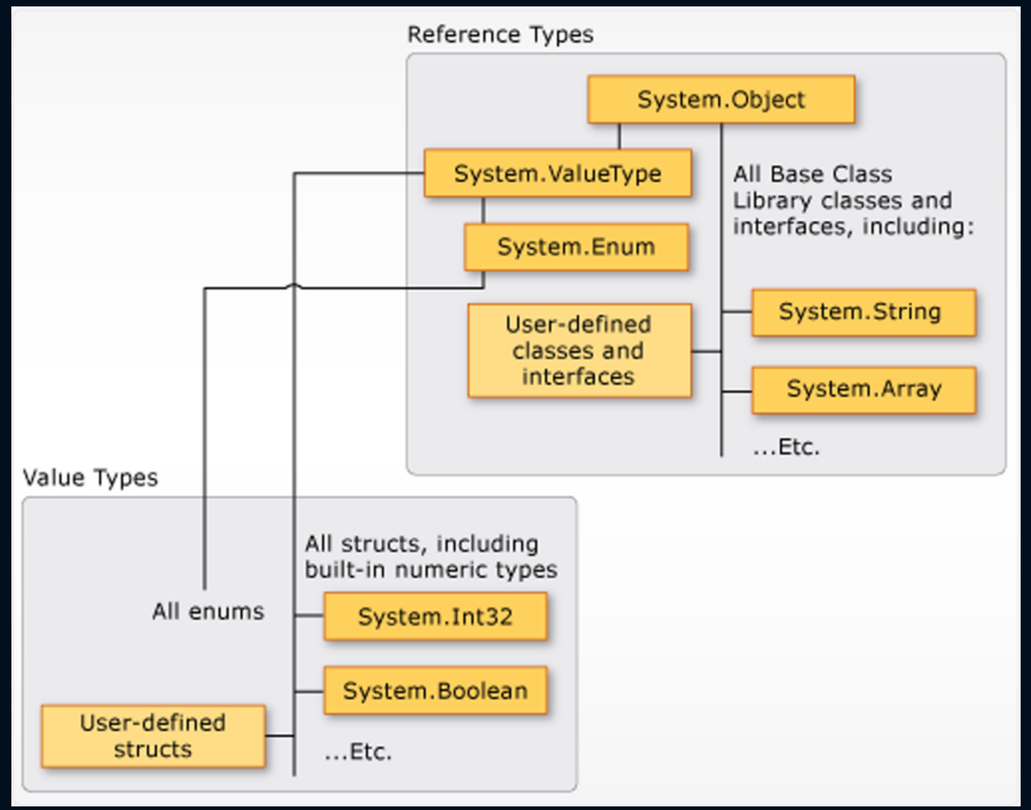
Common Type System

The Common Type System (CTS) is a set of rules and definitions that describe the data types that can be used in C#. The CTS defines primitive data types, structures, classes, enumerations, reference types, and value types.



Common Type System

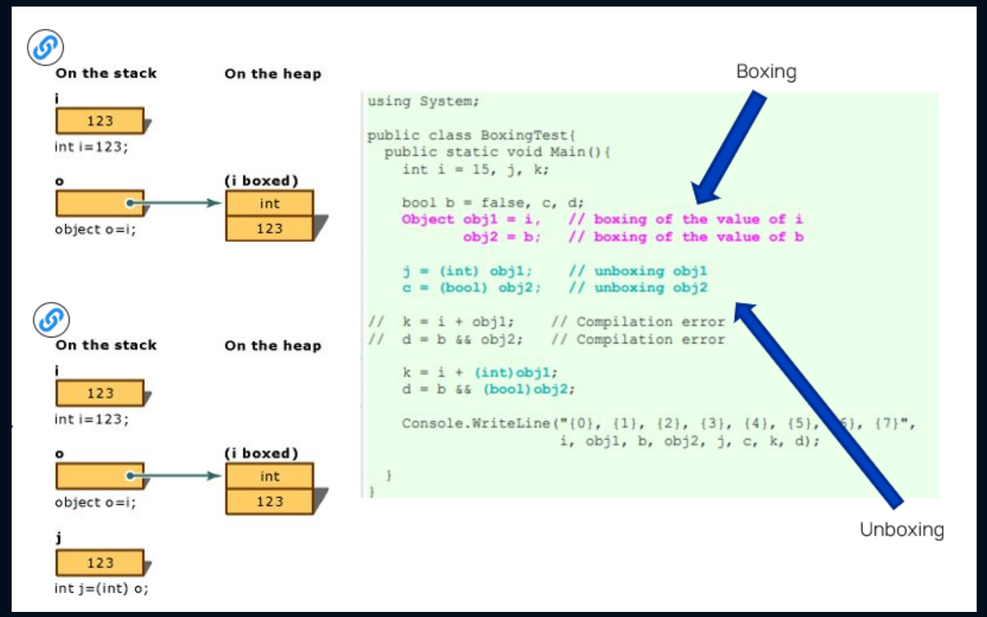
Boxing and unboxing

Boxing is the process of converting a primitive type into an object type.
Unboxing, is the process of converting an object type into a primitive type.
This is one of the building blocks of the unification of the Type System, which was implemented for the C# programming language. It allows different types of data to be used interchangeably by conversion.

What happens when boxing : An example of this would be an int being converted to an System.Int32 object by the compiler. It will be stored on the heap rather than on the stack with the other primitive types

What happens when you unbox a variable: you are performing an explicit conversion from an object type to a primitive type.

For more information here is some documentation



Boxing and unboxing

Conversions

There are two main types of programming languages. The first one is statically typed, the second one is dynamically typed, statically typed languages are usually faster than dynamically typed ones.

More information in this documentation

```
// Implicit conversion
int age = 12;
double era = age;
```

Implicit conversion

This is the safest type of casting. No data is lost. E.g converting a smaller to larger integral types

```
// Explicit conversion
double pi = 3.14;
int roundedPi = (int)pi;
```

Explicit conversion

Involves the cast operator. It is associated with information loss. E.g Convert from larger precision datatype to lower precision

User defined conversion

They can be introduced by declaring conversion operators in class and struct types.
These user-defined conversions convert a value from its type, called source type, to another type which is called target type.

Conversion with helper classes

Similar to the previous conversion type

Conversion examples

Implicit conversion

```
int a = 10;
double b = a; // an int can be safely cast into a double
WriteLine(b);
```

// Result is 10

Explicit conversion

```
double c = 9.8;
int d = c; // compiler gives an error for this line
WriteLine(d);
```

Compilation error

```
int d = (int)c;
WriteLine(d); // d is 9 losing the .8 part
```

Using cast operator
Result is 9

```
using static System.Console;
```

← Importing namespace

Using System.Convert

```
using static System.Convert;
double g = 9.8;
int h = ToInt32(g); // a method of System.Convert
WriteLine($"g is {g} and h is {h}");
```

Result

```
g is 9.8 and h is 10
```

is, as and typeof

Alongside the cast operators, these operators and expressions perform type checking or type conversion. More information in the documentation

is operator

Checks if the run-time type of an expression result is compatible with a given type. The is operator also tests an expression result against a pattern.
The is operator doesn't consider user-defined conversions.

as operator

Explicitly converts the result of an expression to a given reference or nullable value type.
If the conversion isn't possible, the as operator returns null. Unlike a cast expression, the as operator never throws an exception.
Considers only reference, nullable, boxing, and unboxing conversions.

typeof operator

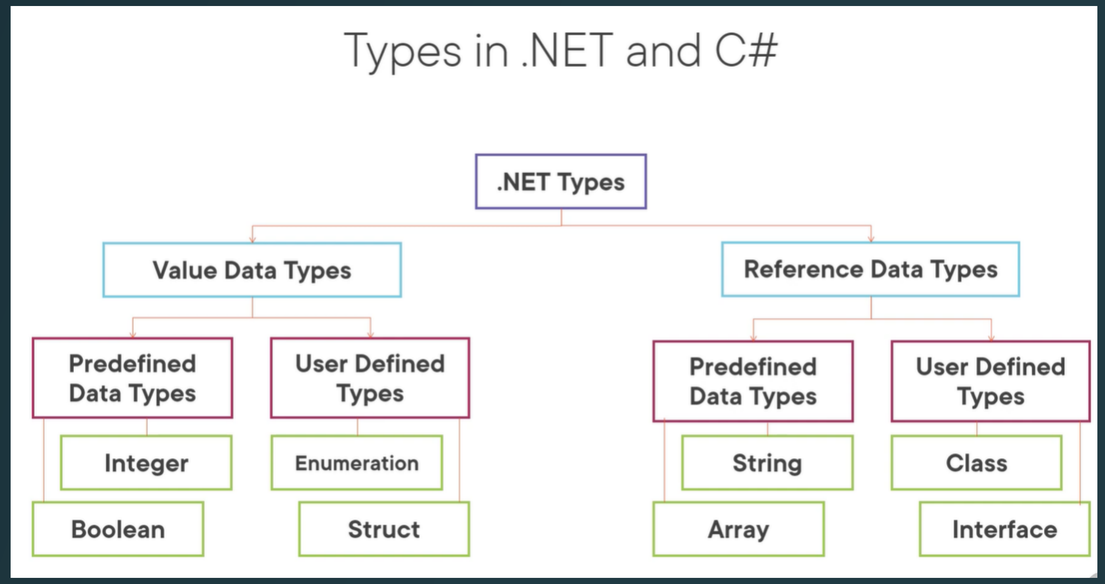
Obtains the System.Type instance for a type.
The argument to the typeof operator must be the name of a type or a type parameter.
The argument mustn't be a type that requires metadata annotations (dynamic, string?)

Value and reference types

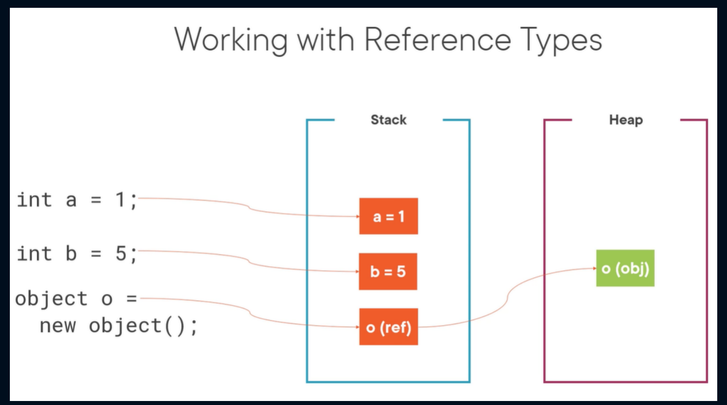
Data types can be classified into two large groups : Value types and reference types

- Fixed size, allocated by compiler on stack
- Value is copied to this memory location
- Independent instances/copies
- Value change does not affect other copies

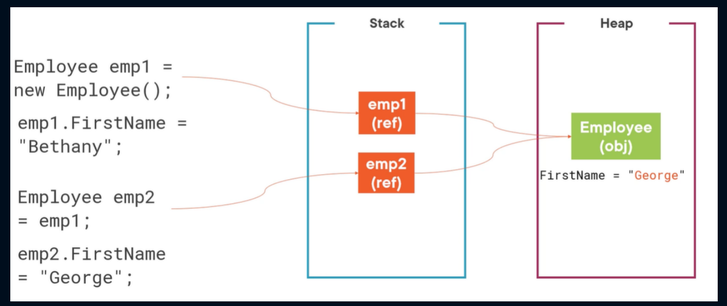
- Stack contains just a pointer to the memory address
- Classes and collections are reference types
- Single shared instance
- Reference may point to nothing (null)
- Null checking code may be required



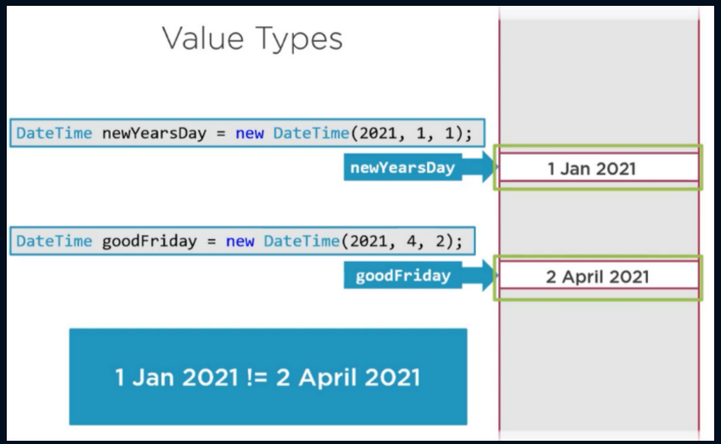
Value and reference types when creating copies



Reference types and value types assignation



Copying using a reference type



Using a value type

Visual studio commands

Some useful command to use in Visual Studio

Ctrl + TAB
Shows active files in editor

Alt + F7
Shows active tools

Ctrl + G
Specify a line number in the active editor

Ctrl + ,
Navigate to any file in the solution

Ctrl + 1 + T
Search for types in the current editor

Ctrl + T
Search anything in the solution

Ctrl + -
Go to the previous position (similar to Undo but for navigation)

Ctrl + Shift + -
Go back to the position (Re do but for navigation)

F12
When selecting a type, to see the definition

Alt+F12
When selecting a type, to peek definition

Ctrl + G
Use the Visual studio general search

Ctrl + .
Show refactoring actions

Ctrl + F
Search in file

Ctrl + H
Find and replace

Ctrl + M + O
Expand file to show details

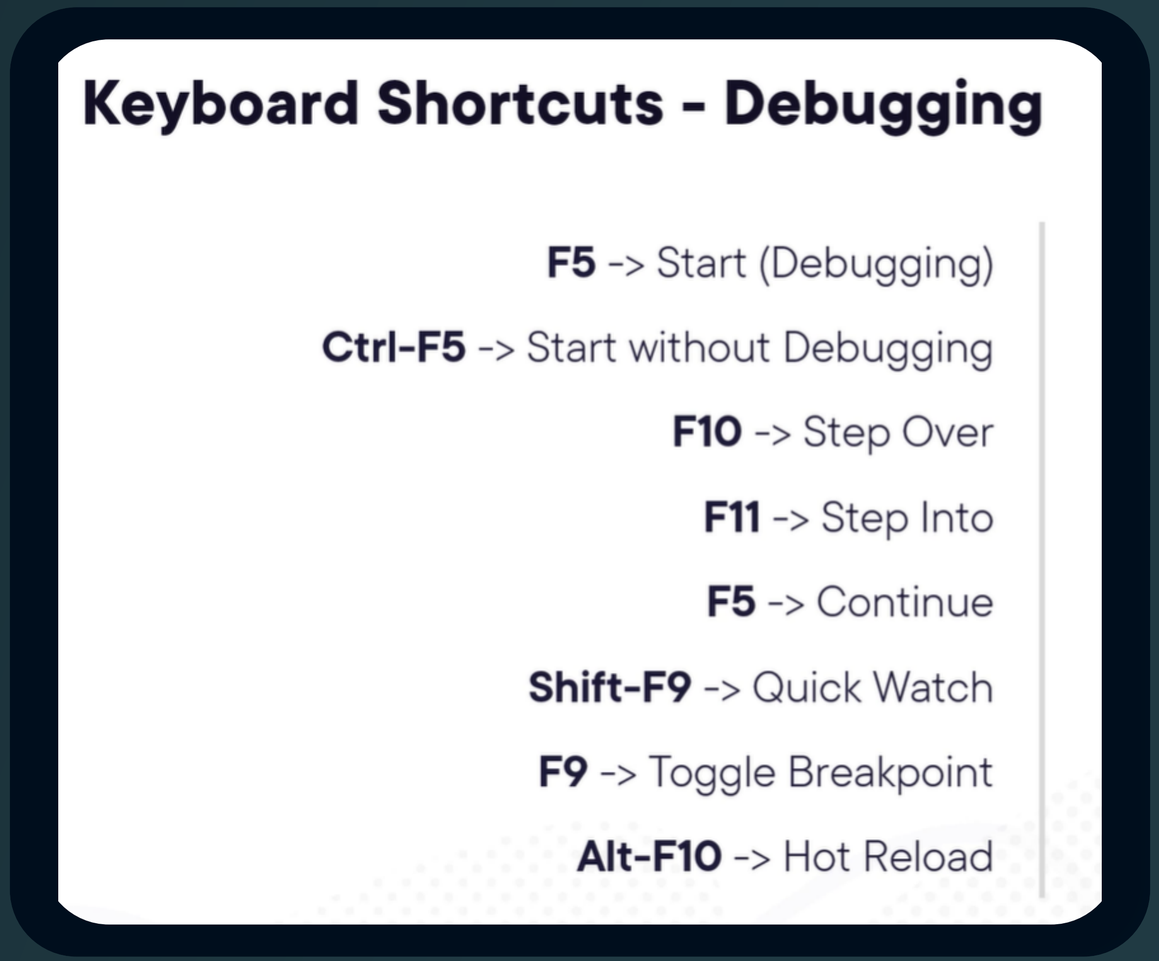
Ctrl + K + E
Run cleanup configuration

Ctrl + R +R
Rename selected value

Ctrl + R + G
Clean usings and reorder

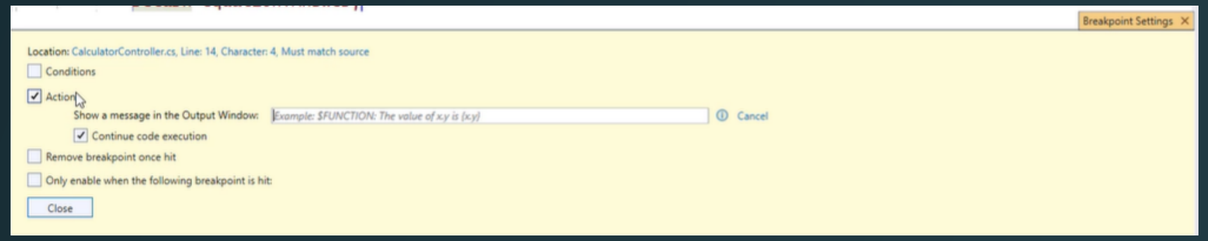
Some keyboard shortcuts

Some benefits of debugging: clearly understand flow and conditions in code. Help me code easier to read / maintain / understand. Improve reliability of code or application

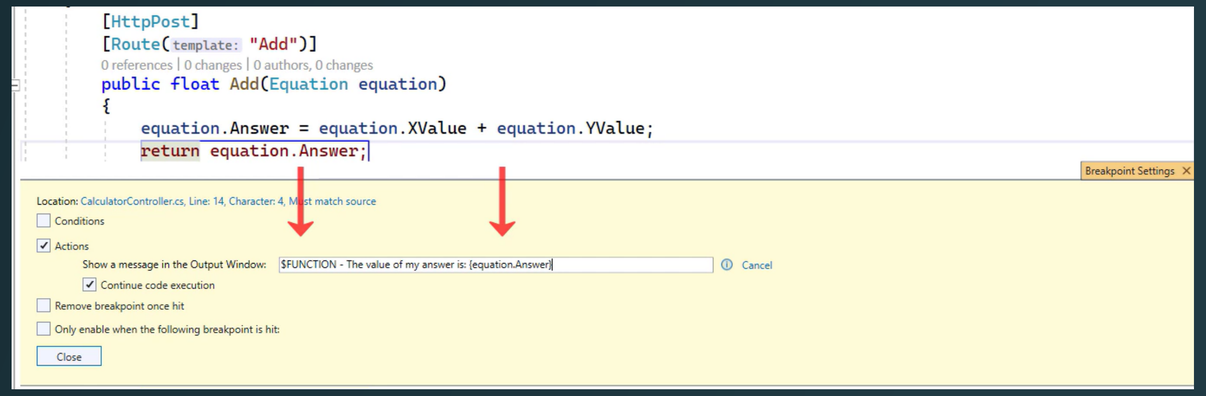


Tracepoints

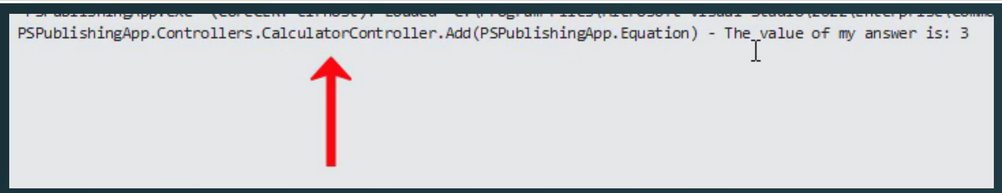
Breakpoints in Visual Studio IDE are essential debugging tools that allow developers to pause the execution of their code at specific points to inspect variables, step through the code, and troubleshoot issues.



Using action inside breakpoints



Using Function metadata when breakpoint hit



Result

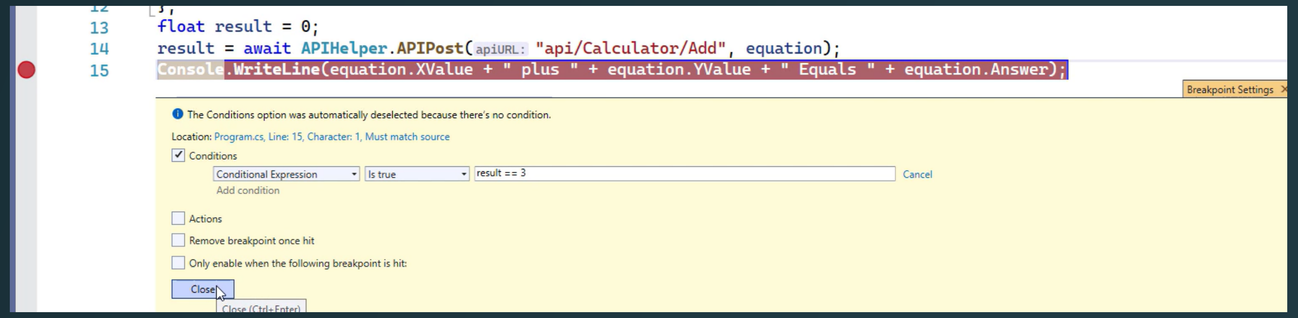
Conditions

Breakpoint Types:

- Standard Breakpoints: These are the regular breakpoints that pause the execution of the program when reached.
- Tracepoints: Tracepoints are special breakpoints that don't cause the program to pause but instead allow developers to log custom messages or perform specific actions when hit.
- Conditional Breakpoints: Conditional breakpoints are set with a condition, such as a specific variable value, and will only pause the execution if the condition is true.
- Hit Count Breakpoints: Hit count breakpoints pause the execution after a certain number of hits, useful for repetitive code or loops.

Breakpoint Conditions: Breakpoints can have conditions associated with them, allowing developers to specify when a breakpoint should be hit.

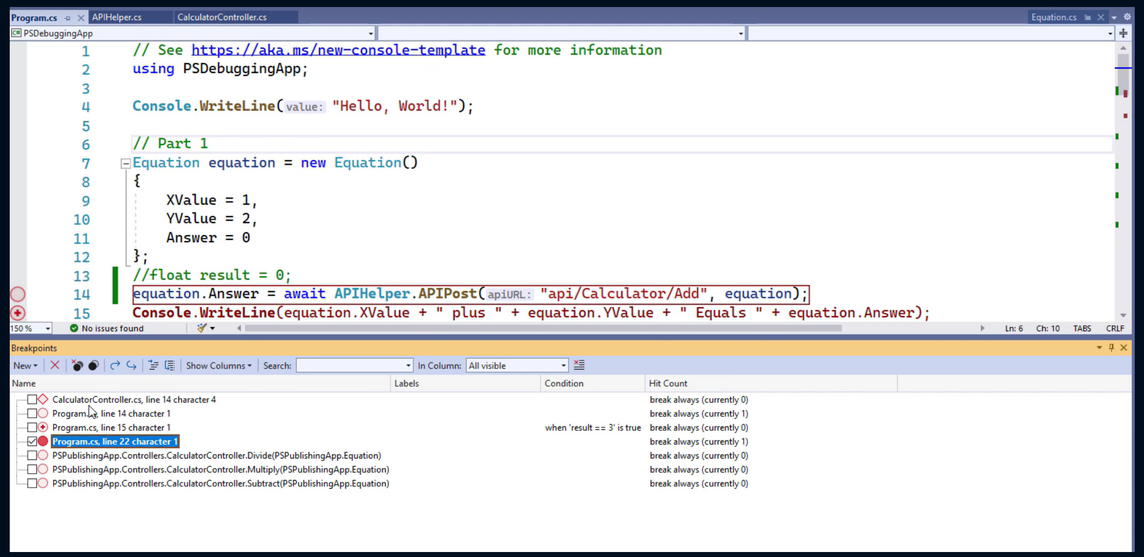
Dependent Breakpoints: Dependent breakpoints are breakpoints that are triggered only when a specific preceding breakpoint is hit.



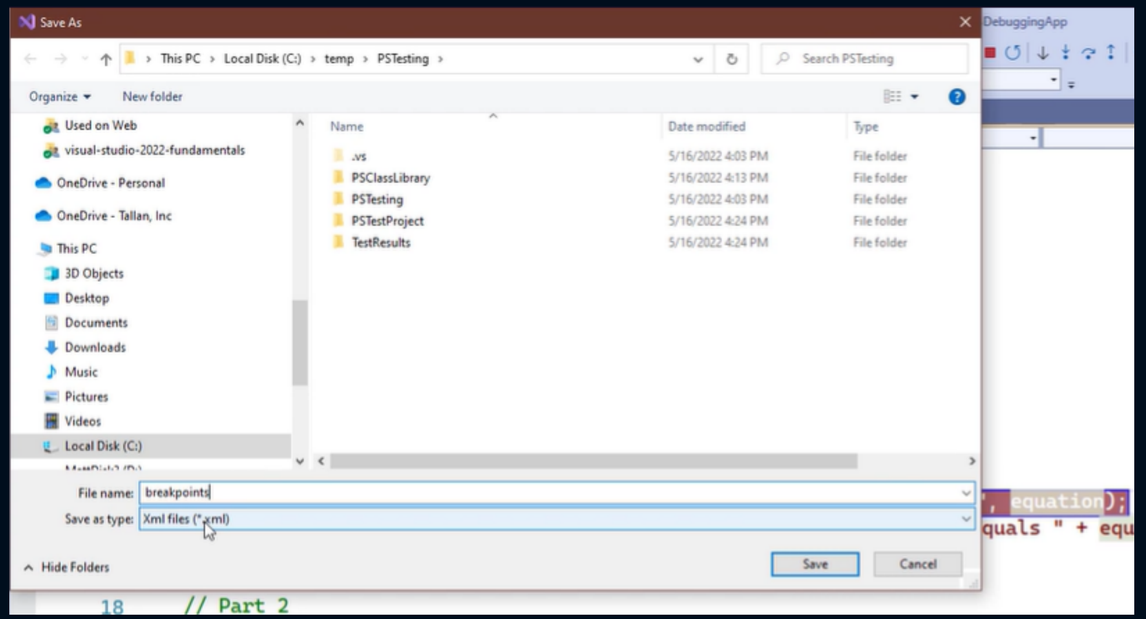
Using Condition to hit breakpoint

Breakpoint View and Export

Breakpoint export is a feature in Visual Studio IDE that allows developers to export their configured breakpoints to share or reuse them across different development environments or with other team members.



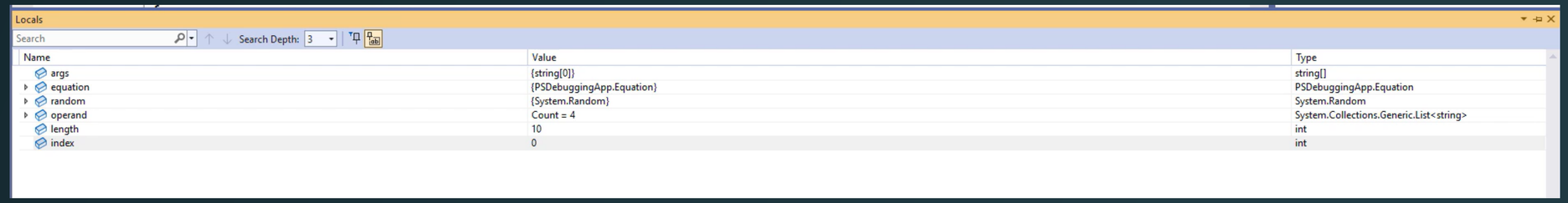
Break point view

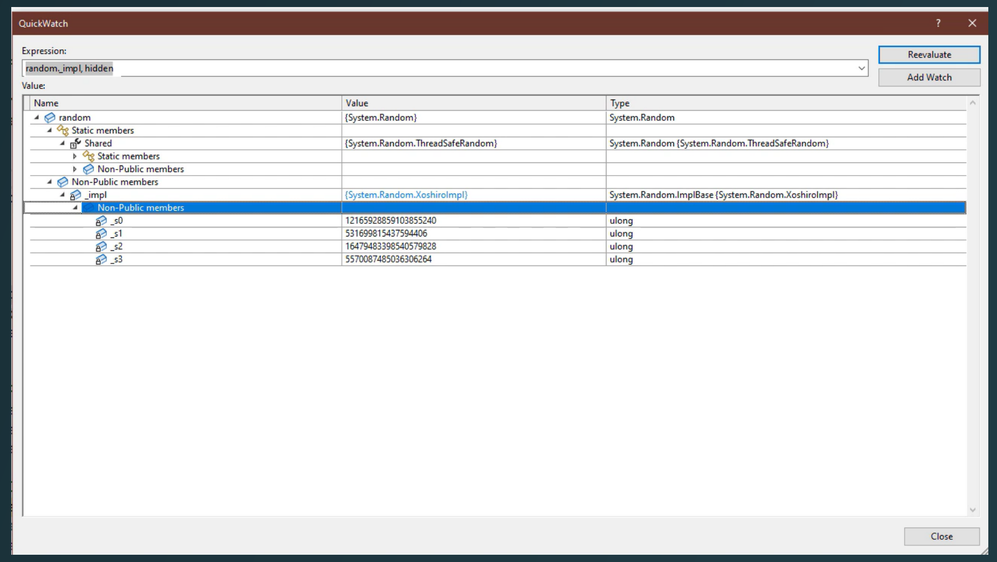


Export break points

Watch and quick watch

In Visual Studio IDE, the Watch and QuickWatch features are used during the debugging process to inspect and monitor the values of variables, expressions, and properties.



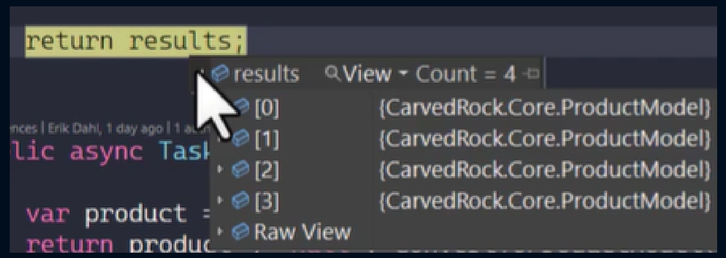


Watch - displays the current values of these items, these can be updated during debugging.

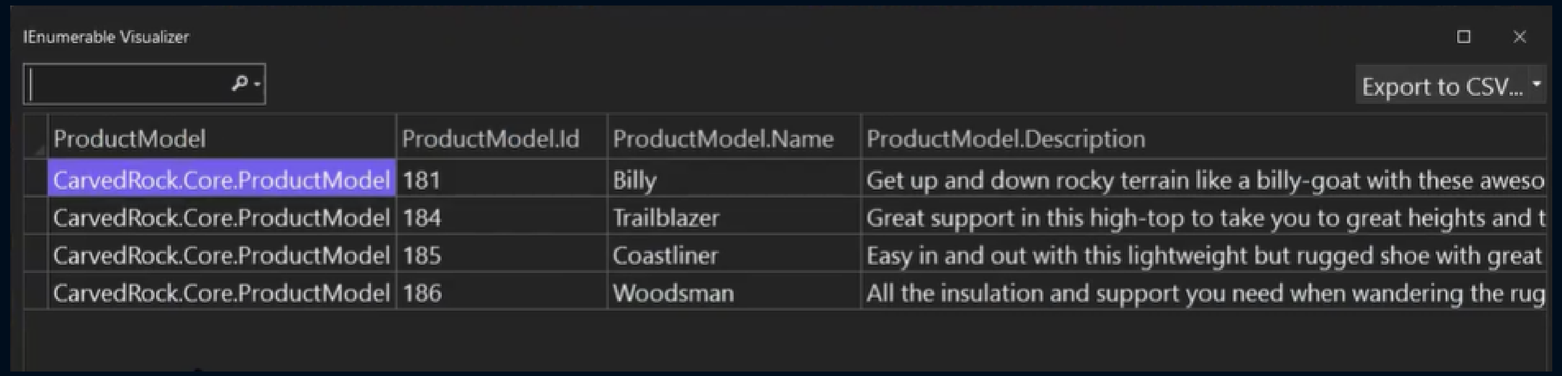
Quick watch - provides an on-the-spot evaluation of an expression or variable during debugging.

Other debugger tools

The debugger can also display data inline, some of the information available is: scalar values, classes (uses default ToString()) , for collections you can use a visualizer and for strings visualizers are extended.



Debugger inline display



Debugger Visualizer

Demo

We will check the following:

Conversions

Debugging tools

You can check the following Repository for some examples:

C# fundamentals

Tasks

Quick Exercises

Create a Console application that gives the user 2 options:

1. Input Commands
2. Show categories
3. Exit

When selecting 1 the user can input anything, this information should be categorized between text and numbers, and separated in different collections.

When selecting 2 the user should see a List of Text commands and a List of Numeric commands.

When selecting 3 the application should terminate its execution