Amilious Core

Tools, Editors, and Extension Methods for Unity

**USER GUIDE**

Release 1.0.0  
Oct 2022

Copyright © 2022 Amilious, Brandon James Anderson – All Rights Reserved

This manual, as well as the software described in it, is furnished under license, and may be used or copied only in accordance with the terms of such license. The content of this manual is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by its authors. The author assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual.

Table of Contents

[Getting Started 5](#_Toc96947038)

[Overview 5](#_Toc96947039)

[Installation 5](#_Toc96947040)

[Quick Start 5](#_Toc96947041)

[Support 6](#_Toc96947042)

[Console 7](#_Toc96947043)

[Writing to the Console 7](#_Toc96947044)

[Write 7](#_Toc96947045)

[Write Format 7](#_Toc96947046)

[Write Line 7](#_Toc96947047)

[Write Line Format 7](#_Toc96947048)

[Console Input 8](#_Toc96947049)

[Old Input System 8](#_Toc96947050)

[New Input System 8](#_Toc96947051)

[Other Options 8](#_Toc96947052)

[Commands 9](#_Toc96947053)

[Creating Commands 9](#_Toc96947054)

[Manually Registering Commands 9](#_Toc96947055)

[Single Command Registering 9](#_Toc96947056)

[Register All Commands Within an Object 10](#_Toc96947057)

[Manually Unregistering Commands 10](#_Toc96947058)

[Single Command Unregistering 10](#_Toc96947059)

[Commands Behavior 11](#_Toc96947060)

**Table of Contents**  
  
  
  
[Command Attributes 12](#_Toc96947061)

[Command Attribute 12](#_Toc96947062)

[Group Attribute 12](#_Toc96947063)

[Alias Attribute 12](#_Toc96947064)

[No Help Attribute 13](#_Toc96947065)

[Conditional Command Attribute 13](#_Toc96947066)

[Parameter Attributes 14](#_Toc96947067)

[Description Attribute 14](#_Toc96947068)

[Valid Values Attribute 14](#_Toc96947069)

[Command Name Attribute 14](#_Toc96947070)

[Group Name Attribute 15](#_Toc96947071)

[Min Value Attribute 15](#_Toc96947072)

[Max Value Attribute 15](#_Toc96947073)

[Conditional Parameter Attribute 16](#_Toc96947074)

[IValidValuesForTypeProvider 16](#_Toc96947075)

[Parameter Converters 17](#_Toc96947076)

[Parameter Converters 17](#_Toc96947077)

[Creating A Custom Converter 17](#_Toc96947078)

[Settings 18](#_Toc96947079)

[Console Settings 18](#_Toc96947080)

[Amilious Console Style Sheet 18](#_Toc96947081)

# Getting Started

## Overview

This User Guide was designed to provide *Amilious Core* users with a basic overview of the features and functionality of the library.

## Installation

Once you have downloaded *Amilious Core* from Unity’s Asset Store, go to: “**Assets > Import Package > Custom Package..**.”. In the Import Asset window, find and select the Amilious Core.unitypackage file. After the “Importing package” window appears in Unity, verify that all items to import are selected and then click the Import button in the bottom right of the window.

Support

Qr code

Description automatically generatedShould you have questions or require assistance after watching the videos on YouTube [Amilious Core Youtube Videos](https://www.youtube.com/playlist?list=PLFaVfDE8su4FSS_zG0Qmw7hkMeGMrU0nC) , please join our discord by going to <https://discord.gg/ZeZV5Nycsz/> or scanning the QR code.

# Extensions

## Writing to the Console

### Write

Use the **Write(**“message”**)** or **Write(**style**,**“message”**)** methods of the *AmiliousConsole* class to write to the console. These methods will not add a new line character after the message.

### Write Format

Use the **WriteFormat(**“format”**,** args…**)** or **WriteFormat(**style**,**“format”,args…**)** methods of the *AmiliousConsole* class to write to the console. These methods will not add a new line character after the message.

### Write Line

Use the **WriteLine(**“message”**)** or **WriteLine(**style**,**“message”**)** methods of the *AmiliousConsole* class to write lines to the console. These methods will add a new line character after the message.

### Write Line Format

Use the **WriteLineFormat(**“format”**,** args…**)** or **WriteLineFormat(**style,“format”**,**args…**)** methods of the *AmiliousConsole* class to write to the console. These methods will add a new line character after the message.

# Console Input

***\*\*\* If you change input systems, make sure that you change the EventSystem as well. If you don’t, you will get errors. \*\*\****

## Old Input System

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application

Description automatically generatedIf you are using the old Input system you can adjust the keys by selecting them in the inspector on the *Console Drawer* (ConsoleGUIController).

## New Input System

Graphical user interface, application

Description automatically generatedIf you decide to use the new input system in your project, make sure that the **Amilious.Console** assembly definition has a reference to the input system. If it does not add it and hit apply on the bottom of the inspector.



Graphical user interface, application

Description automatically generatedOnce you have made sure that the reference is set, you can adjust the input by selecting them in the inspector on the *Console Drawer* (ConsoleGUIController).

## Other Options

If you want to set up your own input you can uncheck the AllowInput on the *Console Drawer* (ConsoleGUIController). You can can call the **ToggleConsole**, **NextSuggestion**, **PreviousSuggestion**, and **TakeSuggestion** methods to control the *Console Drawer*.

# Commands

## Creating Commands

**Creating Commands Video:** <https://youtu.be/mqSQzhyZREA/>

A screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generatedCreating a new command is easy. All you need to do is add either the attribute **[Cmd(**“name”**,**”description”**)]**   
or **[Command(**“name”**,**”description”**)]** attribute above the method which you want to make a command. You also need to remember that the first parameter of the method must be “**CommandContext** ctx” and it must have a **void** return type. Any parameters that are required by the command should follow the *CommandContext* ctx parameter.

## Manually Registering Commands

**Registering Commands Video:** <https://youtu.be/Ybwv6zJNCPI/>

Commands need to be registered to use them. Only commands that are not within a *CommandsBehavior* need to be manually registered. This is best done from the **OnEnable()** *MonoBehavior* method. To register commands, you just need to use one of the following methods:

### Single Command Registering

A screenshot of a computer

Description automatically generated with medium confidence“**AmiliousConsole.Instance.RegisterCommand(ComandsObject,nameOfCommand)**”  
With this command you can register a single command using the object the command is in and the name of the command method.

**Commands**

### Register All Commands Within an Object

Text

Description automatically generated“**AmiliousConsole.Instance.RegisterCommands(ComandsObject)**”  
With this command you can register all of the commands within an object.

## Manually Unregistering Commands

**Unregistering Commands Video:**  <https://youtu.be/7xjiTNvHrWI/>

Commands need to be unregistered when their parent object is being destroyed or when the command is no longer needed. Only commands that are not within a *CommandsBehavior* need to be manually unregistered. This is best done from the **OnDisable()** *MonoBehavior* method. To unregister commands, you just need to use one of the following methods:

### Single Command Unregistering

“**AmiliousConsole.Instance.UnregisterCommand(ComandsObject,nameOfCommand)**”

A screenshot of a computer

Description automatically generated with medium confidenceWith this command you can register a single command using the object the command is in and the name of the command method.  
  
  
Unregister All Commands Within an Object

Text

Description automatically generated“**AmiliousConsole.Instance.RegisterCommands(ComandsObject)**”  
With this command you can register all of the commands within an object.

**Commands**

## Commands Behavior

**Commands Behavior Video:** <https://youtu.be/dEzTgE_SuKA/>

Graphical user interface, text, application, chat or text message

Description automatically generatedText

Description automatically generatedA *CommandsBehavior* is the preferred and automated way of managing commands. It will automatically register commands when it is enabled and unregister commands when disabled. A *CommandsBehavior* is an extension of *MonoBehavior* so it can be used in the same way as a *MonoBehavior*. To use a *CommandsBehavior* simply extend the class that contains commands with *CommandsBehavior*. As long as you have the script on an active GameObject in the scene the commands within the class should be registered.

# Command Attributes

## Command Attribute

**Command Attribute Video:** <https://youtu.be/mqSQzhyZREA/>

This attribute is used to create commands that can be used with the *Amilious Console* and must be used on the method that you want to be executed with the command. You can use **[Cmd(**“name”**,**”description”**)]** or **[Command(**“name”**,**”description”**)].** The description is optional but recommended for the attribute.

## Group Attribute

**Group Attribute Video:** <https://youtu.be/me7yAzRoFZU/>

A screenshot of a computer

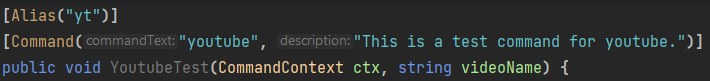
Description automatically generated with medium confidenceText

Description automatically generatedThis attribute can be used on either a class or a method to make the command or commands part of a command group. You can use **[CommandGroup(“**name”**,**”description”**)]** or **[CmdGroup(**“name”**,** “description”**)]**. The description is optional but recommended. It is also recommended that you only set the description once because each attribute will override the groups description. Either that or use a constant for the group description.

## Alias Attribute

**Alias Attribute Video:** <https://youtu.be/3CzOO95ay20/>

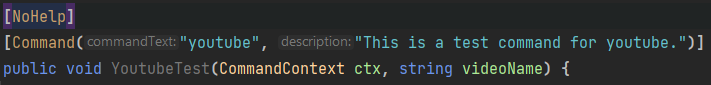
This attribute can be used to add aliases for a command. You can add as many aliases as you wish. If you try to create an alias that is the name of another command, the alias will be disabled. If two commands have the same alias, it will only work for the first command registered with that alias. You can use **[Alias(**“alias1”**,**”alias2”**,**…**)]** to add aliases.



**Command Attributes**

## No Help Attribute

**No Help Attribute Video:** <https://youtu.be/LYAfx43s_os/>

This attribute can be added to commands to hide the command from being displayed with the help command. This will also prevent commands from appearing in the suggestions.

## Conditional Command Attribute

**Conditional Command Attribute Video:** <https://youtu.be/q8NqQ1V6Oko/>

Text

Description automatically generatedConditional command attributes can be created to preform checks before executing a command. To create a conditional command attribute just create a new class that extends from **ConditionalCommandAttribute**. Checks should be preformed by overriding the **ValidateCommand** method. If this method returns true the command will be executed, otherwise it will not be executed.

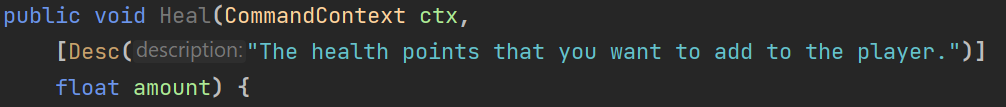
A screenshot of a computer

Description automatically generatedYou can create a description for the attribute by overriding the **AttributeHelpDescription property.**

# Parameter Attributes

## Description Attribute

**Description Attribute Video:** <https://youtu.be/r_oizz2UJbE/>

This attribute can be used to add descriptions to parameters. These descriptions will be displayed by both the help command and the suggestion manager. You can use either **[Desc(**“description”**)]** or **[Description(**“description”**)]**.

## Valid Values Attribute

**Valid Values Attribute Video:** <https://youtu.be/hO-eLoqVo68/>

Text

Description automatically generatedThis attribute is used to confine a parameter to the given values. Only the provided values will be accepted by the parameter.

## Command Name Attribute

**Command Name Attribute:** <https://youtu.be/BjIYrHvkK7A/>

This attribute specifies that a string parameter should only contain command names. There are two optional parameters that can be used to allow aliases or groups. By default, both aliases and groups will be enabled.

**Parameter Attributes**

## Group Name Attribute

**Group Name Attribute Video:** <https://youtu.be/8fS9c5_nSjo/>

This attribute specifies that a string parameter should only contain group names.

## Min Value Attribute

**Min Value Attribute Video:**  <https://youtu.be/tNwSUpPSU-A/>

Graphical user interface, text, application

Description automatically generatedThis attribute indicates that the value of the parameter should be greater than or equal to the provided value. This will work with all numeric types as well as with strings. When using with a string parameter it will compare to the string length.

## Max Value Attribute

**Max Value Attribute Video:**  <https://youtu.be/ffY5SmijWgI/>

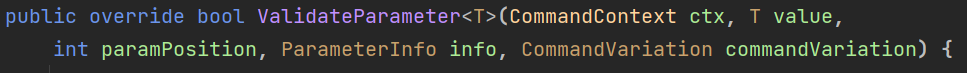
A screenshot of a computer

Description automatically generatedThis attribute indicates that the value of the parameter should be less than or equal to the provided value. This will work with all numeric types as well as with strings. When using with a string parameter it will compare the string length.

**Parameter Attributes**

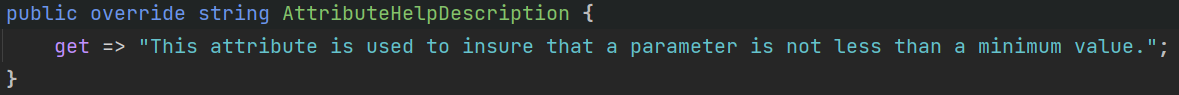
## Conditional Parameter Attribute

**Conditional Parameter Attribute:**  <https://youtu.be/NZgOdqOKV6w/>

Conditional parameter attributes can be created to preform checks before executing a command. To create a conditional parameter attribute just create a new class that extends from **ConditionalParameterAttribute**. Checks should be performed by overriding the **ValidateParameter** method. If this method returns true the command will be executed, otherwise it will not be executed.

Text

Description automatically generatedYou will also need to list the types that are valid for the attribute. You do this by returning an array of types from the **ValidTypes** property.

You can create a description for the parameter by overriding the **AttributeHelpDescription**.

## IValidValuesForTypeProvider

**IValidValuesForTypeProvider Video:** <https://youtu.be/exF9ZfcBx3U/>

This interface can be added to a **ConditionalParameterAttribute** to indicate that it should provide values for the suggestion manager. The interface will add a **GetValidValueNames** that should return all valid values for the given type.

# Parameter Converters

## Parameter Converters

In order to turn the text parameters that are entered into the console into the correct values of the correct type parameter converters are used. You need to have a parameter converter for each type of parameter that you plan on using for commands in the Amilious Console. Bellow is a list of the built-in converters for Amilious console:

|  |  |  |
| --- | --- | --- |
| * Bool | * Color | * DateTime |
| * DateTimeOffset | * TimeSpan | * Enums |
| * Float128(decimal) | * Float64(double) | * Float32(float) |
| * Int64(long) | * Uint64(ulong) | * Int32(int) |
| * Uint32(uint) | * Int16(short) | * Uint16(ushort) |
| * Int8(sbyte) | * Uint8(byte) | * Uri |
| * Vector2 | * Vector2Int | * Vector3 |
| * Vector3Int | * Quaternion | * String |
| * Nullable<T> |  |  |

## Creating A Custom Converter

**Custom Converter Video:**  <https://youtu.be/qfbxGJyxXMQ/>

To create a custom **ParameterConverter** you need to extend from the **AbstractParameterConverter<T>** class. **T** should be the type of parameter the converter is for. You will need to create a constructor that contains a reference to the *AmiliousConsole*. Pass the console along with a friendly name for the type to the base constructor. Use override the Convert method to parse the value. Return true if the value is valid and pass the converted value to the out parameter.







# Settings

## Console Settings

Graphical user interface, text

Description automatically generatedThe console settings are a scriptable object that you can easily save and use for different projects. All the fields have descriptive tool tips and easy to understand names. The settings are applied by adding the settings scriptable object to the Amilious Console in the inspector. For convenience there are fields both of a style sheet and a sprite asset. When applied here they will be applied to the TextMeshPro objects in the Console Drawer prefab.

## Amilious Console Style Sheet

A picture containing text, electronics

Description automatically generatedAmilious Console uses styles for auto generated text. You can modify the look by modifying the style sheet that is referenced by the command settings scriptable object. If you want to know what styles are used, just look at the default Amilious Console style sheet.