C# - Tuple

The Tuple<T> class was introduced in .NET Framework 4.0. A tuple is a data structure that contains a sequence of elements of different data types. It can be used where you want to have a data structure to hold an object with properties, but you don't want to create a separate type for it.

Tuple<T1, T2, T3, T4, T5, T6, T7, TRest>

The following example creates a tuple with three elements:

Tuple<int, string, string> person =

new Tuple <int, string, string>(1, "Steve", "Jobs");

In the above example, we created an instance of the Tuple which holds a record of a person. We specified a type for each element and passed values to the constructor. Specifying the type of each element is cumbersome. So, C# includes a static helper class Tuple which returns an instance of the Tuple<T> without specifying the type of each element, as shown below.

var person = Tuple.Create(1, "Steve", "Jobs");

A tuple can only include maximum eight elements. It gives a compiler error when you try to include more than eight elements.

var numbers = Tuple.Create(1, 2, 3, 4, 5, 6, 7, 8);

Accessing Tuple Elements

The elements of a tuple can be accessed with Item<elementNumber> properties e.g. Item1, Item2, Item3 and so on up to Item7 property. The Item1 property returns the first element, Item2 returns the second element and so on. The last element (the 8th element) will be returned using the Restproperty.

Example: Accessing Tuple Elements

var person = Tuple.Create(1, "Steve", "Jobs");

person.Item1; // returns 1

person.Item2; // returns "Steve"

person.Item3; // returns "Jobs"

var numbers = Tuple.Create("One", 2, 3, "Four", 5, "Six", 7, 8);

numbers.Item1; // returns "One"

numbers.Item2; // returns 2

numbers.Item3; // returns 3

numbers.Item4; // returns "Four"

numbers.Item5; // returns 5

numbers.Item6; // returns "Six"

numbers.Item7; // returns 7

numbers.Rest; // returns (8)

numbers.Rest.Item1; // returns 8

Generally, the 8th position is for the nested tuple which you can access using the Rest property.

Nested Tuples

If you want to include more than eight elements in a tuple, you can do that by nesting another tuple object as the eighth element. The last nested tuple can be accessed using the Rest property. To access the nested tuple's element, use the Rest.Item1.Item<elelementNumber> property.

Example: Nested Tuple

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

numbers.Item1; // returns 1

numbers.Item7; // returns 7

numbers.Rest.Item1; //returns (8, 9, 10, 11, 12, 13)

numbers.Rest.Item1.Item1; //returns 8

numbers.Rest.Item1.Item2; //returns 9

[Try it](http://www.tutorialsteacher.com/codeeditor?cid=cs-BxE3wy)

You can include the nested tuple object anywhere in the sequence. However, it is recommended to place the nested tuple at the end of the sequence so that it can be accessed using the Rest property.

Example: Nested Tuple

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

numbers.Item1; // returns 1

numbers.Item2; // returns 2

numbers.Item3; // returns (3, 4, 5, 6, 7, 8)

numbers.Item3.Item1; // returns 3

numbers.Item4; // returns 9

numbers.Rest.Item1; //returns 13

[Try it](http://www.tutorialsteacher.com/codeeditor?cid=cs-0zm3HR)

Tuple as a Method Parameter

A method can have a tuple as a parameter.

static void Main(string[] args)

{

var person = Tuple.Create(1, "Steve", "Jobs");

DisplayTuple(person);

}

static void DisplayTuple(Tuple<int,string,string> person)

{

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

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

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

}

Tuple as a Return Type

A Tuple can be return from a method.

static void Main(string[] args)

{

var person = GetPerson();

}

static Tuple<int, string, string> GetPerson()

{

return Tuple.Create(1, "Bill", "Gates");

}

Usage of Tuple

Tuples can be used in the following scenarios:

1. When you want to return multiple values from a method without using ref or out parameters.
2. When you want to pass multiple values to a method through a single parameter.
3. When you want to hold a database record or some values temporarily without creating a separate class.

Tuple Limitations:

1. Tuple is a reference type and not a value type. It allocates on heap and could result in CPU intensive operations.
2. Tuple is limited to include 8 elements. You need to use nested tuples if you need to store more elements. However, this may result in ambiguity.
3. Tuple elements can be accessed using properties with a name pattern Item<elementNumber> which does not make sense.

C# 7 includes ValueTuple to overcome the limitations of Tuple and also makes it even easier to work with Tuple. Learn about it in the next chapter.