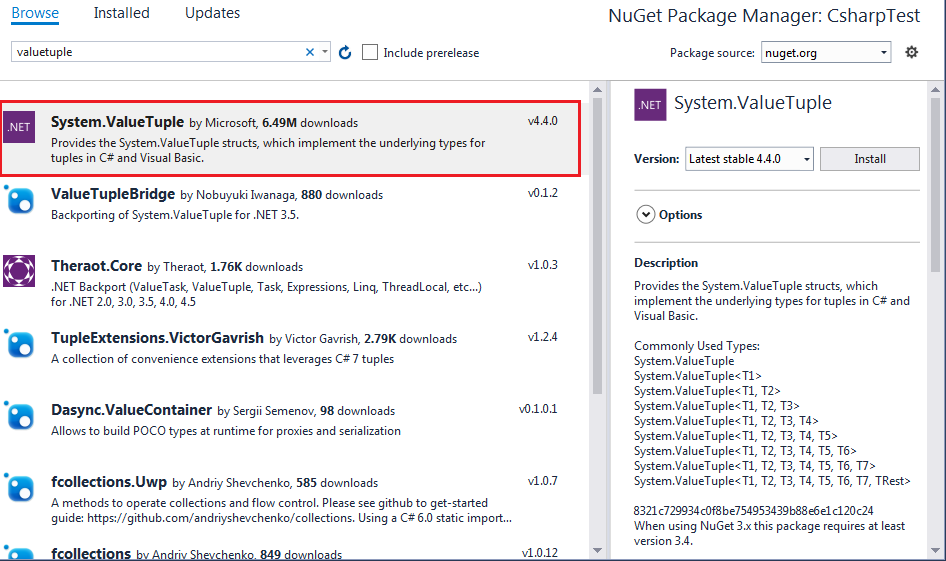
C# - ValueTuple

C# 7.0 (.NET Framework 4.7) introduced ValueTuple, a structure which is a value type representation of the [Tuple](http://www.tutorialsteacher.com/csharp/csharp-tuple).

The ValueTuple is only available in .NET Framework 4.7. If you don't see ValueTuple in your project then you need to install the ValueTuple. (.NET Framework 4.7 or higher, or .NET Standard Library 2.0 or higher already includes ValueTuple.)

To install the ValueTuple package, right click on the project in the solution explorer and select **Manage NuGet Packages..**. This will open the NuGet Package Manager. Click the **Browse** tab, search for ValueTuple in the search box and select the **System.ValueTuple** package, as shown below.

[](http://www.tutorialsteacher.com/Content/images/csharp/install-valuetuple.png)

ValueTuple Initialization

It is easy to create and initialize ValueTuple. It can be created and initialized using parentheses () and specifying the values between them.

var person = (1, "Bill", "Gates");

//equivalent Tuple

//var person = Tuple.Create(1, "Bill", "Gates");

ValueTuple can also be initialized by specifying the type of each element, as shown below.

ValueTuple<int, string,string> person = (1, "Bill", "Gates");

person.Item1; // returns 1

person.Item2; // returns "Bill"

person.Item3; // returns "Gates"

The following is a short way of declaring types for each member.

(int, string, string) person = (1, "Bill", "Gates");

person.Item1; // returns 1

person.Item2; // returns "Bill"

person.Item3; // returns "Gates"

Please notice that we have not used var in the above tuple initialization statement; instead we provided the type of each member values inside the brackets.

Tuple requires at least two values. The following is NOT a tuple.

var number = (1); // int type, NOT a tuple

var numbers = (1,2); //valid tuple

Unlike Tuple, a ValueTuple can include more than eight values.

var numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14);

Named Members:

We can assign names to ValueTuple properties instead of having the default property names Item1, Item2 and so on.

(int Id, string FirstName, string LastName) person = (1, "Bill", "Gates");

person.Id; // returns 1

person.FirstName; // returns "Bill"

person.LastName; // returns "Gates"

We can also assign member names at the right side with values, as below.

var person = (Id:1, FirstName:"Bill", LastName: "Gates");

Please note that we can provide member names either at the left side or at the right side but not at both side. The left side has precedence over the left side. The following will ignore names at the right side.

// PersonId, FName, LName will be ignored.

(int Id, string FirstName, string LastName) person = (PersonId:1, FName:"Bill", LName: "Gates");

// PersonId, FirstName, LastName will be ignored. It will have the default names: Item1, Item2, Item3.

(string, string, int) person = (PersonId:1, FName:"Bill", LName: "Gates");

We can also assign variables as member values.

string firstName = "Bill", lastName = "Gates";

var per = (FirstName: firstName, LastName: lastName);

ValueTuple as Return Type:

The following method returns a ValueTuple.

static void Main(string[] args)

{

DisplayTuple(1, "Bill", "Gates");

}

static void DisplayTuple((int, string, string) person)

{

Console.WriteLine($"Id = { person.Item1}");

Console.WriteLine($"First Name = { person.Item2}");

Console.WriteLine($"Last Name = { person.Item3}");

}

We can also specify different member names for a ValueTuple returned from a method.

static void Main(string[] args)

{

var person = GetPerson();

}

static (int, string, string) GetPerson()

{

return (Id:1, FirstName: "Bill", LastName: "Gates");

}

Deconstruction

Individual members of a ValueTuple can be retrieved by deconstructing it. A deconstructing declaration syntax splits a ValueTuple into its parts and assigns those parts individually to fresh variables.

static void Main(string[] args)

{

// change property names

(int PersonId, string FName, string LName) person = GetPerson();

}

static (int, string, string) GetPerson()

{

return (Id:1, FirstName: "Bill", LastName: "Gates");

}

We can also use var instead of explicit data type names.

static void Main(string[] args)

{

// use var as datatype

(var PersonId, var FName, var LName) person = GetPerson();

}

static (int, string, string) GetPerson()

{

return (Id:1, FirstName: "Bill", LastName: "Gates");

}

ValueTuple also allows "discards" in deconstruction for the members you are not going to use.

// use discard \_ for the unused member LName

(var id, var FName, \_) = GetPerson();