DTT
APPS WEB GAMES

SINCE 2010

Runtime Utilities

Documentation | V1.0.0 | 17-11-21

DTT APPS WEB GAMES

ˆ package logo

Table of Contents

1. Get started quickly

To start using the utility tools of the asset, import one of the utility namespaces into your script file.

Using the **DTT.Utils.Extensions** namespace you are provided with additional methods for a variety of purposes. For example, you can now start setting values for your **RectTransform** components more easily.

```
using DTT.Utils.Extensions;
using UnityEngine;

public class CustomBehaviour : MonoBehaviour
{
    private void Start()
    {
        // Retrieve the RectTransform component.
        RectTransform rectTransform = this.GetRectTransform();

        // Set the anchor values with one method.
        rectTransform.SetAnchor(0.5f, 0.5f, 0.5f, 0.5f);

        // Or set the anchor values using one of unity's predefined anchor settings.
        bool alsoSetPosition = true;
        bool alsoSetPivot = false;
        rectTransform.SetAnchor(RectAnchor.STRETCH_CENTER, alsoSetPivot, alsoSetPosition);
    }
}
```

Using the **DTT.Utils.Workflow** namespace you can boost your project workflow by using methods that would normally require you to write a lot of boilerplate code. For Example, you can now start using the **PathUtility** class to manage paths more easily.

```
using DTT.Utils.Workflow;
using UnityEditor;
using UnityEngine;

public static class TestScript
{
    public static void ManageObjectAtPath(string fullPathOfObject)
    {
        // Convert the full path to an asset path that Unity can work with.
        string assetPath = PathUtility.ToAssetPath(fullPathOfObject);
        Object asset = AssetDatabase.LoadAssetAtPath(assetPath, typeof(Object));
    }
}
```

Using the **DTT.Utils.Optimization** namespace you can optimize your code by for example using a lazy dictionary instead of a normal one.

```
using DTT.Utils.Optimization;
using UnityEngine;

public class TestScript : MonoBehaviour
{
    private readonly LazyDictionary<string, GameObject[]> _taggedGameObjects = new LazyDictionary<string,
GameObject[]>();

    private void Awake()
    {
        // Fill up the enemies array with values when you are first trying to retrieve them.
        _taggedGameObjects.Add("Enemies", () => GameObject.FindGameObjectsWithTag("Enemy"));
        _taggedGameObjects.Add("Friends", () => GameObject.FindGameObjectsWithTag("Friend"));
    }
}
```

2. Introduction

The goal of the **Runtime Utilities** asset is to provide users with a broad range of tools to reduce work time, optimize project code and boost workflow. It contains three namespaces. Each with its own set of tools.

The **DTT.Utils.Extensions** namespace contains static classes with extension methods for things like Math, string modification, Rich Text and unity its RectTransform component.

The **DTT.Utils.Workflow** namespace helps to boost your workflow by providing methods that would normally require you to write a lot of boilerplate code.

The **DTT.Utils.Optimization** namespace provides datastructes that help you optimize your code.

3. Changelog

1. version 1.0.0 – Initial release

4. Set-Up

Setting up your project with this asset is as easy as importing one of the three provided namespaces: **DTT.Utils.Extensions**, **DTT.Utils.Workflow** or **DTT.Utils.Optimization**.

5. API

5.1 Enum Extensions

```
public enum Values
{
    One,
    Two,
    Three,
    Four
}

private void Awake()
{
    // Use the 'Next' extension method to iterate forward over your enum values.
    Values value = Values.One;
    Debug.Log(value.Next()); /* Log: Two */

    // Use the 'Previous' extension method iterate backward over your enum values.
    Debug.Log(value.Previous()); /* Log: Four */
}
```

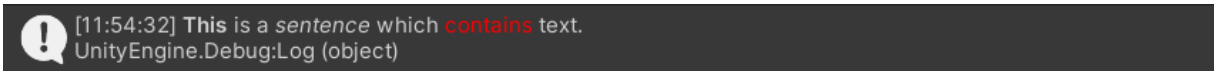
5.2 Math Extensions

```
int value = 1;

// Use the 'Inverse' method to inverse a number.
int inverse = value.Inverse(); /* inverse: -1 */

// Use the 'InRange' method to check whether a value is in range of a min and max.
bool inRange = value.InRange(0, 2); /* inRange: true */
```

```
float value = 0.5f;

// Use the 'Inverse' method to inverse a number.
float inverse = value.Inverse(); /* inverse: -0.5 */

// Use the 'Complement' to get 1 minus your value.
float complement = value.Complement(); /* complement: 0.5 */

// Use the 'Normalize' method to normalize a value between a min and max.
float normalized = value.Normalize(0f, 5f); /* normalized: 0.1 */

// Use the 'Map' method to map a value to a new scale.
float mapped = value.Map(0f, 5f, 5f, 10f); /* mapped: 5.5 */
```

5.3 Rich Text Extensions

```
string text = "This is a sentence which contains text.";

// Use the rich text extensions to add styling to your text.
text = text.Replace("This", "This".Bold());
text = text.Replace("sentence", "sentence".Italics());
text = text.Replace("contains", "contains".Color(RichTextColor.RED));

// Unity's console supports rich text. This can also be used with Text Mesh Pro and the GUIStyle.
Debug.Log(text);
```



5.4 String Extensions

```
// Use extension methods for modifying text capitalization.
string spacedOut = "MyCustomString".AddSpacesBeforeCapitals(); /* spacedOut: My Custom String */
string readable = "MY_CONSTANT_NAME".FromAllCapsToReadableFormat(); /* readble: My Constant Name */
string constant = "My readable name".FromReadableFormatToAllCaps(); /* constant: MY_CONSTANT_NAME */

// Use 'StripHtmlTags' to remove html tags from your strings.
string stripped = "<b>bold text</b>".StripHtmlTags(); /* stripped: bold text */

// Use 'Ellipsis' to truncate your string after a certain length and at a character to indicate its end.
string ellipsis = "This text is to long.".Ellipsis(5, GUI.skin.font); /* This text is... */
```

5.5 Remaining API

The above covers the Runtime Utilities API you are most likely to use. This, however, is not an exhaustive list and you can find further functionality by exploring the namespaces. If you have any questions regarding one of the undocumented methods or classes, feel free to contact us!

6. Support and feedback

If you have any questions regarding the use of this asset, we are happy to help you out. Always feel free to contact us at:

unity-support@d-tt.nl

(We typically respond within 1-2 business days)

We are actively developing this asset, with many future updates and extensions already planned. We are eager to include feedback from our users in future updates, be they 'quality of life' improvements, new features, bug fixes or anything else that can help you improve your experience with this asset. You can reach us at the email above.

Reviews and ratings are very much appreciated as they help us raise awareness and to improve our assets.

DTT stands for Doing Things Together

DTT is an app, web and game development agency based in the centre of Amsterdam. Established in 2010, DTT has over a decade of experience in mobile, game, and web based technology.

Our game department primarily works in Unity where we put significant emphasis on the development of internal packages, allowing us to efficiently reuse code between projects. To support the Unity community, we are publishing a selection of our internal packages on the Asset Store, including this one.

More information about DTT (including our clients, projects and vacancies) can be found here:

https://www.d-tt.nl/en/