Smooth Save for PlayMaker

Smooth Save is an asset for PlayMaker 1.7 - 1.8 and Unity 4.6 - 5.x. It provides 37 PlayMaker actions for easy and robust saving and loading of game data.

Smooth Save requires the PlayMaker asset, which is sold separately by Hutong Games.
https://www.assetstore.unity3d.com/#/content/368

If you have any questions, problems, or feedback, feel free to drop me a line. You can contact me at: support@ChristopherCreates.com

Enabling PlayMaker 1.7 Support

By default, Smooth Save is configured for PlayMaker 1.8. If you want to use it with 1.7, just delete the `Scripts` folder and extract the replacement from `PlayMaker1_7.zip`.

Supported Variable Types

Smooth Save fully supports PlayMaker variables of types `Bool`, `Color`, Enum, `Float`, `Int`, `Quaternion`, `Rect`, `String`, `Vector2`, and `Vector3`. These are the “basic” types.

For `GameObject` variables, it will save their active state, layer, name, tag, position, rotation, and scale. (Each of these is a basic type which is saved individually behind the scenes.)

With PlayMaker 1.8 or higher, the new `Array` variable is supported in the types above.

Any other value of a basic type in the game can be saved indirectly. Just use a `Get Property` action to copy the value into a basic variable. That variable can then be saved in the usual manner. Similarly, use a `Set Property` action to restore the value after loading.

Using the Walkthrough and Example Scenes

The easiest way to get started with Smooth Save is with the demo scenes. Look in the `Smooth Save for PlayMaker` folder and you'll find `Smooth Save Walkthrough`. This scene is an annotated tour of the actions. To see those actions in action, look in the `Examples` folder. There you'll find various scenes that demonstrate saving and loading in common scenarios. Between them, that might be all you need to get started. For more details, just keep reading this documentation.

One-Action Saving and Loading

If you want to keep things simple, you don't really need to know much about how Smooth Save works to use it. Just put all the variables you need to save and load in either one FSM or the Globals, and then you can save or load your entire game with just one action.

To save, use either the `Copy FSM To Pool` or `Copy Globals To Pool` action, depending on where you put your variables. Just leave all the options at their default settings (especially `Save Now` as checked). When this action is run, it will automatically save all of the variables in that location. There's nothing else you need to do.

Loading is very similar. Just use either the `Copy Pool To FSM` or `Copy Pool To Globals` action. Again, leave the options at their default settings (especially `Load Now` as checked). When this action is run, it will load the saved data and automatically copy it into the matching variables in that location. It's that easy!

Note that this may not work the way you want with variables whose owners have the same name, such as in instances of a prefab. See *Identifying Variable Data* below for details.

How Smooth Save Works

The Data Pool

Smooth Save keeps everything you want to save in a data pool. This includes the values of your variables and information to identify where the data came from. Every data pool has a file that it is stored in. Only one pool can be loaded at a time. For details, see *Managing Data Pools* below.

Saving Variables

1. `Copy Variable To Pool`
2. `Save Data Pool`

All data starts where you put it in a PlayMaker variable. To save it, you copy that variable into the data pool. This is repeated for each variable you want to save. When it's all ready, you then save the data pool to its file.

Loading Variables

1. Load Data Pool
2. Copy Pool To Variable

When loading, the same process works in reverse. The data pool file is first loaded. Then data from the pool is copied back into each variable.

Saving and Loading Groups of Variables

- `Copy FSM To Pool`, `Copy Pool To FSM`
- `Copy Globals To Pool`, `Copy Pool To Globals`

You can also save or load some or all of the variables in an FSM or the Globals at one time using the actions above.

The `Filter` option lets you select variables by name. Use the format “abc*” to match the beginning of names, “*abc” for the end of names, or just “abc” for anywhere in the name. Leaving it blank will match all variables.

The `Type` option lets you select variables of just one type.

`Load Now` and `Save Now` are convenience options that are the same as using the `Load Data Pool` or `Save Data Pool` actions before or after the copy.

Action Result Events

There are four types of events that can be called as a result of the actions.

The `Data Not Found Event` can be specified when trying to copy data from the pool to a variable. This event is called if no data is found in the pool that matches the variable you're trying to copy it into. See *Identifying Variable Data* below for details.

The `Not Loaded Event` can be specified when trying to access the current data pool. This event is called if there is no data pool loaded.

The `File Not Found Event` can be specified when trying to access a data pool file. This event is called if the file is not found.

The File Error Event can be specified when trying to access a data pool file. This event is called if there is an error accessing the save data or the data is corrupted.

Identifying Variable Data

When data is saved to file, it loses any direct connection to the variable it came from. So each piece of data must be identified in some way to know where to put it when it's loaded. It's important to understand these identifications to avoid misplaced data.

FSM Variables: Level Name + Owner Name + FSM Name + Variable Type + Variable Name
Global Variables: Variable Type + Variable Name
Keyed Variables: Variable Type + Key

Important: FSM variable identification uses the *names* of various items. So if different variables have the same set of names, such as in instances of a prefab, they will all identify the same and all refer to the same data in the pool. If you need each variable to have its own data, then either use keys or give each a unique owner or FSM name.

Using Keys To Identify Data

If you need more control over how your data is saved and loaded, you can specify a `Key`. This is simply a string that uniquely identifies a piece of data. It's up to you to create the string in a way that is meaningful for your program. Keys are useful for saving and loading where automatic identifiers might fail. Even more powerfully, keys also allow you to save and load the same data into completely different variables of the same type.

Managing Data Pools

Default Data Pool

For most actions, if no data pool is specified Smooth Save will find or create one by default. First, the currently loaded pool will be used. If no pool is loaded, then it will load the data pool file with the lowest ID. If there are no data pool files, it will create a new data pool.

This means that if you only need one data pool, you never have to think about them at all. Smooth Save will handle everything automatically.

Multiple Data Pools

To create additional pools, use the `Make Data Pool` action. Every data pool has an ID that is unique in the current set of pools. Use this ID to load or delete a specific pool. Note that if you delete a pool, its ID is freed up and will be used again if another pool is created.

Pool Names

Each data pool has a name associated with it. This is purely for your convenience. The name may be null or empty, and can be the same between pools. You can set the name when the pool is created with `Make Data Pool` or any time with `Set Data Pool Name`.

Special Platforms and PlayerPrefs

Some platforms like the web player and some consoles lack a conventional file system. If you're publishing to these, Smooth Save cannot save its data to files in the usual manner. In this event, use the `Use PlayerPrefs` action.

This action tells Smooth Save to save data as XML and store it in a string in PlayerPrefs. This is a bit slower and uses a bit more memory than saving to files, but it should work on any platform.

If the `Encode Data` option is checked, it will encode the XML as Base64 to make it cheat-resistant. This is similar to the default binary serialization, see *Cheat-Resistant Data* below. Encoding the data uses more memory.

NOTE: You only need to use this action once, but be sure to use it before any other Smooth Save actions.

Other Actions

The `Get Data Pool Info` action returns the ID, name, and size of the current data pool.

The `Unload Data Pool` action removes the current data pool from memory. It does not affect the saved data pool file or any variables that have used it.

The `Delete Data Pool` action deletes a data pool file and, if loaded, unloads it.

The `Debug Data Pool` action will display the contents of the data pool on the console. This includes all of the information Smooth Save is using to identify the data, so it can be very useful for figuring out what is going where.

Final Details

Data Location

Smooth Save stores each data pool in a “SmoothSaveNNNN” file, where NNNN is the data pool ID. This file is created in `UnityEngine.Application.persistentDataPath`, which varies by which platform you are publishing on. It is safe to manually delete any data pool file you don't need.

Cheat-Resistant Data

Some forms of saving data in Unity ultimately store it in a human-readable text file (such as XML or JSON). This works fine, but it also means that illicitly modifying the save game data is trivial. Smooth Save uses binary serialization to store the data, which is not human readable and cannot be changed with a simple text editor. This is not encryption and anyone who has programming skills could still modify the data, but it does prevent casual cheating by general users.

If you like Smooth Save, please tell everyone!

Leave a review on the Asset Store.

If you don't like Smooth Save, please tell me!

Let me know how I can help: support@ChristopherCreates.com

Thank you!

These creations are a labor of love for me. It's very rewarding to know that what I make goes out into the world and helps people. Please feel free to contact me if you need any assistance or have any feedback.

Christopher