C# Reflection

In c#, **Reflection** is useful to get the type information that describes assemblies, modules, members, parameters and other entities in the managed code by examining their metadata.

In c#, the **System.Reflection** [namespace](https://www.tutlane.com/tutorial/csharp/csharp-namespaces-with-examples) will contain all the [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) that provide access to the metadata of managed code to get the type information.

We can also use reflection to create type instances dynamically at runtime, bind the type to an existing object or get the type from an existing object and invoke its [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) or access its fields and [properties](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set). In case, if we define any attributes in our code, those also can be accessed by using reflection.

## C# Reflection Example

Following is the example of using reflection to get the [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) object [properties](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) and values in c#.

using System;

using System.Collections.Generic;

using System.Reflection;

namespace TutlaneExamples

{

    class Program

    {

        static void Main(string[] args)

        {

            List<userdetails> items = new List<userdetails>();

            items.Add(new userdetails { userid = 1, username = "suresh", location = "chennai" });

            items.Add(new userdetails { userid = 2, username = "rohini", location = "guntur" });

            items.Add(new userdetails { userid = 3, username = "praveen", location = "bangalore" });

            items.Add(new userdetails { userid = 4, username = "sateesh", location = "vizag" });

            items.Add(new userdetails { userid = 5, username = "madhav", location = "nagpur" });

            items.Add(new userdetails { userid = 6, username = "honey", location = "nagpur" });

            string strmsg = string.Empty;

            foreach (var user in items)

            {

                strmsg = GetPropertyValues(user);

                Console.WriteLine(strmsg);

            }

            Console.ReadLine();

        }

        private static string GetPropertyValues(userdetails user)

        {

            Type type = user.GetType();

            PropertyInfo[] props = type.GetProperties();

            string str = "{";

            foreach (var prop in props)

            {

                str += (prop.Name + ":" + prop.GetValue(user) + ":" + prop.PropertyType.Name) + ",";

            }

            return str.Remove(str.Length - 1) + "}";

        }

    }

    class userdetails

    {

        public int userid { get; set; }

        public string username { get; set; }

        public string location { get; set; }

    }

}

If you observe the above example, we added **System.Reflection** [namespace](https://www.tutlane.com/tutorial/csharp/csharp-namespaces-with-examples) to get the **userdetails** class object [property](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) details. Here, we used a **Type** & **PropertyInfo** classes to get the required object [property](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) details.

Try executing the above program.

If you observe the above result, we are able to get the **userdetails** class object [property](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) details including **name**, **value**, and **type**.

## C# Reflection Overview

Following are the important points which we need to remember about reflection in c#.

* **Reflection** is useful to get the type information that describes assemblies, modules, members, parameters and other entities in the managed code by examining their metadata.
* By reflection, we can create type instances dynamically at runtime, bind or get the type from an existing object and invoke its [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) or access its fields and [properties](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set).
* It provides access to perform [late binding](https://www.tutlane.com/tutorial/csharp/csharp-polymorphism#divcsrunply) and get [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) type information created at runtime.