SimpleConsole

1 Table of Contents

1. Table of Contents
2. Quick Start
3. General Information
4. Usage
5. Built-in Commands
6. Console Functions from Code
7. Customize Console Behaviour
8. Contact

2 Quick Start

1. Import the downloaded package.
2. Drag the "__ConsoleObject" prefab from "Plugins/SimpleConsole" folder into the scene.
3. Press Play.
4. During play, press the backquote (`) key to open the console (on a standard QWERTY keyboard, it's to the left of the '1' key.)
5. Press 'esc' or the backquote key to close the console.

3 General Information

Download and import the SimpleConsole package from the asset store. This should create a folder called Plugins (if it didn't already exist) with a folder inside called SimpleConsole that contains an ease-of-use console prefab and various scripts.

The console provides a GUI frontend (currently, the GUI frontend cannot be changed), which displays all output and accepts input (with a box displaying command suggestions/descriptions). The GUI frontend uses the Unity GUI system but does not support changing the GUI skin currently.

4 Usage

The usage of the console is relatively straightforward. You can open the console with the backquote key "`". With the console opened, you can type in commands in the input field that is automatically focused. When typing commands, a box underneath the input field will appear to display command suggestions (if there are any), you can press the "tab" key to have the first suggestion placed into the input field. If you wish to use a command that you have used previously, you can press the up and down arrow keys to go through your command history (if any), placing the command automatically in the input field. When finished, you can press the "esc" key or the backquote key to close the console.

Entered commands will take the form:

Command parameter1 parameter2 ... parameter

These are simple commands that are directly linked to a method defined in your scripts. By default the console comes with a few commands “help”, “history” and “clear”. Commands cannot contain spaces, except between parameters (i.e. “history –c”). Commands are also case-sensitive. It is possible to get more information about a specific command by typing “help command” by replacing command with the command you want to know more about (i.e. “help history”). These commands output are generally the bulk of what is displayed in the console. Parameters can have spaces if they are placed inside of double quotation marks (i.e. command “hello world” parameter2…). Parameters inside of double quotation marks will be used as one long string, so if all of the parameters are accidentally placed inside of a set of double quotation marks, then everything will come out as one parameter.

5 Built-in Commands

help
Outputs information about getting started with commands to the console.

history
Outputs all of the commands that you have entered into the console.

clear
Clears all output that has been sent to the console (does not clear command history).

quit
Exits the application.

exit
Exits the application.

object.list
Lists all of the active GameObjects.

object.select
Selects or deselects a GameObject.

object.delete
Destroys the selected GameObject.

object.position
Gets or sets the position of the selected GameObject.

object.rotation
Gets or sets the position of the selected GameObject.

object.scale
Gets or sets the scale of the selected GameObject.

6 Console Functions From Code

print

```
public static void print(string msg)
public static void print(string msg, Color clr)
```

Prints the given message to the console in white (first print) or in the color of your choice (second print). Can be used for debugging messages.

eval

```
public static void eval(string cmdLine)
```

Executes the given command as if entered in the console.

7 Customize Console Behaviour

Adding a Custom Command

You can easily add a custom command by calling the AddCommand on the console instance.

```
SimpleConsole.AddCommand("customCommand", "command description", CustomCommand);
```

The first parameter is the name of the command. The second parameter is the description of the command. The third parameter is the method to call when the command is used. This is the basic version of the AddCommand method, there is one overload that accepts a string as a third parameter for the help text (text output to the console when "help command" is called) and the fourth parameter the method to call.

The method called when the command is used is required to be in a specific format:

C#

```
public string CustomCommand(SimpleContainer param)
```

JS

```
function CustomCommand(param : SimpleContainer)
```

The method can be either public or private, it does not make a difference as it will get called either way. The returned string will be output to the console.

Example 7.1a: a complete MonoBehaviour with a custom console command (C#)

```
using UnityEngine;
using Simple;

public class CustomClass : MonoBehaviour
{
   void Start() {
      SimpleConsole.AddCommand("hello", "Outputs \"Hello World!\" to the console.", Hello);
   }

   private string Hello(SimpleContainer args) {
```

```
        // print the number of arguments that were provided to the console
        SimpleConsole.print("Arg Count: " + args.size);

        // output all arguments that were entered to the command, getting the proper data for each arg
        foreach(SimpleObject i in args) {
            SimpleConsole.print("Param: " + (i.isString()?i.sData: i.isFloat()?i.fData: i.isInt()?i.iData:"None"));
        }
        return "Hello World!";
    }
}
```

Example 7.1b: a MonoBehaviour with a custom console command (JS)

```
import UnityEngine;
import Simple;
function Start() {
    SimpleConsole.AddCommand("hello", "Outputs \"Hello World!\" to the console.", Hello);
}

function Hello(args: SimpleContainer) {
    // print the number of arguments that were provided to the console
    SimpleConsole.print("Arg Count: " + args.size);

    // output all arguments that were entered to the command, getting the proper data for each arg
    for(var i : SimpleObject in args) {
        SimpleConsole.print("Param: " + (i.isString()?i.sData: i.isFloat()?i.fData: i.isInt()?i.iData:"None"));
    }
    return "Hello World!";
}
```

Adding a Debug Command

The difference between a debug command and a normal command is that debug commands are only added to the console if it is a debug build. You can easily add a debug command by calling AddDebugCommand on the console instance.

```
SimpleConsole.AddDebugCommand("customCommand", "command description", CustomCommand);
```

The parameters are exactly the same as the normal AddCommand method, including the one overloaded method that accepts four parameters. The method called when the command is used is also required to be in the same format as the normal commands methods.

Removing Commands

It is possible to remove commands that you do not wish to have available anymore by calling the RemoveCommand method (this will not remove commands provided by the console).

```
SimpleConsole.RemoveCommand("customCommand");
```

The only parameter is the name of the command to be removed (remember: commands are case-sensitive). If the command exists, then it will be removed and the method will return true, if not, it will return false and an error will be logged in Unity's debug log.

Example 7.2: a MonoBehaviour that adds and removes a command (C#)

```
using UnityEngine;
using Simple;

public class UseOnce : MonoBehaviour {
   void Start() {
      // the help text output when "help useonce" is called
      string help = "This command can only be used once.\n\n";
      help += "This command restarts the current scene, and then removes itself from the list of available commands.";
      SimpleConsole.AddDebugCommand("useonce", "Restart the current scene once.", help, useOnce);
   }

   private string useOnce(SimpleContainer args) {
      SimpleConsole.RemoveCommand("useonce");
      SimpleConsole.print("Removed command useonce.");
      SimpleConsole.print("Restarting the scene.");
      Application.LoadLevel(Application.loadedLevel);
      return "Restarted the scene.";
   }
}
```

Requiring a Slash for Commands

```
SimpleConsole.RequireSlash(bool val)
```

It is possible to require slashes before commands. This would require a command to look like this instead: /command [argument1] [argument2] … [argument n]. Where command is replaced by the command name. To require slashes, the method RequireSlash should be called with a boolean value of true (false to disable).

8 CONTACT

I would really appreciate comments, questions, bug reports, feature requests, etc. You can contact me at asmith@anthonysjournal.info or you can reach me through my Twitter account @anthonytrilli