_drum` for a regular hit vs. `add_bass_drum_short` for staccato/damped/muted.

Example: `add_bass_drum_short(55, CHANNEL=3, VEL, 127)`

NotePerformer Override Sounds

Available Override Sounds

Instruments must call `set_noteperformer_override_sound(STRING)` to link with one of our predefined *Override Sounds*, from the list below.

The Override Sound must be *exactly* specified. If the Override Sound is misspelled, **Engine Tools will display an error** and the engine won't show up on the main NPPE screen.

- Accordion
- Alto (ah)
- Alto (oh)
- Alto flute
- Alto horn
- Alto recorder
- Alto saxophone
- Alto trombone
- Altos (ah)
- Altos (oh)
- Aluphone (bowed)
- Aluphone (mallets)
- Bagpipes
- Bandoneon
- Banjo
- Bansuri
- Baritone horn
- Baritone saxophone
- Bass (ah)
- Bass (oh)
- Bass clarinet
- Bass flute
- Bass recorder
- Bass saxophone
- Bass trombone
- Bass trumpet
- Basses (ah)
- Basses (oh)
- Basset-horn
- Bassoon
- Celesta
- Chimes (bowed)
- Chimes (mallets)
- Choir (ah)
- Choir (oh)
- Choir men (ah)
- Choir men (oh)
- Choir women (ah)
- Choir women (oh)
- Church organ (Great)
- Church organ (Pedal)
- Church organ (Swell)
- Cimbasso
- Clarinet
- Clavinet
- Contrabass clarinet
- Contrabass recorder

- Contrabasses
- Contrabassoon
- Cornet
- Crotales (bowed)
- Crotales (mallets)
- Dizi
- Drawbar organ
- Drum set (brushes)
- Drum set (electronic)
- Drum set (rock)
- Drum set (sticks)
- Dulcimer
- Eb-clarinet
- Electric bass
- Electric guitar
- Electric piano 1
- Electric piano 2
- English horn
- Erhu
- Euphonium
- Flugelhorn
- Flute
- French horn
- Garklein recorder
- Glockenspiel (bowed)
- Glockenspiel (mallets)
- Great bass recorder
- Guzheng
- Hand bells
- Harmonica
- Harp
- Harpsichord
- Koto
- Mandolin
- Marimba (bowed)
- Marimba (mallets)
- Master Live Program NP
- Nylon guitar
- Oboe
- Oboe d'amore
- Percussive organ
- Piano
- Piccolo
- Piccolo trumpet
- Rock organ
- Shakuhachi
- Shamisen
- Sitar
- Solo cello
- Solo contrabass
- Solo viola
- Solo violin
- Sopranino recorder
- Sopranino saxophone
- Soprano (ah)
- Soprano (oh)
- Soprano cornet
- Soprano recorder
- Soprano saxophone
- Sopranos (ah)
- Sopranos (oh)
- Steel-string guitar
- Steelpan
- Synth bass 1

- Synth bass 2
- Synth lead 1
- Synth lead 2
- Synth pad 1
- Synth pad 2
- Taiko
- Tenor (ah)
- Tenor (oh)
- Tenor recorder
- Tenor saxophone
- Tenors (ah)
- Tenors (oh)
- Theremin (classic)
- Theremin (modern)
- Timpani
- Timpani (hard mallets)
- Timpani (soft mallets)
- Trombone
- Trumpet
- Tuba
- Ukulele
- Upright bass
- Upright piano
- Vibraphone (bowed)
- Vibraphone (mallets)
- Violas
- Violins
- Violoncellos
- Wagner tuba
- Xylophone (bowed)
- Xylophone (mallets)

Disclaimer

NPPE provides a framework for integrating VST3 sample libraries into notation-based playback, **but the quality of a Playback Engine ultimately depends on the original sample content and the articulations included in the engine**.

Since NPPE adapts sample libraries for real-time notation playback, it does not guarantee that a Playback Engine will reproduce the sound exactly as it would in a DAW. If an **exact** reproduction of a sample library's sound is needed, the library should instead be loaded directly into the notation software, using your own **Expression Maps** or **Sound Sets** to configure articulations and playback behavior. A DAW is typically required to fully utilize a sample library's capabilities.

NPPE is designed as a user-friendly and efficient way to use VST3 sample libraries with notation programs, but it is not a replacement for full DAW-based playback. Users can experiment with different articulations and features in a Playback Engine to find the best setup for their needs.

Playback Engines may require ongoing updates to VST3 presets for compatibility. **We can't provide or maintain Playback Engines for every sample library**, but we rely on the user community or VST3 developers to maintain and expand the compatibility.