Wind Json Parser

Json Unity Parser

Author

Winddy.He

Email: hgplan@126.com

Introduction

So far, I have not found a satisfied Json parser written in C# on the Internet. In order to use very little code to perfectly implement the interconversion between Json string, C# object and json object, the three kinds of commonly used data types, I wrote this.
WindJson is a lightweight json parser, which can parse standard Json format from object to Json or from Json to object,support IOS and Android system.

Feature

- Conversion from Json object to Json string.
- Conversion from Json object to C# object.
- Support parsing standard Json string.
- Support parsing Json string with comments format "//"and "/* */ ".
- Provide a convenient interface to convert from json string to List or Dictionary.
- Provide a lightweight dictionary class, Dict, to help you reduce the capacity of the generated code.

Script Reference

JsonParser

class in WindJson, File in JsonParser.cs.

Description

A main functional interface class, which provides all the interface associated with Json parsing.

Functions

- public JsonParser(string rOriginData);

  A constructor, passing a Json string in a JsonParser object.

- public string PretreatmentProc();

  Preprocessing to remove the comments and some separators (', '\ t', '\ r', '\ n')in the Json string.

- public JsonNode Parser();

  Parsing the incoming Json string to JsonNode object.

Static Functions

- public static JsonNode Parse(string jsonStr);

  Parsing Json string to JsonNode object.

- public static JsonNode ToJsonNode(object rObject);

  A static method parsing a C# object to a JsonNode object.

JsonNode

class in WindJson, File in JsonData.cs

Description

A Json object class, used to represent the logical structure of Json format in the program.

Functions

- public override string ToString();

  Parsing a JsonNode object to a Json string.

- public virtual object ToObject(Type rType);

  Parsing a JsonNode object to a C# object whose type is rType.

- public T ToObject();

  Parsing a JsonNode object to a C# object whose type is T.

- public List ToList();

  Parsing a JsonNode object to a C# array object whose type is List.

- public T[] ToArray();

  Parsing a JsonNode object to a C# array object whose type is T[].

- public Dict<TKey, TValue> ToDict<TKey, TValue>();

  Parsing a JsonNode object to a C# dictionary object whose type is Dict<TKey, TValue>.

- public Dictionary<TKey, TValue> ToDictionary<TKey, TValue>();

  Parsing a JsonNode object to a C# dictionary object whose type is Dictionary<TKey, TValue>.

JsonArray

class in WindJson, File in JsonData.cs, Inherits from JsonNode.

Description

A Json array object.

Functions And Variables

- public JsonNode this[int nIndex]

  Index the nIndexth value of the Json array.

- public int Count;

  The number of elements in the array of Json array object.

  Add an element to the JsonArray.

- public override void Add(JsonNode rItem);

Add an element to the JsonArray.

JsonClass

class in JsonNode, File in Jsondata.cs, Inherits from JsonNode.

Description

A Json class object.

Functions And Variables

- public JsonNode this[string rKey];

  Index the JsonNode object whose keyword is rKey in the JsonClass object.

- public int Count;

  The number of elements in the JsonClass object.

- public void Add(string rKey, JsonNode rItem);

  Add a JsonNode object whose key value is rKey to a JsonClass object.

JsonData

class in WindJson, File in JsonData.cs, Inherits from JsonNode.

Description

AA JsonData object, used to store specific data.

Dict

class in WindJson, File in Dict.cs

Description

A lightweight dictionary class, which encapsulates Dictionary< object object > object.Use the extended syntax feature of C#, encapsulate those uncommonly used method in a static class. In this way, the size of DLL generated by the AOT compiler in IOS can be reduced.

Usage

A Demo.Unity scenes is provided in the project,it is used to illustrate the use of WindJson.

- This is a json string

  Note: Json string is supported parsing the comments format of "//", /**/.

```
{
    "B1": 1,
    "B2": 200000,
    "B3": 123.5,
    "B4": 1234567.899,
    "B5": false,
    "A1":
    [
        {"a":  2.001  },
        {"a":  2.00223}
    ],
    "D": [ 1, 2, 333 ],
    "E":
    {
        "1": [{"a": 233.4 }, {"a": 12233.4   }],
        "2": [{"a": 2233.4}, {"a": 2222233.4 }]
    },
    "F":
    {
        "1": { "11":{ "a": 334.6 }, "22": { "a": 4456.7 } },
        "2": { "21":{ "a":98.8 } }
    }
}
// Support comments
/*{
    "A1":
    [
        {"a":  2.001  }
    ],
    "D": [ 1, 2, 333 ]
}*/
```

- Here is a corresponding C# class.

  Note: Only public attributes and variables will be parsed.

```
public class A
{
    public int      B1 { get; set; }
    public long     B2 { get; set; }
    public float    B3 { get; set; }
    public double   B4 { get; set; }
    public bool     B5 { get; set; }

    public A1[]     A1 { get; set; }

    public List<int>                D;
    public Dictionary<int, A1[]>    E;
    public Dict<int, Dict<int, A1>> F;
}

public class A1
{
    public double a;
}

public class C
{
    public int[] c;
}
```

- Parse Json string to JsonNode

```
JsonParser rJsonParser = new JsonParser(File.ReadAllText(path));
JsonNode rNode = rJsonParser.Parser();
```

- Parse JsonNode to Json string

```
rNode.ToString();
```

- Parse JsonNode to C# object

```
A a = rNode.ToObject(typeof(A)) as A;
```

- Parse C# object to JsonNode

```
JsonNode rJsonNode = JsonParse.ToJsonNode(a);
```

- Any composed JsonNode, here is a example

```
JsonNode rRootNode = new JsonClass();
rRootNode["name"] = new JsonData("Winddy");
rRootNode["age"] = new JsonData(12);
JsonNode rArray = new JsonArray();
rArray.Add(new JsonData("book1"));
rArray.Add(new JsonData("book2"));
rArray.Add(new JsonData("book3"));
rRootNode["books"] = rArray;
Debug.Log(rRootNode.ToString());
```

- If you want to convert from JsonNode to other objects such as List， Array，ToDictionary<TKey, TValue> and ToDict<TKey, TValue>, the corresponding interfaces are also provided.

```
List<A> rLists =  rJsonNode.ToList<A>();
A[] rArrays = rJsonNode.ToArray<A>();
Dict<string, A> rDict1 = rJsonNode.ToDict<string, A>();
Dictionary<string, A> rDict2 = rJsonNode.ToDictionary<string, A>();
```