Nullable Reference Types

Filip Ekberg
Principal Consultant & CEO

@fekberg fekberg.com

What if **null** never existed?

Nullable Reference Types

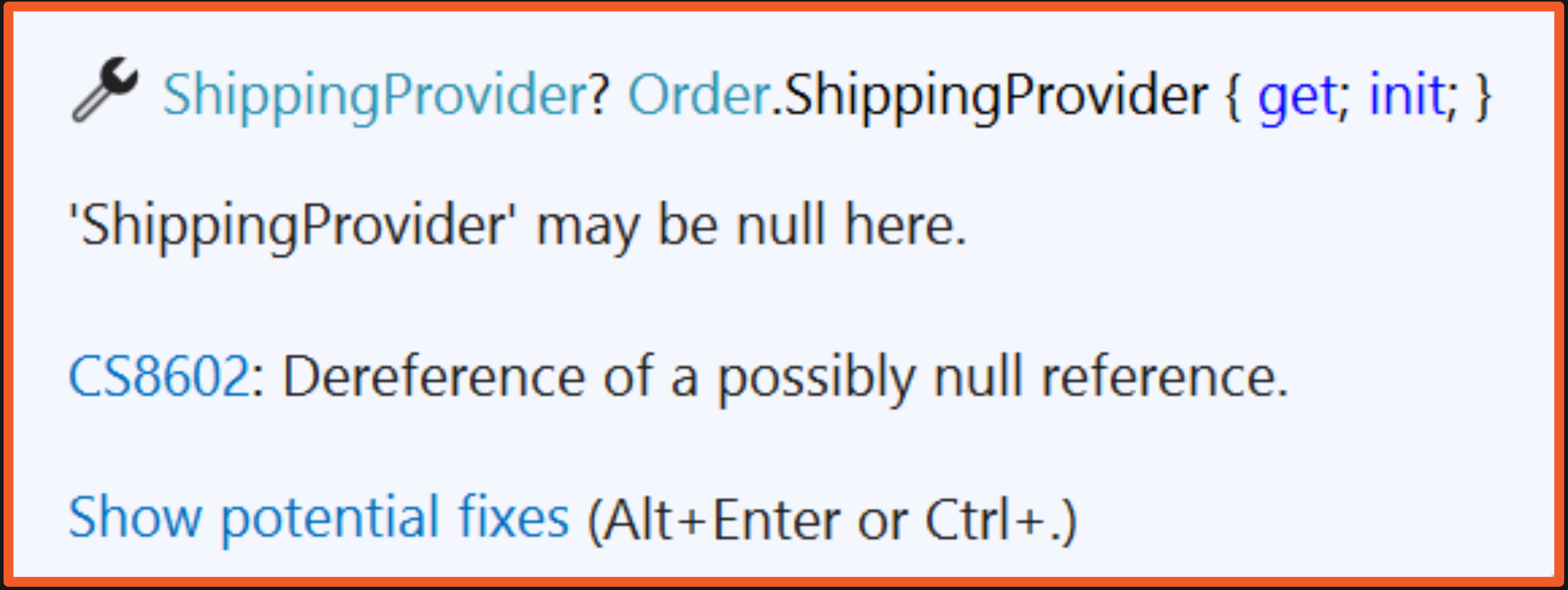
```
Order order = new()
{
    ShippingProvider = null
};

var providerName = order.ShippingProvider.Name;
```

Nullable Reference Types

```
Order order = new()
{
    ShippingProvider = null
};

var providerName = order.ShippingProvider.Name;
```



ShippingProvider? Order.ShippingProvider { get; init; }

'ShippingProvider' may be null here.

CS8602: Dereference of a possibly null reference.

Show potential fixes (Alt+Enter or Ctrl+.)

What does it mean to return null?

Returning Null Is Ambiguous

```
class OrderProcessor
{
    public IEnumerable<Summary> Process(IEnumerable<Order> orders)
    {
        return null;
    }
}
```

Method Returns **null**: What Does That Mean?

Unable to process?

Unexpected issues?

Everything went fine but a summary was not necessary

It’s impossible for the caller of the method to know the intent!

Return a Default Value Instead

```
class OrderProcessor
{
    public IEnumerable<Summary> Process(IEnumerable<Order> orders)
    {
        return Enunmerable.Empty<Summary>();
    }
}
```

Just a matter of time before you run into a **null reference exception**

Nullable Reference Types

Find potential problems!

Highlights where you need to guard against nulls, or where you are using nulls and should not

Now the default behaviour!

All reference types are looked at as non-nullable

Are we safe from nulls?

No.

There is **definitely** a **benefit** to **enable** this when it **builds** on a **build server**

Nullable Reference Type

```
record Customer(Address address);
record Customer(Address? address);

Customer   customer = null;
Customer?  customer = null;
```

Not allowed!
Produces a warning!

Allowed to be null

Allow Null

```
record Customer(Address? address);
```

Anyone using this need
to make sure it is not null

Can you **find good solutions** to the rest of the potential **null reference exceptions**?

The **compiler** has **no idea** about if the method performs a **null check**

Refactor your code **one** small **portion at a time**

Enable or Disable **Nullable Reference Types**

Enable or Disable **Nullable Reference Types**

```
class OrderProcessor
{
    #nullable enable
    public IEnumerable<Summary> Process(IEnumerable<Order> orders)
    {
        return null;
    }
}

#nullable disable
OrderProcessor orderProcessor = null;
```

Enable or Disable **Nullable Reference Types**

```
class OrderProcessor
{
    #nullable enable
    public IEnumerable<Summary> Process(IEnumerable<Order> orders)
    {
        return null;
    }
}

#nullable disable
OrderProcessor orderProcessor = null;
```

Null Conditional Operator

```
var report = order?.GenerateReport();
```

Let’s **explore more** ways to work with **nulls** and **avoid null** reference exceptions

Handling Nulls

Handling Nulls

```
// null coalescing operator
Order? order = GetOrder() ?? new();
```

Handling Nulls

```
// null coalescing operator
Order? order = GetOrder() ?? new();

// null coalescing assignment
order ??= new();
```

Handling Nulls

```
// null coalescing operator
Order? order = GetOrder() ?? new();

// null coalescing assignment
order ??= new();

// null forgiving operator
order!.GenerateReport();
```

The compiler will **not always** be able to **determine** the **null state**

Can we **hint** to the compiler that **we know** for certain that it **is** an **instance**?

If your instance is in fact null you'd **end up with** a **null reference exception**

We Checked for Nulls

```
bool ValidateShippingProvider(
      ShippingProvider? provider) { ... }
```

Attributes for Null State Static Analysis

```
bool ValidateShippingProvider(
        [NotNullWhen(true)]
        ShippingProvider? provider) { ... }
```

Attributes for Null State Static Analysis

```
bool ValidateShippingProvider(
        [NotNullWhen(true)]
        ShippingProvider? provider) { ... }
```

Aim to **avoid null** and nullable reference types!

Aim to avoid null and nullable reference types!

In reality that is unlikely to always be easy!

```
AllowNull          NotNullIfNotNull

DisallowNull       MemberNotNull

MaybeNull          MemberNotNullWhen

NotNull            DoesNotReturn

MaybeNullWhen      DoesNotReturnIf

NotNullWhen
```

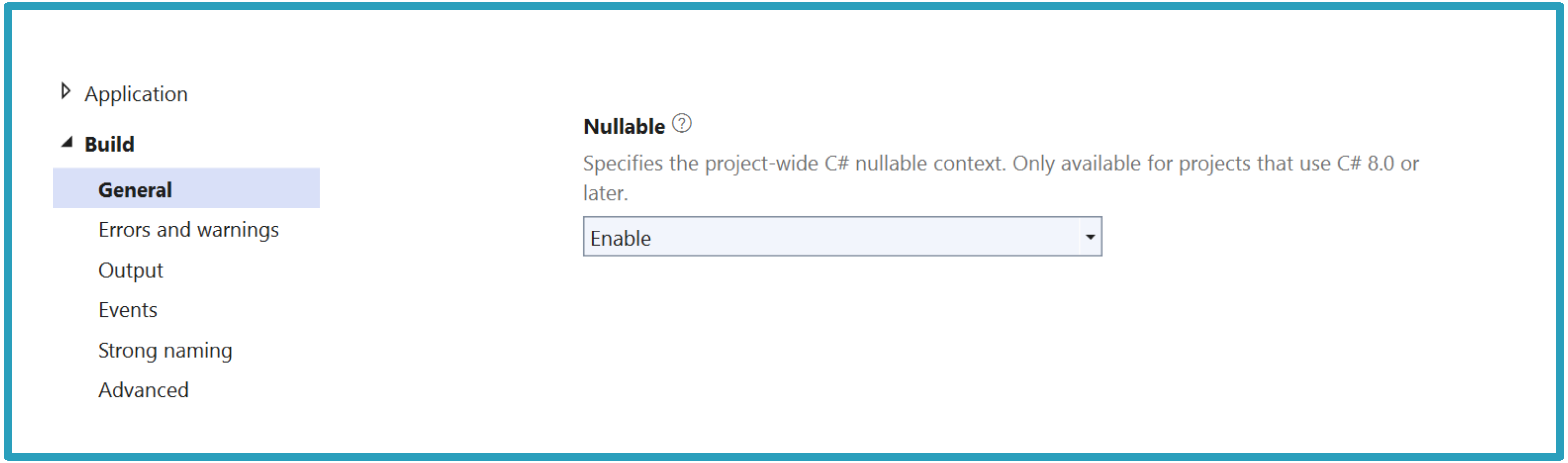
System.Diagnostics.**CodeAnalysis**

Instruct the compiler how you handle, or do not handle, nulls

We can **allow null** for a **default parameter** and **promise** that we **handle it** accordingly

Use these **attributes** to **communicate** the **intent** when working with null

Enable Nullable Reference Types

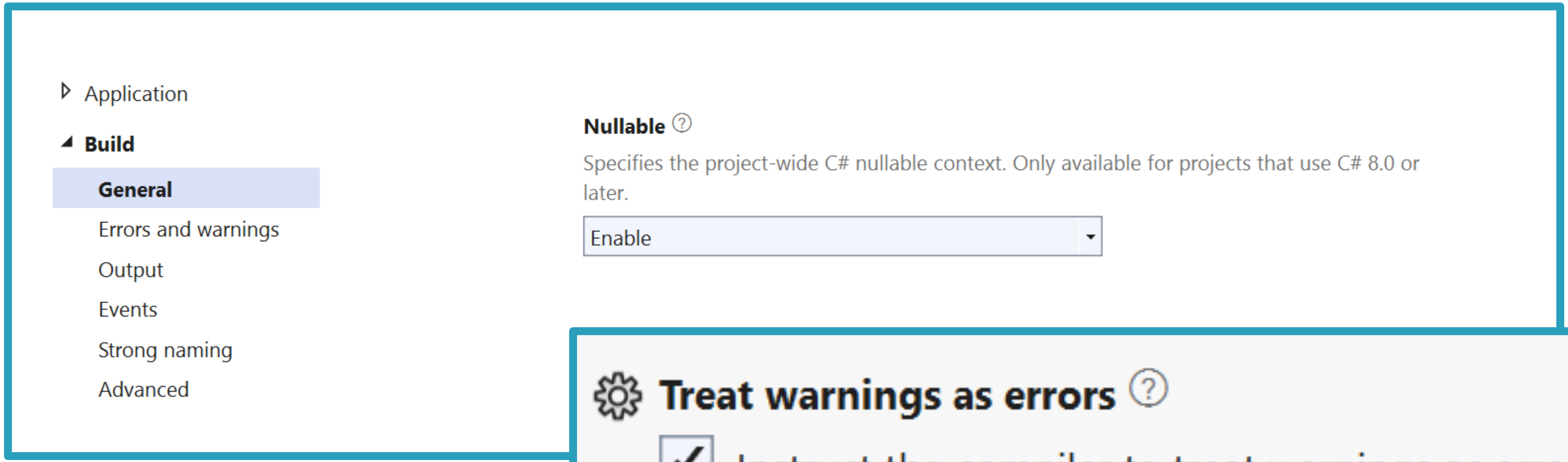


Nullable

Specifies the project-wide C# nullable context. Only available for projects that use C# 8.0 or later.

Enable

Treat Warnings as Errors



Application

Build

General

Errors and warnings

Output

Events

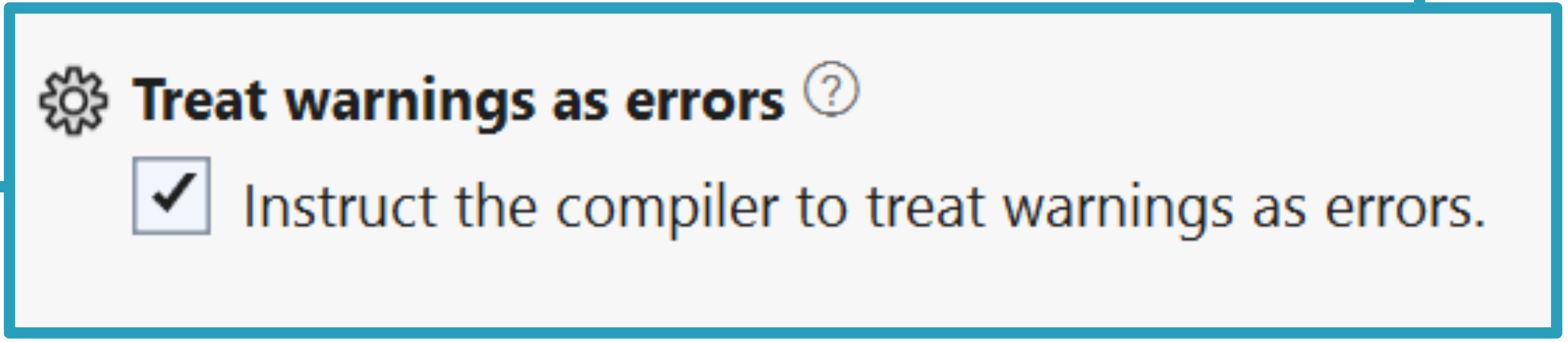
Strong naming

Advanced

Nullable

Specifies the project-wide C# nullable context. Only available for projects that use C# 8.0 or later.

Enable



Treat warnings as errors

✓ Instruct the compiler to treat warnings as errors.

Explicitly **Allowing Nulls**

```
Order? order = null;
```

Explicitly **Allowing Nulls**

```
Order? order = null;
```

Allowed to be null

Requires **Null Guards**

```
Order? order = GetOrder();

if(order is not null)
{
    order.GenerateReport();
}

order?.GenerateReport();
```

The **compiler** will **track** the **null state** and **warn you** if it determines if it may be null

Compiler **Warnings**

```
Order? order = GetOrder();

order.GenerateReport();
```

Order **may be null** and you should consider **adding a null check**

Null Forgiving Operator

Null Forgiving Operator

```
Order? order = GetOrder();

if(ValidateOrder(order))
{
    order!.GenerateReport();
}
```

Null Forgiving Operator

```
Order? order = GetOrder();

if(ValidateOrder(order))
{
    order!.GenerateReport();
}
```

Instruct the **compiler** that a **null check** have been **made**

It would have been appropriate to **use the attribute** for **null state** tracking

ArgumentNullException.ThrowIfNull

```
Order? order = GetOrder();

ArgumentNullException.ThrowIfNull(order);

order.GenerateReport();
```

No warning because ThrowIfNull uses [NotNullWhen]

Next: **Indexers, Ranges, and Indices**