Table of Contents

Save Extension Documentation

Save Extension allows your projects to be saved and loaded.

This plugin is for Unreal Engine 4 and has support for versions **4.20** and **4.19**

Introduction

At Piperift we like to release the technology we create for ourselves.

Save Extension is part of this technology, and as such, we wanted it to be public so that others can enjoy it too, making the job of the developer considerably easier.

This plugin was designed to fulfill the needs for automatic world saving and loading that are unfortunately missing in the engine at this moment in time. Automatic in the sense that any actor in the world can be saved, including AI, Players, controllers or game logic without any extra components or setups.

Intended Usage

All our technology is designed to work for very different games and needs, but, naturally, it was created around certain requirements.

In the case of SaveExtension, it has been developed to support games with high amounts of content in the world like open worlds or narrative games.

What I mean by this is that you usually wouldn't serialize a world for a mario game. It can do it, but may not be worth it. Other games might have items to be picked, player states, AI, or streaming levels that require this serialization and here's where the strength of SaveExtension comes.

Supported Features

SaveGame tag saving

Any variable tagged as `SaveGame` will be saved.

Full world serialization

All actors in the world are susceptible to be saved.

Only exceptions are for example StaticMeshActors

Asynchronous Saving and Loading

Loading and saving can run asynchronously, splitting the load between frames.
This states can be tracked and shown on UI.

Level Streaming and World Composition

Sublevels can be loaded and saved when they get streamed in or out. This allows games to keep the state of the levels even without saving the game.

If the player exists an area where 2 enemies were damaged, when he gets in again this enemies will keep their damaged state

Data Modularity

All data is structured in the way that levels can be loaded at a minimum cost.

Compression

Files can be compressed, getting up to 20 times smaller file sizes.

Asynchronous

Level Streaming & World Composition

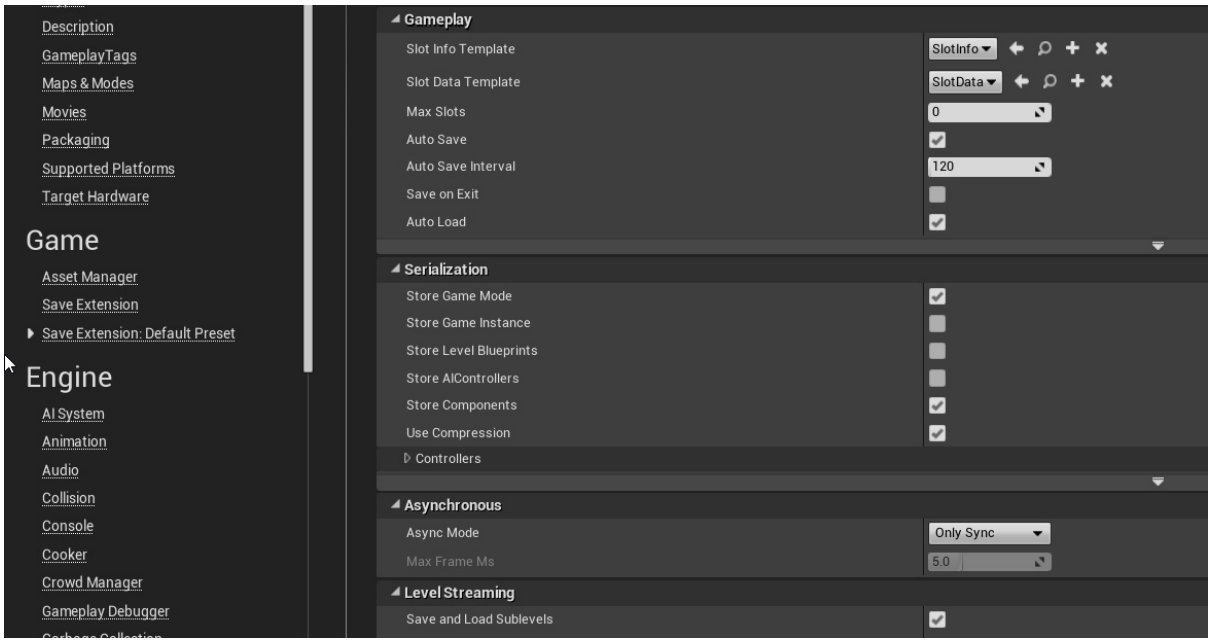
Presets

A preset is an asset that serves as a configuration preset for Save Extension.

Within other settings, presets define how the world is saved, what is saved and what is not.

Default Preset

Under *Project Settings -> Game -> Save Extension: Default Preset* you will find the default values for all presets.



This default settings page is useful in case you have many presets, or in case you have none (because without any preset, default is used).

> All settings have defined tooltips describing what they are used for

Gameplay

Defines the runtime behavior of the plugin. Slot templates, maximum numbered slots, autosave, autoload, etc. Debug settings are inside Gameplay as well.

Check Saving & Loading

Serialization

What should be stored and what should not. Here you can decide if you want to store for example AI.

You can also enable compression to reduce drastically

Asynchronous

Should load be asynchronous? Should save be asynchronous?

Check Asynchronous

Level Streaming

Should sublevels be saved and loaded individually?

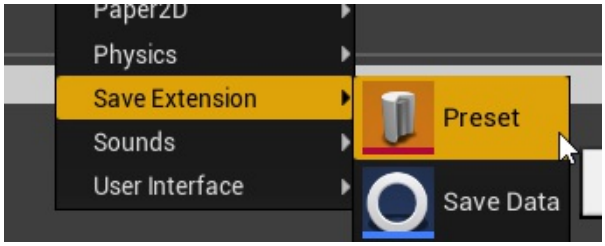
Sublevels will be loaded and saved when they get shown or loaded.

Check Level Streaming

Multiple presets

Because presets are assets, the active preset can be switched in runtime allowing different saving setups for different maps or gamemodes.

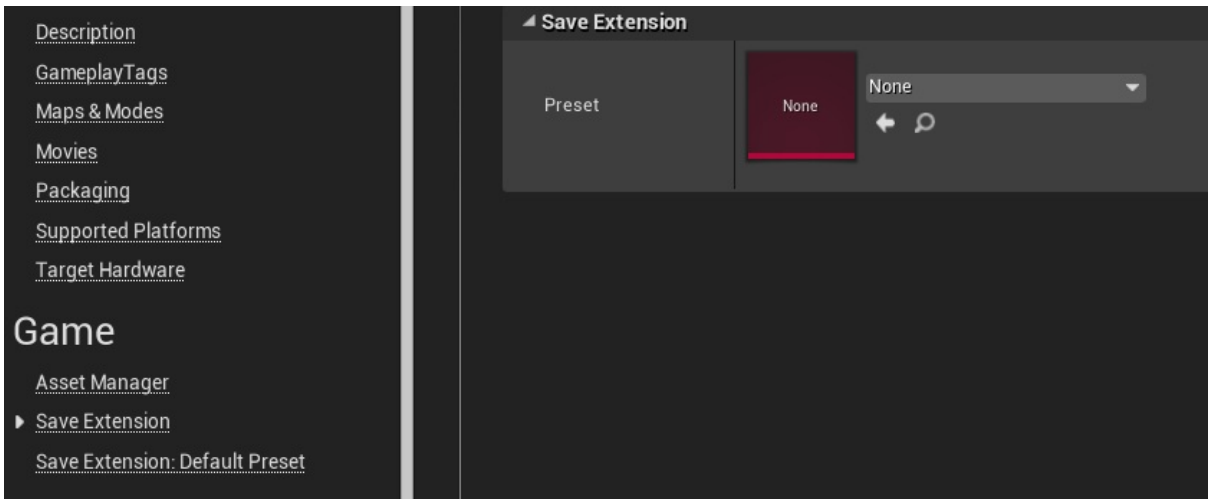
Creating a Preset



You can create a new preset by right-clicking on the content browser -> *Save Extension* -> *Preset*

Setting the active Preset

You can set the active preset in editor inside *Project Settings* -> *Game* -> *Save Extension*



Quick Start

Save Extension's setup is actually very simple.

When using this plugin

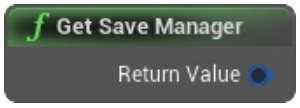
Saving & Loading

Early concepts

Save Manager

The save manager is in charge of loading, saving and all the rest of the logic (auto load, auto save, level streaming...). It will be initialized the first time **GetSaveManager** gets called. Usually at BeginPlay.

GetSaveManager



Slots

When we say **slot** we refer to the **integer that identifies a saved game**.

This slot identifies a saved game even during gameplay, meaning that if an slot gets loaded it will be "active" until we load another slot.

With this we can for example do "*SaveCurrentSlot*" to pick the current slot if any and save it.

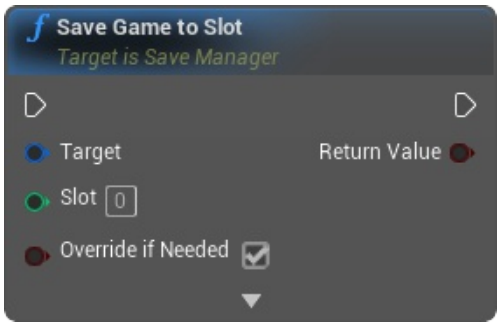
Auto-Save will also save the current slot, and **Auto-Load** will load the last current slot

Saving

To save a game you can just get the Save Manager and then call *SaveGame to Slot*

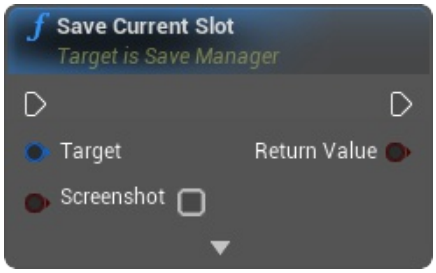
Save to Slot

Save into certain slot



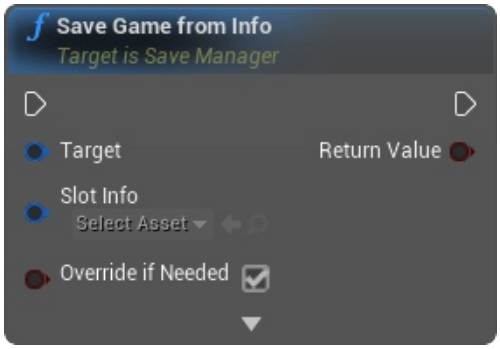
Save Current Slot

Save into the last slot that was loaded



Save Game from Info

Saves a game based on a Slot-Info (Check Slot Infos)



Loading

Slot Templates

Slot Info

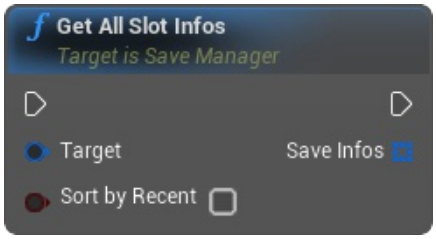
Slot Infos are SaveGame Objects used to identify a saved game. They hold all information that needs to be accessible while the game is **NOT** loaded.

It can for example hold the name of the game, the progress of the player, the current quest or the level.

Slot Infos are also used directly to load or save games. You can load all available infos and make the player decide which one to load from UI.

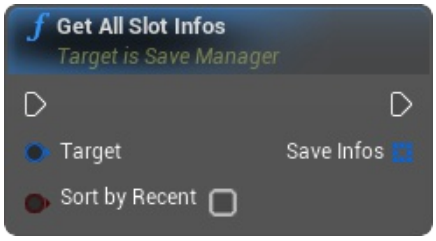
Get All Slot Infos

May be expensive, use with care



Get Slot

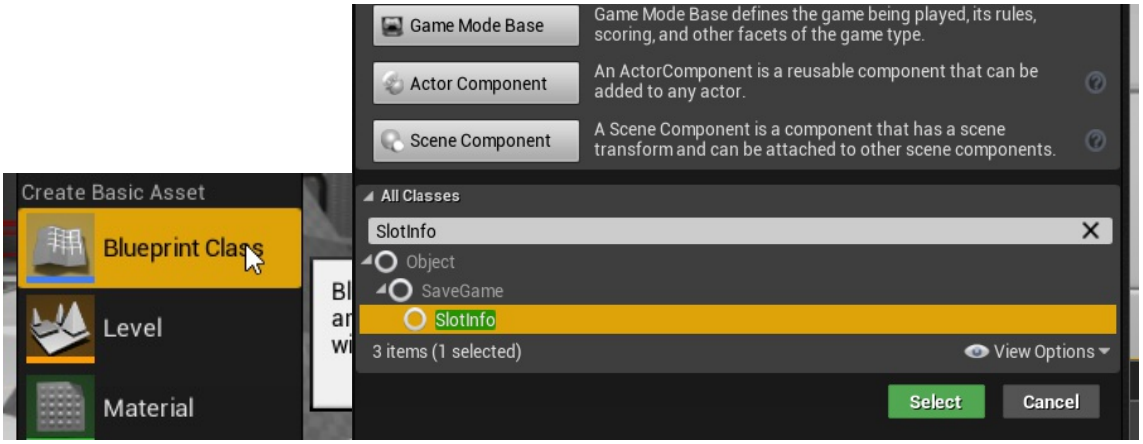
May be expensive, use with care



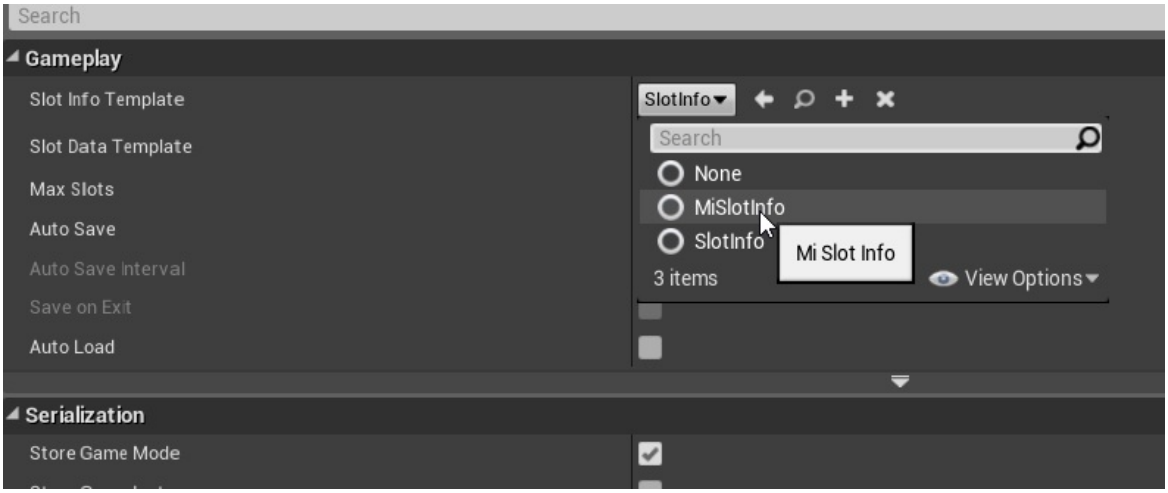
Custom Slot Infos

You can also create your own slot infos and store anything you want on them. Like a normal SlotInfo, your variables will be available when you load the slot (it will only need casting to your class).

1. Create a Blueprint child of *"SlotInfo"*



2. Assign it into your active Preset



Slot Data

Same as Slot Info objects, **SlotData** is a SaveGame Object, meaning all its properties will be saved.

Slot Datas are designed to hold all the information about the world and the player. All information that will be only accessible during gameplay.

Installation

Manually

This are the general steps for installing the plugin into your project:

1. Download the last release from here
Make sure you download the same version than your project

2. Extract the folder “SaveExtension” into the **Plugins folder** of your existing project (e.g "MyProject/Plugins")

3. Done! You can now open the project

From Marketplace

Install from the launcher: AVAILABLE HERE