Tuples and Deconstruction

Filip Ekberg
Principal Consultant & CEO

@fekberg | fekberg.com

Tuple

“The tuples feature provides concise syntax to group multiple data elements in a lightweight data structure“

Example:

```
var group = (order.OrderNumber, order.LineItems);
```

docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-tuples

Tuples are **strongly typed**

Creating a Tuple

Creating a Tuple

```
var summary = (order.OrderNumber, order.Total);
```

Creating a Tuple

```
var summary = (order.OrderNumber, order.Total);
```

Tuple vs Anonymous Type

Tuple

Value type

Values stored in fields

Not immutable

Anonymous Type

Reference type

Values stored in properties with backing fields

Read-only

Tuple field **names** only **tracked by** the **compiler**

Tuples are **not** read-only

Creating and Using a Tuple

```
// Tuple assignment
var summary = (order.OrderNumber,
               order.Items,
               Sum: GetSumFor(order));
```

Creating and Using a Tuple

```
// Tuple assignment
var summary = (order.OrderNumber,
               order.Items,
               Sum: GetSumFor(order));
```

```
ValueTuple<Guid, IEnumerable<Item>, decimal>
```

Creating and Using a Tuple

```
(Guid, IEnumerable<Item>, decimal) summary = (order.OrderNumber,
                                              order.Items,
                                              Sum:
GetSumFor(order));
```

Tuple Assignment

Tuple Assignment

```
var summary = (order.OrderNumber, order.Items, Sum: GetSumFor(order));
```

Tuple Assignment

```
var summary = (order.OrderNumber, order.Items, Sum: GetSumFor(order));

(Guid, IEnumerable<Item>, decimal)
    summary = (order.OrderNumber, order.Items, Sum: GetSumFor(order));
```

Tuple Assignment

```
(Guid, IEnumerable<Item>, decimal) summary = (OrderNumber, Items, Sum);
```

Tuples support **different types** of **assignments** and **deconstruction**

The order of the elements is important!

The right hand side is assigned into the corresponding position on the left

Tuples can be deconstructed!

This is a much **more flexible approach than** what we saw with the **anonymous type**

Tuple Deconstruction

```
Guid orderNumber;
decimal sum;

(orderNumber, var items, sum) = (OrderNumber, Items, Sum);
```

Tuple Deconstruction

```
Guid orderNumber;
decimal sum;

(orderNumber, var items, sum) = (OrderNumber, Items, Sum);
```

The order and number of elements must match!

Tuples in C#

Powerful language feature

Deconstruction is useful in many situations

Converting an Anonymous Type to a Tuple

Converting an Anonymous Type to a Tuple

```
var group = new { order.OrderNumber, order.Total, Sum = order.Sum };

var group = (order.OrderNumber, order.Total, Sum: order.Sum);
```

Converting an Anonymous Type to a Tuple

```
var group = new { order.OrderNumber, order.Total, Sum = order.Sum };

var group = (order.OrderNumber, order.Total, Sum: order.Sum);
```

Returning a Tuple

Returning a Tuple

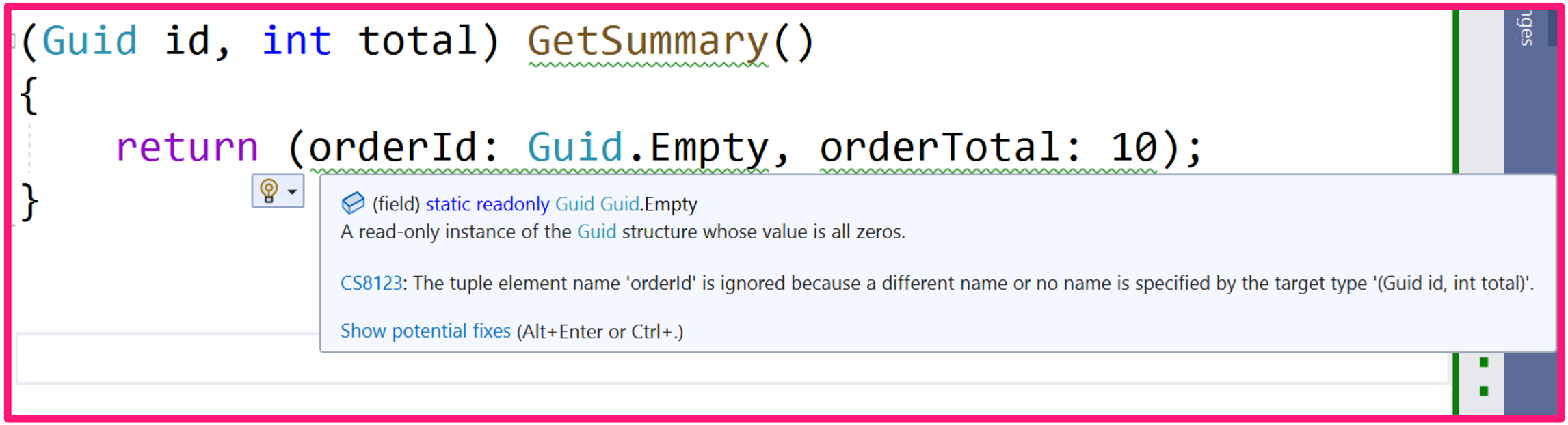
```
public (Guid, int, decimal, IEnumerable<Item>) Process(...)
{
    return (order.OrderNumber, amountOfItems, GetTotalFor(order), items);
}
```

Returning a Tuple

```
public (Guid, int, decimal, IEnumerable<Item>) Process(...)
{
    return (order.OrderNumber, amountOfItems, GetTotalFor(order), items);
}
```

When Elements May Be Ignored

Only when explicitly defining a new tuple that is returned



Tuples can be **parameters** to **methods** as well

Can you create an extension method for a tuple?

Yes!

The **extension method** will work for **all tuples** with a **matching sequence** of **elements**

Data binding?

Stick to anonymous types!

Deconstruction and Pattern Matching

```
if(order is (total: > 100, ready: true))
{

}
```

Deconstruction and Pattern Matching

```
if(order is (total: > 100, ready: true))
{

}
```

Deconstruct the Order

Deconstruct the Order

```
var order = new Order();
```

Deconstruct the Order

```
var order = new Order();

var (total, isReady) = order;
```

Introducing the Deconstruct Method

Introducing the Deconstruct Method

```
public void Deconstruct()
{
```

Introducing the Deconstruct Method

```
public void Deconstruct(out decimal total, out bool ready)
```

Introducing the Deconstruct Method

```
public void Deconstruct(out decimal total, out bool ready)
{
    total = Total;
    ready = IsReadyForShipment;
}
```

The result is not a tuple!

Deconstructing an object is always done into separate variables

Deconstruction

```
var (total, isReady) = order;
```

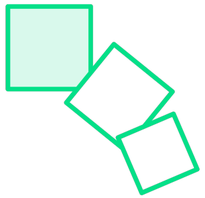
```
decimal total2;
bool ready;
order.Deconstruct(out total2, out ready);
decimal total = total2;
bool isReady = ready;
```

You can have multiple deconstruct methods

They need to have a different number of parameters!

You can create an extension method that deconstructs an object

Tuple

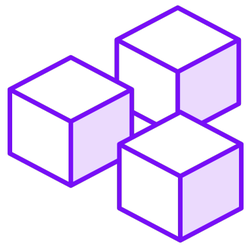


A sequence of elemenets

```
var summary = (orderNumber, total, items)
```

Support named parameters

```
(id: GetId(), total: GetTotal(), items: GetItems())
```



Compiles to a generic struct

```
System.ValueTuple<T1,T2,T3,...>
```

Tuple Assignment

Tuple Assignment

```
var summary = (order.OrderNumber,
               order.Items,
               Sum: GetSumFor(order));
```

Tuple Deconstruction

Tuple Deconstruction

```
(Guid orderNumber, IEnumerable<Item> items, decimal sum) summary =
(order.OrderNumber, order.Items, GetSumFor(order));
```

Tuple Deconstruction

```
(Guid orderNumber, IEnumerable<Item> items, decimal sum) summary =
(order.OrderNumber, order.Items, GetSumFor(order));
```

```
var (orderNumber, items, sum) summary =
(order.OrderNumber, order.Items, GetSumFor(order));
```

Tuple Deconstruction into Existing Variables

```
Guid orderNumber;
IEnumerable<Item> items;
decimal sum;

(orderNumber, items, sum) summary =
(order.OrderNumber, order.Items, GetSumFor(order));
```

Discard

```
(orderNumber, items, _) summary = Process();
```

Discard

```
(orderNumber, items, _) summary = Process();
```

If this **returns** a **tuple** with **3 elements** it **has to be assigned** to a tuple with a **matching number** of **elements**

Tuples as Return Types

Tuples as Return Types

```
(Guid id, int total) GetSummary() => (order.Id, order.Total)

var (orderNumber, total) = GetSummary();
```

Tuples as Return Types

```
(Guid id, int total) GetSummary() => (order.Id, order.Total)

var (orderNumber, total) = GetSummary();

var (total , orderNumber) = GetSummary();
```

The id is stored in the local variable total
The order is important!

Deconstruct any type!

Just add a deconstruct method to the type or as an extension method

Deconstruct

Deconstruct

```
public class Order
{




}
```

Deconstruct

```
public class Order
{
    public void Deconstruct(                                    )
    {


    }
}
```

Deconstruct

```
public class Order
{
    public void Deconstruct(out decimal total, out bool ready)
    {
        total = Total;
        ready = IsReadyForShipment;
    }
}
```

Deconstruct

```
public class Order
{
    public void Deconstruct(out decimal total, out bool ready)
    {
        total = Total;
        ready = IsReadyForShipment;
    }
}

var order = new Order();
var (orderTotal, isReady) = order;
```

Deconstruct can be an extension method

Example: KeyValuePair<TKey, TValue>

Example: KeyValuePair<TKey, TValue>

```
var dictionary = new Dictionary<string, Order>();

foreach (var (orderId, theOrder) in dictionary)
{

}
```

Example: KeyValuePair<TKey, TValue>

```
var dictionary = new Dictionary<string, Order>();

foreach (var (orderId, theOrder) in dictionary)
{

}
```

The **KeyValuePair** implements a **deconstruct** that supports this

Up Next:

Pattern Matching