# Save System for Realistic FPS Prefab

User Guide

v1.1

Save System for Realistic FPS Prefab

# Contents

# Chapter 1: Welcome to Save System for RFPSP

Thanks for using the Save System for RFPSP. This asset adds save support to Azuline Studios' Realistic FPS Prefab. It's designed to require minimal changes to Realistic FPS Prefab's source code.

This asset requires Unity 5.3.6+ and Azuline Studios' Realistic FPS Prefab 1.24+.

## How to Get Help

We're here to help! If you get stuck, have questions, or want to request a feature, please email us:

Email: support@pixelcrushers.com

## Setup & Demo

To use the Save System for RFPSP, you'll need to edit two Realistic FPS Prefab script files. We'll walk you through the simple process here. You won't have to touch any other scripts to use the Save System.

First, edit the script RFPSP ► Scripts ► HUD ► MainMenu.cs. On line 322, insert the text highlighted below:

```
        if (GUI.Button(new Rect(buttonH + buttonWidthAmt + buttonSpacing, buttonV + ((buttonHeightAmt + 
buttonSpacing) * 4), buttonWidthAmt, buttonHeightAmt), "Exit Game", mainButtonSyle)){
            PlayerPrefs.SetInt("Game Type", 0);
            PlayButtonFx(buttonClickFx, buttonFxVol);
            Application.Quit();
            #if UNITY_EDITOR
            UnityEditor.EditorApplication.isPlaying = false;
            #endif
        }

        // START INSERTED TEXT...
        if (GUI.Button(new Rect(buttonH, buttonV + ((buttonHeightAmt + buttonSpacing) * 5), 
buttonWidthAmt, buttonHeightAmt), "Save Game", mainButtonSyle))
        {
            PixelCrushers.SaveSystem.SaveToSlot(1);
            resumePress = true;
        }

        GUI.enabled = PixelCrushers.SaveSystem.HasSavedGameInSlot(1);
        if (GUI.Button(new Rect(buttonH + buttonWidthAmt + buttonSpacing, buttonV + ((buttonHeightAmt + 
buttonSpacing) * 5), buttonWidthAmt, buttonHeightAmt), "Load Game", mainButtonSyle))
        {
            PixelCrushers.SaveSystem.LoadFromSlot(1);
            this.enabled = false;
        }
        GUI.enabled = true;
        // ...END INSERTED TEXT

    }

    //play button click sound effects
    private void PlayButtonFx(AudioClip clip, float vol){
```

For easy pasting, a copy of this text is in the file Scripts ► _PASTE_INTO_MAINMENU.txt.

***Only required for Realistic FPS Prefab versions prior to 1.2x:***

1. Select the menu item **Edit** → **Project Settings** → **Player**.
2. In the Inspector, set **Other Settings** → **Scripting Define Symbols** to `RFPS_1_2`. If this field already contains symbols, add `RFPS_1_2` separated by a semicolon.

***Only required for Realistic FPS Prefab version 1.24 (not 1.25+):***

Next, edit the script RFPSP ▸ Scripts ▸ AI ▸ CharacterDamage.cs. On line 7, insert one line of text:

```
public class CharacterDamage : MonoBehaviour {
    public UnityEngine.Events.UnityEvent onDie; // ADD THIS LINE!
```

Then on line 182, add one more line of text:

```
//Kill NPC
if (hitPoints <= 0.0f){
    onDie.Invoke(); // ADD THIS LINE!
    AIComponent.vocalFx.Stop();
```

That's it! You'll now be able to use the Save System without having to touch any other script files.

## Demo

To play the demo, add these scenes to your build settings:

- Pixel Crushers ▸ Save System for RFPSP ▸ Demo > DemoScene1
- Pixel Crushers ▸ Save System for RFPSP ▸ Demo > DemoScene2

Then play DemoScene1. You can do the following:

- Press Escape to access the RFPSP Main Menu, which will have new buttons to save and load your game.

- Use the portals to move between DemoScene1 and DemoScene2. The portals are to the left when you start the scene, just behind the shack.

- Next to the portal in DemoScene1, there's a Save Checkpoint trigger. If you enter this, it will automatically save your game.

Only a few enemies have been configured to remember in saved games when they've been killed: the "SoldierBadNPCColliders" in front of the house near the beginning of DemoScene1, and two of the robots further down the trail.

Only one destructible has been configured for saved games: the "ExplosiveBarrel" next to the house.

Similarly, only a few pickups have been configured to remember in saved games when they've been picked up, such as the "Sniper_Pickup" leaning against the fence by the house near the beginning of DemoScene1.

The rest of this manual describes how to set up your scenes and objects for saved games.

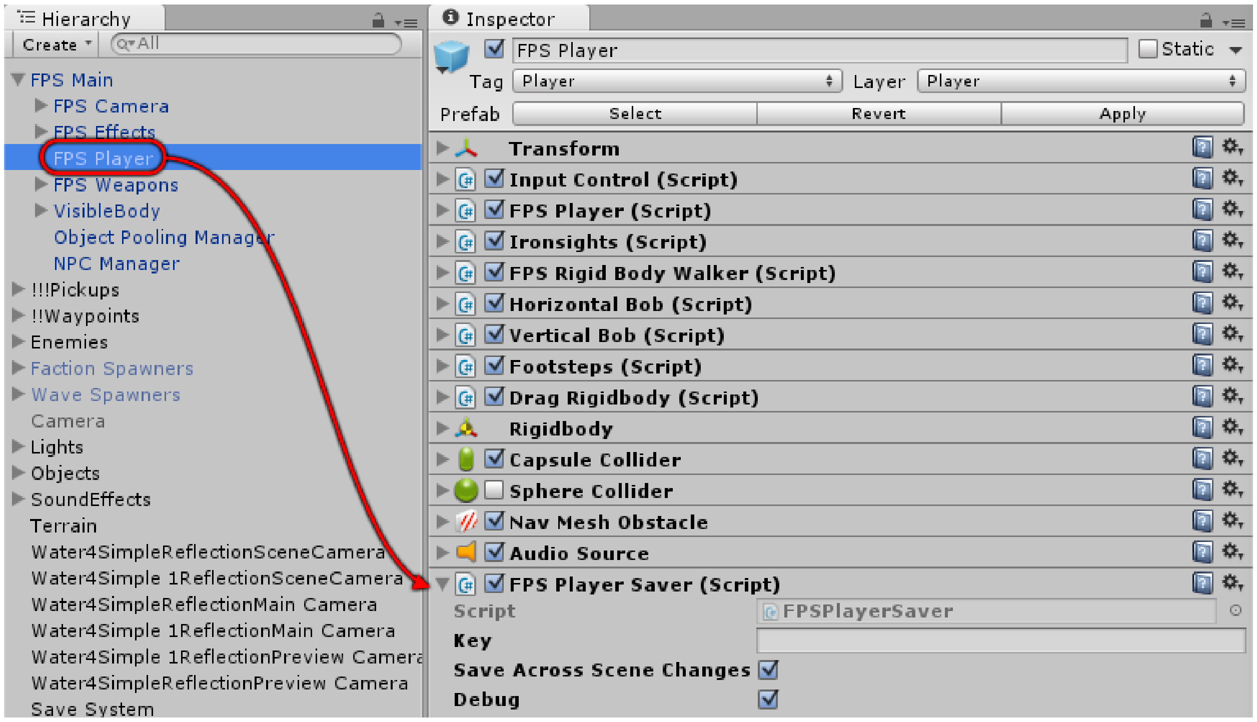
# Chapter 2: Configuration

Follow the steps in this chapter to set up your project with the Save System for RFPSP.

In each scene, make these changes:

## FPS Player

Inspect FPS Main > FPS Player. Add an **FPS Player Saver** component.



```
FPS Main
  FPS Camera
  FPS Effects
  FPS Player
  FPS Weapons
  VisibleBody
  Object Pooling Manager
  NPC Manager
!!!Pickups
!!Waypoints
Enemies
Faction Spawners
Wave Spawners
Camera
Lights
Objects
SoundEffects
Terrain
Water4SimpleReflectionSceneCamera
Water4Simple 1ReflectionSceneCamera
Water4SimpleReflectionMain Camera
Water4Simple 1ReflectionMain Camera
Water4Simple 1ReflectionPreview Camera
Water4SimpleReflectionPreview Camera
Save System
```

```
FPS Player
Tag  Player      Layer  Player
Prefab

Transform
Input Control (Script)
FPS Player (Script)
Ironsights (Script)
FPS Rigid Body Walker (Script)
Horizontal Bob (Script)
Vertical Bob (Script)
Footsteps (Script)
Drag Rigidbody (Script)
Rigidbody
Capsule Collider
Sphere Collider
Nav Mesh Obstacle
Audio Source
FPS Player Saver (Script)
  Script                       FPSPlayerSaver
  Key
  Save Across Scene Changes    ✓
  Debug                        ✓
```

This will save the player's stats, inventory, and position.

Data in saved games are stored under keys. You can specify a value in the **Key** field, or just leave it blank to default to the GameObject's name (FPS Player).

Make sure **Save Across Scene Changes** is ticked.

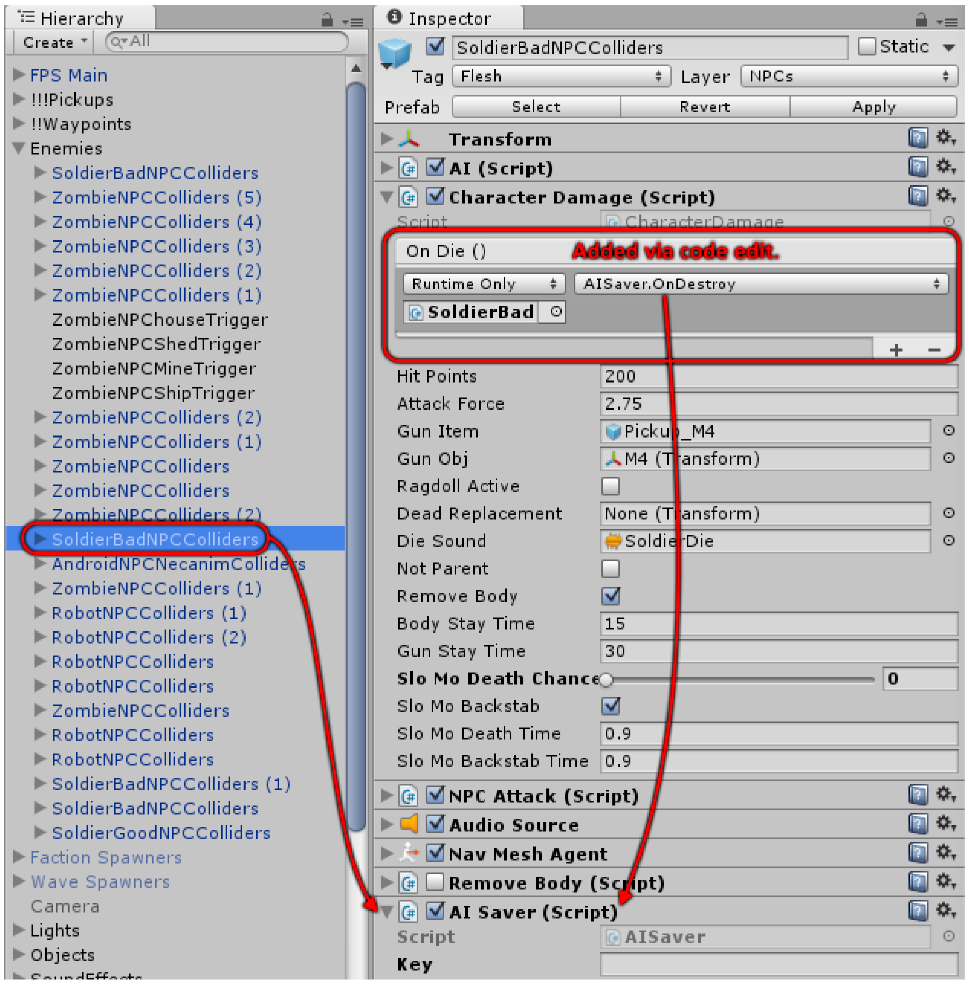
To log debug information to the Console, tick **Debug**.

You can apply the Save System for RFPSP to prefabs as well as scene GameObjects.

## AI

To configure AI NPCs to remember in saved games when they've been killed, make sure you've edited the script RFPSP / Scripts / AI / CharacterDamage.cs as described above on page 5 of this manual. This will add an **OnDie()** event in the Inspector.

Add an **AI Saver** component. Set the **OnDie()** event to AISaver.OnDestroy. If you have more than one AI with the same name, assign a unique **Key** to each.



SoldierBadNPCColliders Static
Tag Flesh Layer NPCs
Prefab

Transform
AI (Script)
Character Damage (Script)
Script CharacterDamage
On Die ()
Runtime Only AISaver.OnDestroy
SoldierBad

| | |
|---|---|
| Hit Points | 200 |
| Attack Force | 2.75 |
| Gun Item | Pickup_M4 |
| Gun Obj | M4 (Transform) |
| Ragdoll Active | ☐ |
| Dead Replacement | None (Transform) |
| Die Sound | SoldierDie |
| Not Parent | ☐ |
| Remove Body | ✓ |
| Body Stay Time | 15 |
| Gun Stay Time | 30 |
| Slo Mo Death Chance | 0 |
| Slo Mo Backstab | ✓ |
| Slo Mo Death Time | 0.9 |
| Slo Mo Backstab Time | 0.9 |

NPC Attack (Script)
Audio Source
Nav Mesh Agent
Remove Body (Script)
AI Saver (Script)
Script AISaver
Key

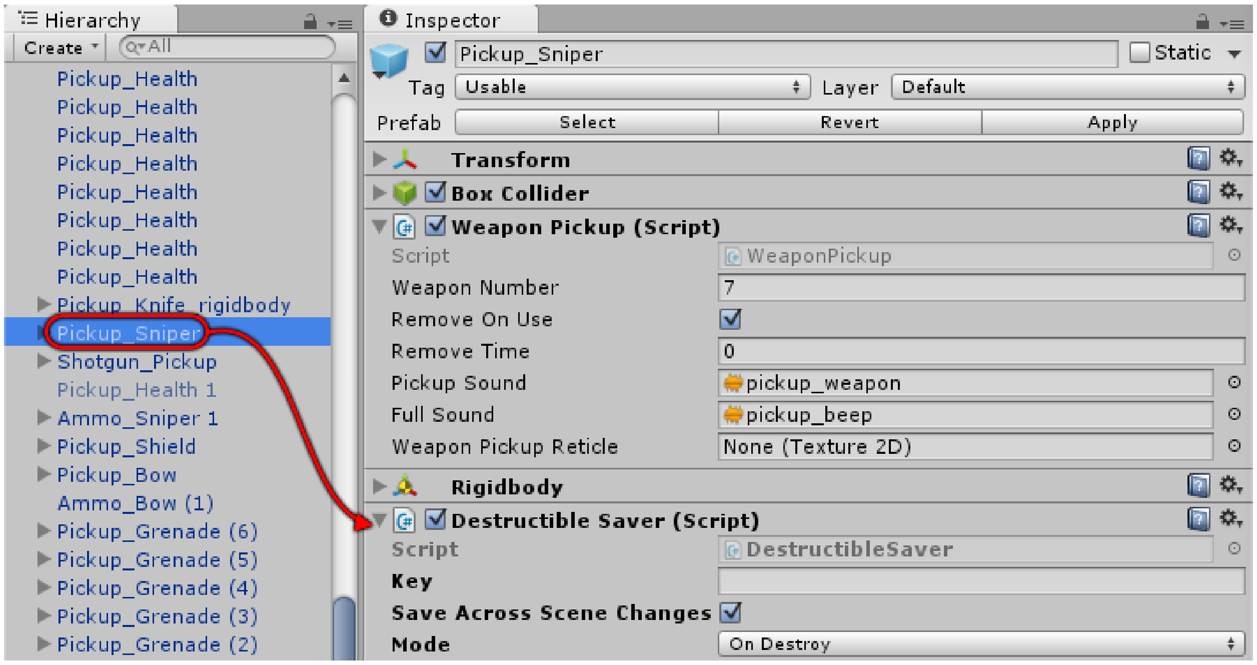
Optionally, inspect the Character Damage component, and assign a corpse prefab to the **Dead Replacement** field. (If you've enabled the Remove Body component, don't assign a corpse prefab.)

## Other Savers: Position, Active, Enabled, Animator

To save a GameObject’s position, add a **Position Saver**. To save its animator state, add an **Animator Saver**. To save a GameObject’s active/inactive state add an **Active Saver**. To save a component’s enabled state, add an **Enabled Saver**. In both cases, put the saver on a GameObject that’s guaranteed to be active. For RFPSP pickups, use Destructible Savers instead as described below.

## Pickups and Destructibles

To remember the state of a pickup or destructible in saved games and across scene changes, add a **Destructible Saver**.



```
Pickup_Sniper
Tag     Usable        Layer  Default
Prefab

Weapon Pickup (Script)
Script                  WeaponPickup
Weapon Number           7
Remove On Use           ✓
Remove Time             0
Pickup Sound            pickup_weapon
Full Sound              pickup_beep
Weapon Pickup Reticle   None (Texture 2D)

Destructible Saver (Script)
Script                      DestructibleSaver
Key
Save Across Scene Changes   ✓
Mode                        On Destroy
```

If you’re configuring multiple GameObjects with the same name, assign a unique **Key** to each.

Tick **Save Across Scene Changes** to remember the change when leaving the scene and returning to it. This will increase the size of the saved game file.

## Saver Keys

Every saver component needs a unique key. If the Key field is blank, it will use the GameObject name (e.g., “Orc”). To automatically assign unique keys to every saver in a scene, select menu item **Tools > Pixel Crushers > Common > Save System > Assign Unique Keys…**.

## Spawned (Dropped) Objects

This section describes how to configure dropped objects to be saved in saved games.

Dropped objects are managed by a component called **Spawned Object Manager**. Although it has the word "Spawned" in the name, it has nothing to do with RFPSP's NPC spawn system. "Spawned" in this case means RFPSP has dropped a pickup object into the scene, typically when the player drops a weapon. The Spawned Object Manager keeps track of these dropped pickup objects. When you load a game, it re-instantiates the pickup object there.

This is an overview of the configuration process:

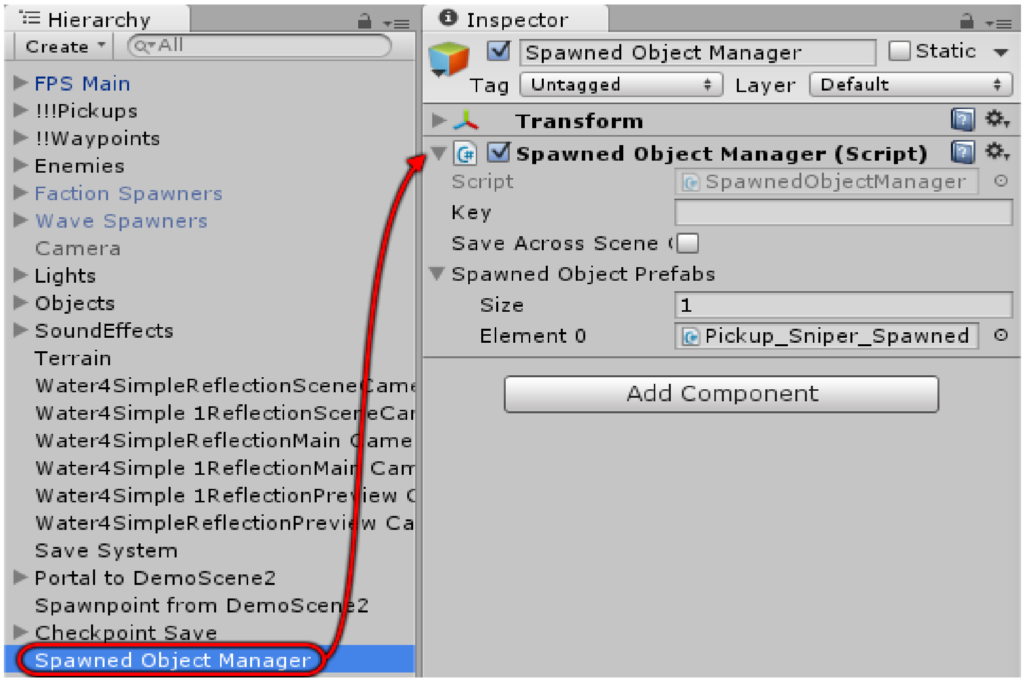
1. Create a copy of a pickup prefab, and add a **Spawned Object** component.
2. Add a **Spawned Object Manager** to the scene, and assign the prefab to it.
3. Configure the player to drop instances of this prefab instead of the regular RFPSP pickup.

### Create Spawned Object Prefab

Find the prefab (such as Pickup_Sniper), and make a copy, for example named Pickup_Sniper_Spawned. Add a **Spawned Object** component to it. Repeat for all items that the player can drop. The Prefabs folder contains preconfigured prefabs for the standard RFPSP pickups.

### Create Spawned Object Manager

Drag the **Spawned Object Manager** prefab from the Prefabs folder into the scene, or create an empty GameObject and add an **FPS Spawned Object Manager** component. Each scene should have its own Spawned Object Manager.



Spawned Object Manager — Static
Tag: Untagged — Layer: Default

Spawned Object Manager (Script)
Script: SpawnedObjectManager
Key:
Save Across Scene: ☐
Spawned Object Prefabs
Size: 1
Element 0: Pickup_Sniper_Spawned

Add *all* Spawned Object prefabs that the player can drop to the **Spawned Object Prefabs** list. If a prefab is missing from the list, the Spawned Object Manager will not be able to re-spawn it when loading games or returning to the scene. The Spawned Object Manager prefab's Spawned Object Prefabs list contains all of the standard RFPSP pickups already.

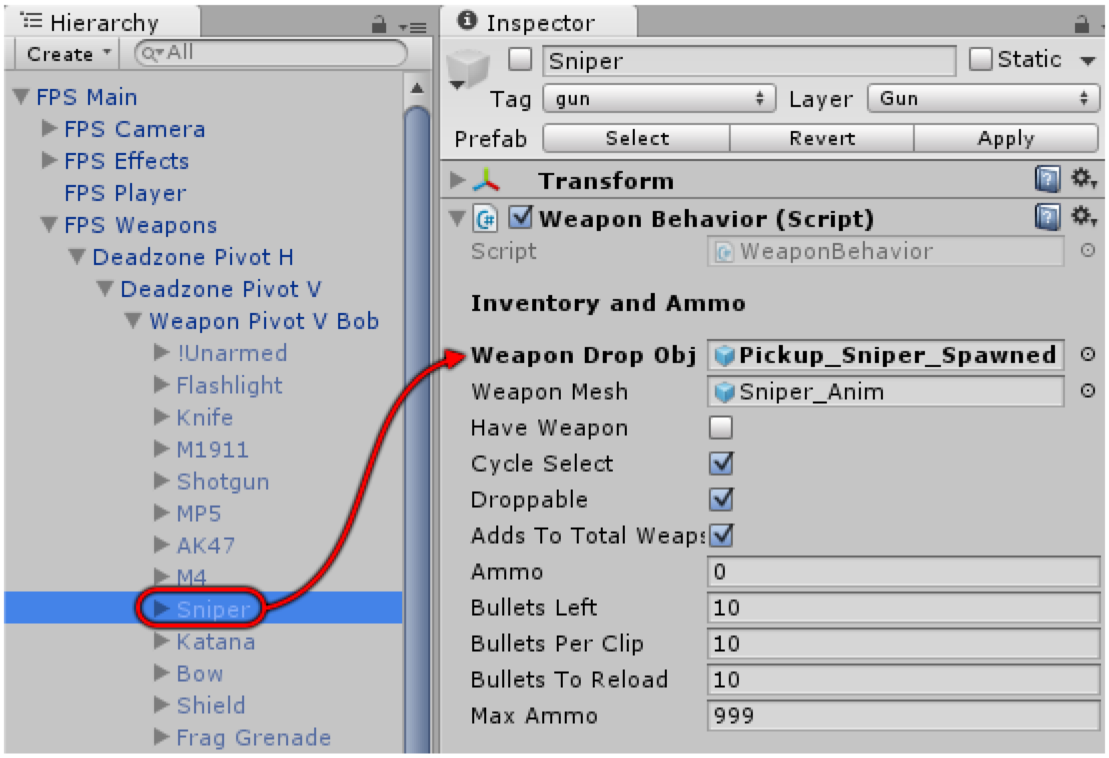
## Arrows

If you want to save the state of arrows that have been shot into the scene so players can retrieve them, add these two lines to the beginning of the Update() method in ArrowObject.cs:

```
void Update () {
 if (FPSPlayerComponent == null) return; // ADD THIS LINE (1/2)
 if (myMeshRenderer == null) InitializeProjectile(); // ADD THIS LINE (2/2)
```

## Configure Player's Spawned Object Drops

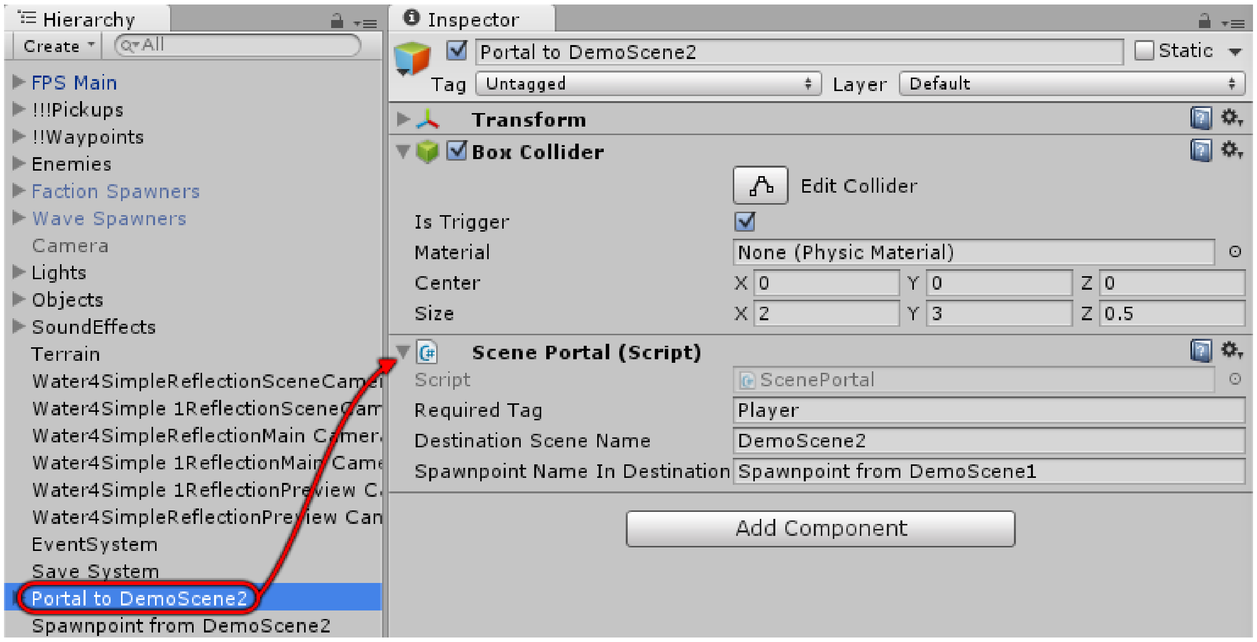
Inspect each of the player's weapons (located under **FPS Weapons**). Assign the spawned object prefab to the **Weapon Drop Obj** field as shown below, replacing the original version of the prefab.



Sniper
Tag gun
Layer Gun
Script WeaponBehavior
Weapon Drop Obj Pickup_Sniper_Spawned
Weapon Mesh Sniper_Anim
Have Weapon ☐
Cycle Select ✓
Droppable ✓
Adds To Total Weaps ✓
Ammo 0
Bullets Left 10
Bullets Per Clip 10
Bullets To Reload 10
Max Ammo 999

## Scene Portals

To set up a transition to another scene, drag the **Scene Portal** prefab from the Prefabs folder into your scene, or add a GameObject with a trigger collider and add a **Scene Portal** component.



FPS Main
!!!Pickups
!!Waypoints
Enemies
Faction Spawners
Wave Spawners
Camera
Lights
Objects
SoundEffects
Terrain
Water4SimpleReflectionSceneCamer
Water4Simple 1ReflectionSceneCam
Water4SimpleReflectionMain Camer
Water4Simple 1ReflectionMain Came
Water4Simple 1ReflectionPreview C
Water4SimpleReflectionPreview Can
EventSystem
Save System
Portal to DemoScene2
Spawnpoint from DemoScene2

Portal to DemoScene2
Tag Untagged
Layer Default

Box Collider
Is Trigger ✓
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 2 Y 3 Z 0.5

Scene Portal (Script)
Script ScenePortal
Required Tag Player
Destination Scene Name DemoScene2
Spawnpoint Name In Destination Spawnpoint from DemoScene1

Set **Destination Scene Name** to the scene that this portal leads to. Make sure the destination scene is in your project’s build settings.

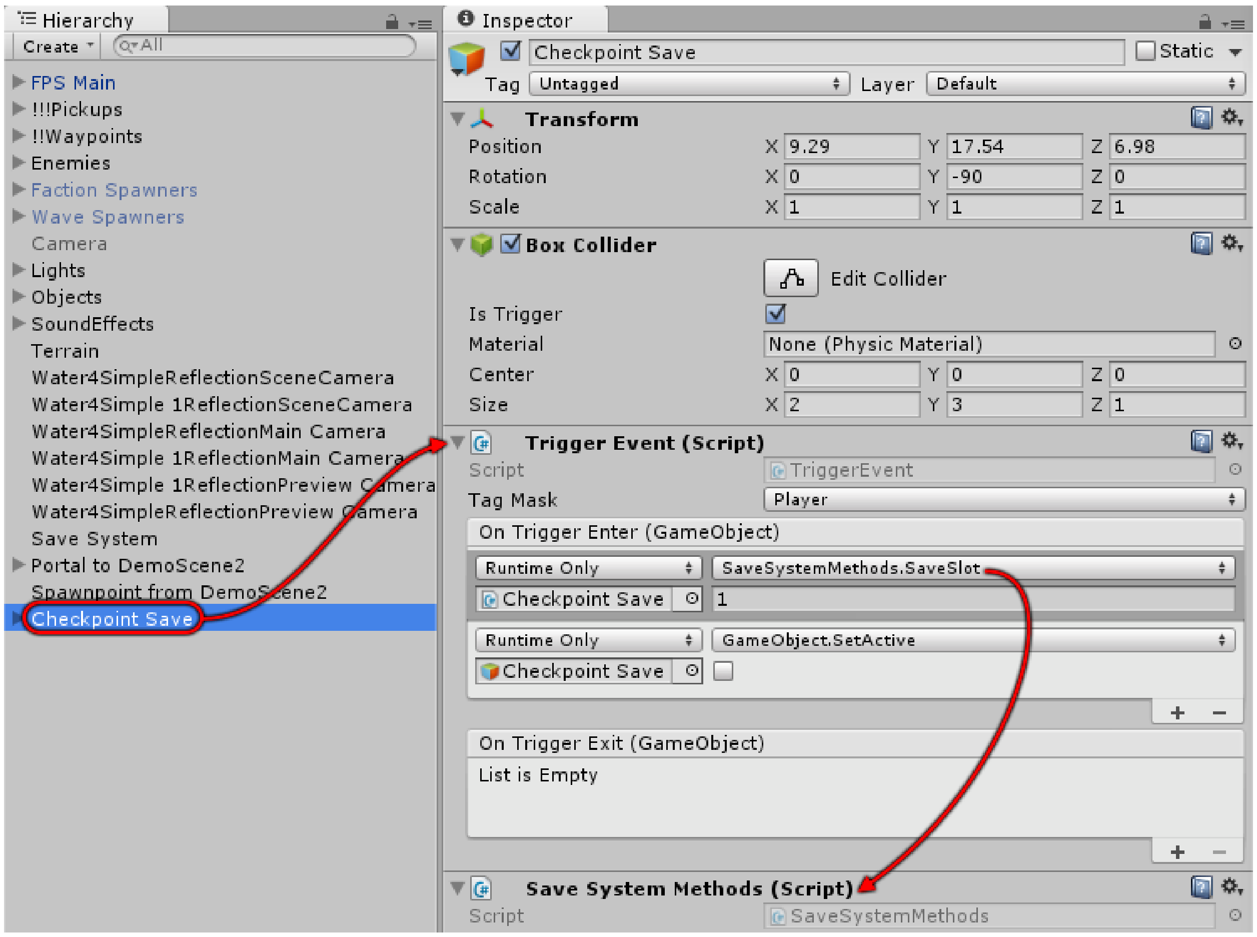
In the destination scene, create an empty GameObject where the player should appear. This is called a *spawnpoint*. In the original scene’s Scene Portal component,set the **Spawnpoint Name In Destination Scene** field to the name of that spawnpoint.

Make sure your spawnpoint is far enough away from any scene portals in the destination scene so it won’t immediately trigger another scene change.

If you don’t want to use trigger colliders, you can manually call the **ScenePortal.UsePortal** method.

## Checkpoint Saves

To set up a checkpoint save trigger, drag the **Checkpoint Save** prefab from the Prefabs folder into the scene, or add a GameObject with a trigger collider and add **Trigger Event** and **Save System Methods** components.



Hierarchy
Create
All

FPS Main
!!!Pickups
!!Waypoints
Enemies
Faction Spawners
Wave Spawners
Camera
Lights
Objects
SoundEffects
Terrain
Water4SimpleReflectionSceneCamera
Water4Simple 1ReflectionSceneCamera
Water4SimpleReflectionMain Camera
Water4Simple 1ReflectionMain Camera
Water4Simple 1ReflectionPreview Camera
Water4SimpleReflectionPreview Camera
Save System
Portal to DemoScene2
Spawnpoint from DemoScene2
Checkpoint Save

Inspector
Checkpoint Save — Static
Tag: Untagged — Layer: Default

**Transform**

| | X | Y | Z |
|---|---|---|---|
| Position | 9.29 | 17.54 | 6.98 |
| Rotation | 0 | -90 | 0 |
| Scale | 1 | 1 | 1 |

**Box Collider**
Edit Collider
Is Trigger: ✓
Material: None (Physic Material)

| | X | Y | Z |
|---|---|---|---|
| Center | 0 | 0 | 0 |
| Size | 2 | 3 | 1 |

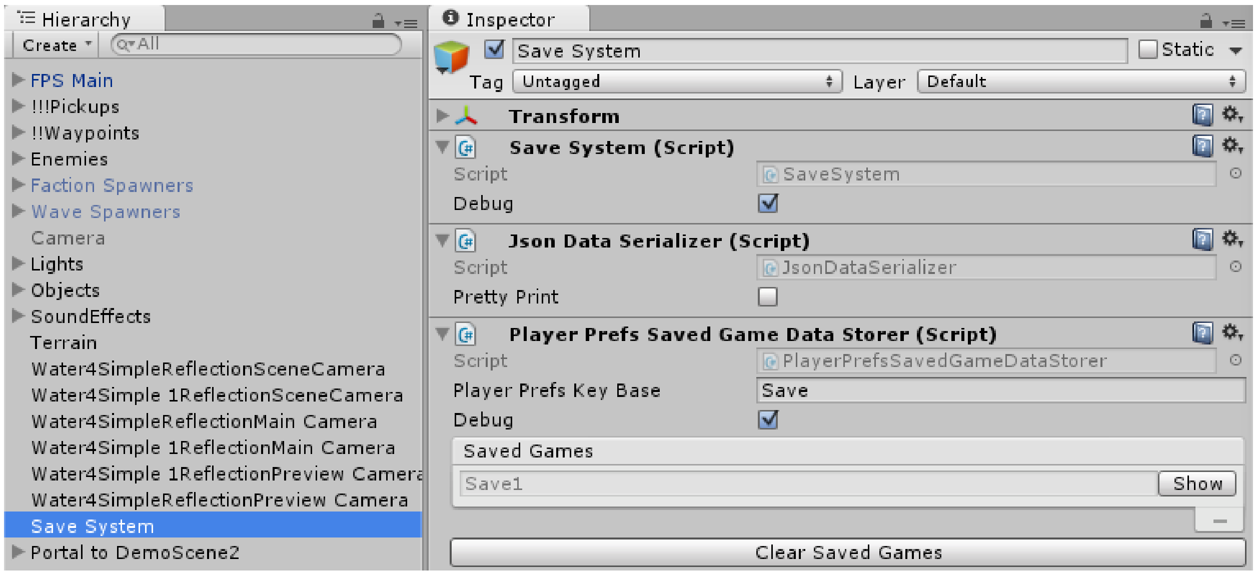
**Trigger Event (Script)**
Script: TriggerEvent
Tag Mask: Player
On Trigger Enter (GameObject)
Runtime Only — SaveSystemMethods.SaveSlot
Checkpoint Save — 1
Runtime Only — GameObject.SetActive
Checkpoint Save — ☐
On Trigger Exit (GameObject)
List is Empty

**Save System Methods (Script)**
Script: SaveSystemMethods

If you added the components manually, configure the Trigger Event's **On Trigger Enter** event to call **SaveSystemMethods.SaveSlot**, and specify a slot number. Optionally, disable the trigger after the checkpoint save by adding another event handler for GameObject.SetActive as shown above.

## Save System

This step is optional. The Save System will automatically create a Save System GameObject at runtime. However, you can manually add one if you want to customize it. To add it, drag the **Save System** prefab from the Prefabs folder into your scene, or create a GameObject and add a **Save System** component. This GameObject acts as a *persistent singleton*, meaning it survives scene changes and typically there is only one instance.


Hierarchy
Create
All
FPS Main
!!!Pickups
!!Waypoints
Enemies
Faction Spawners
Wave Spawners
Camera
Lights
Objects
SoundEffects
Terrain
Water4SimpleReflectionSceneCamera
Water4Simple 1ReflectionSceneCamera
Water4SimpleReflectionMain Camera
Water4Simple 1ReflectionMain Camera
Water4Simple 1ReflectionPreview Camera
Water4SimpleReflectionPreview Camera
Save System
Portal to DemoScene2
Inspector
Transform
Show
Clear Saved Games

Save System
Static
Tag Untagged
Layer Default

Save System (Script)
Script SaveSystem
Debug ✓

Json Data Serializer (Script)
Script JsonDataSerializer
Pretty Print

Player Prefs Saved Game Data Storer (Script)
Script PlayerPrefsSavedGameDataStorer
Player Prefs Key Base Save
Debug ✓
Saved Games
Save1

The Save System relies on two components:

- **Data Serializer**: Converts binary saved game data into a saveable format.
- **Saved Game Data Storer**: Writes serialized data to persistent storage such as PlayerPrefs or local disk files.

By default, the Save System uses **Json Data Serializer**. If you want to use a different serializer, you can add a **Binary Data Serializer** or your own implementation of DataSerializer to the Save System.

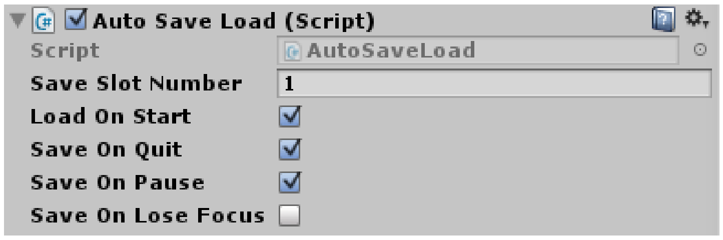
By default, the Save System uses **PlayerPrefs Saved Game Data Storer** to save games to PlayerPrefs. The Save System also ships with a **Disk Saved Game Data Storer** that saves games to encrypted local disk files; to use it, remove the PlayerPrefs Saved Game Data Storer (if present) and add Disk Saved Game Data Storer. If you want to store games a different way, you can add your own implementation of the SavedGameDataStorer class.

If you have a menu system, to load and save games you can assign the methods **SaveSystem.SaveGameToSlot** and **SaveSystem.LoadGameFromSlot** to your UI buttons' **OnClick()** events. Or in scripts you can use the equivalent static methods **SaveSystem.SaveToSlot** and **SaveSystem.LoadFromSlot** described in the next chapter.

Similarly, to change scenes without using the Scene Portal component, you can call **SaveSystem.LoadScene** in scripts or assign **SaveSystem.LoadScene** to your UI buttons' **OnClick()** events. Since the Save System GameObject might not be in all of your scenes, you can add a **Save System Methods** component and direct OnClick() to its LoadScene method instead.

## Auto Save Load

Mobile games typically auto-save when the player closes the game and auto-load when the player resumes the game. To add this behavior to your game, add an **Auto Save Load** component to the Save System:



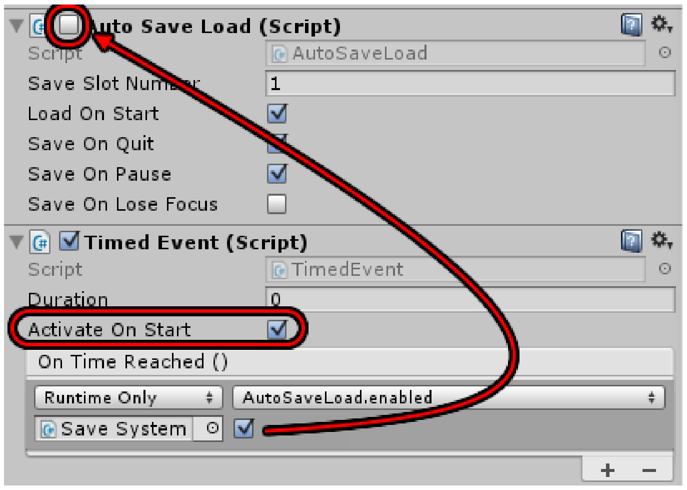
Auto Save Load (Script)
Script: AutoSaveLoad
Save Slot Number: 1
Load On Start: ✓
Save On Quit: ✓
Save On Pause: ✓
Save On Lose Focus: ☐

Tick **Load On Start** to load the saved game (if it exists) on start, and **Save On Quit** to save the game when quitting the app.

Tick **Save On Pause** to also save the game when the player pauses/minimizes it. This way the game will be saved correctly if the player pauses the app and then kills it, instead of exiting it normally in the app itself.

Tick **Save On Lose Focus** to also save the game when the app loses focus.

If your game uses a splash screen, you may not want to enable Auto Save Load until the first scene actually loads. Otherwise, if the player minimizes the app during the splash screen, the Auto Save Load component could save the game in its initial state, before it has loaded its previously-saved data. To set this up, disable Auto Save Load, and use a **Timed Event** component to enable it on Start:

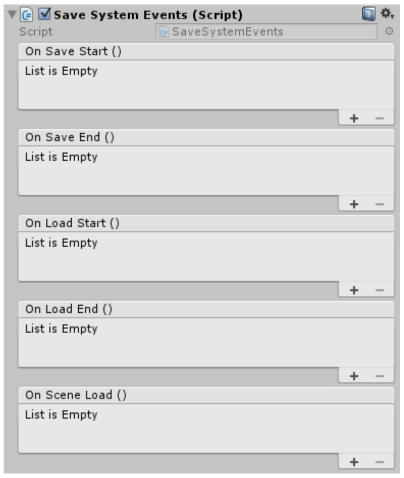


Auto Save Load (Script)
Script: AutoSaveLoad
Save Slot Number: 1
Load On Start: ✓
Save On Quit: ✓
Save On Pause: ✓
Save On Lose Focus: ☐

Timed Event (Script)
Script: TimedEvent
Duration: 0
Activate On Start: ✓
On Time Reached ()
Runtime Only | AutoSaveLoad.enabled
Save System | ✓

## Save System Events

The Save System Events components allows you to hook up events in the inspector.



Save System Events (Script)

Script SaveSystemEvents

On Save Start ()

List is Empty

On Save End ()

List is Empty

On Load Start ()

List is Empty

On Load End ()

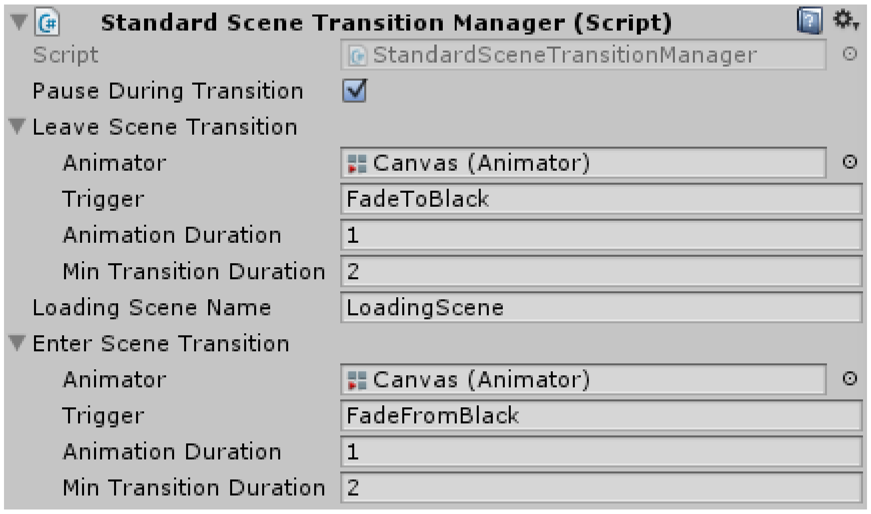
List is Empty

On Scene Load ()

List is Empty

## Scene Transition Manager

To play outro and intro animations, and/or show a loading scene while loading the next actual scene, add a **Standard Scene Transition Manager** to your Save System:



Standard Scene Transition Manager (Script)

| | |
|---|---|
| Script | StandardSceneTransitionManager |
| Pause During Transition | ✓ |
| Leave Scene Transition | |
| Animator | Canvas (Animator) |
| Trigger | FadeToBlack |
| Animation Duration | 1 |
| Min Transition Duration | 2 |
| Loading Scene Name | LoadingScene |
| Enter Scene Transition | |
| Animator | Canvas (Animator) |
| Trigger | FadeFromBlack |
| Animation Duration | 1 |
| Min Transition Duration | 2 |

If **Pause During Transition** is ticked, make sure your Animator(s) are set to Unscaled Time.

If a Scene Transition Manager is present, the Save System will:

1. Set the Leave Scene Transition's animator trigger (if specified).
2. Load the loading scene (if specified).
3. Asynchronously load the next actual scene.
4. After the actual scene has been loaded, set the Enter Scene Transition's trigger (if specified).

## How To Respawn With Saved Data

If you want the player to respawn with their saved inventory after dying, you will need to edit RFPSP's FPSPlayer.cs script in two places.

**Edit #1**: Replace the last "}" in the Start() method with this:

```
    if (!string.IsNullOrEmpty(savedDataToLoadOnStart)) //[PixelCrushers]
    {
        Debug.Log("Applying saved inventory.");
        GetComponent<PixelCrushers.SaveSystemForRFPS.FPSPlayerSaver>().
            ApplyInventory(savedDataToLoadOnStart);
        savedDataToLoadOnStart = string.Empty;
    }
}
public static string savedDataToLoadOnStart = string.Empty; //[PixelCrushers]
```

**Edit #2**: Add this line (marked with `[PixelCrushers]`) at line ~1269:

```
//Call Die function if player's hitpoints have been depleted
if (hitPoints < 1.0f){
    savedDataToLoadOnStart = GetComponent<PixelCrushers.SaveSystemForRFPS.FPSPlayerSaver>().
        RecordData(); //[PixelCrushers]
    SendMessage("Die");//use SendMessage() to allow other script components on this object to detect player death
}
```

# Chapter 3: Scripting

The Save System has a very simple scripting interface.

## SaveSystem class

These methods are available though SaveSystem:

```
public static void LoadFromSlot(int slotNumber)
```
Loads a previously-saved game from a slot.

```
public static void SaveToSlot(int slotNumber)
```
Saves the current game to a slot.

```
public static void DeleteSavedGameInSlot(int slotNumber)
```
Deletes the data in a saved game slot.

```
public static void LoadScene(string sceneNameAndSpawnpoint)
```
Loads a scene, optionally positioning the player at a specified spawnpoint. The parameter is a string containing the name of the scene to load, optionally followed by "@*spawnpoint*" where "*spawnpoint*" is the name of a GameObject in that scene. The player will be spawned at that GameObject's position. This method implicitly calls RecordSavedGameData() before leaving the current scene and calls ApplySavedGameData() after loading the new scene.

```
public static SavedGameData RecordSavedGameData()
```
Returns a SavedGameData object containing the saved data from the current game.

```
public static void ApplySavedGameData(SavedGameData savedGameData)
```
Applies a SavedGameData object to the current game.

```
public static void LoadGame(SavedGameData savedGameData)
```
Loads a game previously saved in a SavedGameData object.

```
public static void LoadScene(string sceneName, string spawnpointName = null)
```
Loads a scene, optionally positioning the player at a spawnpoint in the new scene.

```
public static void RestartGame(string startingSceneName)
```
Clears the current save game data and restart the game at the specified scene.

## Saver class

Saver is the base class for any components that record data for saved games. You can create subclasses to extend the data that the Save System saves. A starter template with detailed comments is provided in Plugins ► Pixel Crushers ► Common ► Templates ► SaverTemplate.cs .

## DataSerializer class

DataSerializer is the base class for data serializers that the Save System can use to serialize and deserialize saved game data. The default subclass is JsonDataSerializer.

### SavedGameDataStorer class

SavedGameDataStorer is the base class for data storers. A data storer writes and reads a SavedGameData object to and from some storage location, such as PlayerPrefs or a disk file. The default subclass is PlayerPrefsSavedGameDataStorer, but the Save System also includes DiskSavedGameDataStorer which saves to encrypted disk files on supported platforms (e.g., desktop).

### Events

You can register listeners for these C# events:

```
public static event SceneLoadedDelegate sceneLoaded = delegate { };
public static event System.Action saveStarted = delegate { };
public static event System.Action saveEnded = delegate { };
public static event System.Action loadStarted = delegate { };
public static event System.Action loadEnded = delegate { };
```

## Save System Methods

To access Save System methods without scripting, such as in a UI Button's OnClick() event, add a **Save System Methods** component to the scene. This component exposes the methods of the SaveSystem class to the inspector.

# Chapter 4: Serializers

The default serializer is JSON Data Serializer. However, you can remove this and add a Binary Data Serializer if you want to serialize to binary format. If your Savers use types that are not serializable, you will need to make a subclass of Binary Data Serializer and add serialization surrogates. For examples, see the scripts BinaryDataSerializer.cs, Vector3SerializationSurrogate.cs, and QuaternionSerializationSurrogate.cs.

# Chapter 5: Third Party Support

Integration support for third party assets is included the unitypackages listed below. Documentation is included in each third party support package.

## Compass Navigator Pro Integration

*Compass Navigator Pro © Kronnect*

Plugins ► Pixel Crushers ► Common ► Third Party Support ► Compass Navigator Pro Support

## PlayMaker Integration

*PlayMaker © Hutong Games*

Plugins ► Pixel Crushers ► Common ► Third Party Support ► PlayMaker Support

## Tactical Shooter AI Integration

*Tactical Shooter AI © Squared 55*

Plugins ► Pixel Crushers ► Common ► Third Party Support ► Tactical Shooter AI Support