

Namespace Cobilas

Classes

[TypeUtilitarian](#)

Utility static class to obtain type or assembly.

Structs

[Interrupter](#)

Represents a list of switches.

[NullObject](#)

This class represents a null object.

Interfaces

[INullObject](#)

This interface is used to demarcate if a specific object is a null representation.

Cobilas Core

Description

Cobilas Core Net4x is a utility library for CSharp.

Json

(namespace:Cobilas.IO.Serialization.Json)

Only present in the NuGet version.

The static class `Json` grants static read and write functions.

JsonContractResolver

Used by `JsonSerializer` to resolve a `JsonContract` for a given `Type`. Furthermore, `JsonContractResolver` determines how the fields of an `Object` will be serialized.

ATLF(Arquivo de tradução de leitura facil)

ATLF (Easy to Read Translation File) can be used to create and load translations for apps.

```
#>Header
```

```
The use of the header is not mandatory.<#
```

```
#! version:/*std:1.0*/
```

```
#! encoding:/*utf-8*/
```

```
#> Comment <#
```

```
#> ATLF format(1.0) <#
```

```
#> Uni-line marking <#
```

```
#! Tag1:/*value1*/
```

```
#> Multi-line marking <#
```

```
#! Tag2:/*
```

```
value1
```

```
value2
```

```
value3
```

```
value4
```

```
*/
```

How to read ATLF

```
static void Main(string[] args) {  
    using ATLFReader reader = ATLFReader.Create(@"C:\folder1\file.txt");  
    reader.Reader();  
}
```

```

    Console.WriteLine($"tag.value.1:{reader.GetTag("tag.value.1")}");
    Console.WriteLine($"tag.value.2:{reader.GetTag("tag.value.2")}");
    Console.WriteLine($"tag.value.3:{reader.GetTag("tag.value.3")}");
}

```

The other reading functions.

- The `ATLFNode[]:ATLFReader.GetHeader()` function allows you to get the header tags.
- The `ATLFNode[]:ATLFReader.GetAllComments()` function allows you to get all comments. The `ATLFNode[]:ATLFReader.GetTagGroup(string path)` function allows you to obtain tags that belong to the same path.

```

/*C:\folder1\file.txt
* #! version:/*std:1.0* /
* #! encoding:/*utf-8* /
*
* #! tag.value.cop1:/*value1* /
* #! tag.value.map.cop1:/*value1* /
* #! tag.value.map.cop2:/*value1* /
* #! tag.value.cop2:/*value1* /
* #! tag.value.cop3:/*value1* /
*/
static void Main(string[] args) {
    using ATLFReader reader = ATLFReader.Create(@"C:\folder1\file.txt");
    reader.Reader();
    foreach(var item in reader.GetTagGroup("tag.value.map"))
        Console.WriteLine(item);
}

```

How to write ATLF

```

static void Main(string[] args) {
    using ATLFWriter writer = ATLFWriter.Create(File.OpenWrite(@"C:\folder1\file.txt"));
    writer.WriteHeader();//The header is not mandatory but if you add a header, call this
function first.
    writer.WriteComment("my tag1");
    writer.WriteNode("tag1", "value1");
    writer.WriteWhitespace("\r\n");//This function is called automatically when the `Indent`
property is `true`. By default the `Indent` property is `true`.
    writer.WriteComment("my tag2");
    writer.WriteNode("tag2", "value2");
    writer.WriteWhitespace(2, "\r\n");//This function is called automatically when the
`Indent` property is `true`. By default the `Indent` property is `true`.
    writer.WriteComment("my tag3");
}

```

```
writer.WriteNode("tag3", "value3");  
}
```

Encoders and decoders

Regarding encoders and decoders, ATLF allows the creation of customized encoders and decoders. To use a custom encoder or decoder, assign a version to your custom encoder or decoder using the `Version` property and then assign the version of the custom encoder or decoder in the `TargetVersion` property of the `ATLFWriter` and `ATLFReader` classes.

Creating a custom encoding class

To create a custom encoding class, the class must inherit the `ATLFVS10Encoding` class.

Creating a custom decoding class

To create a custom decoding class, the class must inherit the `ATLFVS10Decoding` class.

[Cobilas.Core.Net4x](#) is on nuget.org

To include the package, open the `.csproj` file and add it.

```
<ItemGroup>  
  <PackageReference Include="Cobilas.Core.Net4x" Version="2.1.0" />  
</ItemGroup>
```

Or use command line.

```
dotnet add package Cobilas.Core.Net4x --version 2.1.0
```

Namespace Cobilas.GodotEditor.Utility. Serialization

Classes

[BuildSerialization](#)

Class allows to build a serialization list of properties of a node class.

[HidePropertyAttribute](#)

The attribute allows you to hide and save the value of a field or property in the editor.

[MemberItem](#)

Represents a property or field.

[PropertyItem](#)

The class stores the information for drawing in the editor.

[SerializationCache](#)

Class to handle property caching.

[SerializeFieldAttribute](#)

Base attribute for field and property serialization attributes.

[ShowPropertyAttribute](#)

The attribute allows you to show a field or property in the editor.

[ShowRangePropertyAttribute](#)

The attribute allows you to display a field or property in the editor in range form.

Structs

[SNInfo](#)

Represents the information of a [SerializedNode](#).

Interfaces

[ISerializedPropertyManipulation](#)

The interface allows property manipulation.

Cobilas Godot Utility

Description

The package contains utility classes in csharp for godot engine(Godot3.5)

RunTimeInitialization

(namespace: Cobilas.GodotEngine.Utility.Runtime)

The `RunTimeInitialization` class allows you to automate the `Project>Project Settings>AutoLoad` option.

To use the `RunTimeInitialization` class, you must create a class and make it inherit `RunTimeInitialization`.

```
using Cobilas.GodotEngine.Utility.Runtime;
//The name of the class is up to you.
public class RunTimeProcess : RunTimeInitialization {}
```

And remember to add the class that inherits `RunTimeInitialization` in `Project>Project Settings>AutoLoad` .

Remembering that the `RunTimeInitialization` class uses the virtual method `_Ready()` to perform the initialization of other classes.

And to initialize other classes along with the `RunTimeInitialization` class, the class must inherit the `Godot.Node` class or some class that inherits `Godot.Node` and use the `RunTimeInitializationClassAttribute` attribute.

```
using Godot;
using Cobilas.GodotEngine.Utility.Runtime;
[RunTimeInitializationClass]
public class ClassTest : Node {}
```

RunTimeInitializationClass

```
/*
bootPriority: Represents the boot order
{ (enum Priority)values
    StartBefore,
    StartLater
}
name:The name of the object
subPriority: And the execution priority order.
*/
```

```

//RunTimeInitializationClassAttribute(string? name, Priority bootPriority =
Priority.StartBefore, int subPriority = 0, bool lastBoot = false)
[RunTimeInitializationClassAttribute(string?, [Priority:Priority.StartBefore],
[int:0], [bool:false])]
[RunTimeInitializationClass()]

```

CoroutineManager

The `CoroutineManager` class is responsible for creating and managing coroutines for godot.
How to create a coroutine?

```

using Godot;
using System.Collections;
using Cobilas.GodotEngine.Utility;

public class ClassTest : Node {
    private Coroutine coroutine;
    public override void _Ready() {
        coroutine = CoroutineManager.StartCoroutine(Corroutine1());
        coroutine = CoroutineManager.StartCoroutine(Corroutine2());
        coroutine = CoroutineManager.StartCoroutine(Corroutine3());
    }

    private IEnumerator Corroutine1() {
        GD.Print("Zé da manga");
        //When the return is null, by default the coroutine is executed as
        _Process().
        yield return null;
    }

    private IEnumerator Corroutine2() {
        GD.Print("Zé da manga");
        //When the return is RunTimeSecond the coroutine is executed as _Process()
        with a pre-defined delay.
        yield return new RunTimeSecond(3);
    }

    private IEnumerator Corroutine3() {
        GD.Print("Zé da manga");
        When the return is RunTimeSecond the coroutine is executed as
        _PhysicProcess() with a pre-defined delay.
        yield return new FixedRunTimeSecond(3);
    }
}

```

With the `IYieldVolatile` interface you can switch coroutine execution between `_Process(float)` and `_PhysicsProcess(float)`.

IYield Classes

- `RunTimeSecond` is a framework that allows you to delay your coroutine in seconds. This class inherits `IYieldUpdate`.
- `FixedRunTimeSecond` is a framework that allows you to delay your coroutine in seconds. This class inherits `IYieldFixedUpdate`.
- `IYieldUpdate` is an interface that allows the coroutine to run in the `_Process(float)` function.
- `IYieldFixedUpdate` is an interface that allows the coroutine to run in the `_PhysicsProcess(float)` function.
- `IYieldVolatile` is an interface that allows the coroutine to run in the `Process(float)` or `_PhysicsProcess(float)` function.
- `IYieldCoroutine` is the base interface for Yield interfaces.

Stop coroutines

Now to stop a coroutine.

```
public static void StopCoroutine(Coroutine Coroutine);  
public static void StopAllCoroutines();
```

SerializedPropertyCustom

Now a class has been added for custom serialization of properties in the Godot inspector.

With the `HideProperty` and `ShowProperty` attributes you can serialize properties in the Godot inspector.

Example

Below is an example of usage.

```
public class Exe1 : Node {  
    [ShowProperty] string var1;  
    [ShowProperty] string var2;  
    [ShowProperty] string var3;  
    //The property will not be shown but its value will be saved.  
    [HideProperty] string var4;  
    [ShowProperty] vec2d var5;  
  
    public override GArray _GetPropertyList() =>  
SerializedNode.GetPropertyList(BuildSerialization.Build(this).GetPropertyList());  
    public override bool _Set(string property, object value) =>  
BuildSerialization.Build(this).Set(property, value);  
}
```



```

        public override object _Get(string property) =>
BuildSerialization.Build(this).Get(property);
    }
    [Serializable]
    public struct vec2d {
        //When ShowProperty or HideProperty has its parameter true the value will be saved
in cache.
        [ShowProperty(true)] public float x;
        [ShowProperty(true)] public float y;
    }

```

The [Cobilas Godot Utility](#) is on nuget.org

To include the package, open the `.csproj` file and add it.

```

<ItemGroup>
    <PackageReference Include="Cobilas.Godot.Utility" Version="4.4.1" />
</ItemGroup>

```

Or use command line.

```
dotnet add package Cobilas.Godot.Utility --version 4.4.1
```