dotCommand Documentation

Release 2018.1.x

Well Fired Development

Aug 23, 2018

General

Welcome to the official documentation for .Command a powerful Command Line Processor for Unity Applications.

If you are reading the .pdf version of this documentation, you may find the online version easier to navigate. It is accessible on https://dotcommand-documentation.readthedocs.io/en/latest/.

We recommend you read the *introduction page* to get an overview of what this documentation has to offer.

The table of contents below and in the sidebar should let you easily access the documentation for your topic of interest. You can also use the search function in the top left corner.

Note: Notice something wrong with our documentation? Feel free to submit a pull request.

If you have a technical question, please feel free to contact us through our keybase team wellfiredltd.technicalsupport

---

The main documentation for the site is organized into the following sections:

CHAPTER 1

About

1.1 Introduction

This page aims at giving a broad presentation of the tool and of the contents of this documentation, so that you know where to start if you are a beginner or where to look if you need info on a specific feature.

1.1.1 About .Command

.Command has a small set of core features that make it a must have for any development team.

- Fully view your applications log, as you would in Unity in any built out player.
- Strong filtering and searching functionality from with the global log, which works in Editor and in Player.
- Expose any C# Method or Property to a simple UI, exposing them to touch or click.
- Email any callstack or full log to any email address at the touch of a button.
- Auto open when an exception is throw to instantly alert you of an error.
- Multiple themes to choose from.

1.1.2 Why .Command is better for you

Developers

Instanly expose any of your c# Methods or Properties (public, private, internal, static or instance) to the *All Commands Window* window. Allowing you to access them in game on the command line instantly. If you're a power user, the command interface works similar to a regular terminal. All the usual hotkeys work here too. All built in types are parsed and displayed as suggestions, with support for complex types.

Quality Assurance

As soon as an error happens, you'll know about it. You'll be able to quickly send the offending exception or callstack to your nearest Developer for fixing. With automatic command exposing you'll be able to instantly Spawn new characters, change levels, generally make the game easier for you to test.

Administrators

Your team will be more productive from day 1. Whatever type of product you're making, .Command will enable you to build it faster. .Command builds upon and enhances your existing application, making it easier to test, develop and ship.

1.1.3 About the documentation

This documentation is continuously written, corrected, edited and revamped by members of the .Command team and community. It is edited via text files in the reStructuredText markup language and then compiled into a static website/offline document using the open source Sphinx and ReadTheDocs tools.

Note: You can contribute to .Command's documentation by opening issues through YouTrack or sending patches via pull requests on its GitHub source repository.

1.1.4 Organisation of the documentation

This documentation is organised in five sections, the way it is split up should be relatively intuitive:

- The *General* section contains this introduction as well as general information on the tool It also contains the *Frequently asked questions*.
- The *Getting Started* section is the the main entry point of this documentation, as it contains all the necessary information on using the tool.
- Finally, the *Class API reference* is the documentation of the .Command API. It is generated automatically from a files in the main repository, and the generated files of the documentation are therefore not meant to be modified.

1.2 Frequently asked questions

1.2.1 Can you add some more commands to .Command

We tried to provide a nice set of out of the box functionality, but we're definitely open to adding more! If there's some missing commands you would like to see in .Command, feel free to create an issue on our YouTrack page, or join and contact us through our keybase team wellfiredltd.technicalsupport.

1.2.2 Why do I need to register my objects and types

We could have made this automatic, and that would have been fine for some development teams, but for anyone with a large number of classes in their unity project, this would have caused major slowdown. Another benefit of using manual registering and un-registering is that you can register all kinds of objects and you're not limited to MonoBehaviours or UnityObjects. This provides the user with far greater flexibility.

CHAPTER 2

Installing

2.1 Package Contents

Each .unitypackage downloaded from the AssetStore or from the WellFired website will have the same contents.

- **/WellFired/WellFired.Command/Code** Here you'll find all code related to the .Command project
- **/WellFired/WellFired.Command/Platform/** Here you can find some useful utilities, such as quick start prefabs, link.xml and test scenes.

2.2 Installing

1. Import the .unitypackage into your unity project.

2.3 To be continued. . . ..

This section walked you through installing .Command, in the next section you'll get to use .Command in your Unity Project.

CHAPTER 3

Quick Start

.Command is built to be incredibly easy to use, in here, you're going to find two sections, one will get you running with no work, and the other is a bit more manual, but provides you more control.

Its recommended that you follow the *Slightly less Quick Start* for more control over .Command, but for those of you who are unfamiliar with writing code (or simply don't want to), you can get started with the *Quickest Start* section.

3.1 Quickest Start

Simply drag the prefab DebugConsoleLoader from the /WellFired/WellFired.Command/Platform/Prefabs directory into your scene and press play. You can now toggle .Command with the tilde (~) key, or by pressing the button at the top left of the screen. Once open .Command should look something like this.

3.2 Slightly less Quick Start

The recommended approach when loading .Command is as follows.

For this quick start, we assume you already have an entry point, or a single place in time where you'd like to load and instantiate .Command.

1. Decide upon your entry point for .Command. This will be where you load .Command.
2. Add the required using

```
using WellFired.Command.Unity.Runtime.CommandHandlers;
using WellFired.Command.Unity.Runtime.Console;
```

3. Load .Command

```
// Loads .Command with the default settings.
DevelopmentConsole.Load();
```

4. Tell .Command to register built in commands.

```
// Register built in .Command Inspect commands
DevelopmentCommands.Register(typeof(Inspect));
```

5. Optionally Customise .Command behaviour. (.Command has sensible defaults).

```
// Dont auto open when an exception is thrown.
DevelopmentConsole.Instance.DisableAutoOpen();

// Show the 'show .Command' for quick access if users don't have a tilde␣
↪key (for instance mobile platforms)
DevelopmentConsole.Instance.DrawShowDotCommandButton = true;

// Set a special message on the 'show .Command' button
DevelopmentConsole.Instance.ShowDotCommandButtonMessage = "Show .Command␣
↪(~)";

// Tell the 'open console' button to display in the top left corner
DevelopmentConsole.Instance.DisplayCorner = DisplayCorner.TopLeft;

// Get a notification when the visible state of the console changes. I.E.␣
↪It becomes visible
DevelopmentConsole.Instance.VisibleStateChange += (visible) => {
    if(visible)
        // Disable In-Game Controls
    else
        // Enable In-Game Controls
};
```

Now press play in Unity and you can toggle .Command with the tilde (~) key.

3.3 And Then?

In the coming pages, you'll learn more about .Command's User Interface, how to filter the log as well as how to add custom commands.

CHAPTER 4

UI Overview

4.1 Introduction

The UI for .Command is not made of many components, so it shouldn't take long to familiarise yourself. If you're already familiar with the UI or don't want to read this section, it's fine to skip and move on to *Logs and filters*.

4.2 Main Window

.Command is split into a couple of main areas, these are highlighted in the image below. Each of which is covered under the sub headings below.

4.2.1 Action Area

Here you can find a selection of commonly used actions.

- **Close** will close .Command.
- **Maximise** will ensure .Command uses all available space.
- **Disable** will ensure .Command does not auto open (if this setting is toggled on when .Command loads)
- **Filters Active** This will bring up the *Filter Window*, allowing you to filter content displayed in the *Log History* area.
- **All Commands** This will bring up a quick action window, allowing users to interact with .Command without typing.

4.2.2 Log History

Over time the Log History will fill up with your applications logs. these items will be color coded to show you which type of message you are receiving, this works similar to Unity's Console Window.

Each item in this list can be clicked (or touched) to bring up a separate window showing a full callstack.

Tip: Note that the list can be scrolled up and down by directly sliding on it. This allows you to scroll the list without grabbing the scrollbar which comes as a great convenience on small screen devices.

4.2.3 Input Field

You can type commands by hand into the input field. .Command also provides you with some autocomplete functionalities, whilst you have focus on the input field you can press the tab or shift tab keys to cycle forward and backwards through the suggestions shown in the *Input Field*.

4.2.4 Suggestion Area

As you type in the *Input Field*, suggestions will appear here. These suggestions are buttons and you can click (or touch) them. .Command will provide you with suggestions for the following things.

- Recently used commands when the *Input Field* is empty
- Commands whilst using the *Input Field*
- Command Parameters after having input a command. Most built in c# types will be automatically parsed and presented to the user in the suggestion box (including enums).

4.3 Filter Window

The filter window allows you to quickly toggle which filters the *Log History* will display.

Filters can be toggled on and off, you can also see *Logs and filters* for an overview on how you can add your own custom filters to .Command.

Filters can be toggled on individually and also simultaneously, in the example above, if you toggled on Info and Warning only, your *Log History* would only show messages with the Info or Warning Filter.

4.4 All Commands Window

The all commands window allows you to quickly see and execute all commands exposed to .Command

From this window, you can simply click any command you'd like to execute and .Command will do the rest for you.

4.5 Log Entry Window

The log entry window allows you to quickly see a full callstack for any given item in the *Log History*. Simply click or tap the item in the *Log History* to open this window.

From this window, you can copy the full callstack, or press the email button to open your email client and send the callstack to any given email address.

4.6 To be continued. . . ..

In the next section you'll find out how to add your own custom filters to .Command and how you can filter use those in your game.

CHAPTER 5

Logs and filters

5.1 Introduction

Unity has built in support for four types of filters, and allows you to filer the in editor log with these filters.

- **Info** Messages about your game, this corresponds to the Debug.Log().
- **Warnings** Warnings about your game, this corresponds to the Debug.LogWarning().
- **Errors** Warnings or exceptions in your game, this corresponds to the Debug.LogError().
- **Exceptions** These are tracked when exceptions are thrown.

These filters are great and allow you to split up what's visible in your in-editor log, however when a product grows, searching through thousands of entries becomes burdensome and frankly a waste of time. On top of this, you cannot see the in-editor log window when you have a built out player running on an iPad or Windows computer for example.

.Command solves all these problems by giving you a powerful suite for filtering and displaying logs in your player.

5.2 Displaying Logs in built out players

This functionality works out of the box and requires no effort from you whatsoever. The in game .Command window will mimic the Unity Editor Log exactly. It also out of the box provides the same filtering functionality, providing info, warning and error filtering.

5.3 Adding custom filters to .Command

.Command ships with three built in filters, Info, Warning and Error. These can all be toggled on in the *Filter Window*.

As an extension to this you can add as many Filters as you like, and they will automatically populate the *Filter Window* window. In order to do this, you must follow the following steps.

Note: If you followed the *Quickest Start* when you installed .Command, you might want to have a read over the *Slightly less Quick Start* for more information on loading .Command.

1. Create an enum that will hold your custom Filter definition, for example...

```
private enum AdditionalFilters
{
    dotCommand,

    Networking,
    Sound,
    Graphics
}
```

2. When you load .Command, pass your custom enum to the Load method.

```
DevelopmentConsole.Load(typeof(AdditionalFilters));
```

3. Your new filters; dotCommand, Networking, Sound and Graphics should now be available in the *Filter Window*.

5.4 Logging with your custom filter

Now you've told .Command about your custom filters, .Command knows how to perform custom filter, the only thing left to do is start Logging with your custom filter. In order to do that, you'll want to perform the following steps.

1. Add the correct Namespace for your Debug.Log

```
using Debug = WellFired.Command.Log.Debug;
```

2. Perform your logging

```
Debug.Log(AdditionalFilters.dotCommand, "Hello Log!");
```

.Command provides overloads for everyone of Unity's Log methods, as an extension that takes an enum as the first parameter. It's also worth mentioning that Unity's build in logging functionality still works, so you can slowly migrate your logs over to a filter safe approach.

5.5 Next?

This section walked you through custom filters with .Command, and next we're going to over .Command's ready to use out of the box commands!

CHAPTER 6

Built in Commands

.Command comes with a selection of useful built in commands, these will change over time, and new commands will be added. It's also incredibly easy to add your own commands, as you'll see in the next section *Custom Commands*.

At any point during execution of your game, you can view the *All Commands Window* for a full list of all commands. Commands may also contain small description to help you understand their purpose.

6.1 AutoScroll

A command that allows you to jump to last log entry and to activate autoscrolling. Autoscrolling will get disabled as soon as you scroll the list of log entries.

6.2 Clear

A simple command that allows you to clear the *Log History*.

6.3 ConsoleScale

If you'd like to change the scale of .Command, you can call this. Call it if the UI is too small with a larger number or if it's too big with a smaller number.

- ConsoleScale 1.5

  This will scale .Command up by 1.5 times, making .Command appear bigger.

- ConsoleScale 0.5

  This will half the scale, making .Command appear smaller

6.4 DeviceId

A simple command that will print out the current Device Id, this can be useful if you're running on Mobile Devices.

6.5 InspectAllGameObjects

Run this command to print a list of all game objects in your currently active scene to the *Log History*. You might use this command if you wanted to know which objects are in your scene currently, it would return you a list similar to Unity's built in Hierarchy window.

After running this command you will see a new entry in the *Log History*, stating 'Click to see a list of all game objects in the scene'. simply click this log entry to see all game objects in your current scene.

6.6 InspectGameObject

Run this command to inspect all components on a game object. You might want to do this to check a GameObject has the correct components.

This command takes parameters, the .Command interface will inform you of these required parameters, here is an example usage:

- InspectGameObject MyGameObject

  This will inspect all components on the game object : MyGameObject

After running this command you will see a new entry in the *Log History*, beginning 'Click to see the result of your inspection of gameObject : '. simply click this log entry to see all the state of the requested game object.

6.7 InspectGameObjectComponent

Run this command to inspect a component on a game object. You could do this if you wanted to check a specific value on a component. Similar to being able to use the Inspector in Unity Editor but in your built out players. In the example below, for example, you could check all of the settings on the Camera component are as expected.

This command takes parameters, the .Command interface will inform you of these required parameters, here is an example usage:

- InspectGameObjectComponent MyGameObject Camera

  This will inspect the Camera component on the game object : MyGameObject

After running this command you will see a new entry in the *Log History*, beginning 'Click to see the result of your inspection of gameObject : '. simply click this log entry to see all the state of the requested component on the requested game object.

6.8 InspectGameObjectProperty

Run this command to inspect a property on a component on a game object. You could use this if you want to check a specific value that is usually exposed to the unity inspector for example, checking the scale on a GameObject.

This command takes parameters, the .Command interface will inform you of these required parameters, here is an example usage:

- InspectGameObjectProperty MyGameObject Camera fov

  This will inspect the fov property on the Camera component on the game object : MyGameObject

After running this command you will see a new entry in the *Log History*, beginning 'Click to see the result of your inspection of gameObject : '. simply click this log entry to see all the state of the requested property on the component of the game object.

6.9 And Then?

Now you've seen an overview of all the commands available to you out of the box with .Command, it's time to learn about how to add your own completely custom commands to .Command.

CHAPTER 7

Custom Commands

7.1 Introduction

.Command comes with a selection of *Built in Commands*, though it's completely possible to expose any method in your codebase to .Command. In most cases it requires setting an attribute.

- **Methods** Methods can be fully exposed both static and non static
- **Properties** Both the get and set Property can be exposed to .Command

7.2 Custom Command Examples

The following 'real world' use cases should help you with exposing your existing codebase as Commands with .Command.

Each time you want to expose a method to .Command you will go through the following steps.

1. Register your object with .Command so .Command knows where to look for your commands.
2. Mark up your code with a custom attribute.
3. You are now fully exposed to .Command

7.2.1 Creating a static cheats class

One common use cases when developing games is to create a central class which acts as a cheat container. We're going to assume the following class already exists.

```
public static class HeroCheats
{
    private static int _heroHealth = 100;
```

(continues on next page)

(continued from previous page)

```
    public static int HeroHealth
    {
        get { return _heroHealth; }
        set { _heroHealth = value; }
    }

    public static void DamageHero(int damage)
    {
        HeroHealth -= damage;
    }
}
```

We want to follow the steps introduced in the *Custom Command Examples* section :

1. Register the object with .Command. (You can do this anywhere in your code base).

```
DevelopmentCommands.Register(typeof(HeroCheats));

.. note:: After registering a type or object, when you want to remove those commands␣
↪from .Command you can also call
            DevelopmentCommands.Unregister to unregister that object
```

2. Mark up your code. (See the highlighted lines for additions)

```
 public static class HeroCheats
 {
     private static int _heroHealth = 100;

     [ConsoleCommand(Description = "Gets or Sets the heroes health")]
     public static int HeroHealth
     {
         get { return _heroHealth; }
         set { _heroHealth = value; }
     }

     [ConsoleCommand(Description = "Damages our hero")]
     public static void DamageHero(int damage)
     {
         HeroHealth -= damage;
     }
 }
```

3. These methods and properties are now exposed to .Command. If you press play in Unity and open the *All Commands Window*, you should see the following additions:

- **HeroHealth** The Property exposed to .Command has been automatically added to the *All Commands Window*, here we can see that the description has also been extracted as well as a little description of the type of data that this command can consume, along with it's current value.
- **DamageHero** The Method exposed to .Command has been automatically added to the *All Commands Window*, here we can see that the description has also been extracted as well as a little description of the type of data that this command can be passed to this method.

Have a play around with these exposed methods to get a feel for how they work, and what they do, then feel free to go and add the ConsoleCommand attribute to your own codebase!

Note: Ensure you only call *Register* with a type once or you may end up with duplicate commands in your *All*

Commands Window

7.2.2 ConsoleCommands on instance classes

It's worth noting that you can add the ConsoleCommand attribute to any property or method on any class, static or not. If you're going to add an instance class, you must change slightly the first step, instead of

```
DevelopmentCommands.Register(typeof(HeroCheats));
```

you would add the following

```
// This would be instantiated somewhere else
var someInstantiatedObject = new SomeInstantiatedObject();

// Add your instantiated object instead of the type
DevelopmentCommands.Register(someInstantiatedObject);
```

Note: In this case, instantiated object could be anything, from a plain old class, to a MonoBehaviour or EditorScript, it really doesn't matter, as long as you register the object or type (for static classes), your ConsoleCommand Attributes will be automatically added to .Command

Warning: After registering an instantiated object, be sure to unregister it with *Unregister* when the object is destroyed. If you don't do this, nothing bad will happen, however it's good practise to match your register calls with an unregister call.

7.3 Next Up

Skinning .Command and tweaking global settings.

CHAPTER 8

Theme and Settings

8.1 Introduction

.Command ships with a couple of options that can be modified to suit your needs. These options reside in the Unity Preference window. To access them, go to Unity > Preferences (on macOS) or Edit > Preferences... (Windows).

From here, select the .Command menu in the left hand pane.

8.2 Settings

There are currently only two global .Command settings.

1. **Allow sending of usage data** .Command will by default send anonymous analytics data to WellFired Development to help improve .Command functionality, if you don't want this feature, be sure to disable this option.
2. **Theme** .Command ships with a selection of themes that are ready to use out of the box, you can select the theme here. By default, .Command uses the dark theme.

8.3 Going deeper?

You've now completed all the basic's! In the next section you'll learn how to provide suggestions for a ConsoleCommand.

CHAPTER 9

Programmatic Control

Every method and property listed here can be found in greater detail in our *Class API reference*, specifically be sure to check out the *DevelopmentConsole* api page.

9.1 Hiding the Show .Command Button

The show .Command button can be set to display or hidden with the *DrawShowDotCommandButton* property. Pass true or false to this depending on if you want to show or hide the button.

```
DevelopmentConsole.Instance.DrawShowDotCommandButton = true;
```

9.2 Change the text of the show .Command button

ShowDotCommandButtonMessage will allow you to override the default message on the 'show console' button.

```
DevelopmentConsole.Instance.ShowConsoleButtonMessage = "My custom message";
```

9.3 Change the location of the show .Command button

By default the 'show console' button is displayed in the top left corner, you can override this behaviour with the *DisplayCorner* property.

```
DevelopmentConsole.Instance.DisplayCorner = DisplayCorner.TopRight;
```

9.4 Receiving callbacks when .Command appears or disappear

It's possible to receive a callback when .Command opens, either manually or automatically, with the *VisibleStateChange* property.

```
DevelopmentConsole.Instance.VisibleStateChange += (visible) => {
    if(visible)
        // Disable In-Game Controls
    else
        // Enable In-Game Controls
};
```

9.5 Adding additional Custom Filters

At any point during execution you can add additional filters to .Command, this can be useful if you support dynamic module loading, or if your team wants to distribute .Command as part of a central package and doesn't know ahead of time what Filters they will have

```
private enum DynamicModuleFilter
        {
                ModuleFIlter
        }

DevelopmentConsole.Instance.AddCustomFilters(typeof(DynamicModuleFilter));
```

9.6 Next

Now we'll find out how to improve your already created custom commands.

CHAPTER 10

Suggestions

When providing *Custom Commands* to .Command, you can also present suggestions to the user on what they can input.

This could be useful when a function can be called with any data, but you want to provide hints to content creators who might not know all the possibilities, it provides a more user friendly interface to .Command.

10.1 Improving an existing Command

Suppose our 'real world' example has the following class with the following method.

```
public static class Cheats
{
    [ConsoleCommand(Description = "Damages all enemies of the given type")]
    public static void DamageAllEnemiesOfType(string type)
    {
        if (type == "enemy_type_sausage")
        {
            // Damage all sausages
        }
        if (type == "enemy_type_heroes")
        {
            // Damage all heroes
        }
        if (type == "enemy_type_projectiles")
        {
            // Damage all projectiles
        }
    }
}
```

The method DamageAllEnemiesOfType takes a type string, but inside the body, we're expecting certain values. You can provide hints to .Command, so that it can present a friendly user interface.

Using the example above, we know that the data received should be one of the following values "enemy_type_sausage", "enemy_type_heroes" or "enemy_type_projectiles". So, let's present that to the user of .Command.

We firstly need to create a class which implements ISuggestion. Inside this class we want to return a list of suggested data, for example

```
public class EnemyTypeSuggestion : ISuggestion
{
    public IEnumerable<string> Suggestion(IEnumerable<string> previousArguments)
    {
            var suggestions = previousArguments.ToList();
            suggestions.AddRange( new[] { "enemy_type_sausage", "enemy_type_heroes",
↪"enemy_type_projectiles" } );
            return suggestions;
    }
}
```

The final step is to tell the ConsoleCommand to use our suggestion, you can do this by adding a parameter Attribute, as shown in the highlighted line.

```
 public static class Cheats
 {
     [ConsoleCommand(Description = "Damages all enemies of the given type")]
     public static void␣
↪DamageAllEnemiesOfType([Suggestion(typeof(EnemyTypeSuggestion))]string type)
     {
         if (type == "enemy_type_sausage")
         {
             // Damage all sausages
         }
         if (type == "enemy_type_heroes")
         {
             // Damage all heroes
         }
         if (type == "enemy_type_projectiles")
         {
             // Damage all projectiles
         }
     }
 }
```

If you press play in Unity and open .Command you should see that it now starts to show you suggestions for your parameter.

Note: It's possible to add more than one suggestion to a method, in fact you can add one per parameter.

CHAPTER 11

dotCommand API

11.1 Classes

11.1.1 ExtensionMethods

Namespace: *WellFired.Command*

Description

Public Static Methods

void	CopyProperties (this TFrom @ from, TTo to)

Breakdown

- void **CopyProperties< TFrom, TTo >** (this TFrom @ from, TTo to)

11.1.2 Debug

Namespace: *WellFired.Command*

Description

Public Static Methods

void	*LogFormat* (Enum filter, string message, params object[] parameters)

Continued on next page

Table 1 – continued from previous page

void	*LogError* (Enum filter, object message)
void	*LogError* (Enum filter, object message, Object ping)
void	*LogError* (object message, Object ping)
void	*LogErrorFormat* (Enum filter, string message, params object[] parameters)
void	*LogErrorFormat* (string message, params object[] parameters)
void	*LogWarning* (Enum filter, object message)
void	*LogWarning* (object message)
void	*LogWarningFormat* (Enum filter, string message, params object[] parameters)
void	*LogWarningFormat* (string message, params object[] parameters)
void	*LogWarningFormat* (Enum filter, string message, Object ping, params object[] parameters)
void	*LogWarningFormat* (string message, Object ping, params object[] parameters)
void	*LogWarningFormat* (Enum filter, object message, Object ping)
void	*LogWarning* (object message, Object ping)
void	*Log* (Enum filter, object message)
void	*Log* (object message)
void	*Log* (Enum filter, string message, Color color)
void	*Log* (string message, Color color)
void	*Log* (Enum filter, string message, Object ping)
void	*Log* (string message, Object ping)
void	*Log* (Enum filter, string message, Object ping, Color color)
void	*Log* (string message, Object ping, Color color)
void	*LogError* (object message)
void	*LogFormat* (string message, params object[] parameters)
void	*LogFormat* (Enum filter, string message, Color color, params object[] parameters)
void	*LogFormat* (string message, Color color, params object[] parameters)
void	*LogFormat* (Enum filter, string message, Object ping, params object[] parameters)
void	*LogFormat* (string message, Object ping, params object[] parameters)
void	*LogFormat* (Enum filter, string message, Object ping, Color color, params object[] parameters)
void	*LogFormat* (string message, Object ping, Color color, params object[] parameters)
void	*LogException* (Exception exception)
void	*ClearDeveloperConsole* ( )
void	*DebugBreak* ( )
void	*DrawLine* (Vector3 start, Vector3 end)
void	*DrawLine* (Vector3 start, Vector3 end, UnityEngine.Color color)
void	*DrawLine* (Vector3 start, Vector3 end, UnityEngine.Color color, float duration)
void	*DrawLine* (Vector3 start, Vector3 end, UnityEngine.Color color, float duration, bool depthTest)
void	*DrawRay* (Vector3 start, Vector3 dir)
void	*DrawRay* (Vector3 start, Vector3 dir, UnityEngine.Color color)
void	*DrawRay* (Vector3 start, Vector3 dir, UnityEngine.Color color, float duration)
void	*DrawRay* (Vector3 start, Vector3 dir, UnityEngine.Color color, float duration, bool depthTest)
void	*Break* ( )
void	*Assert* (bool condition, string message = null, string message1 = null)
void	*WriteLine* (string line)
void	*LogException* (Exception exception, Object context)

Breakdown

- void **LogError** (object message)
- void **LogFormat** (Enum filter, string message, params object[] parameters)
- void **LogError** (Enum filter, object message, Object ping)

- void **LogError (** object message, Object ping **)**
- void **LogErrorFormat (** Enum filter, string message, params object[] parameters **)**
- void **LogErrorFormat (** string message, params object[] parameters **)**
- void **LogWarning (** Enum filter, object message **)**
- void **LogWarning (** object message **)**
- void **LogWarningFormat (** Enum filter, string message, params object[] parameters **)**
- void **LogWarningFormat (** string message, params object[] parameters **)**
- void **LogWarningFormat (** Enum filter, string message, Object ping, params object[] parameters **)**
- void **LogWarningFormat (** string message, Object ping, params object[] parameters **)**
- void **LogWarningFormat (** Enum filter, object message, Object ping **)**
- void **LogWarning (** object message, Object ping **)**
- void **Log (** Enum filter, object message **)**
- void **Log (** object message **)**
- void **Log (** Enum filter, string message, Color color **)**
- void **Log (** string message, Color color **)**
- void **Log (** Enum filter, string message, Object ping **)**
- void **Log (** string message, Object ping **)**
- void **Log (** Enum filter, string message, Object ping, Color color **)**
- void **Log (** string message, Object ping, Color color **)**
- void **LogError (** Enum filter, object message **)**
- void **LogFormat (** string message, params object[] parameters **)**
- void **LogFormat (** Enum filter, string message, Color color, params object[] parameters **)**
- void **LogFormat (** string message, Color color, params object[] parameters **)**
- void **LogFormat (** Enum filter, string message, Object ping, params object[] parameters **)**
- void **LogFormat (** string message, Object ping, params object[] parameters **)**
- void **LogFormat (** Enum filter, string message, Object ping, Color color, params object[] parameters **)**
- void **LogFormat (** string message, Object ping, Color color, params object[] parameters **)**
- void **LogException (** Exception exception **)**
- void **ClearDeveloperConsole ( )**
- void **DebugBreak ( )**
- void **DrawLine (** Vector3 start, Vector3 end **)**
- void **DrawLine (** Vector3 start, Vector3 end, UnityEngine.Color color **)**
- void **DrawLine (** Vector3 start, Vector3 end, UnityEngine.Color color, float duration **)**
- void **DrawLine (** Vector3 start, Vector3 end, UnityEngine.Color color, float duration, bool depthTest **)**
- void **DrawRay (** Vector3 start, Vector3 dir **)**
- void **DrawRay (** Vector3 start, Vector3 dir, UnityEngine.Color color **)**

- void **DrawRay** (Vector3 start, Vector3 dir, UnityEngine.Color color, float duration)
- void **DrawRay** (Vector3 start, Vector3 dir, UnityEngine.Color color, float duration, bool depthTest)
- void **Break** ( )
- void **Assert** (bool condition, string message = null, string message1 = null)
- void **WriteLine** (string line)
- void **LogException** (Exception exception, Object context)

11.1.3 Settings

Namespace: WellFired

Description

Properties

Theme	*Theme* { get; set; }
SkinData	*SkinData* { get; set; }

Public Methods

void	*LoadSkinData* ( )

Breakdown

- Theme **Theme** { get; set; }
- *SkinData* **SkinData** { get; set; }
- void **LoadSkinData** ( )

11.1.4 DarkOlive

Namespace: WellFired.Command.Skins

Inherits: *WellFired.Command.Skins.Customisation.DarkSkin*

Description

Properties

override Color	*MainColor* { get; set; }
override Color	*MainFontColor* { get; set; }
override Color	*SecondaryColor* { get; set; }
override Color	*ButtonColor* { get; set; }
override Color	*TextEntryFontColor* { get; set; }

Breakdown

- override Color **MainColor** { get; set; }
- override Color **MainFontColor** { get; set; }
- override Color **SecondaryColor** { get; set; }
- override Color **ButtonColor** { get; set; }
- override Color **TextEntryFontColor** { get; set; }

11.1.5 DarkSkin

Namespace: WellFired.Command.Skins

Implements: *WellFired.Command.Skins.ISkinData*

Description

Properties

Color	*DetailedLogMessageBackgroundColor* { get; set; }
Color	*ButtonColor* { get; set; }
Color	*MainColor* { get; set; }
Color	*SecondaryColor* { get; set; }
Color	*MainFontColor* { get; set; }
Color	*TextEntryColor* { get; set; }
Color	*TextEntryFontColor* { get; set; }
Color	*EntryExceptionColor* { get; set; }
Color	*EntryErrorColor* { get; set; }
Color	*EntryWarningColor* { get; set; }
Color	*EntryInfoColor* { get; set; }
Color	*ButtonHoverColor* { get; set; }
Color	*DetailedLogMessageColor* { get; set; }
Color	*GeneralLabelFontColor* { get; set; }
int	*FontSize* { get; set; }
int	*ButtonSpacing* { get; set; }
int	*ButtonSpacingTouch* { get; set; }
int	*EntryHeight* { get; set; }
int	*ButtonPaddingKeyboard* { get; set; }
int	*ButtonPaddingTouch* { get; set; }
int	*HeaderPaddingKeyboard* { get; set; }
int	*HeaderPaddingTouch* { get; set; }

Breakdown

- Color **ButtonHoverColor** { get; set; }
- Color **DetailedLogMessageBackgroundColor** { get; set; }
- Color **MainColor** { get; set; }
- Color **SecondaryColor** { get; set; }

- Color **MainFontColor** { get; set; }
- Color **TextEntryColor** { get; set; }
- Color **TextEntryFontColor** { get; set; }
- Color **EntryExceptionColor** { get; set; }
- Color **EntryErrorColor** { get; set; }
- Color **EntryWarningColor** { get; set; }
- Color **EntryInfoColor** { get; set; }
- Color **ButtonColor** { get; set; }
- Color **DetailedLogMessageColor** { get; set; }
- Color **GeneralLabelFontColor** { get; set; }
- int **FontSize** { get; set; }
- int **ButtonSpacing** { get; set; }
- int **ButtonSpacingTouch** { get; set; }
- int **EntryHeight** { get; set; }
- int **ButtonPaddingKeyboard** { get; set; }
- int **ButtonPaddingTouch** { get; set; }
- int **HeaderPaddingKeyboard** { get; set; }
- int **HeaderPaddingTouch** { get; set; }

11.1.6 GreenSkin

Namespace: WellFired.Command.Skins

Inherits: *WellFired.Command.Skins.Customisation.DarkSkin*

Description

Properties

override Color	*MainColor* { get; set; }
override Color	*MainFontColor* { get; set; }
override Color	*SecondaryColor* { get; set; }
override Color	*ButtonColor* { get; set; }
override Color	*TextEntryFontColor* { get; set; }

Breakdown

- override Color **MainColor** { get; set; }
- override Color **MainFontColor** { get; set; }
- override Color **SecondaryColor** { get; set; }
- override Color **ButtonColor** { get; set; }

- override Color **TextEntryFontColor** { get; set; }

11.1.7 LightSkin

Namespace: WellFired.Command.Skins

Inherits: *WellFired.Command.Skins.Customisation.DarkSkin*

Description

Properties

override Color	*MainColor* { get; set; }
override Color	*SecondaryColor* { get; set; }
override Color	*ButtonColor* { get; set; }
override Color	*GeneralLabelFontColor* { get; set; }
override Color	*MainFontColor* { get; set; }
override Color	*TextEntryFontColor* { get; set; }
override Color	*EntryInfoColor* { get; set; }
override Color	*EntryWarningColor* { get; set; }
override Color	*EntryErrorColor* { get; set; }
override Color	*EntryExceptionColor* { get; set; }

Breakdown

- override Color **MainColor** { get; set; }
- override Color **SecondaryColor** { get; set; }
- override Color **ButtonColor** { get; set; }
- override Color **GeneralLabelFontColor** { get; set; }
- override Color **MainFontColor** { get; set; }
- override Color **TextEntryFontColor** { get; set; }
- override Color **EntryInfoColor** { get; set; }
- override Color **EntryWarningColor** { get; set; }
- override Color **EntryErrorColor** { get; set; }
- override Color **EntryExceptionColor** { get; set; }

11.1.8 PurpleSkin

Namespace: WellFired.Command.Skins

Inherits: *WellFired.Command.Skins.Customisation.DarkSkin*

Description

Properties

override Color	*MainColor* { get; set; }
override Color	*MainFontColor* { get; set; }
override Color	*SecondaryColor* { get; set; }
override Color	*ButtonColor* { get; set; }
override Color	*TextEntryFontColor* { get; set; }

Breakdown

- override Color **MainColor** { get; set; }
- override Color **MainFontColor** { get; set; }
- override Color **SecondaryColor** { get; set; }
- override Color **ButtonColor** { get; set; }
- override Color **TextEntryFontColor** { get; set; }

11.1.9 SkyBlueSkin

Namespace: WellFired.Command.Skins

Inherits: *WellFired.Command.Skins.Customisation.DarkSkin*

Description

Properties

override Color	*MainColor* { get; set; }
override Color	*MainFontColor* { get; set; }
override Color	*SecondaryColor* { get; set; }
override Color	*ButtonColor* { get; set; }
override Color	*TextEntryFontColor* { get; set; }

Breakdown

- override Color **MainColor** { get; set; }
- override Color **MainFontColor** { get; set; }
- override Color **SecondaryColor** { get; set; }
- override Color **ButtonColor** { get; set; }
- override Color **TextEntryFontColor** { get; set; }

11.1.10 Skin

Namespace: *WellFired.Command*

Description

Properties

SkinData	*Data* { get; set; }

Public Static Methods

Skin	*From* (*SkinData* skinData)

Breakdown

- *SkinData* **Data** { get; set; }
- *Skin* **From** (*SkinData* skinData)

11.1.11 SkinData

Namespace: *WellFired.Command*

Implements: *WellFired.Command.Skins.ISkinData*

Description

Properties

Color	*DetailedLogMessageBackgroundColor* { get; set; }
Color	*MainColor* { get; set; }
Color	*MainFontColor* { get; set; }
Color	*TextEntryFontColor* { get; set; }
Color	*TextEntryColor* { get; set; }
Color	*ButtonColor* { get; set; }
Color	*ButtonHoverColor* { get; set; }
Color	*EntryExceptionColor* { get; set; }
Color	*EntryErrorColor* { get; set; }
Color	*EntryWarningColor* { get; set; }
Color	*EntryInfoColor* { get; set; }
Color	*SecondaryColor* { get; set; }
Color	*DetailedLogMessageColor* { get; set; }
Color	*GeneralLabelFontColor* { get; set; }
int	*FontSize* { get; set; }
int	*ButtonSpacing* { get; set; }
int	*ButtonSpacingTouch* { get; set; }
int	*EntryHeight* { get; set; }
int	*ButtonPaddingKeyboard* { get; set; }
int	*ButtonPaddingTouch* { get; set; }
int	*HeaderPaddingKeyboard* { get; set; }
int	*HeaderPaddingTouch* { get; set; }

Public Static Methods

SkinData	*From* (*ISkinData* from)

Breakdown

- Color **SecondaryColor** { get; set; }
- Color **DetailedLogMessageBackgroundColor** { get; set; }
- Color **MainFontColor** { get; set; }
- Color **TextEntryFontColor** { get; set; }
- Color **TextEntryColor** { get; set; }
- Color **ButtonColor** { get; set; }
- Color **ButtonHoverColor** { get; set; }
- Color **EntryExceptionColor** { get; set; }
- Color **EntryErrorColor** { get; set; }
- Color **EntryWarningColor** { get; set; }
- Color **EntryInfoColor** { get; set; }

- Color **MainColor** { get; set; }
- Color **DetailedLogMessageColor** { get; set; }
- Color **GeneralLabelFontColor** { get; set; }
- int **FontSize** { get; set; }
- int **ButtonSpacing** { get; set; }
- int **ButtonSpacingTouch** { get; set; }
- int **EntryHeight** { get; set; }
- int **ButtonPaddingKeyboard** { get; set; }
- int **ButtonPaddingTouch** { get; set; }
- int **HeaderPaddingKeyboard** { get; set; }
- int **HeaderPaddingTouch** { get; set; }
- *SkinData* **From** (*ISkinData* from)

11.1.12 Preferences

Namespace: WellFired.Command.Unity

Description

Public Static Methods

void	*PreferencesGUI* ( )

Breakdown

- void **PreferencesGUI** ( )

11.1.13 ConsoleCommandAttribute

Namespace: WellFired.Command.Unity.Runtime

Description

Place this attribute on any method that you want exposed to the .:ref:*Command<namespacewellfired_command>*.

Properties

string	*Name* { get; set; }
string	*Description* { get; set; }

Breakdown

- string **Name** { get; set; }

  Description

  The Name of this attribute. .:ref:*Command<namespacewellfired_command>* will use this as the actual command the user must type of select.

- string **Description** { get; set; }

  Description

  This will give the user a nice overview of the command they are about to use.

11.1.14 DevelopmentCommands

Namespace: WellFired.Command.Unity.Runtime

Description

This is a static class that the user can register use to register command wrappers. If you have any DevelopmentConsole Attributes inside your class, you will want to call one of the following methods:

Events

Action< *CommandWrapper* >	*CommandHandlerAdded*
Action< *CommandWrapper* >	*CommandHandlerRemoved*
Action< *CommandWrapper* >	*CommandExecuted*

public-static-attrib

IEnumerable< *CommandWrapper* >	*Handlers*

Public Static Methods

void	*HandleCommand* (string commandLine)
CommandWrapper	*FindCommandFromPartial* (string partialCommand, int index)
IEnumerable< *CommandWrapper* >	*FindCommandFromPartial* (string partialCommand)
CommandWrapper	*GetCommandWrapper* (string commandName)
void	*Register* (Type type)
void	*Register* (object obj)
void	*Unregister* (object obj)
void	*Unregister* (Type type)

Breakdown

- Action< *CommandWrapper* > **CommandHandlerAdded**
- Action< *CommandWrapper* > **CommandHandlerRemoved**
- Action< *CommandWrapper* > **CommandExecuted**
- IEnumerable< *CommandWrapper* > **Handlers**
- void **HandleCommand** (string commandLine)
- *CommandWrapper* **FindCommandFromPartial** (string partialCommand, int index)
- IEnumerable< *CommandWrapper* > **FindCommandFromPartial** (string partialCommand)
- *CommandWrapper* **GetCommandWrapper** (string commandName)
- void **Register** (Type type)

  Description

  Call this method to register an object by type. Objects that are registered will be parsed for the ConsoleCommand attribute.

  Parameters

type	The type of object that you would like to register with .:ref:*Command<namespacewellfired_command>*

- void **Register** (object obj)

  Description

  Call this method to register an object by instance. Objects that are registered will be parsed for the ConsoleCommand attribute.

  Parameters

obj	The object that you would like to register with .:ref:*Command<namespacewellfired_command>*

- void **Unregister** (object obj)

  Description

  If you have called Register on an object, you should match that call with an unregister

  Parameters

obj	The object that you would like to unregister from .:ref:*Command<namespacewellfired_command>*

- void **Unregister** (Type type)

  Description

  If you have called Register on a type, you should match that call with an unregister

  Parameters

type	The type of object that you would like to unregister from .:ref:*Command<namespacewellfired_command>*

11.1.15 MethodCommandWrapper

Namespace: WellFired.Command.Unity.Runtime

Inherits: *WellFired.Command.Unity.Runtime.Wrapper.CommandWrapper*

Description

The command wrapper that contains cached information about Methods.

Properties

override :ref:‘ParameterWrapper<classwellfired_command_unity_runtime_helpers_parameterwrapper>‘[]	*Parameters* { get; set; }

Public Methods

	MethodCommandWrapper (string commandName, string description, Type type, object referenceObject, MethodInfo methodInfo)
override bool	*Equals* (object otherObject)
override int	*GetHashCode* ( )
override void	*Invoke* (params string[] arguments)

protected-func

bool	*GetArgumentList* (string[] commandArguments, out object[] argumentValues)

protected-static-func

ISuggestion	*GetSuggestionMethod* (ParameterInfo paramInfo)

Breakdown

- override :ref:‘ParameterWrapper<classwellfired_command_unity_runtime_helpers_parameterwrapper>‘[] **Parameters** { get; set; }
- **MethodCommandWrapper** (string commandName, string description, Type type, object referenceObject, MethodInfo methodInfo)

- override bool **Equals** (object otherObject)
- override int **GetHashCode** ( )
- override void **Invoke** (params string[] arguments)
- bool **GetArgumentList** (string[] commandArguments, out object[] argumentValues)
- *ISuggestion* **GetSuggestionMethod** (ParameterInfo paramInfo)

11.1.16 PropertyCommandWrapper

Namespace: WellFired.Command.Unity.Runtime

Inherits: *WellFired.Command.Unity.Runtime.Wrapper.CommandWrapper*

Description

Properties

override :ref:‘ParameterWrapper<classwellfired_command_unity_runtime_helpers_parameterwrapper>‘[]	*Parameters* { get; set; }

Public Methods

	PropertyCommandWrapper (string commandName, string description, Type type, object obj, PropertyInfo propertyInfo)
override bool	*Equals* (object otherObject)
override int	*GetHashCode* ( )
override void	*Invoke* (params string[] arguments)

Breakdown

- override :ref:‘ParameterWrapper<classwellfired_command_unity_runtime_helpers_parameterwrapper>‘[] **Parameters** { get; set; }
- **PropertyCommandWrapper** (string commandName, string description, Type type, object obj, PropertyInfo propertyInfo)
- override bool **Equals** (object otherObject)
- override int **GetHashCode** ( )
- override void **Invoke** (params string[] arguments)

11.1.17 CommandRegistration

Namespace: WellFired.Command.Unity.Runtime

Description

This class is here to provide users with a single easy location to register all command objects with .:ref:*Command<namespacewellfired_command>*.

Public Static Methods

void	*RegisterCommandsOnConsoleStartup* ( )
void	*UnRegisterCommandsOnConsoleExit* ( )

Breakdown

- void **RegisterCommandsOnConsoleStartup** ( )

  Description

  Called automatically when .:ref:*Command<namespacewellfired_command>* launches, you can add your own Registrations here. Don't forget to match your DevelopmentCommands.Register call with a DevelopmentCommands.Unregister call by calling DevelopmentCommands.Unregister in *CommandRegistration.UnRegisterCommandsOnConsoleExit*

- void **UnRegisterCommandsOnConsoleExit** ( )

  Description

  Called automatically when .:ref:*Command<namespacewellfired_command>* is destroyed, you can add your own Unregistrations here.

11.1.18 ComponentProperties

Namespace: WellFired.Command.Unity.Runtime

Description

Properties

List< PropertyInfo >	*Properties* { get; set; }
List< FieldInfo >	*Fields* { get; set; }
string	*ComponentName* { get; set; }
Component	*Component* { get; set; }

Breakdown

- List< PropertyInfo > **Properties** { get; set; }
- List< FieldInfo > **Fields** { get; set; }
- string **ComponentName** { get; set; }
- Component **Component** { get; set; }

11.1.19 DevelopmentConsole

Namespace: WellFired.Command.Unity.Runtime

Description

Console is a MonoBehaviour that opens an in game .:ref:*Command<namespacewellfired_command>*, this console can call any Property / Method that is marked up with the ConsoleCommandAttribute.

Properties

string	*ShowDotCommandButtonMessage* { get; set; }
DisplayCorner	*DisplayCorner* { get; set; }
DevelopmentConsole	*Instance* { get; set; }
bool	*IsMaximized* { get; set; }
bool	*DrawShowDotCommandButton* { get; set; }
bool	*ForceMinimize* { get; set; }
int	*InputBlockerSortingOrder* { get; set; }
bool	*JustMadeVisible* { get; set; }
bool	*IsVisible* { get; set; }
ISkinData	*SkinData* { get; set; }

Public Properties

Action< bool >	*VisibleStateChange*
bool	*ShouldAcceptGameInput*
IEnumerable< string >	*RecentCommands*

public-static-attrib

float	*ScreenWidth*

Public Methods

void	*AddCustomFilters* (Type customFilterProvider)
void	*DisableAutoOpen* ( )
void	*EnableAutoOpen* (bool openOnException = true, bool openOnError = false)
void	*HideAllOpenPopups* ( )
void	*ClearTypedInput* ( )
void	*SetCommandInputTextAsIfUserHadTyped* (string text)
string	*CheckInputForTilde* (string input)
void	*InspectLogEntry* (*LogEntry* logEntry)

Public Static Methods

void	*Load* (Type customFilterType = null)

Breakdown

- string **ShowDotCommandButtonMessage** { get; set; }

  Description

  Set this if you would like to change the message displayed on the 'open .:ref:*Command<namespacewellfired_command>*' button.

- DisplayCorner **DisplayCorner** { get; set; }

  Description

  Set this if you would like to change corner of the screen the 'open .:ref:*Command<namespacewellfired_command>*' button is located.

- *DevelopmentConsole* **Instance** { get; set; }

  Description

  Gets or sets the singleton instance of .:ref:*Command<namespacewellfired_command>*.

- bool **IsMaximized** { get; set; }

  Description

  Is the console maximised

- bool **DrawShowDotCommandButton** { get; set; }

  Description

  Should we draw the 'open .:ref:*Command<namespacewellfired_command>*' button or not.

- bool **ForceMinimize** { get; set; }

  Description

  Has the user clicked on the Force Hide button. If so, they can re-open the console with the ~ key or by setting this value to false.

- int **InputBlockerSortingOrder** { get; set; }

  Description

  Since .:ref:*Command<namespacewellfired_command>* is based on Unity legacy GUI, we use the new UI system to block input behind the console. This property allows to set the sorting order of the canvas used to block input. It is by default in front of everyting.

- bool **JustMadeVisible** { get; set; }
- bool **IsVisible** { get; set; }
- *ISkinData* **SkinData** { get; set; }
- Action< bool > **VisibleStateChange**

  Description

This Action will be triggered when the visible state of .:ref:*Command<namespacewellfired_command>* changes. If it's called with true, it means .:ref:*Command<namespacewellfired_command>* became visible, if it's called with false, it means .:ref:*Command<namespacewellfired_command>* was hidden

- bool **ShouldAcceptGameInput**

  Description

  You can query this in your game, to see if the game should accept Input. This will return false if the console is showing for any reason.

- IEnumerable< string > **RecentCommands**
- float **ScreenWidth**
- void **AddCustomFilters** (Type customFilterProvider)

  Description

  This method allows you to add more filters to the consol at runtime, .:ref:*Command<namespacewellfired_command>* calls this internally when you pass filters to load, but you can additionally call this if you need to append more. (I.E.) you load modules at runtime and can't be sure of which enums will be needed at load time.

  Parameters

customFilter-Provider	This type should be an enum, which holds the types you'd like to use when filtering.

- void **DisableAutoOpen** ( )

  Description

  This method will stop .:ref:*Command<namespacewellfired_command>* from auto opening if an error is fired, you can still open it manually.

- void **EnableAutoOpen** (bool openOnException = true, bool openOnError = false)

  Description

  This method will make sure .:ref:*Command<namespacewellfired_command>* auto opens if an error is fired.

  Parameters

openOnException	Should .:ref:*Command<namespacewellfired_command>* auto open on exception
openOnError	Should .:ref:*Command<namespacewellfired_command>* auto open on error

- void **HideAllOpenPopups** ( )

  Description

  This method will hide all open popups.

- void **ClearTypedInput** ( )
- void **SetCommandInputTextAsIfUserHadTyped** (string text)

  Description

You can call this method if you'd like to set input in .:ref:*Command<namespacewellfired_command>* as though the user had typed it.

Parameters

text	The text to enter into .:ref:*Command<namespacewellfired_command>*

- string **CheckInputForTilde** (string input)

  Description

  Checks the input for the close key and Closes the Development console if it is found.

  Parameters

input	The Input.

- void **InspectLogEntry** (*LogEntry* logEntry)

  Description

  Opens a the history of a specific Item.

  Parameters

logEntry	Log Entry.

- void **Load** (Type customFilterType = null)

  Description

  Call this method to load a single instance of .:ref:*Command<namespacewellfired_command>*. You can then access the instance through the Instance property.

11.1.20 Inspect

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

string	*WildcardToRegex* (string pattern)

Breakdown

- string **WildcardToRegex** (string pattern)

11.1.21 SampleCommands

Namespace: WellFired.Command.Unity.Runtime

Description

A selection of provided sample commands that will help you get started with writing your own.

11.1.22 ScaleSuggestion

Namespace: WellFired.Command.Unity.Runtime

Implements: *WellFired.Command.Unity.Runtime.Suggestion.ISuggestion*

Description

Present sensible scale suggestions to the user, + / - x around currentScale

Public Methods

	ScaleSuggestion ( )
IEnumerable< string >	*Suggestion* (IEnumerable< string > previousArguments)

Breakdown

- **ScaleSuggestion** ( )
- IEnumerable< string > **Suggestion** (IEnumerable< string > previousArguments)

  Description

  This method will be used by the .:ref:*Command<namespacewellfired_command>* to determine the auto complete values that should be used.

  Parameters

previousArguments	The Previous Arguments

11.1.23 UIBlocker

Namespace: WellFired.Command.Unity.Runtime

Description

This class create a UnityUI canvas allowing to place input blockers below our Unity legacy GUI system. This ensure that input events happening in the *Console* are not used by element of the game. This is a temporary solution, in the future, .:ref:*Command<namespacewellfired_command>* will use Unity most recent UI system.

Public Methods

	UIBlocker (Transform parent)
void	*SetSortingOrder* (int sortingOrder)
void	*BlockOpenConsoleArea* (Rect rect)
void	*BlockConsoleArea* (Rect rect)
void	*BlockFilterArea* (Rect rect)
void	*BlockCommandsArea* (Rect rect)
void	*BlockLogEntryArea* (Rect rect)
void	*BlockScreen* ( )
void	*UnblockScreen* ( )

Breakdown

- **UIBlocker** (Transform parent)
- void **SetSortingOrder** (int sortingOrder)

  Description

  The blocking canvas is placed on top of every canvas by default. This function allows to change it.

  Parameters

sortingOrder

- void **BlockOpenConsoleArea** (Rect rect)

  Description

  Will set the size of the input blocker corresponding to the open console button.

  Parameters

rect

- void **BlockConsoleArea** (Rect rect)

  Description

  Will set the size of the input blocker corresponding to the console window.

  Parameters

rect

- void **BlockFilterArea** (Rect rect)

  Description

  Will set the size of the input blocker corresponding to the filter window.

  Parameters

rect

- void **BlockCommandsArea** (Rect rect)

  Description

  Will set the size of the input blocker corresponding to the commands window.

  Parameters

rect

- void **BlockLogEntryArea** (Rect rect)

  Description

  Will set the size of the input blocker corresponding to the log entry window.

  Parameters

rect

- void **BlockScreen** ( )
- void **UnblockScreen** ( )

11.1.24 EmailFactory

Namespace: WellFired.Command.Unity.Runtime

Description

If you want to provide specific email support for your platform, you can add your platform to this Email Factory, returning a new instance of your custom *IEmailSender*.

Public Static Methods

IEmailSender	*GetEmailSender* ( )

Breakdown

- *IEmailSender* **GetEmailSender** ( )

11.1.25 EmailSender

Namespace: WellFired.Command.Unity.Runtime

Implements: *WellFired.Command.Unity.Runtime.Email.IEmailSender*

Description

Public Methods

bool	*CanSendEmail* ( )
void	*Email* (string attachmentPath, string mimeType, string attachmentFilename, string recipientAddress, string subject, string body)

Public Static Methods

string	*Base64Encode* (string plainText)

Breakdown

- bool **CanSendEmail** ( )

  Description

  If this instance of an Email Sender can send an email, you should return true from here, if you do this, your Development *Console* will have an Email button in certain bits of UI.

- void **Email** (string attachmentPath, string mimeType, string attachmentFilename, string recipientAddress, string subject, string body)

  Description

  Implement this method if your custom email sender needs to send email. You can implement this in any way you see fit.

  Parameters

attachmentPath	File path to attachment.
mimeType	MIME type.
attachmentFilename	Attachment filename.
recipientAddress	Recipient address.
subject	Subject.
body	Body.

- string **Base64Encode** (string plainText)

11.1.26 NoEmailSender

Namespace: WellFired.Command.Unity.Runtime

Implements: *WellFired.Command.Unity.Runtime.Email.IEmailSender*

Description

Public Methods

bool	*CanSendEmail* ( )
void	*Email* (string attachmentPath, string mimeType, string attachmentFilename, string recipientAddress, string subject, string body)

Breakdown

- bool **CanSendEmail** ( )

  Description

  If this instance of an Email Sender can send an email, you should return true from here, if you do this, your Development *Console* will have an Email button in certain bits of UI.

- void **Email** (string attachmentPath, string mimeType, string attachmentFilename, string recipientAddress, string subject, string body)

  Description

  Implement this method if your custom email sender needs to send email. You can implement this in any way you see fit.

  Parameters

attachmentPath	File path to attachment.
mimeType	MIME type.
attachmentFilename	Attachment filename.
recipientAddress	Recipient address.
subject	Subject.
body	Body.

11.1.27 ArrayExtensions

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

T[]	SubArray (this T[] data, int index, int length)
void	Populate (this T[] data, T value)

Breakdown

- T[] **SubArray< T >** (this T[] data, int index, int length)

  Description

  This method gets a sub section of another array.

- void **Populate< T >** (this T[] data, T value)

  Description

  Populates an array with the specified value.

  Parameters

data	The array we will populate
value	The value to populate this array with

11.1.28 Color

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

UnityEngine.Color	*ToColor* (this Skins.Color color)

Breakdown

- UnityEngine.Color **ToColor** (this Skins.Color color)

11.1.29 StringExtensions

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

IEnumerable< string >	*Split* (this string str, Func< char, bool > controller)
IEnumerable< string >	*SplitCommandLine* (string commandLine)
string	*TrimMatchingQuotes* (this string input, char quote)

Breakdown

- IEnumerable< string > **Split** (this string str, Func< char, bool > controller)
- IEnumerable< string > **SplitCommandLine** (string commandLine)

  Description

  When passed a command, this method will split it into it's individual components.

- string **TrimMatchingQuotes** (this string input, char quote)

  Description

This method will trim matching quotes

Parameters

input	The string to trim
quote	The character that represents the quote

11.1.30 Texture

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

Texture2D	*Texture2D* (UnityEngine.Color color, int size)

Breakdown

- Texture2D **Texture2D** (UnityEngine.Color color, int size)

11.1.31 Helper

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

object	*GetArgumentValueFromString* (string argument, Type type)
IEnumerable< string >	*GetDefaultParameterPossibleOptions* (Type type)

Breakdown

- object **GetArgumentValueFromString** (string argument, Type type)
- IEnumerable< string > **GetDefaultParameterPossibleOptions** (Type type)

11.1.32 ParameterWrapper

Namespace: WellFired.Command.Unity.Runtime

Description

Public Properties

string	*Name*
bool	*IsOptional*
Type	*Type*
object	*DefaultValue*
ISuggestion	*SuggestionObject*

Public Methods

List< string >	*GetParameterPossibleValues* (string parameterValue, IEnumerable< string > lastTypedParameters)

Breakdown

- string **Name**
- bool **IsOptional**
- Type **Type**
- object **DefaultValue**
- *ISuggestion* **SuggestionObject**
- List< string > **GetParameterPossibleValues** (string parameterValue, IEnumerable< string > lastTypedParameters)

11.1.33 Suggestions

Namespace: WellFired.Command.Unity.Runtime

Description

Public Methods

	Suggestions (*ISkinData* skinData, *IInputField* inputField)
void	*Update* ( )
void	*Draw* (*ISkinData* skinData, Rect rect)

Breakdown

- **Suggestions** (*ISkinData* skinData, *IInputField* inputField)
- void **Update** ( )
- void **Draw** (*ISkinData* skinData, Rect rect)

11.1.34 SuggestionButton

Namespace: WellFired.Command.Unity.Runtime.Helpers

Description

Properties

float	*Width* { get; set; }
GUIContent	*Content* { get; set; }
bool	*AutomaticallyExecute* { get; set; }
string	*Input* { get; set; }

Public Methods

	SuggestionButton (*ISkinData* skinData, string label, string input, bool automaticallyExecute)

Breakdown

- float **Width** { get; set; }
- GUIContent **Content** { get; set; }
- bool **AutomaticallyExecute** { get; set; }
- string **Input** { get; set; }
- **SuggestionButton** (*ISkinData* skinData, string label, string input, bool automaticallyExecute)

11.1.35 InputFieldFactory

Namespace: WellFired.Command.Unity.Runtime

Description

If you want to provide specific input support for your platform, you can add your platform to this Input Factory, returning a new instance of your custom *IInputField*.

Public Static Methods

IInputField	*GetInputField* (*LogHistoryGui* logHistoryView)

Breakdown

- *IInputField* **GetInputField** (*LogHistoryGui* logHistoryView)

11.1.36 KeyboardInputField

Namespace: WellFired.Command.Unity.Runtime

Implements: *WellFired.Command.Unity.Runtime.Input.IInputField*

Description

Public Properties

const string	*FocusOutId*
bool	*HasFocus*

Properties

Rect	*Rect* { get; set; }
string	*PreviousCompleteInput* { get; set; }
string[]	*PreviousCompleteParameters* { get; set; }
int	*CurrentParameterIndex* { get; set; }
string	*Input* { get; set; }

Public Methods

	KeyboardInputField (*LogHistoryGui* logHistoryGuiView)
void	*Draw* (*ISkinData* skinData)
void	*LoseFocus* ( )
void	*Focus* ( )
void	*FinaliseInput* ( )

Breakdown

- const string **FocusOutId**
- bool **HasFocus**
- Rect **Rect** { get; set; }
- string **PreviousCompleteInput** { get; set; }
- string[] **PreviousCompleteParameters** { get; set; }
- int **CurrentParameterIndex** { get; set; }
- string **Input** { get; set; }
- **KeyboardInputField** (*LogHistoryGui* logHistoryGuiView)
- void **Draw** (*ISkinData* skinData)
- void **LoseFocus** ( )
- void **Focus** ( )
- void **FinaliseInput** ( )

11.1.37 TouchInputField

Namespace: WellFired.Command.Unity.Runtime

Implements: *WellFired.Command.Unity.Runtime.Input.IInputField*

Description

Public Properties

const string	*FocusOutId*
bool	*HasFocus*

Properties

Rect	*Rect* { get; set; }
string	*PreviousCompleteInput* { get; set; }
string[]	*PreviousCompleteParameters* { get; set; }
int	*CurrentParameterIndex* { get; set; }
string	*Input* { get; set; }

Public Methods

	TouchInputField (*LogHistoryGui* logHistoryGuiView)
void	*Draw* (*ISkinData* skinData)
void	*LoseFocus* ( )
void	*Focus* ( )
void	*FinaliseInput* ( )

Breakdown

- const string **FocusOutId**
- bool **HasFocus**
- Rect **Rect** { get; set; }
- string **PreviousCompleteInput** { get; set; }
- string[] **PreviousCompleteParameters** { get; set; }
- int **CurrentParameterIndex** { get; set; }
- string **Input** { get; set; }
- **TouchInputField** (*LogHistoryGui* logHistoryGuiView)
- void **Draw** (*ISkinData* skinData)
- void **LoseFocus** ( )
- void **Focus** ( )
- void **FinaliseInput** ( )

11.1.38 SettingsData

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

Settings	*Load* ( )
void	*Save* (*Settings* settings)

Breakdown

- *Settings* **Load** ( )
- void **Save** (*Settings* settings)

11.1.39 DebugLog

Namespace: WellFired.Command.Unity.Runtime

Description

Breakdown

11.1.40 DebugLogHistory

Namespace: WellFired.Command.Unity.Runtime

Description

Events

Action< *LogEntry* >	*LogEntryAdded*
Action< *LogEntry* >	*LogEntryRemoved*
Action	*LogHistoryCleared*

Public Properties

IList< *LogEntry* >	*LogItems*

Properties

DebugLogHistory	*Instance* { get; set; }

Public Methods

	DebugLogHistory (Action< *LogEntry* > onLogEntryAdded)
void	*Dispose* ( )
void	*Update* ( )
void	*Clear* ( )
void	*LogException* (string message)
void	*LogMessage* (string message, LogType type = LogType.Log)
void	*LogMessage* (string message, string stacktrace, LogType type = LogType.Log)

Breakdown

- Action< *LogEntry* > **LogEntryAdded**
- Action< *LogEntry* > **LogEntryRemoved**
- Action **LogHistoryCleared**
- IList< *LogEntry* > **LogItems**
- *DebugLogHistory* **Instance** { get; set; }
- **DebugLogHistory** (Action< *LogEntry* > onLogEntryAdded)
- void **Dispose** ( )
- void **Update** ( )
- void **Clear** ( )
- void **LogException** (string message)
- void **LogMessage** (string message, LogType type = LogType.Log)
- void **LogMessage** (string message, string stacktrace, LogType type = LogType.Log)

11.1.41 LogEntry

Namespace: WellFired.Command.Unity.Runtime

Description

Properties

LogType	*Type* { get; set; }
string	*StackTrace* { get; set; }
string	*LogMessage* { get; set; }
string	*FirstLineOfLogMessage* { get; set; }
float	*Time* { get; set; }
Enum	*Filter* { get; set; }
string	*FilterName* { get; set; }

Public Methods

	LogEntry (LogType type, string message, float time, string stackTrace = “”)

Breakdown

- LogType **Type** { get; set; }
- string **StackTrace** { get; set; }
- string **LogMessage** { get; set; }
- string **FirstLineOfLogMessage** { get; set; }
- float **Time** { get; set; }
- Enum **Filter** { get; set; }
- string **FilterName** { get; set; }
- **LogEntry** (LogType type, string message, float time, string stackTrace = “”)

11.1.42 LogHandler

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

void	*RegisterLogCallback* (Application.LogCallback callback)
void	*UnRegisterLogCallback* (Application.LogCallback callback)

Breakdown

- void **RegisterLogCallback** (Application.LogCallback callback)
- void **UnRegisterLogCallback** (Application.LogCallback callback)

11.1.43 EmailViewModal

Namespace: WellFired.Command.Unity.Runtime

Description

Public Methods

	EmailViewModal (*LogHistory* viewModel)
string	*GetAsBody* ( )
string	*GetHtml* ( )

Breakdown

- **EmailViewModal** (*LogHistory* viewModel)
- string **GetAsBody** ( )
- string **GetHtml** ( )

11.1.44 LogHistory

Namespace: WellFired.Command.Unity.Runtime

Description

Properties

string	*FilterString* { get; set; }
IList< *LogEntry* >	*LogEntries* { get; set; }

Public Properties

bool	*IsFiltered*
bool	*HasFiltersToggled*

Public Methods

	LogHistory (*DebugLogHistory* debugLogHistory, *Filter* filter)
void	*FilterLogItems* ( )
string	*ActiveFilterNames* ( )

Breakdown

- string **FilterString** { get; set; }
- IList< *LogEntry* > **LogEntries** { get; set; }
- bool **IsFiltered**
- bool **HasFiltersToggled**
- **LogHistory** (*DebugLogHistory* debugLogHistory, *Filter* filter)
- void **FilterLogItems** ( )
- string **ActiveFilterNames** ( )

11.1.45 LogHistoryGui

Namespace: WellFired.Command.Unity.Runtime

Description

Properties

bool	*AutoScrolling* { get; set; }

Public Methods

	LogHistoryGui (*ISkinData* skinData, *LogHistory* viewModel, *DevelopmentConsole* console)
void	*Draw* (*ISkinData* skinData, bool mouseUpOutsideOfWindow)

Breakdown

- bool **AutoScrolling** { get; set; }
- **LogHistoryGui** (*ISkinData* skinData, *LogHistory* viewModel, *DevelopmentConsole* console)
- void **Draw** (*ISkinData* skinData, bool mouseUpOutsideOfWindow)

11.1.46 Skin

Namespace: WellFired.Command.Unity.Runtime

Description

Properties

GUIStyle	*TooltipMoreBackgroundStyle* { get; set; }
GUISkin	*GuiSkin* { get; set; }
GUIStyle	*PopupHeaderStyle* { get; set; }
GUIStyle	*PopupWidowStyle* { get; set; }
GUIStyle	*ScrollViewStyle* { get; set; }
GUIStyle	*ConsoleItemBorderlessLabelStyle* { get; set; }
GUIStyle	*FullSizeBorderlessLabelStyle* { get; set; }
GUIStyle	*TooltipNormalStyle* { get; set; }
GUIStyle	*TooltipHighlightedStyle* { get; set; }
GUIStyle	*TooltipOptionBackgroundStyle* { get; set; }
GUIStyle	*ConsoleTextField* { get; set; }
GUIStyle	*ConsoleWindowBackgroundStyle* { get; set; }
GUIStyle	*TooltipLabelBackground* { get; set; }
GUIStyle	*HeaderStyle* { get; set; }
GUIStyle	*ItemAlternateBackgroundStyle* { get; set; }
GUIStyle	*HighlightedItemBackgroundStyle* { get; set; }
GUIStyle	*SuggestionButtonBackgroundStyle* { get; set; }
GUIStyle	*SuggestionButtonMoreBackgroundStyle* { get; set; }
int	*ConsoleRowTextHeight* { get; set; }
int	*ConsoleRowHeight* { get; set; }
int	*ConsoleRowTextLeftMargin* { get; set; }

Breakdown

- GUIStyle **ConsoleTextField** { get; set; }
- GUIStyle **TooltipMoreBackgroundStyle** { get; set; }
- GUIStyle **PopupHeaderStyle** { get; set; }
- GUIStyle **PopupWidowStyle** { get; set; }
- GUIStyle **ScrollViewStyle** { get; set; }
- GUIStyle **ConsoleItemBorderlessLabelStyle** { get; set; }
- GUIStyle **FullSizeBorderlessLabelStyle** { get; set; }
- GUIStyle **TooltipNormalStyle** { get; set; }
- GUIStyle **TooltipHighlightedStyle** { get; set; }
- GUIStyle **TooltipOptionBackgroundStyle** { get; set; }
- GUISkin **GuiSkin** { get; set; }
- GUIStyle **ConsoleWindowBackgroundStyle** { get; set; }
- GUIStyle **TooltipLabelBackground** { get; set; }
- GUIStyle **HeaderStyle** { get; set; }
- GUIStyle **ItemAlternateBackgroundStyle** { get; set; }
- GUIStyle **HighlightedItemBackgroundStyle** { get; set; }
- GUIStyle **SuggestionButtonBackgroundStyle** { get; set; }
- GUIStyle **SuggestionButtonMoreBackgroundStyle** { get; set; }
- int **ConsoleRowTextHeight** { get; set; }
- int **ConsoleRowHeight** { get; set; }
- int **ConsoleRowTextLeftMargin** { get; set; }

11.1.47 SuggestionAttribute

Namespace: WellFired.Command.Unity.Runtime

Description

This attribute can be used on parameters only. to specify that you will provide auto complete behaviour for that element. The type that is passed to the constructor must be a type that implements the *ISuggestion* interface.

Properties

Type	*Type* { get; set; }

Public Methods

	SuggestionAttribute (Type type)

Breakdown

- Type **Type** { get; set; }
- **SuggestionAttribute** (Type type)

11.1.48 Scroller

Namespace: WellFired.Command.Unity.Runtime

Description

Properties

ScrollerState	*State* { get; set; }

Public Methods

	Scroller (float maxReleaseSpeed, float decelerationSpeed)
void	*DoTouch* (Vector2 touchPosition)
void	*DoUntouch* ( )
Vector2	*Update* (Vector2 touchPosition, float deltaTime)
void	*ZeroSpeed* ( )

Breakdown

- ScrollerState **State** { get; set; }
- **Scroller** (float maxReleaseSpeed, float decelerationSpeed)
- void **DoTouch** (Vector2 touchPosition)
- void **DoUntouch** ( )
- Vector2 **Update** (Vector2 touchPosition, float deltaTime)
- void **ZeroSpeed** ( )

11.1.49 CommandsWindow

Namespace: WellFired.Command.Unity.Runtime.UI

Inherits: *WellFired.Command.Unity.Runtime.UI.Windows.PopupWindow*

Description

Public Methods

	CommandsWindow (*ISkinData* skinData)

protected-func

override void	*OnShow* ( )
override void	*OnHide* ( )
override void	*DrawWindow* (int windowId)

Breakdown

- **CommandsWindow** (*ISkinData* skinData)
- override void **OnShow** ( )
- override void **OnHide** ( )
- override void **DrawWindow** (int windowId)

11.1.50 Filter

Namespace: WellFired.Command.Unity.Runtime.UI

Description

Properties

IEnumerable< *FilterState* >	*Custom* { get; set; }
IEnumerable< *FilterState* >	*Global* { get; set; }

Public Properties

int	*ActiveFilterCount*

Public Methods

	Filter ( )
void	*AddCustomFilters* (IEnumerable< *FilterState* > filterStates)

Breakdown

- IEnumerable< *FilterState* > **Custom** { get; set; }
- IEnumerable< *FilterState* > **Global** { get; set; }
- int **ActiveFilterCount**
- **Filter** ( )
- void **AddCustomFilters** (IEnumerable< *FilterState* > filterStates)

11.1.51 FilterState

Namespace: WellFired.Command.Unity.Runtime.UI.Windows

Description

Properties

string	*Name* { get; set; }
bool	*CustomFilter* { get; set; }
bool	*State* { get; set; }

Public Methods

	FilterState (string name, bool custom)

Breakdown

- string **Name** { get; set; }
- bool **CustomFilter** { get; set; }
- bool **State** { get; set; }
- **FilterState** (string name, bool custom)

11.1.52 FilterWindow

Namespace: WellFired.Command.Unity.Runtime.UI

Inherits: *WellFired.Command.Unity.Runtime.UI.Windows.PopupWindow*

Description

Properties

LogHistory	*LogHistory* { get; set; }

Public Methods

	FilterWindow (*ISkinData* skinData, *Filter* filter)
void	*Show* (*LogEntry* logEntry)

protected-func

override void	*OnShow* ( )
override void	*OnHide* ( )
override void	*DrawWindow* (int windowId)

Breakdown

- *LogHistory* **LogHistory** { get; set; }
- **FilterWindow** (*ISkinData* skinData, *Filter* filter)
- void **Show** (*LogEntry* logEntry)
- override void **OnShow** ( )
- override void **OnHide** ( )
- override void **DrawWindow** (int windowId)

11.1.53 LogEntryPopupWindow

Namespace: WellFired.Command.Unity.Runtime.UI

Inherits: *WellFired.Command.Unity.Runtime.UI.Windows.PopupWindow*

Description

Public Methods

	LogEntryPopupWindow (*ISkinData* skinData)
void	*Show* (*LogEntry* logEntry)

protected-func

override void	*OnShow* ( )
override void	*OnHide* ( )
override void	*DrawWindow* (int windowId)

Breakdown

- **LogEntryPopupWindow** (*ISkinData* skinData)
- void **Show** (*LogEntry* logEntry)
- override void **OnShow** ( )
- override void **OnHide** ( )
- override void **DrawWindow** (int windowId)

11.1.54 PopupWindow

Namespace: WellFired.Command.Unity.Runtime.UI

Description

protected-attrib

readonly *ISkinData*	*SkinData*

Properties

bool	*IsVisible* { get; set; }

Public Properties

float	*WindowX*
float	*WindowY*
float	*WindowWidth*
float	*WindowHeight*

protected-func

abstract void	*OnShow* ( )
abstract void	*OnHide* ( )
abstract void	*DrawWindow* (int windowId)

Public Methods

	PopupWindow (*ISkinData* skinData, string windowTitle, int windowId)
void	*Show* ( )
void	*Hide* ( )
Rect	*Draw* ( )

Breakdown

- readonly *ISkinData* **SkinData**
- bool **IsVisible** { get; set; }
- float **WindowX**
- float **WindowY**
- float **WindowWidth**
- float **WindowHeight**
- abstract void **OnShow** ( )
- abstract void **OnHide** ( )
- abstract void **DrawWindow** (int windowId)
- **PopupWindow** (*ISkinData* skinData, string windowTitle, int windowId)
- void **Show** ( )
- void **Hide** ( )
- Rect **Draw** ( )

11.1.55 GuiBeginHorizontal

Namespace: WellFired.Command.Unity.Runtime

Description

Public Methods

	GuiBeginHorizontal ( )
	GuiBeginHorizontal (params GUILayoutOption[] layoutOptions)
	GuiBeginHorizontal (GUIStyle guiStyle, params GUILayoutOption[] layoutOptions)
void	*Dispose* ( )

Breakdown

- **GuiBeginHorizontal** ( )
- **GuiBeginHorizontal** (params GUILayoutOption[] layoutOptions)
- **GuiBeginHorizontal** (GUIStyle guiStyle, params GUILayoutOption[] layoutOptions)
- void **Dispose** ( )

11.1.56 GuiBeginVertical

Namespace: WellFired.Command.Unity.Runtime

Description

Public Methods

	GuiBeginVertical ( )
	GuiBeginVertical (params GUILayoutOption[] layoutOptions)
	GuiBeginVertical (GUIStyle guiStyle, params GUILayoutOption[] layoutOptions)
void	*Dispose* ( )

Breakdown

- **GuiBeginVertical** ( )
- **GuiBeginVertical** (params GUILayoutOption[] layoutOptions)
- **GuiBeginVertical** (GUIStyle guiStyle, params GUILayoutOption[] layoutOptions)
- void **Dispose** ( )

11.1.57 GuiChangeColor

Namespace: WellFired.Command.Unity.Runtime

Description

Public Methods

	GuiChangeColor (Color newColor)
void	*Dispose* ( )

Breakdown

- **GuiChangeColor** (Color newColor)
- void **Dispose** ( )

11.1.58 GuiChangeContentColor

Namespace: WellFired.Command.Unity.Runtime

Description

Public Methods

	GuiChangeContentColor (Color newColor)
void	*Dispose* ( )

Breakdown

- **GuiChangeContentColor** (Color newColor)
- void **Dispose** ( )

11.1.59 GuiEnable

Namespace: WellFired.Command.Unity.Runtime

Description

Public Methods

	GuiEnable (bool newState)
void	*Dispose* ( )

Breakdown

- **GuiEnable** (bool newState)
- void **Dispose** ( )

11.1.60 GuiLayoutReflectionHelper

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

object	*GetFieldValue* (object obj, string fieldName)
void	*SetFieldValue* (object obj, string fieldName, object val)

Breakdown

- object **GetFieldValue** (object obj, string fieldName)
- void **SetFieldValue** (object obj, string fieldName, object val)

11.1.61 GuiScaler

Namespace: WellFired.Command.Unity.Runtime

Description

Public Properties

float	*Scale*

Public Methods

	GuiScaler ( )
	GuiScaler (float scale)
void	*ReScale* (float scale)
void	*Begin* ( )
void	*End* ( )

Breakdown

- float **Scale**
- **GuiScaler** ( )
- **GuiScaler** (float scale)
- void **ReScale** (float scale)
- void **Begin** ( )
- void **End** ( )

11.1.62 Helper

Namespace: WellFired.Command.Unity.Runtime

Description

Properties

float	*Scale* { get; set; }

Public Static Methods

IDisposable	*BodyBeginVertical* (*ISkinData* skinData)
bool	*IsValidConsoleScale* (float scale)
void	*Label* (*ISkinData* skinData, GUIContent content, params GUILayoutOption[] options)
void	*Label* (*ISkinData* skinData, GUIContent content)
void	*Label* (*ISkinData* skinData, Rect rect, GUIContent content)
bool	*Button* (*ISkinData* skinData, string message)
bool	*Button* (*ISkinData* skinData, string message, params GUILayoutOption[] options)
bool	*Button* (*ISkinData* skinData, Rect rect, string message)
Vector2	*LabelSizeWithContent* (*ISkinData* skinData, GUIContent content)
IDisposable	*HeaderBeginHorizontal* (*ISkinData* skinData)
IDisposable	*HeaderBeginVertical* (*ISkinData* skinData)
IDisposable	*BodyBeginHorizontal* (*ISkinData* skinData)
GUIStyle	*Window* (*ISkinData* skinData)
void	*Space* (*ISkinData* skinData)
void	*ShrinkableSpace* (*ISkinData* skinData)
void	*LogEntry* (*ISkinData* skinData, Rect itemRect, string message, LogType type, bool hover, bool active, bool on, bool keyboardFocus)
string	*TextEntry* (*ISkinData* skinData, string commandInput, params GUILayoutOption[] options)
Vector2	*DrawTooltip* (*ISkinData* skinData, Vector2 topLeft, *CommandWrapper* commandWrapper)
GUIStyle	*SuggestionButtonStyle* (*ISkinData* skinData)
GUIStyle	*SuggestionLabelStyle* (*ISkinData* skinData)
Vector2	*BeginScrollView* (*ISkinData* skinData, Vector2 scrollPosition, params GUILayoutOption[] options)
void	*TextArea* (*ISkinData* skinData, GUIContent content, params GUILayoutOption[] layoutOptions)
void	*DrawArgument* (*ISkinData* skinData, Rect rect, GUIContent content, bool current)
string	*SearchField* (*ISkinData* skinData, string commandInput, float screenWidth)
int	*EntryHeight* (*ISkinData* skinData)

Breakdown

- float **Scale** { get; set; }
- GUIStyle **Window** (*ISkinData* skinData)
- IDisposable **BodyBeginVertical** (*ISkinData* skinData)
- void **Label** (*ISkinData* skinData, GUIContent content, params GUILayoutOption[] options)
- void **Label** (*ISkinData* skinData, GUIContent content)
- void **Label** (*ISkinData* skinData, Rect rect, GUIContent content)
- bool **Button** (*ISkinData* skinData, string message)
- bool **Button** (*ISkinData* skinData, string message, params GUILayoutOption[] options)
- bool **Button** (*ISkinData* skinData, Rect rect, string message)
- Vector2 **LabelSizeWithContent** (*ISkinData* skinData, GUIContent content)
- IDisposable **HeaderBeginHorizontal** (*ISkinData* skinData)

- IDisposable **HeaderBeginVertical** (*ISkinData* skinData)
- IDisposable **BodyBeginHorizontal** (*ISkinData* skinData)
- bool **IsValidConsoleScale** (float scale)
- void **Space** (*ISkinData* skinData)
- void **ShrinkableSpace** (*ISkinData* skinData)
- void **LogEntry** (*ISkinData* skinData, Rect itemRect, string message, LogType type, bool hover, bool active, bool on, bool keyboardFocus)
- string **TextEntry** (*ISkinData* skinData, string commandInput, params GUILayoutOption[] options)
- Vector2 **DrawTooltip** (*ISkinData* skinData, Vector2 topLeft, *CommandWrapper* commandWrapper)
- GUIStyle **SuggestionButtonStyle** (*ISkinData* skinData)
- GUIStyle **SuggestionLabelStyle** (*ISkinData* skinData)
- Vector2 **BeginScrollView** (*ISkinData* skinData, Vector2 scrollPosition, params GUILayoutOption[] options)
- void **TextArea** (*ISkinData* skinData, GUIContent content, params GUILayoutOption[] layoutOptions)
- void **DrawArgument** (*ISkinData* skinData, Rect rect, GUIContent content, bool current)
- string **SearchField** (*ISkinData* skinData, string commandInput, float screenWidth)
- int **EntryHeight** (*ISkinData* skinData)

11.1.63 Padding

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

RectOffset	*OnPlatform* (int mouseAndKeyboard, int touch)

Breakdown

- RectOffset **OnPlatform** (int mouseAndKeyboard, int touch)

11.1.64 Platform

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

bool	*IsMouseAndKeyboard* ( )

Breakdown

- bool **IsMouseAndKeyboard** ( )

11.1.65 Selector

Namespace: WellFired.Command.Unity.Runtime

Description

Public Static Methods

T	OnPlatform (T mouseAndKeyboard, T touch)

Breakdown

- T **OnPlatform< T >** (T mouseAndKeyboard, T touch)

11.1.66 CommandWrapper

Namespace: WellFired.Command.Unity.Runtime

Description

Each *Command* has an instance of something which derives from *CommandWrapper*, this simply contains the details for the call. Basically caching it. It is possible to implement custom commands, simply by deriving from this type

protected-attrib

string	*TheMethodOrPropertyName*

Properties

string	*Description* { get; set; }
string	*CommandName* { get; set; }
WeakReference	*ObjectReference* { get; set; }
Type	*Type* { get; set; }
abstract :ref:‘ParameterWrapper<classwellfired_command_unity_runtime_helpers_parameterwrapper	*Parameters* { get; set; }

Public Properties

string	*MethodOrPropertyName*

Public Methods

abstract void	*Invoke* (params string[] arguments)
override bool	*Equals* (object otherObject)
override int	*GetHashCode* ( )
bool	*IsValid* ( )
string	*GetParametersAsString* ( )

protected-func

	CommandWrapper (string commandName, string description, Type type, object referenceObject)

Breakdown

- string **TheMethodOrPropertyName**
- string **Description** { get; set; }
- string **CommandName** { get; set; }
- WeakReference **ObjectReference** { get; set; }
- Type **Type** { get; set; }
- abstract :ref:‘ParameterWrapper<classwellfired_command_unity_runtime_helpers_parameterwrapper>‘[] **Parameters** { get; set; }
- string **MethodOrPropertyName**
- abstract void **Invoke** (params string[] arguments)
- override bool **Equals** (object otherObject)

  Description

  Your custom *Command* Wrappers can implement this to check for object equality.

  Parameters

otherObject	The *CommandWrapper* to compare with the current *CommandWrapper*.

- override int **GetHashCode** ( )
- bool **IsValid** ( )

  Description

  Determines whether this instance is valid.

- string **GetParametersAsString** ( )

  Description

  Getsa string containing all the method parameters.

- **CommandWrapper** (string commandName, string description, Type type, object referenceObject)

11.2 Interfaces

11.2.1 ISkinData

Namespace: *WellFired.Command*

Description

Properties

Color	*DetailedLogMessageBackgroundColor* { get; set; }
Color	*MainColor* { get; set; }
Color	*MainFontColor* { get; set; }
Color	*TextEntryFontColor* { get; set; }
Color	*TextEntryColor* { get; set; }
Color	*ButtonColor* { get; set; }
Color	*ButtonHoverColor* { get; set; }
Color	*EntryExceptionColor* { get; set; }
Color	*EntryErrorColor* { get; set; }
Color	*EntryWarningColor* { get; set; }
Color	*EntryInfoColor* { get; set; }
Color	*SecondaryColor* { get; set; }
Color	*DetailedLogMessageColor* { get; set; }
Color	*GeneralLabelFontColor* { get; set; }
int	*FontSize* { get; set; }
int	*ButtonSpacing* { get; set; }
int	*ButtonSpacingTouch* { get; set; }
int	*EntryHeight* { get; set; }
int	*ButtonPaddingKeyboard* { get; set; }
int	*ButtonPaddingTouch* { get; set; }
int	*HeaderPaddingKeyboard* { get; set; }
int	*HeaderPaddingTouch* { get; set; }

Breakdown

- Color **SecondaryColor** { get; set; }
- Color **DetailedLogMessageBackgroundColor** { get; set; }
- Color **MainFontColor** { get; set; }
- Color **TextEntryFontColor** { get; set; }
- Color **TextEntryColor** { get; set; }
- Color **ButtonColor** { get; set; }
- Color **ButtonHoverColor** { get; set; }
- Color **EntryExceptionColor** { get; set; }
- Color **EntryErrorColor** { get; set; }
- Color **EntryWarningColor** { get; set; }

- Color **EntryInfoColor** { get; set; }
- Color **MainColor** { get; set; }
- Color **DetailedLogMessageColor** { get; set; }
- Color **GeneralLabelFontColor** { get; set; }
- int **FontSize** { get; set; }
- int **ButtonSpacing** { get; set; }
- int **ButtonSpacingTouch** { get; set; }
- int **EntryHeight** { get; set; }
- int **ButtonPaddingKeyboard** { get; set; }
- int **ButtonPaddingTouch** { get; set; }
- int **HeaderPaddingKeyboard** { get; set; }
- int **HeaderPaddingTouch** { get; set; }

11.2.2 IEmailSender

Namespace: WellFired.Command.Unity.Runtime

Description

You can implement this interface if you would like to provide specific functionality for your debug console to send email logs.

Public Methods

bool	*CanSendEmail* ( )
void	*Email* (string attachmentPath, string mimeType, string attachmentFilename, string recipientAddress, string subject, string body)

Breakdown

- bool **CanSendEmail** ( )

  Description

  If this instance of an Email Sender can send an email, you should return true from here, if you do this, your Development *Console* will have an Email button in certain bits of UI.

- void **Email** (string attachmentPath, string mimeType, string attachmentFilename, string recipientAddress, string subject, string body)

  Description

  Implement this method if your custom email sender needs to send email. You can implement this in any way you see fit.

  Parameters

attachmentPath	File path to attachment.
mimeType	MIME type.
attachmentFilename	Attachment filename.
recipientAddress	Recipient address.
subject	Subject.
body	Body.

11.2.3 IInputField

Namespace: WellFired.Command.Unity.Runtime

Description

Properties

bool	*HasFocus* { get; set; }
Rect	*Rect* { get; set; }
string	*PreviousCompleteInput* { get; set; }
string[]	*PreviousCompleteParameters* { get; set; }
int	*CurrentParameterIndex* { get; set; }
string	*Input* { get; set; }

Public Methods

void	*FinaliseInput* ( )
void	*Focus* ( )
void	*LoseFocus* ( )
void	*Draw* (*ISkinData* skinData)

Breakdown

- bool **HasFocus** { get; set; }
- Rect **Rect** { get; set; }
- string **PreviousCompleteInput** { get; set; }
- string[] **PreviousCompleteParameters** { get; set; }
- int **CurrentParameterIndex** { get; set; }
- string **Input** { get; set; }
- void **FinaliseInput** ( )
- void **Focus** ( )
- void **LoseFocus** ( )
- void **Draw** (*ISkinData* skinData)

11.2.4 ISuggestion

Namespace: WellFired.Command.Unity.Runtime

Description

The Interface for an IAutoCompletable Object, this is used by the DevelopmentConsole to auto complete suggestions for the user. You can provide your own Auto Complete method, E.G.

Public Methods

IEnumerable< string >	*Suggestion* (IEnumerable< string > previousArguments)

Breakdown

- IEnumerable< string > **Suggestion** (IEnumerable< string > previousArguments)

  Description

  This method will be used by the .:ref:*Command<namespacewellfired_command>* to determine the auto complete values that should be used.

  Parameters

previousArguments	The Previous Arguments

11.3 Namespaces

11.3.1 Command

Namespace: <namespace>

Description

Breakdown

11.3.2 Console

Namespace: WellFired.Command.Unity

Description

Breakdown

11.4 Enums

11.4.1 Theme

Namespace: *WellFired.Command*

Description

Default
Light
SkyBlue
Green
Purple
DarkOlive

11.4.2 DisplayCorner

Namespace: *WellFired.Command.Unity.Runtime.Console*

Description

TopLeft
TopRight
BottomRight
BottomLeft