

# Nullable reference types – advanced guide

Дятлов Андрей

C# Developer, JetBrains

# Обо мне

- Занимаюсь поддержкой языка C# в ReSharper с 2015 года
  - Анализаторы кода, рефакторинги
  - Поддержка новых версий языка
- Ищу баги в Roslyn

- 



**Julien Couvreur**

@jcouv

Still trying to catch up on the series of nullable issues filed by @a\_tessenr  
They're interesting cases, clear and detailed.  
Thanks!

[github.com/dotnet/roslyn/](https://github.com/dotnet/roslyn/) ...

# План доклада

- Краткое описание nullable reference types
- Способы постепенного перевода проекта на их использование
- Взаимодействие с обобщенным кодом
- Аннотации для помощи компилятору
- Что делать если компилятор не прав?
- Common pitfalls
- Warnings as errors

# Что такое nullable reference types?

```
class Employee
{
    public string Name { get; }
    public string Surname { get; }
    public DateTime? Birthday { get; }

    public Employee(string name, string surname)
        => (Name, Surname) = (name, surname);
}
```

# Что такое nullable reference types?

```
class Employee
{
    public DateTime? Birthday { get; }
```

```
public bool HasBirthdayToday()
    => Birthday.HasValue
        && Birthday.Value.Day == DateTime.Today.Day
        && Birthday.Value.Month == DateTime.Today.Month;
```

```
public bool HasBirthdayToday()
    => Birthday?.Day == DateTime.Today.Day
        && Birthday?.Month == DateTime.Today.Month;
```

```
}
```

# Что такое nullable reference types?

```
public class Employee
{
    public string Name { get; }    // do consumers check nulls?
    public string Surname { get; } // do consumers check nulls?
    public DateTime? Birthday { get; }

    public Employee(string name, string surname) // usages that pass nulls?
        => (Name, Surname) = (name, surname);

    public string GetInitials() => $"{Name[0]}. {Surname[0]}.";
}
```

# Что такое nullable reference types?

```
public class Employee
{
    public string Name { get; }
    public string? Surname { get; }
    public DateTime? Birthday { get; }

    public Employee(string name, string? surname)
        => (Name, Surname) = (name, surname);

    public string GetInitials() => $"{Name[0]}. {Surname[0]}.";
}
```



🔧 `string? Employee.Surname { get; }`

'Surname' may be null here.

Dereference of a possibly null reference.

[Show potential fixes \(Ctrl+.\)](#)

# Как это выражено в IL?

```
.method public hidebysig instance string
  GetInitials() cil managed
{
  .maxstack 8

  IL_0000: ldstr      "{0}. {1}."
  IL_0005: ldarg.0     // this
  IL_0006: call       instance string Employee::get_Name()
  IL_000b: ldc.i4.0
  IL_000c: callvirt     instance char [System.Runtime]System.String::get_Chars(int32)
  IL_0011: box         [System.Runtime]System.Char
  IL_0016: ldarg.0     // this
  IL_0017: call       instance string Employee::get_Surname()
  IL_001c: ldc.i4.0
  IL_001d: callvirt     instance char [System.Runtime]System.String::get_Chars(int32)
  IL_0022: box         [System.Runtime]System.Char
  IL_0027: call       string [System.Runtime]System.String::Format(string, object, object)
  IL_002c: ret

} // end of method Employee::GetInitials
```



# Что такое nullable reference types?

```
public class Employee
{
    public string Name { get; }
    public string? Surname { get; }
    public DateTime? Birthday { get; }

    public Employee(string name, string? surname)
        => (Name, Surname) = (name, surname);

    public string GetInitials() => $"{Name[0]}. {Surname?[0]}.";
}
```

# В чем отличие от Nullable<T>?

## **Nullable<T>**

- Специальный тип
- Явное получение значения с помощью `.Value``
- Null-значение проверяется в рантайме

## **Nullable reference types**

- Аннотация в системе типов
- Неявное получение значения
- Только compile-time предупреждения

# Преимущества перед другими аннотациями (например, JetBrains.Annotations)

- `[CanBeNull] string? element;`
- `[NotNull, ItemCanBeNull] IList<string?> collection;`
- Аннотация является частью типа, а не атрибутом
  - `[???] IList<IList<string?>> nestedCollection;`
  - `[???] GenericType<string?, IList<string>> complexType;`

# Преимущества перед другими аннотациями (например, JetBrains.Annotations)

- Можно использовать везде где используется тип
  - `string?` `localVariable`;
  - `class MyStringCollection : IList<string?>`
  - `class C<T> where T : IList<string?>`

# И это все?



**Andy Gocke**  
agocke

Dev on the C# compiler

 Microsoft

 Seattle, WA

Organizations



**Andy Gocke**

@andygocke

Follow



In case anyone was wondering if nullable reference types in C# was an expensive feature, my rough estimate is 15 dev-years (~10 devs full-time for 1.5 years)

1:28 pm - 17 Jul 2019

**17** Retweets **90** Likes



 16

 17

 90



<https://bit.ly/2INzOGj>

# Включаем и пользуемся!

```
string GetStringWithNumber(string input,  
                           bool allowUserInput)  
{  
    input ??= allowUserInput  
        ? GetUserInput().StringData  
        : null;  
    return int.TryParse(input, out _) ? input : null;  
}  
  
UserInputData GetUserInput() { /* ... */ }
```

# Включаем и пользуемся!



```
string GetStringWithNumber(string input,
                           bool allowUserInput)
{
    input ??= allowUserInput
              ? GetUserInput().StringData
              : null;
    return int.TryParse(input, out _) ? input : null;
}
```



 **class** System.String

Represents text as a sequence of UTF-16 code units.

Converting null literal or possible null value to non-nullable type.

[Show potential fixes](#) (Ctrl+.)



 **class** System.String

Represents text as a sequence of UTF-16 code units.

Possible null reference return.

[Show potential fixes](#) (Ctrl+.)


```
UserInputData GetUserInput() { /* ... */ }
```




# Включаем и пользуемся!

```
string? GetStringWithNumber(string? input,  
                             bool allowUserInput)  
{  
    input ??= allowUserInput  
              ? GetUserInput().StringData  
              : null;  
    return int.TryParse(input, out _) ? input : null;  
}  
  
UserInputData GetUserInput() { /* ... */ }
```



# Включаем и пользуемся!


 [JamesNK](#) / [Newtonsoft.Json](#)


 Watch 526  Star 7,294  Fork 2,468

[Code](#) [Issues 323](#) [Pull requests 35](#) [Projects 0](#) [Security](#) [Insights](#)


## Update Newtonsoft.Json project to use nullable reference types (#1950)

Browse files

 master (#1950)

 **JamesNK** committed on 29 Jul Verified

1 parent [f940f21](#)    commit [cdf10151d507d497a3f9a71d36d544b199f73435](#)

 Showing **169 changed files** with 2,265 additions and 1,860 deletions.

[Unified](#) [Split](#)



<https://bit.ly/2IOBY8o>

Я не готов переписывать весь проект!  
Что мне делать?

- Хочу включать анализ гранулярно
- Хочу чтобы компилятор мне помог без дополнительных усилий
- Пишу библиотеку и хочу только проаннотировать ее для пользователей

# Предупреждения компилятора без лишних усилий

```
#nullable enable warnings
public static Transaction Create(IClient client, TransactionInfo info) {
    var transaction = new Transaction();
    if (client.Address?.Country == Countries.Russia) { /* ... */ }

    // ...

    if (info.RequiredFields.HasPostIndex()) {
        transaction.SenderInfo.PostIndex = client.Address.FindPostIndex();
    }

    // ...
    return transaction;
}
```

# Откуда компилятор узнает что нужно предупреждение?

- Присвоение `null`
  - `variable = null;`
- Проверка переменной на `null`
  - `if (variable != null) { /* ... */ }`
  - `variable?.DoSomething();`
- Аннотация типа переменной
  - `string? nullableVar;`

# В чем отличие от включения всей фичи?

```
#nullable enable
```

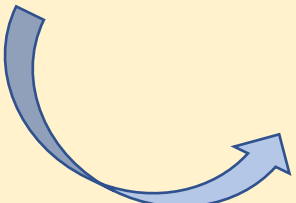
```
public void NullableEnabled(string variable) {  
    variable.ToString();  
    variable = null;  
}
```



**string:[NotNull]**

```
#nullable enable warnings
```

```
public void NullableWarningsOnly(string variable) {  
    variable.ToString();  
    variable = null;  
}
```



**string:[Oblivious]**

# Просто аннотируем библиотеку

```
public interface IClient {  
    string FirstName { get; }  
    string LastName { get; }  
    string MiddleName { get; }  
  
    Address Address { get; }  
    PassportInfo Passport { get; }  
    PhoneNumber ContactPhone { get; }  
    string ContactEmail { get; }  
  
    Notification SendNotification(Transaction transaction, string message);  
}
```

# Просто аннотируем библиотеку

```
#nullable enable annotations
```

```
public interface IClient {  
    string FirstName { get; }  
    string LastName { get; }  
    string? MiddleName { get; }  
  
    Address? Address { get; }  
    PassportInfo Passport { get; }  
    PhoneNumber ContactPhone { get; }  
    string? ContactEmail { get; }  
  
    Notification? SendNotification(Transaction? transaction, string message);  
}
```

# Просто аннотируем библиотеку

```
[NullableContext(1)] public interface IClient {  
    string FirstName { get; }  
    string LastName { get; }  
    [Nullable(2)] string MiddleName { [NullableContext(2)] get; }  
  
    [Nullable(2)] Address Address { get; }  
    PassportInfo Passport { get; }  
    PhoneNumber ContactPhone { get; }  
    [Nullable(2)] string ContactEmail { [NullableContext(2)] get; }  
  
    [Nullable(2)] Notification SendNotification(  
        [Nullable(2)] Transaction transaction, string message);  
}
```



# Включаем анализ на части проекта

Настройка в проекте (.csproj)  
<Nullable>enable</Nullable>

Отдельные файлы \ классы \ методы  
#nullable enable  
#nullable disable

```
#nullable disable
```

```
[Obsolete("JSON Schema validation has been  
moved to its own package. ...")]  
public class JsonValidatingReader  
    : JsonReader, IJsonLineInfo  
{  
    // more than 1000 lines of code here  
}
```



<https://bit.ly/33v96Kg>

# План доклада

- Краткое описание nullable reference types
- Способы постепенного перевода проекта на их использование
- **Взаимодействие с обобщенным кодом**
- Аннотации для помощи компилятору
- Что делать если компилятор не прав?
- Common pitfalls
- Warnings as errors

# Отношения между типами с Nullable Reference Types

- `string:[NotNull] <: string?`

`string? = { string:[NotNull], null }`

`var nullableArray = new[] { nullableStr, nonNullableStr }`



`string?[]`



`string?`



`string:[NotNull]`

# Отношения между типами с Nullable Reference Types

- `string:[NotNull] <: string?`

```
T ChooseOne<T>(T t1, T t2);
```

```
var nullableStr = ChooseOne(nullableStr, nonNullableStr);
```



`string?`



`string?`



`string:[NotNull]`

# Отношения между типами с Nullable Reference Types

- `string:[Oblivious] <=> string:[NotNull]`

```
var notNullableString = NotAnnotatedMethod();
```



`string:[NotNull]`



`string:[Oblivious]`

```
Console.WriteLine(notNullableString.Length) // OK
```

```
notNullableString = null;
```



`IEnumerable<string>` - подтип `IEnumerable<string?>`

```
void Run() {  
  
    IEnumerable<string> nonNullableStrings = new[] {"notNull", "string"};  
    IEnumerable<string?> nullableStrings = new[] {null, "string"};  
  
    nullableStrings = nonNullableStrings; // OK  
    nonNullableStrings = nullableStrings; // warning  
}
```

`IEnumerable<string>` - подтип `IEnumerable<string?>`

```
void AcceptNullable(IEnumerable<string?> nullableStrings)
{
    foreach (var nullable in nullableStrings)
        if (nullable != null)
            Console.WriteLine(nullable.Length); // no `null` => OK
}
```

```
void AcceptNonNullable(IEnumerable<string> nonNullableStrings)
{
    foreach (var nullable in nonNullableStrings)
        Console.WriteLine(nullable.Length); // `null` => NullReferenceException
}
```

## Action<string?> - подтип Action<string>

```
void AcceptNullable(Action<string?> action) => action(null);
```

```
void AcceptNonNullable(Action<string> action) => action("notNull");
```

```
void Run() {  
    Action<string?> nullableAction = x => Console.WriteLine(x);  
    AcceptNonNullable(nullableAction);  
    AcceptNullable(nullableAction);  
  
    Action<string> nonNullableAction = x => Console.WriteLine(x.Length);  
    AcceptNonNullable(nonNullableAction);  
    AcceptNullable(nonNullableAction);  
}
```



# Quiz!

- `Action<string>` = `Action<string?>`
- `IEnumerable<string?>` = `IEnumerable<string>`
  
- `List<string?>` = `List<string>`
- `List<string>` = `List<string?>`
- `List<string>` `<= / =>` `List<string?>`

`List<string>`  $\leq/\geq$  `List<string?>`

```
void NonNullableToNullable() {  
    List<string> listOfNonNullable = new List<string>();  
    List<string?> listOfNullable = listOfNonNullable;  
  
    listOfNullable.Add(null);  
    Console.WriteLine(listOfNonNullable[0].Length); // crash  
}
```

```
void NullableToNonNullable() {  
    List<string?> listOfNullable = new List<string?>();  
    List<string> listOfNonNullable = listOfNullable;  
  
    listOfNullable.Add(null);  
    Console.WriteLine(listOfNonNullable[0].Length); // crash  
}
```

# Nullable Reference Types и вывод типов

```
public string? Example(string? x) {  
    if (x == null) return;  
  
    var inferred = x;  
  
    inferred.ToString();  
    inferred = null;  
  
    return inferred;  
}
```

string? inferred;

inferred.ToString();  
inferred = null;

string:[NotNull] inferred;

inferred.ToString();  
inferred = null;

# Констрейнты на Nullable Reference Types

```
class C<T1, T2, T3, T4, T5>
  where T1: IInterface    // can only be substituted with non-nullable
  where T2: IInterface?
  where T3: class         // can only be substituted with non-nullable class
  where T4: class?
  where T5: notnull       // non-nullable class or struct
{
}
```

# Констрейнты на Nullable Reference Types

```
public T? ClassConstraint<T>() where T : class
{
    ClassConstraint<string>();
    ClassConstraint<string?>();
    ~~~~~
    return ...
}
```

 string? Constraints.ClassConstraint<string?>()

The type 'string?' cannot be used as type parameter 'T' in the generic type or method 'Constraints.ClassConstraint<T>()'. Nullability of type argument 'string?' doesn't match 'class' constraint.

[Show potential fixes \(Ctrl+.\)](#)

# Аннотируем свой фреймворк

```
public static TElement? FindFirstElement<TElement>(
    IEnumerable<TElement> input, string key)
    where TElement : IKeyOwner
{
    foreach (var element in input)
        if (element?.Key == key)
            return element;

    return default;
}
```



📄 TElement in Extensions.FindFirstElement<TElement> where TElement : IKeyOwner

A nullable type parameter must be known to be a value type or non-nullable reference type. Consider adding a 'class', 'struct', or type constraint.

Only non-nullable value type could be underlying of 'System.Nullable'

# Компилятор предлагает добавить `class`

```
public static TElement? FindFirstElement<TElement>(
    IEnumerable<TElement> input, string key)
    where TElement : class, IKeyOwner { /* ... */ }
```

```
StructKeyOwner Example1(StructKeyOwner[] structs)
    => FindFirstElement(structs, "key");
    ~~~~~
```

# Компилятор предлагает добавить `class`

```
public static TElement? FindFirstElement<TElement>(
    IEnumerable<TElement> input, string key)
    where TElement : class, IKeyOwner { /* ... */ }

// TElement -> ReferenceType?
// TElement is constrained to a non-nullable class
ReferenceType? Example2(ReferenceType?[] referenceTypeArray)
    => FindFirstElement(referenceTypeArray, "key");
```



# Попробуем использовать `class?`

```
public static TElement FindFirstElement<TElement>(
    IEnumerable<TElement> input, string key)
    where TElement : class?, IKeyOwner { /* ... */ }

public void Test(ReferenceType[] array) {
    // TElement -> ReferenceType
    // ReferenceType FindFirstElement<ReferenceType>( ... )
    var firstElement = FindFirstElement(array, "key");

    firstElement.ToString(); // no warnings
}
```

# А как теперь работает FirstOrDefault?

```
public static void Main()  
{  
    string[] nonNullableStrings = new string[0];  
    var isItNullable = nonNullableStrings.FirstOrDefault();  
    Console.WriteLine(isItNullable.Length); // runtime crash  
}
```



# А как теперь работает FirstOrDefault?

```
public static TSource FirstOrDefault<TSource>(
    this IEnumerable<TSource> source)
=> source.TryGetFirst(out bool s_);
```

**.NET Core 3.1**



<https://bit.ly/2PAHlfG>

```
[return: MaybeNull]
public static TSource FirstOrDefault<TSource>(
    this IEnumerable<TSource> source)
=> source.TryGetFirst(out bool s_);
```

**.NET Core master branch**



<https://bit.ly/2N4jT8M>

# Что делает MaybeNull?

```
[return: System.Diagnostics.CodeAnalysis.MaybeNull]
public static TElement FindFirstElement<TElement>(
    IEnumerable<TElement> input, string key) where TElement : IKeyOwner { /* ... */ }

public void Test(ReferenceType[] array) {
    // TElement -> ReferenceType
    // ReferenceType? FindFirstElement<ReferenceType>( ... )

    var firstElement = FindFirstElement(array, "key");
    firstElement.ToString();
}
```

А со структурами теперь будет работать?  
Они станут Nullable<T>?

```
[return: System.Diagnostics.CodeAnalysis.MaybeNull]
public static TElement FindFirstElement<TElement>(
    IEnumerable<TElement> input, string key) where TElement : IKeyOwner { /* ... */ }

public void Test(StructKeyOwner[] array) {
    StructKeyOwner firstElement = FindFirstElement(array, "key");

    firstElement.ToString(); // OK; it's a non-nullable struct
}
```

# Добавим assert. Что может пойти не так?

```
public static void Assert(bool condition,  
                           string message) {  
    if (!condition) throw new Exception(message);  
}
```

```
[return: MaybeNull] public static TElement FindFirstElement<TElement>(  
    IEnumerable<TElement> input, string key) where TElement : IKeyOwner  
{  
    Assertion.Assert(input != null, "input should not be null");  
    foreach (var element in input)  
        // ...  
}
```

# Добавим assert. Что может пойти не так?

```
public static void Assert([DoesNotReturnIf(false)] bool condition,  
                           string message) {  
    if (!condition) throw new Exception(message);  
}
```

```
[return: MaybeNull] public static TElement FindFirstElement<TElement>(IEnumerable<TElement> input, string key) where TElement : IKeyOwner  
{  
    Assertion.Assert(input != null, "input should not be null");  
    foreach (var element in input)  
        // ...  
}
```

# Code analysis attributes

```
namespace System.Diagnostics.CodeAnalysis
{
    [AllowNull]
    [DisallowNull]
    [MaybeNull]
    [NotNullWhen(bool)]
    [MaybeNullWhen(bool)]
    [NotNullIfNotNull(string)]
    [NotNull]
    [DoesNotReturn]
    [DoesNotReturnIf(true)]
}
```



# Code analysis attributes

```
[MaybeNull] /* <=> */ [CanBeNull]
```

```
[return:MaybeNull] T IEnumerable.FirstOrDefault<T>(this IEnumerable<T> source);
```

```
[AllowNull] /* <=> */ [CanBeNull]
```

```
bool IEqualityComparer<T>.Equals([AllowNull] T x, [AllowNull] T y);
```

```
[DisallowNull] /* <=> */ [JetBrains.Annotations.NotNull]
```

```
int IEqualityComparer<T>.GetHashCode([DisallowNull] T obj);
```

# Code analysis attributes

```
[NotNull] /* <=> */ [ContractAnnotation("value:null => halt")]  
  
public static void Assertion.AssertNotNull<T>([NotNull] T value);
```

```
[DoesNotReturn] /* <=> */ [ContractAnnotation("=> halt")]  
  
[DoesNotReturn] public static void Debug.Fail();
```

```
[DoesNotReturnIf(true)] /* <=> */ [ContractAnnotation("condition:true => halt")]  
  
public static void Debug.Assert([DoesNotReturnIf(false)] bool condition);
```

# Code analysis attributes

```
[MaybeNullWhen(bool)] /* <=> */ [ContractAnnotation("=> false; value:null")]  
bool IDictionary.TryGetValue(TKey key, [MaybeNullWhen(false)] out TValue value);
```

```
[NotNullWhen(bool)] /* <=> */ [ContractAnnotation("=> true; result:notnull")]  
bool Mutex.TryOpenExisting(string name, [NotNullWhen(true)] out Mutex? result);  
bool string.IsNullOrEmpty([NotNullWhen(false)] string? value);
```

```
[NotNullIfNotNull("path")] /* <=> */ [ContractAnnotation(  
    "path:notnull => notnull")]  
[return: NotNullIfNotNull("path")] string? Path.GetFileName(string? path);
```

# Code analysis attributes

- Подсказка что должен делать метод
- Компилятор полагается на аннотации при анализе использований метода
- Компилятор не проверяет что метод соответствует аннотации
  - В местах вызова проверки отсутствуют
  - В самом методе проверки отсутствуют
  - Рантайм проверки отсутствуют
  - Compile-time проверки отсутствуют

# План доклада

- Краткое описание nullable reference types
- Способы постепенного перевода проекта на их использование
- Взаимодействие с обобщенным кодом
- Аннотации для помощи компилятору
- **Что делать если компилятор не прав?**
- Common pitfalls
- Warnings as errors

# Компилятор не всегда прав

```
[return: MaybeNull] public static TElement FindFirstElement<TElement>(
    IEnumerable<TElement> input, string key) where TElement : IKeyOwner
{
    Assertion.Assert(input != null, "input should not be null");
    foreach (var element in input)
        if (element?.Key == key) return element;

    return default;
}
```

Что вообще такое default?

```
string str = default;
```

```
Console.WriteLine(str.Length);
```

# Будь! настойчивее! и! компилятор! согласится!

```
[return: MaybeNull] public static TElement FindFirstElement<TElement>(
    IEnumerable<TElement> input, string key) where TElement : IKeyOwner
{
    Assertion.Assert(input != null, "input should not be null");
    foreach (var element in input)
        if (element?.Key == key) return element;

    return default!;
}
```

---

# Dammit operator

- Что делать если компилятор не прав?
  - Игнорировать предупреждение
  - Добавить assert / проверку
  - Использовать dammit-оператор

- Да я потом инициализирую переменную...
  - `string x = null!`;




# Dammit operator

- Что делать если компилятор не прав?
  - Игнорировать предупреждение
  - Добавить assert / проверку
  - Использовать dammit-оператор

- Да я потом инициализирую переменную...
  - `public string Name { get; set; } = null!;`

# Dammit operator

- Не защищает от дальнейших предупреждений
  - `string x = y!;`  
`y.ToString(); // warning again`  

- Не добавляет рантайм проверок

# Dammit operator также делает значение not-null

```
Dictionary<Transaction, IClient>? GetFailedUserTransactions() { /* ... */ }  
List<Transaction> GetFailedCommissionDeductions() { /* ... */ }
```

```
void ProcessFailedTransactions() {  
    // Dictionary<Transaction, IClient?>? = Dictionary<Transaction, IClient>?  
    Dictionary<Transaction, IClient?>? failedTransactions  
                                     = GetFailedUserTransactions();  
    foreach (var transaction in GetFailedCommissionDeductions())  
        failedTransactions.Add(transaction, null);  
  
    // ...  
}
```

# Dammit operator также делает значение not-null

```
Dictionary<Transaction, IClient>? GetFailedUserTransactions() { /* ... */ }  
List<Transaction> GetFailedCommissionDeductions() { /* ... */ }
```

```
void ProcessFailedTransactions() {
```

```
Dictionary<Transaction, IClient?>? failedTransactions
```

```
= GetFailedUserTransactions()!;
```

```
foreach (var transaction in GetFailedCommissionDeductions())
```

```
failedTransactions.Add(transaction, null); // no warnings runtime crash
```

```
// ...
```

# А что если компилятор слишком прав?




**David Kean**

davkean

Dev on C#/VB languages team @ Microsoft.  
Working on <https://github.com/dotnet/project-system>.

 @microsoft

 Melbourne, Australia

 [Sign in to view email](#)

 <http://twitter.com/davkean>

## Organizations



**David Kean**

@davkean

Follow



This nullable warning I get from this has thrown me: [sharplab.io/#v2:EYLgZgpghg](https://sharplab.io/#v2:EYLgZgpghg)

...

This feels like the compiler is ignoring me when I say this is a nullable string.

@andygocke @jcouv

12:43 am - 28 Aug 2019



<https://bit.ly/2MBjIGj>

4 Likes



«This feels like the compiler is ignoring me when I say this is a nullable string.»

```
Task<string?> FindFileAsync()  
{  
    string? path = GetMeNonNullableString();  
    return Task.FromResult(path); // Task<string?> = Task<string>  
}
```



Nullability of reference types in value of type 'Task<string>' doesn't match target type 'Task<string?>'.

[Show potential fixes \(Ctrl+.\)](#)

```
string GetMeNonNullableString() => string.Empty;
```

```
public static Task<TResult> Task.FromResult<TResult>(TResult result);
```

# Resolution – By Design

```
private static Task<string> FindFileAsync() {  
    string? path = GetMeSomeString();  
    if (path == null)  
        throw new ArgumentNullException(nameof(path));  
  
    return Task.FromResult(path); // we expect the compiler to know that  
                                   // path is not null here  
                                   // even though it's declared as `string?`  
}  
  
string? GetMeSomeString() { /* ... */ }
```

# И как же тогда быть?

```
private static Task<string?> FindFileAsync1() {  
    string? path = GetMeNonNullableString();  
    return Task.FromResult(path)!;  
}
```

Dammit operator

```
private static Task<string?> FindFileAsync2() {  
    string? path = GetMeNonNullableString();  
    return Task.FromResult<string?>(path);  
}
```

Explicit type arguments

```
private static Task<string?> FindFileAsync3() {  
    var path = (string?) GetMeNonNullableString();  
    return Task.FromResult(path);  
}
```

Cast to nullable type



# Итак, как донести свою мысль до компилятора?

- Я знаю что здесь не бывает `null`
  - Assert / dammit-оператор
- Здесь нужен `nullable`-тип для вывода типов
  - Каст к `nullable` / явные типы-аргументы
- Предупреждение в преобразовании обобщенных типов
  - Dammit-оператор, убедиться в отсутствии нежелательных сайдэффектов

# Где нужно быть особенно внимательным?

- Не проаннотированные библиотеки
- Инициализация массивов
- Ref / in / out параметры и переменные
- Кросс-процедурные зависимости
- Замыкания

# Не проаннотированные библиотеки

```
[CLSCompliant(false)]  
public static Logger GetLogger(string name) {  
    return LogManager.factory.GetLogger(name);  
}
```

NLog

```
class LibConsumer {  
    public void Consume() {  
        var logger = LogManager.GetLogger(name: null); // no warnings; crash  
        logger.Warn("This code passed null to the logger name");  
    }  
}
```

# А какие библиотеки проаннотированы?

#	Package	Downloads
1	<a href="#"><u>newtonsoft.json</u></a>	<a href="#"><u>24,281,200</u></a>
2	<a href="#"><u>serilog</u></a>	<a href="#"><u>7,352,817</u></a>
3	<a href="#"><u>castle.core</u></a>	<a href="#"><u>7,285,094</u></a>
4	<a href="#"><u>moq</u></a>	<a href="#"><u>5,742,965</u></a>
10	<a href="#"><u>xunit</u></a>	<a href="#"><u>4,751,202</u></a>
11	<a href="#"><u>automapper</u></a>	<a href="#"><u>4,505,797</u></a>
14	<a href="#"><u>awssdk.core</u></a>	<a href="#"><u>3,906,135</u></a>
15	<a href="#"><u>nunit</u></a>	<a href="#"><u>3,663,371</u></a>

# Инициализация массивов

```
void ArrayInit() {  
    var array = new string[10];  
    for (var i = 0; i < array.Length; i++)  
        array[i] = Console.ReadLine();  
  
    array[0].ToString();  
}
```

# Инициализация массивов

```
void ArrayInit() {  
    var array = new string[10];  
    // for (var i = 0; i < array.Length; i++)  
    //     array[i] = Console.ReadLine();  
  
    array[0].ToString(); // no warnings; runtime crash  
}
```

# Кросс-процедурные зависимости

```
class SlideContent {  
    public Image? Image { get; }  
    public string? Text { get; }  
    public Column[]? Columns { get; }  
  
    public void FormatContent() {  
        if (Image != null && Text != null) {  
            SetTwoColumnsTemplate();  
            Columns[0].SetImage(Image);  
            Columns[1].SetText(Text);  
        }  
    }  
}
```

# Кросс-процедурные зависимости

```
class SlideContent {  
    public void FormatContent() {  
        if (Image != null && Text != null) {  
            SetTwoColumnsTemplate();  
            Columns![0].SetImage(Image); // no warnings; crash  
            Columns[1].SetText(Text);  
        }  
  
        private void SetTwoColumnsTemplate() {  
            Columns = new [2] {new Column(Text, Image), new Column() };  
            Image = null;  
            Text = null;  
        }  
    }  
}
```



# Ref / in / out параметры

```
private string? myField;
```

```
public void TestArgument() {  
    if (myField == null) return;  
    ArgumentCheck(ref myField);  
}
```

```
public void ArgumentCheck(ref string argument) {  
    myField = null;  
    argument.ToString(); // no warnings; crash  
}
```

# Записи в замыканиях

```
void LambdaExpression1(string? str) {  
    Action action = () => str = Console.ReadLine();  
    action();  
    str.ToString(); // possible derference of a null reference  
}
```

```
void LambdaExpression2(string? str) {  
    Action action = () => str = null;  
    if (str != null) {  
        action();  
        str.ToString(); // no warnings; runtime crash  
    }  
}
```

# Анализ самих замыканий

```
void LambdaExpression3(string? str) {  
    if (str == null) return;  
  
    Action action = () => str.ToString(); // no warnings; runtime crash  
    str = null;  
    action();  
}  
  
void LocalFunction(string? str) {  
    if (str == null) return;  
  
    void Local() => str.ToString(); // warning  
    Local();  
}
```

# Локальные функции

```
private void ReplayReadsAndWrites(LocalFunctionSymbol localFunc,  
                                   SyntaxNode syntax,  
                                   bool writes)  
{  
    // https://github.com/dotnet/roslyn/issues/27233  
    // Support field initializers in local functions.  
}
```

# Аннотация массивов

// rank specifiers order - left to right

```
string ArrayTest1(string[][,][,,] array) => array[1] [2,2] [3,3,3];
```

// rank specifiers order - right to left

```
string ArrayTest2(string[]?[,]?[,]?[,]? array) => array[3,3,3]? [2,2]? [1];
```

# Quiz!

```
void NoAnnotations(string[][,][,,] array) => array [1] [2,2] [3,3,3];
```

// can you guess?

```
string SingleAnnotation(string[]?[,][,,] array)  
    => array[1] [3,3,3] [2,2];  
    => array[3,3,3] [1] [2,2];  
    => array[2,2] [3,3,3] [1];
```

# Quiz!

```
void NoAnnotations(string[][,][,,] array) => array [1] [2,2] [3,3,3];
```

// can you guess?

```
string SingleAnnotation(string[]?[,][,,] array)  
    => array[1] [3,3,3] [2,2];  
    => array[3,3,3] [1] [2,2];  
    => array[2,2] [3,3,3] [1];
```

# Quiz!

```
void NoAnnotations(string[][,][,,] array) => array [1] [2,2] [3,3,3];
```

// can you guess?

```
void SingleAnnotation(string[]?[,][,,] array)  
  
=> array[2,2] [3,3,3]? [1];
```

```
var array = new string[]?[10][] { /* ... */ }
```



# Как такое могло случиться?

```
void SyntaxAmbiguity() {  
    // good old C# 7.3; nullable types are disallowed in pattern matching  
    var result = x is int? y : illegalSyntax;  
    var result = x is int ? y : illegalSyntax;  
  
    // top-level type is nullable -> conditional expression  
    var result = x is string ? y : illegalSyntax;  
  
    // top-level type is nullable -> conditional expression  
    var result = x is string[]? y : illegalSyntax;  
  
    // ??? Let's change the syntax so the last `?` is always top-level!  
    var result = x is string[][]? y : illegalSyntax;  
}
```



<https://bit.ly/323Ypyc>

# `warnings as errors`

## Плюсы

- Быстрое обнаружение возможных ошибок при изменениях в коде
- Легко обнаруживать изменения контрактов подключенных библиотек
  - если они проаннотированы

## Минусы

- Хрупкий код, при рефакторингах появляются ошибки
- Невозможно объявить переменную нужного типа
  - Паттерн / out var переменные
- Бессмысленные проверки когда анализ не справляется

# Зависимости между переменными

```
static void Main() {  
    string? one = Environment.GetEnvironmentVariable("one");  
    string? two = Environment.GetEnvironmentVariable("two");  
    if (one == null && two == null)  
        Console.WriteLine("both null");           // one == null; two == null;  
    else if (one != null && two != null)  
        Console.WriteLine("both non-null");        // one != null; two != null;  
    else if (one != null)  
        Console.WriteLine("one is non-null");      // one != null; two == null;  
    else  
        Console.WriteLine(two.Length);            // one == null; two != null;  
}
```



<https://bit.ly/33d3Fzt>

# Код легко ломается рефакторингами

```
class C {  
    string? Field1;  
    void M1(C c) {  
        if (c.Field1 != null)  
            c.Field1.Equals(...);  
    }  
}
```



<https://bit.ly/2VeBsZE>

# Код легко ломается рефакторингами

```
class C {  
    string? Field1;  
    void M1(C c) {  
        if (c.Field1 != null)  
            M2(c);  
    }  
    void M2(C c) {  
        c.Field1.Equals(...);  
    }  
}
```



<https://bit.ly/2VeBsZE>

# Код легко ломается рефакторингами

```
public void Example1(A a) {  
    if (a?.b is B) {  
        a.ToString(); // no warnings; correct  
    }  
}
```

```
public void Example2(A a) {  
    var b = a?.b as B;  
    if (b != null) {  
        a.ToString(); // potentially nullable-dereference warning (wrong)  
    }  
}
```



<https://bit.ly/2IA32rR>

Grzegorz Galezowski

@GGalezowski

Follow



Why does `@roslyn` issue a warning in this case? The `ItemGroups` is a nullable ref type, but it's assigned in the constructor and there is no way in my class to nullify it...

```
public class XmlProjectBuilder
{
    private readonly XmlProject _xmlProject;

    public XmlProjectBuilder(string assemblyName)
    {
        _xmlProject = new XmlProject
        {
            AbsolutePath = AbsolutePathTo(assemblyName),
            PropertyGroups = new List<XmlPropertyGroup>
            {
                new XmlPropertyGroup
                {
                    AssemblyName = assemblyName
                }
            },
            ItemGroups = new List<XmlItemGroup>();
        };

        public void WithReferences(params string[] names)
        {
            _xmlProject.ItemGroups.Add(
                new XmlItemGroup
                {
                    AssemblyName = assemblyName,
                    References = new List<string> { names }
                }
            );
        }

        public void WithProperties(params string[] names)
        {
            _xmlProject.ItemGroups.Add(
                new XmlItemGroup
                {
                    AssemblyName = assemblyName,
                    Properties = new List<string> { names }
                }
            );
        }
    }
}
```

Possible dereference of a null reference.  
Show potential fixes (Ctrl+.)



<https://bit.ly/2Wrrf8U>

4:43 PM - 27 Apr 2019



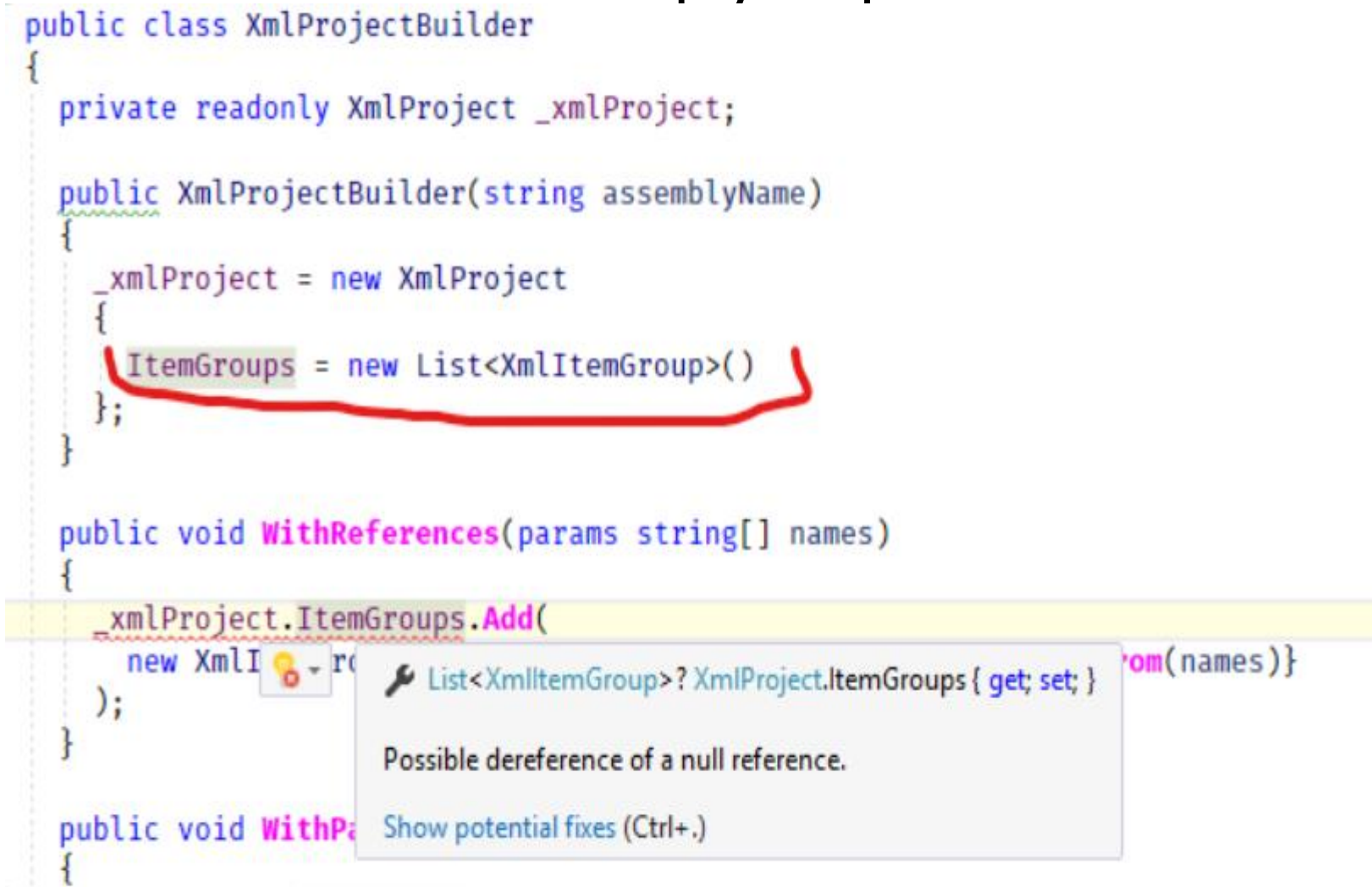
Но ведь я присвоил только **[NotNull]** в конструкторе?

```
public class XmlProjectBuilder
{
    private readonly XmlProject _xmlProject;

    public XmlProjectBuilder(string assemblyName)
    {
        _xmlProject = new XmlProject
        {
            ItemGroups = new List<XmlItemGroup>()
        };

        public void WithReferences(params string[] names)
        {
            _xmlProject.ItemGroups.Add(
                new XmlItemGroup { Name = names[0] });
        }

        public void WithParameters(params string[] names)
        {
            _xmlProject.ItemGroups.Add(
                new XmlItemGroup { Name = names[0] });
        }
    }
}
```





# Не всегда возможно объявить правильный тип переменной

```
var dic = new Dictionary<string, string> { {"key", "value"}};  
dic.TryGetValue("key", out string value);
```



[🔧] (local variable) `string` value

Converting null literal or possible null value to non-nullable type.

[Show potential fixes \(Ctrl+.\)](#)

```
var dic = new Dictionary<string, string> { {"key", "value"}};  
string value;  
dic.TryGetValue("key", out value!);
```



<https://bit.ly/310vBFg>

# Не всегда возможно объявить правильный тип переменной

```
if (SomeCall() is string result)
{
    DoSomething(result);
    result = GetPossibleNullValue();
}
```



Converting null literal or possible null value to non-nullable type.

Show potential fixes (Ctrl+.)

```
if (SomeCall() is string result)
{
    string? nullableResult = result;
    DoSomething(nullableResult);
    nullableResult = GetPossibleNullValue();
}
```

# Заключение

- Больше информации о коде
- Больше ошибок найденных на этапе разработки
- Анализ может ошибаться
- #nullable директивы для постепенного включения фичи в проекте
- System.Diagnostics.CodeAnalysis аннотации помогут компилятору понять ваш код
- `Warnings as errors` может сделать ваш код слишком хрупким

# Спасибо за внимание



[tessenr@gmail.com](mailto:tessenr@gmail.com)



[github.com/TessenR](https://github.com/TessenR)



[twitter.com/a\\_tessenr](https://twitter.com/a_tessenr)