psai Manual

Table of contents

Introduction to psai 4
- Why psai? 4
- How psai works 4
- psai's components 6
  - Themes 6
    - Theme Types 6
    - Table of Theme Types 7
  - Intensity 9
  - Segments 9
    - How Segments are mixed 9
    - Segment suitabilities 10
  - psai States 11
- psai's playback logic 11
  - when a Theme is triggered... 12
  - at the end of a playing Segment... 13
- Best Practice for composers 14

psai for Unity 16
- Getting Started 16
- psai Editor (Light Edition) 16
- Exporting and playing back custom Soundtracks 17
- psai Multi Audio Object Editor 17
- Optimizing performance 17
- Troubleshooting (Unity) 18
- Scripts 18
  - PsaiCoreManager 18
  - PsaiSoundtrackLoader 18
  - PsaiPlayer 19
  - Trigger Scripts 19

psai Player (native SDK only) 19
- Introduction 19
  - Getting Started 19
  - What's new 19
- Functions / Controls 20
  - Trigger Music Theme 20
  - Stop Music 20
  - Return To BasicMood 20
  - Hold Intensity / Resume 21
- Advanced Functions 21
  - Advanced Functions Panel 21
    - Cutscene Mode 21
    - Menu Mode 22
    - State File 22
    - Log Level 22
    - Error Handling 22
  - Quick Controls Window 23
  - Playback Stats Panel 23

psai Editor 23
- Introduction 23
- The Source Tree View 24
- The Target Tree View 25

The Playback Panel 25
The Property Panels 26
Project Property Panel 26
Theme Property Panel 26
Group Property Panel 27
Segment Property Panel 28
Compatibility of Segments 28
Bridge Segments 29
3rd Party Technology / License 31

Introduction to psai

Welcome to psai!
psai is a multi-platform game audio middleware for interactive music. We provide SDKs for:

- Unity (psai sourcecode in C#, no native code involved. Fully compatible with all target platforms, both with Unity Free and Pro.)
- Microsoft Windows (native psai DLLs using either FMOD, XAudio2 or OpenAL)

The psai middleware consists of the following components:

1. The psai API. It provides all functionality to load and play back psai soundtracks that have been authored and exported by the psai Editor.
2. The psai Editor is the standalone application for authoring and exporting psai soundtracks.
3. The psai Player is the standalone application for playing back psai soundtracks. It is not needed for the Unity version of psai. For Unity, please use the psaiPlayer Script that is included in the psai.unitypackage.

Why psai?

My audio middleware already supports interactive music. Why should i use psai?

The concept for psai was designed by film composers, who found that the approaches of conventional game audio middleware to interactive music were either not flexible enough or too complicated, so they could not achieve the desired level of adaptivity and dramaturgy they were used to when composing a film score. This was basically for two reasons:

Problem 1: No intensity

The first problem is that conventional sound middleware solutions do not directly support the concept of musical intensity. Psai uses a segment-based approach to interactive music, where usually the intensity of a segment is the most important criterion upon the evaluation which segment to play next. Basically all the programmer has to do is to keep psai informed about the intensity of the current game situation, simply by periodically passing a single percentage value.

Problem 2: Complexity

The second problem with most conventional audio middleware is that the authoring complexity grows exponentially along with the number of audio clips. In most approaches you need to manually define paths through your musical timeline, making the authoring process both tedious and error prone. In order to keep the soundtrack manageable, one of your options is to make heavy use of looping, what tends to sound boring and repetitive. The other option is to keep the length of your audio clips rather high. This means that your soundtrack doesn't adapt quickly enough to your current game situation.

To tackle these problems, psai uses a different approach to segment-based interactive music, as explained in the next section.

How psai works

How psai achieves interactivity

Psai's concept is based on the dynamic recombination of prerecorded audio Segments, mainly based on their Themes and musical Intensity. While your game progresses, psai generally enqueues those Segments for playback which intensity currently matches the best. For detailed information about psai's concepts and playback logic, please refer to the chapters psai's components and psai's playback logic .

How psai achieves diversity

The selection process of the next Segment can be influenced by various factors, as explained below. The default weightings of these factors work pretty well for most themes, but can easily be fine-tuned per theme if needed.

1. Repetition avoidance

When choosing the next Segment for playback on-the-fly, psai evaluates the number of times that each suitable Segment has been replayed so far. This prevents the same Segments from being played over and over again.

2. Groups of instrumentation

Psai directly supports the concept of separated groups of instrumentations. E.g. if a given Theme contains a lead melody which is played once by violins and once by horns, just assign the related audio files to different Groups within the Theme. Psai will automatically take care of transitions from one instrumentation to another, and will avoid random switches in between.

3. unpredictability (optional)

A random factor can be added to the selection process if desired, for adding more unpredictability and diversity. This is especially useful for themes which are based on atmospheric sound textures rather than harmonic sequences.

How psai helps to keep it simple

1. Use of predefined Theme Types

With psai the complexity of a game score can be much higher than what you are used to from most current games, as psai's authoring concept is based on a small collection of well-defined Theme Types. E.g. themes of type BaseMood are treated as ambient background music, and themes used within battles and action sequences will usually be of type Action Even. By this simple assignment, much of the basic playback-logic is automatically configured, so that e.g. your ambient music will never interrupt your battle theme, just because the player has stepped into some trigger zone.

2. No manual linking of audio-clips

To avoid the tedious manual linking of single musical segments, we use a simple categorization scheme based on Themes, Segment Types and groups of instrumentation. While in most cases the out-of-the-box configuration leads to good results quickly and easily, micro-management of single transitions is always possible if needed.

3. Direct support for option-screens, cut-scenes and serialization

psai directly supports ever-recurring standard demands of modern games, like seamless transitions from a cut-scene back to gameplay, or the (de)serialization of the current playback state, and sudden switches to in-game menus and back. In the psai Player, these functions are selectable from the Advanced Functions Panel.

4. All power to the composer

We try to take away as many responsibilities from the audio programmer as possible. Thereby all creative decisions are passed on to the composer, and can be put into effect using the psai EDITOR without any programming knowledge. This way the code for integrating psai into the game engine is as minimal as possible, and there is low risk that creative decisions are ruined during the integration process. The interactive soundtrack can be fully authored and tested before the first line of game code is programmed. Likewise, the soundtrack can grow independently from the rest of the game, without the need for rebuilding the game project.

Themes

A psai Soundtrack consists of a collection of Themes. Just like themes in a film score, a psai Theme is a piece of music that will be played in a certain situation or a certain setting. In games we can't predict how long such a situation will persist, so the main difference to a film score is that a psai Theme is of variable length. While the game progresses, you may also want to adapt the intensity of the current Theme according to the current game situation. E.g. if the player is involved in a battle which is getting more and more dramatic, you may choose to raise the intensity of the music according to the damage the player has taken. To achieve both dynamic intensity and variable length, psai creates an interactive soundtrack by enqueueing short pieces of pre-composed music on-the-fly. These pieces of music are called Segments. A Theme always consists of one ore multiple Segments.

Theme Types

Any Theme is always classified as one of psai's predefined Theme Types:

1. Basic Mood
2. Basic Mood Alteration
3. Dramatic Event
4. Action Event
5. Shock Event
6. Highlight Layer

By assigning a Theme Type you control the basic playback behavior for that Theme. The set of Theme Types represents a predefined hierarchy that model the urgency of playback. E.g. your action adventure game may contain at least one certain theme to be played back during a battle. Whenever the player is attacked, you want this theme to start immediately, thereby interrupting any unobtrusive music that might be playing in the background. To achieve that, just assign type **Action Event** to your battle music, and **Basic Mood** to your background theme. These two Theme Types are the most basic ones, and are likely to be sufficient for most scenarios.
For an overview of the mutual interruption behavior of all Theme Types, please see section Table of Theme Types .

1. Basic Mood

The type Basic Mood refers to themes that define the overall vibe and atmosphere for relatively long game situations. A Base Mood could be the general background music for a certain setting, level or area of your game world, or it could refer to a certain character or dialog scene.
Once triggered, a Basic Mood lasts until the next Theme is triggered. When the intensity drops to zero, psai goes into the state of musical rest. It will later wake up with the same BasicMood.

2. Basic Mood Alteration

The Basic Mood Alteration enhances a Basic Mood theme for a relatively short amount of time.
Here's an example: A Basic Mood could be playing while the player explores a forest. As the player passes a special location like a magic tree, this event could be illustrated by playing back a Basic Mood Alteration. Then after a short while, the music changes back to the Basic Mood.

3. Dramatic Event

This Theme Type is suited for conversations in which the player obtains important information about the further course of the game. The Dramatic Event always interrupts the Basic Mood or Basic Mood Alteration immediately, but is of lower priority compared to an Action Event. When the Dramatic Event is over, psai returns to the Basic Mood that was triggered the last.

4. Action Event

An Action Event is a theme of high priority. It quickly responds to the action of the player and interrupts any of the previously mentioned Theme Types immediately. The intensity of an Action Event should be set and updated according to the severity of the situation.

Sudden combat sequences are a typical case for an Action Event. The intensity of the music can be linked to any variable of the game, such as the player's health status. To update psai's intensity level, call Trigger Music Theme repeatedly for the Action Event, thereby passing the new intensity value. A triggering frequency of once per second is sufficient.
Like with every other Theme, once you stop triggering psai's intensity level will drop to zero within the authored time period (Theme duration).
When the intensity has dropped to zero, psai will continue with the theme of the highest priority, that has been triggered the last. If no theme has been triggered, psai will continue with the last BasicMood.

5. Shock Event

The Shock Event has the highest priority of all Theme Types and willl interrupt any other Theme immediately. Use Shock for sudden attacks, dramatic turnarounds of the plot, or death of the player.
When the intensity has dropped to zero, psai will continue with playing the Theme of the highest priority that has been triggered most recently. If no Theme has been triggered, psai continue playback by playing the last BasicMood.
A Shock generally behaves a any other Theme Type and may consist of an arbitrary number of Start-, Middle- and End-Segments. You may also choose to use a Shock as a rather short independent effect, like an orchestra stab. In this case, set the Theme Duration in the Editor to zero. This will make sure that the Shock will end directly after the first Segment. You still can add any number of Segments to your Shock Theme, for more variety. Make sure that each Segment has the START-suitability assigned.

6. Highlight Layer

A Highlight Layer is a special type of Theme, which does not interrupt the Theme currently playing. Instead, it is played as an additional layer of sound, much like a sound effect. The difference to a simple sound effect is that your Highlight Layer will always choose a Segment for playback, that will harmonically fit to the Theme that is currently playing. These compatibilities can be defined within the psai EDITOR.
Use Highlight Layers for situations like when the player is “levelling up” in the midst of battle, or when (s)he accidentally finds a treasure while escaping a dungeon.

Table of Theme Types

This table shows what will happen if a Theme of any Type is triggered (top row) while another Theme (left column) is already playing. The bottom section shows the intensity-related behavior of each Theme Type.

	Basic Mood	**Basic Mood Alteration**	**Dramatic Event**	**Action Event**	**Shock Event**	**Highlight Layer**
...interrupts Basic Mood:	at end of current Segment	at end of current Segment	immediately	immediately	immediately	no[1]
...interrupts Basic Mood Alteration:	never	at end of current Segment	immediately	immediately	immediately	no[1]
...interrupts Dramatic Event:	never	never	at end of current Segment	immediately	immediately	no[1]
...interrupts Action Event:	never	never	never	at end of current Segment	immediately	no[1]
...interrupts Shock Event:	never	never	never	never	immediately	no[1]
...interrupts Highlight Layer:	never	never	never	never	never	maybe[2]

When the intensity has dropped to zero...

... and no Following Theme[3] is set:	Play an End-Segment, then go to rest	Play an End-Segment, then go to rest. Wake up with the last Basic Mood.	Play an End-Segment, then go to rest. Wake up with the last Basic Mood.	Play an End-Segment, then go to rest. Wake up with the last Basic Mood.	Play an End-Segment, then go to rest. Wake up with the last Basic Mood.	-
...and a Following Theme[3] is set	Play an End-Segment, then go to rest	transition to the Following Theme if possible (4). If not, continue like no Following Theme is set.	transition to the Following Theme if possible (4). If not, continue like no Following Theme is set.	transition to the Following Theme if possible (4). If not, continue like no Following Theme is set.	transition to the Following Theme if possible (4). If not, continue like no Following Theme is set.	-

1) Highlights don't interrupt the current theme, but are played back as an additional layer

2) Highlights only interrupt each other if the priority value of the newly triggered highlight is higher than the one currently playing

3) The Following Theme is the Theme that will be playing after the current Theme. Following Themes are set in two ways:

1.) by triggering Themes of lower priority while a Theme of higher priority is playing

2.) automatically when a Theme is interrupted by a Theme of higher priority

E.g. if a Basic Mood was playing when an Action Event kicked in, the Basic Mood is set as the Following Theme.

4) The transition is possible if there is a Middle-Segment within the Following Theme, that is marked as a compatible follower to the Segment currently playing.

Intensity

Psai's playback logic is based on the concept of musical intensity. Whenever a Theme is playing, psai is permanently trying to adapt the musical intensity to the current gameplay situation. It does so by continuously enqueueing musical segments ("Segments") of an intensity that currently matches the best. All a game developer has to do is to map the current gameplay intensity to a value between 0% and 100%, and then pass it to psai's function for triggering Themes, along with the id of the desired theme. To map the intensity to a percentage value you can e.g. use the remaining healthpoints of the player, or maybe the number of attacking enemies.

Automatic decrease of Intensity

After you triggered a theme with a certain intensity, the psai's internal intensity will immediately be set to this new level. If no more trigger calls follow, psai's intensity will automatically and consistently decrease. The rate of this fall-off can be authored per Theme within the psai EDITOR. The purpose of this decrease is that game developers don't have to check back on psai's playback status to eventually turn off any themes, of nothing is happening. Instead it is sufficient to place trigger-events within your game world and/or your gameplay logic, and psai will take care of the rest.
To keep psai's intensity level in-line with your current game situation (like during a battle with dynamic intensity), just trigger the same theme repeatedly, and thereby pass the updated intensity value. A trigger interval of one second is usually sufficient for a common soundtrack. Please see the Best Practice section for more information on how to achieve a highly adaptive soundtrack.

If you would like to temporarily deactivate the automatic decrease of intensity, use the Hold Intensity function.

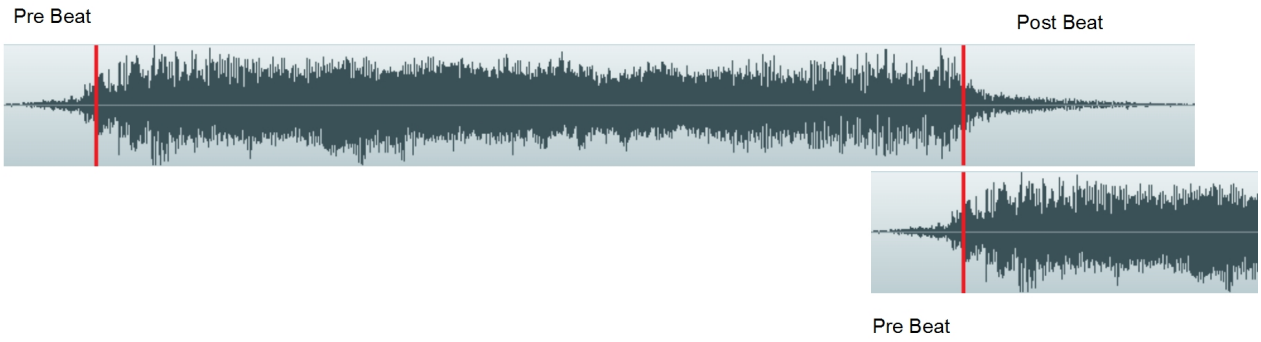
Segments

Psai's concept is based on the dynamic recombination of musical segments. These are automatically chosen and enqueued on-the-fly while your game progresses.
Each Segment contains a segment from the stereo (or surround) sum of your game soundtrack. This means there is only one main track playing at any time, and you don't have to care about multiple layers. The only exception of this are Highlight Layers. A Segment always belongs to exactly one Group, and a Group always belongs to exactly one Theme.

How Segments are mixed

Each Segment consists of three regions:

1. PreBeat
2. Main Region
3. PostBeat



PreBeat

The PreBeat region is meant for short fade-in sections and pickups. The end of the Pre Beat marks the beginning of the first beat of the Main Region.
It is important to keep the PreBeat as short as possible, since it affects psai's responsiveness, as explained in the Best Practice section.

Main Region

This region contains the main musical sequence that could be seamlessly looped within an Audio Editor. It usually consists of one or multiple measures.
The Main Region can generally be as long as you wish, but bear in mind that longer Segments lead to a less responsive soundtrack. See Best Practice
for more information.

PostBeat

The PostBeat region contains the decaying sounds of the preceding Main Region, like reverberation and decaying sounds of string or percussion instruments. The Post Beat starts exactly at the point of time when the beat of the next measure (say: the Main Region of the following Segment) kicks in. To achieve a seamless transition to the next Segment, psai automatically schedules the start of the following Segment in a way so that the beginning of its Main Region aligns with the end of the Main Region currently playing, as shown in the picture below. Thereby both Segments are played on their own internal channels, so that the Post Beat of the first Segment is mixed with the Pre Beat of the following Segment.

Your Post Beat region can basically be as long as you like. Just make sure it does not interfere musically with any Segment that may follow. We therefore recommend to keep the Post Beat shorter than the average Main Region.

Segment suitabilities

Depending on your audio file, a Segment may be suitable for one or multiple parts of its Theme. You can assign any combination of these by enabling the related checkboxes within the Segment Property Panel of the psai Editor.

1. Start-Segments

When a Theme starts to play out of silence, the psai logic will select a Segment which has the Start-Segment property set. A Start-Segment may e.g. contain a crescendo of some sort, e.g. a drumroll of increasing intensity.
Any Theme (with the exception of Highlight-Layers) must at least contain one Start-Segment. Providing multiple Start-Segments may also be beneficial for your soundtrack, like one Start Segment for a sudden kick-in of the Theme with high intensity, and a softer version for not so dramatic situations.

2. Middle-Segments

Middle-Segments are usually by far the most Middle Segments are all Segments that may be played while the Theme is already playing back. The more Middle Segments you provide, the more diversified your Theme will be. The more Middle Segments you provide for a Theme, the less repetition will occur. For a highly dynamic soundtrack make sure to provide a proper number of Segment across different levels of intensity.

3. End-Segments
When a Theme ends, and no other Theme has been queued for playback, psai will end the Theme by playing a Segment of type End-Segment, before the playback goes to a state of Rest or Silence. Make sure each Theme provides at least one End-Segment. The exception to this are Themes of type Highlight Layer, as they only consist of single layering Segments and neither need Start- nor End-Segments.

4. Automatic Bridge-Segments
Bridge Segments will be played as the first Segment of a Group / Theme, when transitioning from another Group / Theme. So in that respect they are like Start-Segments, whereas Start Segments will only be played out of silence. Accordingly, often Start-Segments may also sound good when used as a Bridge. However, be careful dominant chords as they may cause cacophony depending on the tonality of the preceding Theme. As long as there is at least one Bridge Segment within a Group, all other Segments within this Group are implicitly blocked for Group/Theme transitions. For more information on this topic please refer to section Bridge Segments.

psai States

At any point of time, an instance of the psai engine is in one of the following states:

1. Not ready
No soundtrack has been loaded yet.

2. Silence
In this state, psai does not play music and will remain silent until the next call to TriggerMusicTheme.

3. Playing
Psai is currently playing back a Theme.

4. Rest
Psai is letting the music rest for a random period. Although no music is playing at the moment, the last Basic Mood is generally still active, and psai will automatically wake up with this Theme soon, with an intensity as defined in the psai EDITOR. The period of a musical rest is generally of random duration, but you can define minimum and maximum limits for each Theme.

psai's playback logic

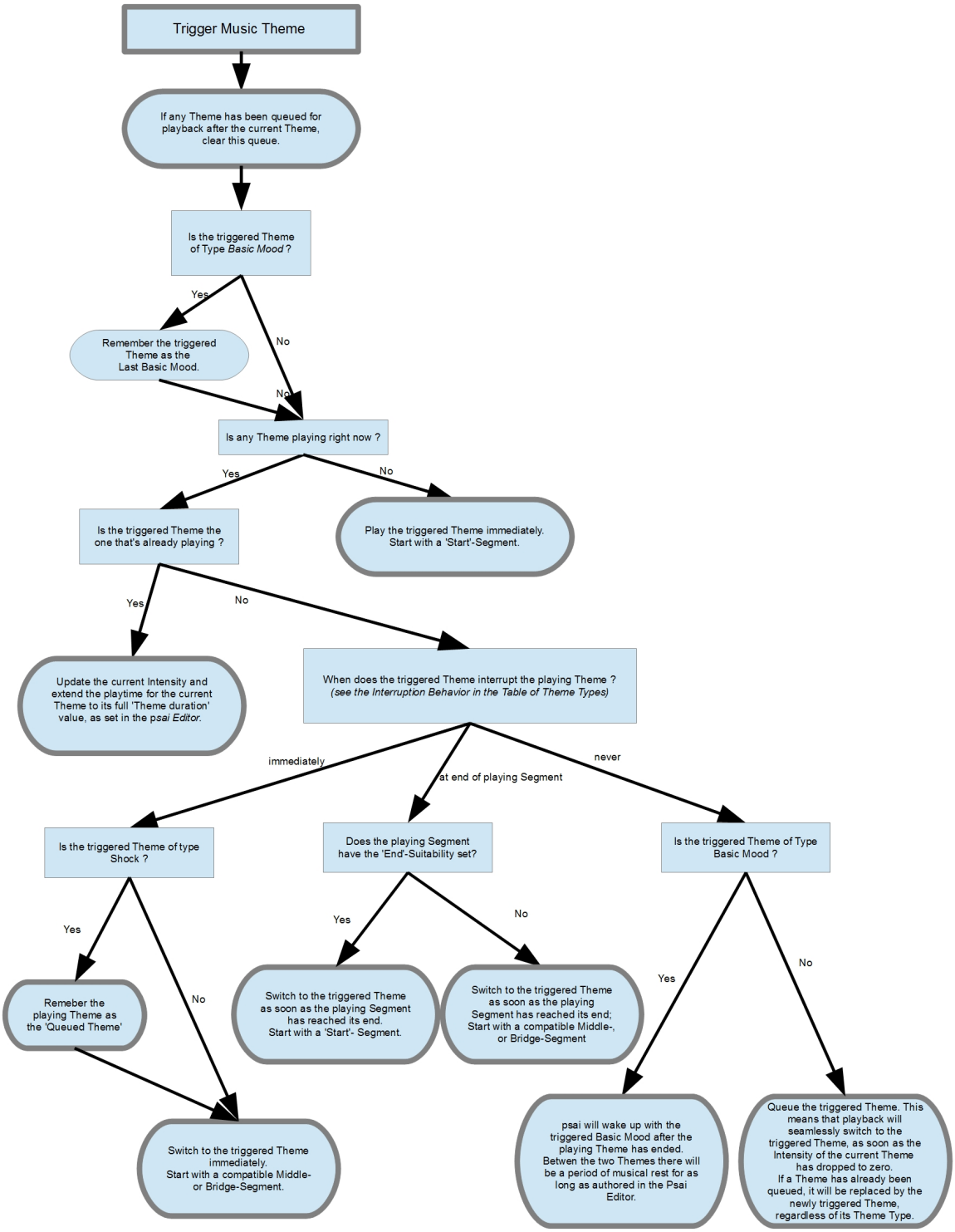
We designed the playback logic to hopefully work as intuitively as possible, so usually you shouldn't have to worry about the details. However there might be times where it helps to get a deeper understanding of how psai works. The following diagrams show how psai reacts to Triggering Music Themes, and how psai evaluates the next action when the playback of a Segment is about to reach its end.

Here's the most important things to know, in human readable format:

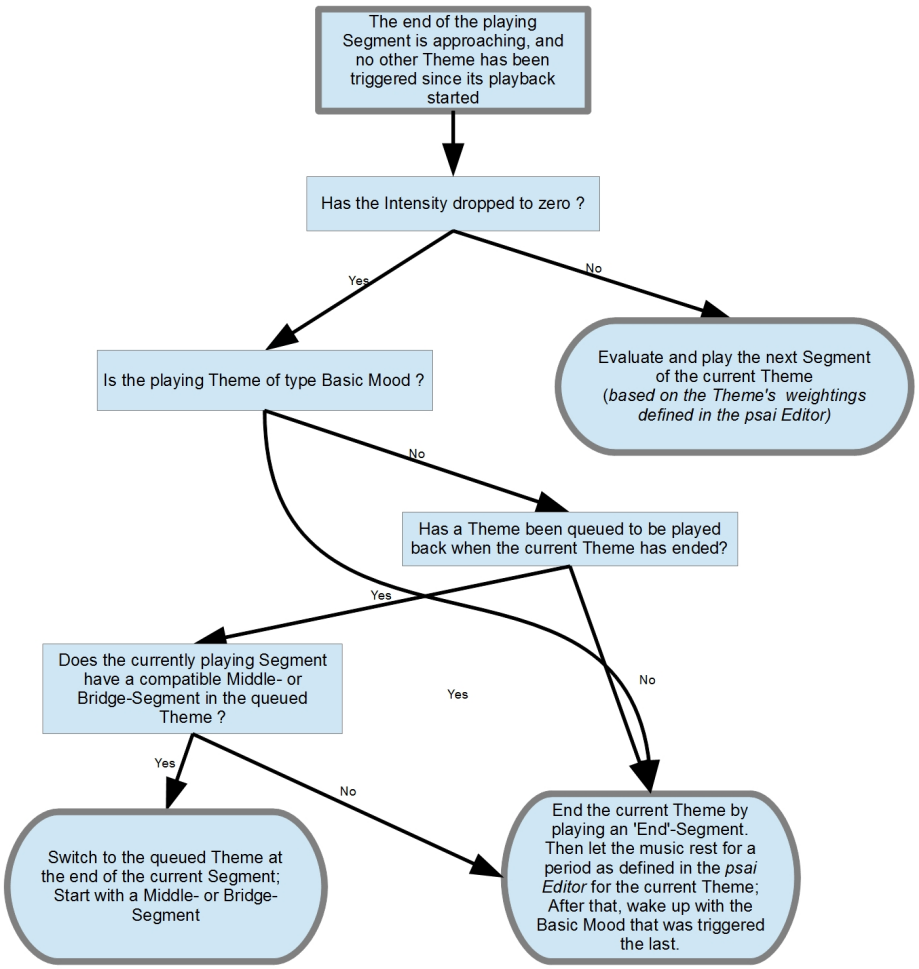
- BasicMoods will be interrupted immediately by every other Theme.
- Highlight Layers are not a regular Theme and will not interrupt anything.
  They are a collection of single Segments that will be layered on top of the playing Theme.
- An immediately interrupting Theme will always start with a 'Start'-Segment
- A Theme that interrupts the playing Theme 'at the end of current Segment' will start with a 'Middle' or 'Bridge'-Segment.
  (make sure there always exists at least one valid transition from each Segment of a Theme to the interrupting Theme!
  So don't be too restrictive when setting your Segment compatibilities)
- If a Theme's intensity has dropped to zero, playback will switch back to the previous interrupted Theme.
- If there is no interrupted Theme (like BasicMood), the music will rest for some time and eventually start with the last BasicMood again.
- When switching back to a interrupted Theme, the Theme transition will happen smoothly with a 'Middle' or 'Bridge' Segment,

like when interrupting non-immediately.

when a Theme is triggered...



at the end of a playing Segment...



Best Practice for composers

Here are two general advices that can help you to keep your soundtrack as reactive as possible.

1. Always keep the PreBeat times short

When the playback of a Segment is approaching its end, psai will start to evaluate the next best-matching Segment. A short interval between that evaluation and its playback means that psai adapts quickly to recent changes of gameplay. The point of time when the evaluation starts is directly affected by the longest PreBeat time among all Segments that could possibly be enqueued next. E.g. if there is a compatible Segment with a prebeat time of 5 seconds, psai will start the evaluation at least 5 seconds before the current Segment is over. This way it is made sure that there will be enough time to fully play that Segment from the beginning, in case it's chosen for playback.
So to to make sure a Theme is as responsive as possible, use the PreBeat periods only for short fade-ins, and try to stay below one or two seconds. Bear in mind that a single compatible Segment with a long PreBeat period is enough to potentially screw the responsiveness of a whole Theme.

2. Keep your Main Regions short to make your soundtrack react faster

The shorter the Main Regions of your Segments are, the faster psai can react to sudden changes of intensity. Therefore to fully tap psai's potential, we recommend to limit the Segment duration to 4-5 seconds for Themes that require a high level of adaptivity (like Action Events). The PostBeat regions can be as long as you wish, as they do not affect the timing of following Segments.

psai for Unity

psai for Unity is the Mono /.NET port of the psai engine, which is completely rewritten in C#. Customers get the full source code.

Getting Started

Like all Unity extensions, the psai for Unity SDK comes as a .unitypackage, which needs to be installed into an existing Unity project. To try out the DemoSoundtrack included in the SDK, just create a new empty Unity Project, like this:

1. Download Unity at http://www.unity3d.com

2. Start the Unity Editor, and create a new Project by selecting Main Menu > File > New Project...

3. In the Main Menu select "Assets > Import Package > Custom Package" and select the psai.unitypackage from the psai SDK.

4. Click "open" and wait for the unitypackage to decompress. After that a window will open where you can select the components to import. To try out the DemoSoundtrack, make sure everything is checked.

5. Please locate the "Project" Window within the Unity Editor. In there you will see that the following files and folders were added to your Project:

a) "Psai" - contains the source code of the psai playback engine
b) "Scripts" - contains scripts for triggering music
c) "PsaiPlayer" - the playground scene for testing psai soundtracks

6. In the Project Window double-click the "PsaiPlayer" Scene to switch to that Scene.

7. Launch the Scene by pressing the "Play"-triangle symbol in the Unity main window.

Now the Psai Player GUI shall appear where you can play around with the Demo Soundtrack.

psai Editor (Light Edition)

The psai.unitypackage comes with a slim version of the psai Editor, that will allow you to open, edit and save your .psai Project files within Unity. Once you have imported the psai.unitypackage to your Unity Project, you will find the psai Editor LE in the menu "Windows > psai Editor (Light Edition)".

What you can do with psai Editor LE:

- Edit the Theme Properties
- Delete Themes, or Import Themes from other psai Projects
- Edit some basic Segment properties

..what you CANNOT do:

- Create new Themes from scratch
- Add or Remove Groups or Segments to/from Themes and Groups
- Change Prebeat/Postbeat times, compatibility settings or anything else that may break your Theme's playback consistency

... so in short, the psai Editor LE is well suited for combining and adapting Themes to a new psai Project, but you will still need the psai Editor (standalone) for creating Themes from scratch.

Please note the following:

- When pressing play in Unity, your project changes will be gone. Make sure to save changes your .psai Project first.
- After saving your Project you still need to build your soundtrack to a binary file! Click "Build Soundtrack"

from the File menu to do so.

- We highly recommend to place the .psai Project file in the same directory where your exported soundtrack file will reside. That is, somewhere within the Resources directory of your Unity project. This will avoid issues with wrong relative paths to your Audio Objects. At runtime, only the binary soundtrack file is loaded, so it's ok to delete the .psai Project file in the release version of your game.

Exporting and playing back custom Soundtracks

You also need Unity to test your own soundtracks. If you already installed the .unitypackage as described in the last section, go on like this:

Within the psai Editor:

1. In the Main Menu select 'File > Build Soundtrack / Export to Unity Project'

2. An Audit sequence is started automatically that will check if your soundtrack for possible deadlocks. If problems where detected, the Console window at the bottom will show more information on how to correct these. When the Audit has succeeded, navigate to "Assets > Resources" folder of your Unity Project. It's recommended to create a subfolder for each Soundtrack, let's name it "MySoundtrack"

3. Choose a name for the binary soundtrack configuration file. We recommend to keep the name "soundtrack.bytes"

4. Click "OK" to start the export.

5. If one or more audio data files did not yet exist within the target folder, the export process will copy these files automatically. In this case you will be asked to use the psai Multi Audio Object Editor extension on your soundtrack folder within your Unity project. It makes sure that the import settings of all Audio Segments are set up correctly.

6. click OK to finish the export.

Within the Unity Editor:

7. Make sure the PsaiPlayer Scene is still active (double-click it in the Project Windows if you are not sure)

8. Locate the Hierarchy Window. In there you should find two entries: a "MainCamera" and a "Psai" object.

9. Select the Psai object.

10. In the Inspector window, the first component shown is the "PsaiSoundtrackLoader". In there edit the variable "Path To Soundtrack File Within Resources Folder" to point to your exported soundtrack file, e.g. "MySoundtrack/soundtrack.bytes"

11. Launch the Scene by pressing the "Play"-triangle symbol in the Unity main window.

psai Multi Audio Object Editor

The psai Multi Audio Object Editor is an Unity Editor extension that makes it very convenient to apply the correct settings for all of your soundtrack's Audio Clips within a (sub-) folder of your Project. Just right-click on the desired folder within Unity's Project window and check the "use psai settings" checkbox. You may also adjust the compression that will be used for each audio file.

Optimizing performance

Optimizing Memory

psai is designed for minimal memory footprint and low CPU hit. It only allocates memory when new AudioClips are loaded dynamically at runtime. Heavy memory allocation is only caused when the

PsaiPlayer script is enabled, as it currently uses Unity's OnGUI framework. So in the release version of your game this is not an issue.

To minimize the file size in the final Release version of your game, consider to...:

- remove all .psai project files from the Resources folder of your project
- select the Psai prefab in your Scene, set the LogLevel property of the PsaiManager component to **off**.
- for minimalists: define the Scripting Symbol **PSAI_NOLOG** for each of your target platforms within the Unity Editor. (Go to Build Settings > Player Settings > Other Settings > Scripting Define Symbols). This will remove all code related to debug output already at compile time.
- also make sure to remove all unneeded files (maybe audio files from Themes you deleted) from your Resources folder

Optimizing playback latency

1. Make sure to keep the PreBeat times of your Segments short, as described here: Best Practice for composers
2. Unity's sound API currently does not provide feedback for the playback latency needed by the host device to buffer or play back an AudioClip. This is why we need to define a maximum latency for each platform, both for a) buffering audio data and b) initiating playing back of buffered audio data. These values can be set in the PsaiPlaybackManager component of the Psai.prefab. The standard values we are using there may be too high for your target devices, so you may consider reducing these values. Very low latency values may result in audible timing problems caused by delayed playback.

Troubleshooting (Unity)

Q: My soundtrack is playing, but way too silently
A: Make sure that each of your Audio Clips uses the correct import settings, e.g. "3D sound" has to be disabled. Click the Audio Clip in the Project window and check the settings in the Inspector window. Use the psai Multi Audio Object Editor to apply the correct settings for multiple files or folders.

Scripts

The psai.unitypackage comes with a set of MonoBehaviour scripts that are used to control or test playback at runtime.

PsaiCoreManager

The PsaiCoreManager is responsible for basic settings of the psai playback engine. It also filters and manages concurrent calls from Trigger Scripts that might otherwise interfere with each other.

Log Level	The amout of logging information written to the Unity Console. Please note that changing this setting while your game is running will have no effect.
Tick Interval In Seconds	The interval in seconds that the PsaiCoreManager will evaluate Trigger scripts that fire continuously. Higher values will save CPU cycles, lower values will make your soundtrack react faster.
Latency Settings	The latency for buffering / playing back buffered sounds that psai will grant the given target platform. Increase if you experience timing problems on a target device.

PsaiSoundtrackLoader

The the path of your binary soundtrack file as exported by the Psai Editor.
Please note that all soundtracks must be located in sub folders of your Unity project's 'Assets/Resources' folder. So for instance if you exported your soundtrack to 'MyUnityProject/Assets/Resources/MySoundtrack/soundtrack.bytes', set this variable to 'MySoundtrack/soundtrack.bytes'

PsaiPlayer

The PsaiPlayer component is your playground for psai Soundtracks within Unity. Please see the video section at www.homeofpsai.com to see it in action, or refer to section Functions / Controls for an explanation of the different functions.

Trigger Scripts

The psai.unitypackage comes with a set of scripts that can be attached to GameObjects in your Unity Scenes, to conveniently place trigger events. These Trigger scripts are usually very short and can easily be extended or adapted as needed.

PsaiContinuousTrigger

This is a base class for scripts that trigger a Theme continuously in a certain frequency. This is necessary when you want to regularly update the dynamic Intensity level, like when matching the musical intensity to the distance (or any other parameter) of an object.
The problem with continuously firing scripts is that they may interfere with each other as soon as they overlap, causing wild jumps of intensity. Therefore PsaiContinousTrigger provides a boolean parameter to have the PsaiCoreManager filter and synchronize overlapping trigger events, using only the trigger with the highest intensity for a given Theme.

Introduction

The psai PLAYER is the standard tool for playing back and testing soundtracks created for the psai middleware. It is also very helpful to get a feeling for how psai works.
Each Button in the psai PLAYER user interface maps to a single call of the psai API, as shown in each subsection of chapter Functions / Controls. For a complete documentation of the API functions, please see the psai API Reference included in the SDK.

If you are new to psai and want to learn about the basic concepts, please refer to Introduction to psai.
If you prefer to learn by playing around with the psai PLAYER, please continue with section Getting Started.

If you have further questions please contact us at devsupport@periscopestudio.de

Getting Started

When you start the player, most of the controls will be disabled until a soundtrack is loaded.

Loading a soundtrack

Load a soundtrack by selecting **Load Soundtrack** from the File Menu.
The psai PLAYER loads files with the extension .pcb. This stands for *psaiCORE binary* and is the format that is exported by the psai EDITOR tool included in the SDK. The EDITOR also exports the compressed audio data into a subfolder of the directory where the .pcb file resides. Please make sure this directory hierarchy is preserved when loading the .pcb file.

Playing around with psai

As soon as your soundtrack has loaded, you are ready for triggering Themes. For a short introduction of psai's basic concepts and terminology, please refer to sections How psai works and psai's components.
You might be wondering why triggering certain Themes seems to have no effect, while others are played back immediately. The Table of Theme Types provides an overview of psai's Theme hierarchy and playback behavior.

Tooltips

Each active control will display a short tooltip if hovered over. You can enable/disable the Tooltips in the Menu ***View***.

What's new

Version 1.0.1.0
- Quick Controls now support intensity per trigger slot

Version 1.0.0.0
first public release

Functions / Controls

This chapter explains the basic functions used for playing back an interactive soundtrack on-the-fly. The most important function hereby is Trigger Music Theme for initiating the playback of a given Theme. While the use of this single function can already be sufficient to achieve very complex results, the other functions in this section may be needed for special situations within your game.

Trigger Music Theme

```
PsaiCore::triggerMusicTheme(int themeId, float intensity)
```

By triggering a Theme you tell psai that some in-game-event occurred, that should generally initiate the playback of the given Theme. Such an event could be that the player has stepped into a certain area of your game world, or some event could be that (s)he has been attacked. Triggering a Theme does not necessarily result in the given Theme to be played back immediately, as there are Themes that have a higher priority than others. E.g. you don't want your furious battle music to be interrupted by some soft ambient Theme, just because the player has stepped into a forest during the battle. To achieve this priority-ranking, psai comes with seven pre-defined Theme Types, that define the basic playback behavior of a Theme.

Regardless of the Theme Type, each Theme will always start immediately upon a call to TriggerMusicTheme if no other Theme is playing at that moment. In any other case, psai will automatically decide if the playing Theme is interrupted or not, and if so, in what manner the transition to the new Theme should be achieved. As stated above, this decision is based on the Theme Types of the triggered Theme and the playing Theme. Please see the Table of Theme Types for the decision matrix.

Stop Music

Stops the music either by fading out quickly, or by playing an End-Segment.
Afterwards psai will enter SILENT-Mode (psai will stay silent until you trigger the next Theme).

StopMusic (by End-Segment)

```
PsaiCore::stopMusic(false)
```

Psai will wait for the current Segment to finish, then play an End-Segment, then stop the music.

StopMusic (immediately)

```
PsaiCore::stopMusic(true)
```

Stops the playback immediately using a quick fadeout.

Return To BasicMood

Ends the current Theme and returns to the Base Mood that was triggered the last.
The transition to the Base Mood will be interrupted by any call to Trigger Music Theme.

Return to BasicMood (by End-Segment)

```
PsaiCore::returnToBaseTheme(false)
```

Returns by playing an End-Segment.

Return to BasicMood (immediately)

```
PsaiCore::stopMusic(true)
```

Returns by fading out quickly.

hold current Intensity

PsaiCore::holdCurrentIntensity(true)

Freezes the intensity on the current level while the current theme is playing.
The automatic decrease will continue as soon as PsaiCore::holdCurrentIntensity(false) is called, or when the playing theme is interrupted or forced to end, e.g. by calling stopMusic() or returnToBase(). Triggering the same theme again will change the constant intensity to the newly triggered intensity, but will not result in reactivating the automatic decrease.

Note: Calls to holdCurrentIntensity() will be ignored while in Menu Mode or in Cutscene Mode. Call MenuModeLeave() or CutsceneLeave() first.

resume

PsaiCore::holdCurrentIntensity(false)
Reactivates the automatic decrease of intensity.

Advanced Functions Panel

The last sections covered the basic functions of psai, which are sufficient to create a dynamic and rich interactive ingame soundtrack. In addition to that, psai also provides functionality for quick and easy support of standard requirements like cutscenes and ingame menus. These advances functions are available from the Advances Functions
Panel (Menu *View*).

Cutscene Mode

The Cutscene Mode is intended for special ingame events (like cinematics), where the normal gameplay is interrupted, and some other musical theme is played back instead. Cutscenes don't affect the normal playback-logic and thus can not be interrupted by any Theme that might be triggered while the cutscene is active.
When a cutscene starts, its music is immediately played back, interrupting any other theme regardless of its theme type. When you leave a cutscene you can choose to either have a smooth transition to the former state by waiting for the current Segment of the cutscene's Theme to finish, or you can immediately fade over to the previous Theme. Furthermore if the cutscene marks a change of the game situation, you can also choose to reset the former psai playback and return to silence mode, allowing you to trigger any other type of theme directly afterwards.
The Theme used for your cutscene can be of any Type, as the Cutscene Mode is unaffected by the usual hierarchy of Theme Types.
For a cinematic cutscene you may want to provide a dedicated Theme of a length made-to-measure. In this case your Theme can just consist of a single Segment of type START.

Cutscene Mode Enter

PsaiCore::cutSceneEnter(int themeId, float intensity)
Enters the Cutscene Mode immediately and fades out any Theme that may be currently playing. The given Cutscene Theme can by of an arbitrary Theme Type. The Cutscene Theme will be played back with the given intensity until the Cutscene Mode is left.

Cutscene Mode Leave

PsaiCore::cutSceneLeave(bool immediately, bool reset)
Leaves the Cutscene Mode either immediately (immediately = true) or by playing an End-Segment (immediately = false). If the reset-parameter is set to true, the former state of the psai soundtrack (the stack of interrupted Themes) will be resettet.

The Cutscene Mode is very similar to the Menu Mode, with only two differences: The first difference is that the Menu Mode is of higher priority than the Cutscene Mode, so that it is possible to switch to an ingame Menu from within a Cutscene, and back. The second difference is that the Cutscene Mode allows for

smooth transitions back to the normal ingame soundtrack, while the Menu Mode always returns by a quick fade.

Menu Mode

The Menu Mode is intended for situations when the player interrupts the current gameplay to switch to the Options screen or some other ingame menu. While in Menu Mode, a normal ingame music is interrupted, and any other Theme can be played back. Like the Cutscene Mode, the Menu Mode will ignore trigger-calls to any other Themes that might occur meanwhile, until the Menu Mode ist left. The Menu Mode can interrupt the Cutscene Mode and will return to the Cutscene when the Menu Mode is left.

Menu Mode Enter

```
PsaiCore::menuModeEnter(int menuThemeId, float intensity)
```

Enters the Menu Mode. Any Theme that might be currently playing is faded out quickly, and the Theme with the given menuThemeId is played instead. This Menu Theme can be of any Theme Type, as the Menu Mode is unaffected by psai's hierarchy of Theme Types. The Menu Theme will be played back with a constant intensity.

Menu Mode Leave

```
PsaiCore::menuModeLeave()
```

Deactivates the Menu Mode and returns to the former state, using a quick fade.

State File

It is possible to save and restore the current state of psai's playback, along with the current intensity, the playcount of all Segments and stack of interrupted Theme Types.

Save to State File

```
PsaiCore::loadState(const char* filename)
```

Serializes the current state of the engine to the file with the given filename.

Load from State File

```
PsaiCore::saveState(const char* filename)
```

Restores the current state of the psai playback engine.

Log Level

```
PsaiCore::setLogLevel(LogLevel newLogLevel)
```

The Debug-Version of the psaiCore library will write to a log file named 'psai.log' with every new instantiation. This file will be overwritten with every new instantiation of psai. You can control the amount of debug information by setting the desired log level. 'Debug' is most verbose.

LOGLEVEL_OFF	don't log any information
LOGLEVEL_FATAL	only log fatal errors that will abort program execution
LOGLEVEL_ERROR	...also log errors that will likely cause severe problems
LOGLEVEL_WARN	...also log warnings that may indicate unexpected behavior
LOGLEVEL_INFO	...also log general information about the current psai state
LOGLEVEL_DEBUG	...also log internal debug information

The default setting is LOGLEVEL_INFO.

Error Handling

```
PsaiCore::setErrorHandling(psai::ErrorHandling errorHandling)
```

The ErrorHandling defines the strategy on how to deal with missing information or Segments, as authored

with the psai EDITOR. E.g. if a Base Mood is being played and an Action Event is being triggered, there may be no matching Segment that would be marked as compatible with the Segment of the Base Mood currently being played. This would practically make the theme transition impossible and result in a musical dead end. Psai provides various strategies for this situation:

ABORT	Fatal Program termination. Use this only for your debug builds.
RECOVER	Try to substitute the Segment with some other of the same theme.
STOPMUSIC	Stop the music and enter a musical rest.
DENYTRIGGER	If a theme is not guaranteed to be compatible with another theme in every situation, ignore TriggerMusicTheme() upon call and return PSAI_INFO_TRIGGER_DENIED.

Quick Controls Window

The Quick Controls Panel does not introduce new functionality, but helps triggering Themes fast and easy. This way you can test your soundtrack for certain in-game situations more conveniently. The Quick Controls Panel provides a collection of slots, where each slot can hold a single Theme along with an intensity value. Trigger the Theme by clicking the slot's trigger-button on the right, or by pressing the keyboard shortcut as shown on the left.

Remapping Keyboard Shortcuts

Right-Click on the displayed key on the left side of each slot to assign a new key. Make sure a soundtrack is loaded before right-clicking.

Playback Stats Panel

The Playback Stats Panel is automatically displayed as soon as a soundtrack is loaded. It displays the Log Window that displays messages about the current playback state, triggered commands and possible problems, like missing Segments.
The lower part of the Panel contains a table that displays the psai soundtrack loaded, along with all Themes and Segments.
The following columns are displayed:

Segment id	the id of the Segment (unique per soundtrack)
Segment name	the name of the Theme
Segment type	like Start-, Middle-, End-Segment and hybrid types.
Segment intensity	an intensity percentage value that reflects the musical intensity of this Segment, as assigned by the composer
Segment playcount	the number of times this Segment has been replayed since the soundtrack was loaded.

Hide / Show Playback Stats Panel

Click on the arrow-button on the right side of the psai PLAYER's main window, to show/hide the Playback Stats Panel.

Sort by value type

To sort the information, click on the desired column header.

Toggle autoscroll

The autoscroll feature automatically scrolls the table view to the Segment that's currently playing.
Right-click on the table to open the context menu and enable/disable the autoscroll feature.

Introduction

The psai EDITOR is the tool for authoring a psai Soundtrack.

Within the psai EDITOR you create a collection of Themes, import their related audio files, and configure the playback behavior.

Basic Tutorial

In this tutorial we will create a small project with two Themes, import audio files and configure our soundtrack for interactive playback.

1. Adding a Theme

1. Select "New Project" from the File menu. A file selector box appears, where you set the name and directory location of your new project.
2. Add your first Theme, either by selecting "Add Theme" from the Project menu or from the Source Tree view's context menu (right-click somewhere in the Source Tree to open the context menu)
3. The new Theme will now appear in your Source Tree view. Left-click the Theme there to select it. The Theme Properties Panel will appear.
4. Click the "Name" field in the Theme Properties Panel to edit the name of your new Theme.

2. Importing Segments

A Segment represents a prerecorded musical segment that consist of an audio file and some associated information, like the level of its musical intensity.
A Segment always belongs to a Theme, and a Theme always consists of one ore more Groups. Groups help to support different instrumentations within a Theme.
Use the "default group" of a Theme if you don't want to use different instrumentations for now. You can always add or remove groups later and move the Segments if desired.

1. Right-click the default group (or any other group) of your Theme within the Source Tree view. This will open the context menu.
2. Select "Add Segment(s) from WAV file(s)". This will open a file selector window.
3. Navigate to the directory where the audio files of your soundtrack reside, and select the audio file(s) to import. Hold [Ctrl] or [Shift] while left-clicking in the selector window to select multiple files.
4. When your selection is complete, click "OK". A new Segment will be created for each selected audio file and added to the selected group of your Theme.

The Source Tree View

The Source Tree view displays the hierarchy of all Themes, Groups and Segments that your soundtrack consists of.

- Left-click a node to select it. The Property Panel for the selected entity (Theme, Group or Segment) will show up, where you can edit all its parameters.
  Selecting a node in the Source Tree will also affect the displayed Target Tree view, as the displayed compatibility-states shown there are always relative to the selected source-entity.
- Left-click the Checkbox of a Node to add it to your checked-selection. Whenever there is more than one entity checked, and you change a value within a Property Panel, you will be prompted if you want this value to be applied to all checked entities of the same type. Hold Shift while clicking to check all nodes between the previously clicked and the latest one clicked.
- Left-click anywhere in the Source Tree to show the Project Property Panel
- Right-click to open the Source Tree Context Menu

Source Tree Context Menu

Depending on the entity clicked-on, the following options are available in the context menu

clicked anywhere:

- clear selection:
  Un-checks all entities
- Add Theme:
  Add a new Theme

clicked on Theme:

- Add Group:
  Add a new Group to the selected Theme

clicked on Group:

- Add Segment(s) from WAV file(s):
  Import multiple audio files as Segments to the selected Group within a Theme
- Add empty Segment:
  Creates a blank Segment with no audio file associated to it yet

The Target Tree View

Like the Source Tree, the Target Tree also displays the hierarchy of all Themes, Groups and Segments that your soundtrack consists of. The Target Tree is used to display and define which Segments may follow directly after another. This compatibility is always displayed and changed relative to the entity that's currently selected in the Source Tree. Please see section Compatibility of Segments for a more detailed explanation.

- Left-click a Segment to load it into the Playback Panel as the Target Segment (the right half of the Playback Panel)
- Right-click a Segment in the Target Tree to toggle its compatibility setting for a Segment selected in the Source Tree. "Compatible" means that the given Target Segment is generally allowed to be played directly after the Source Segment. When your soundtrack is playing during the game and it is time for psai to choose the next Segment, psai will only evaluate those Segments which are generally compatible to the Segment currently playing.

The Playback Panel

The Playback Panel is used to play single Segments or transitions between two Segments. Use it to adjust or check the length of the Pre- and Postbeat regions of each Segment. As soon as these are correct, you can use the Playback Panel to hear how Segment transitions will sound during a game. To do this, first select a Segment in the Source Tree by left-clicking it. This Segment will be loaded to the left half of your Playback Panel as the Segment to be played first (Source Segment). Now left-click another Segment in the Target Tree. This will be loaded to the right section of your Playback Panel, and played right after the Source Segment.

Segment Panels

Both Source and Target section share the same functions for the related Segment.

Play

Play the Segment completely over its full length

w/o Prebeat

Skip the Prebeat region and play the Segment directly from the beginning of its loop-able section.

Postbeat

Only play the region after the loop-able region (decaying sounds and reverb)

Transition Section
This section contains controls for playing both Source and Target Segment one after another. If it is disabled, make sure that the selected entities in both the Source Tree and Target Tree are Segments.

Full Playback
Play both Segments from start to finish, like they will be played in the game.

Play Transition
This skips the beginning of the Source Segment and jumps to a cue-point x milliseconds before the Target Segment kicks in, where x is defined in the field below.

Cue (ms)
The cuetime (in milliseconds) before the actual transition, that should be played when you press the "Play Transition" button.

Project Property Panel

Path to Audio Pool
The path to the directory where all the audio files of your soundtrack reside. This can either be an absolute path or relative to the directory where your psai project file is located.
We strongly recommend creating a single subdirectory for your audio files within your project file's directory, like 'wav'. Then just enter 'wav' here.

Path to DirectX SDK
This is needed if you want to export your soundtrack for Microsoft XACT, as the external xact compression tool will be called during export.

Volume Boost
Supported by XACT export. Raises the output level of the exported audio file. Use with caution, as this may cause clipping / distortion.

Sound Quality
Supported by all export formats. Sets the trade-off between disk space and sound quality. Set this to the far right for maximum quality and biggest file size.

Force full rebuild
Enabling this will force a complete recompression of all audio files during the next export. Disabling this will speed up the exporting process greatly, but does not guarantee that changes to your audio files are included in your exported soundtrack. Disable this if you made no changes to the audio files since the last export.

Default Segment Settings
Newly created Segments will be created with these default settings.

Theme Property Panel

id
This id is passed to the TriggerMusicTheme() function to trigger this Theme in your game code. Theme ids within a soundtrack must be unique.

Name
An arbitrary name of the Theme.

Type
The Theme Type of this Theme.

Rest Time min
The minimum number of seconds the music will remain silent after the Intensity for this Theme has dropped

to zero.
See state "musical rest" in section psai States.

Rest Time max
The maximum number of seconds the music will remain silent after the Intensity for this Theme has dropped to zero.
See state "musical rest" in section psai States.

Theme duration
The timespan (in seconds) a Theme will keep playing after a single call to TriggerMusicTheme(). The musical intensity will fall off during this period and eventually reach zero. To keep the intensity up and extend the playtime, either keep triggering the Theme with new intensity values, or use the hold intensity function.

Theme duration after Rest
The timespan (in seconds) a Theme will keep playing after waking up from a state of musical rest.

Intensity after Rest
The musical intensity a Theme will start with after waking up from a state of musical rest.

Weightings
These sliders affect the importance of the related factors when it comes to selecting the next Segment to play on-the-fly.

Jump between Groups
This slider to controls how important it will be for psai to stay within the same Group while playing this Theme.
"Never" means that psai will never change a group while a Theme is playing. You will have to switch the Theme completely to ever hear some other Group of this Theme.
"Often" means that the Groups are practically ignored, so that the Segments of this Theme are treated as if they all belonged to the same Group.

Importance of Intensity
This slider controls the importance of a best-matching Intensity when the next Segment is evaluated for playback. Setting this to "exact match of intensity" means maximum priority, even if this means that the same Segment is repeated over and over again. The opposite choice is maximum "variety". This means that repetition avoidance is preferred over matching intensity.

Variety
This slider only affects the percentage of variety, as defined by the "Importance of Intenstiy" slider above. Setting this to "maximum repetition avoidance" means, that it is most important to always play the Segment with the least repetition (playcount). Practically this setting has the effect that the Segments will be played in the order of their appearance in the Tree views. Using this setting is especially useful If you have a melodic Theme that reaches over multiple Segments. This way you can break out of the Theme quickly, but the replay order is generally maintained.
Moving this slider to the right will add a random factor to the choice, that gets more and more weighting. Choose a maximum random factor if your Theme consists of atmospheric sound textures that can be played in arbitrary order and don't make use of varying intensity levels. Maintaining a strict Segment order here would tend to sound predictable, as this would have the same effect as if your Theme consisted of a long single audio track.

Group Property Panel

Name
Enter an arbitraty name for this Group

Description
A textfield for notes about this Group, like its usage or open tasks.

Segment Property Panel

Name
An arbitrary name of this Segment. Shall be unique within your soundtrack to avoid confusion.

Intensity
The musical intensity expressed by this Segment

usable at Theme Start
check this if this Segment may be used to start a Theme. See Segment application properties.

usable at Theme Middle
check this if this Segment may be used in the middle of a Theme. See Segment application properties.

usable at Theme End
check this if this Segment may be used to end a Theme. See Segment application properties.

Automatic Bridge Segment
Check this if you generally want to use this Segment for all transitions from any other Theme to this Segment's Theme. As long as there is at least one Segment marked as a Bridge Segment within a Theme, all other Segments within this Theme are implicitly blocked.

Default compatiblity as follower
Here you can set if this Segment usually works well when played after most other Segments. If is does, set the default compatibility to "allowed" (default behavior). If it doesn't, set the compatibility to "blocked". In this case you need to manually allow transitions to this Segment by selecting all its compatible predecessors in the Source Tree, and right-click this Segment until it is marked as light green in the Target Tree view.

Prebeat/Postbeat Length - Set Manually
Check this if you want to enter the Pre- and Postbeat regions manually,in case you know their exact number of samples. The other option is to have the Prebeat/Postbeat times calculated automatically, based on the BPM of your Theme and the number of beats kept blank before the loop-able region of your Segment.

PreBeat Length (Samples)
If the Set Manually option is enabled, enter the length of the PreBeat region in samples. The PreBeat region is the region before the loop-able region of a Segment. See How Segments are mixed.

PostBeat Length (Samples)
If the Set Manually option is enabled, enter the length of the PostBeat region in samples. The PostBeat is the region of an audio sample after the loop-able musical measure, usually containing decaying sounds and reverb. See How Segments are mixed.

Beats per minute
Enter the beats per minute of the associated audio sample. (used to calculate the Pre- and PostBeat regions based on their length in beats and the bpm of the audio sequence.)

Pre beats
The length of your PreBeat region, measured in beats.

Post beats
The length of your PostBeat region, measured in beats.

Compatibility of Segments

A Segment is said to be "compatible" with some other Segment, if it can be played directly after that other

Segment, without sounding awkward.
To test if a Segment A is compatible with a Segment B, select Segment B in the Source Tree View, and select Segment A in the Target Tree View. This will have Segment B (the "source" Segment) loaded to the left section, and Segment A (the "target" Segment) to the right section of the Playback Panel, where you can replay their transition to check how it sounds. In case the playback-buttons are disabled, check the path to your audio pool in the Project Property Panel.
To change the compatibility of a pair of Segments, see the instructions in the Target Tree View.

The default setting within psai is that any Segment is compatible with any other Segment, unless declared otherwise. Therefore their compatibilities will be indicated as "implicitly allowed". You can change this behavior for each Segment within the Segment Property Panel ("default compatibility as follower"). Changing this will result in "implicitly blocked" compatibility.
Compatibilities can not only be defined for single Segment transitions, but also for whole Groups or Themes. E.g. if you selected some Theme A in the Source Tree, and some other Theme B in the Target Tree, and set this transition to "blocked explicitly", this will disallow all transitions from any Segment within Theme A to any Segment in Theme B, along all Groups. Their transitions will be indicated as "blocked implicitly", as you defined the compatibility not on the same hierarchy level, but at least one level above. In practice it does not make a difference if a transition is blocked implicitly or explicitly - this transition will be forbidden during your game in any case. This distinction is just made to help in the authoring process.

Types of compatibility:

1. allowed implicitly (dark green)
This setting indicates that a transition from the selected source entity to the given target entity will be allowed. "Implicitly" indicates that this is either the default behavior, or the compatibility is inherited from the parent Group / Theme.

2. blocked implicitly (dark red)
This setting indicates that a transition from the selected source entity to the given target entity will be forbidden. "Implicitly" indicates that this is either the default behavior, or the compatibility is inherited from the parent Group / Theme.

3. allowed explicitly (light green)
This setting shows that a transition from the selected source entity to the given target entity was manually / explicitly allowed. This transition will stay allowed, independently from the compatibility of the parent Groups / Themes.

4. blocked explicitly (light red)
This setting shows that a transition from the selected source entity to the given target entity was manually and explicitly blocked. This transition will stay forbidden, independently from the compatibility of the parent Groups / Themes.

5. impossible transition (white square with black X)
Psai's playback rules imply that Segment transitions will never occur at runtime. E.g. if there is a Segment that is only usable at the end of a certain Theme, this will never be followed directly by some other Segment that is only usable to end some other Theme or Group. Impossible Transitions like this are indicated by an X-Icon.

Bridge Segments
Bridge Segments are a concept that help to make your soundtrack manageable when it comes to transitions from one Group (or Theme) to another. You can route your audio flow through these Bridge Segments upon those transitions, so that you gain more control and avoid transitions to other Segments that are likely to sound awkward in that context.
Here's an example. E.g. let's say you composed two Themes, one ambient music for a Cave and one for a Forest, and you provided loads of Segments for both Themes. While all those Segments might work very

well as long as you stay within the same Theme, the transition from Cave to Forest might sound too abrupt or dissonant in many cases. In fact there might be only very few Segments in Theme Forest that are suited to be played directly after most Segments of Theme Forest. Without the concept of Bridge-Segments you would have to manually block all the transitions from all Cave to all other Forest-Segments. Even worse, you would have to keep that in mind for every other Segment you might add later to Theme Cave, and block that too. The solution to that problem is to declare the well-suited Segments in Theme Forest as Bridge Segments to Theme Cave. You do that by first selecting the Source Group in the Source Tree View, which in our example would be the default group of theme Cave. Now you make sure that the Forest-Segment suitable for the transition is visible in the Target Tree View, and right-click that Segment. Its compatibility is now shown as "Bridge", and the compatibility of all other Segments of theme Forest's default group is shown as "implicitly blocked".

Automatic Bridge Segments

In the last section we learned how to define a certain Segment within a Group as a Bridge Segment for a given source Group. Now there are some Segments that generally work well for all kinds of Group or Theme transitions, like those starting with drum-crescendos or swoosh-effects. In this case we might decide to use this Segment generally as a Bridge Segment for transitions to its Group, no matter which Theme or Group was played before. To achieve this, we can set the "Automatic Bridge Segment" flag in the Segment Property Panel. This is especially useful when multiple composers are working on a project, as the Automatic Bridge Segment flag makes it obsolete to manually define that Bridge Segment multiple times for all groups after importing Themes.

3rd Party Technology / License

The following sections contain copyright information about the software components used by the psai playback engine and the psai Editor application. Please make sure that this information is contained unchanged within your software distribution.

Google Protocol Buffers

The core Protocol Buffers technology is provided courtesy of Google.
At the time of writing, this is released under the BSD license.
Full details can be found here:

http://code.google.com/p/protobuf/

This .NET implementation is Copyright 2008 Marc Gravell

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.