# Applying String Comparisons and Sorting

**Steve Gordon**
.NET Engineer and Microsoft MVP

@stevejgordon www.stevejgordon.co.uk

## Overview

**Validate string values**

**Check string equality**

- Operators
- String.Equals

**Extract multiple matches using regex**

**Apply TryParseExact**

**Compare strings**

**Optimise regex and consider security**

**Sort strings**

## Requirements

- Implement the IsValid property on HistoricalSalesData.
- Ensure all value are present and within expected ranges.

## Demo

**Validate strings**

- Complete the IsValid implementation

**Perform equality checks**

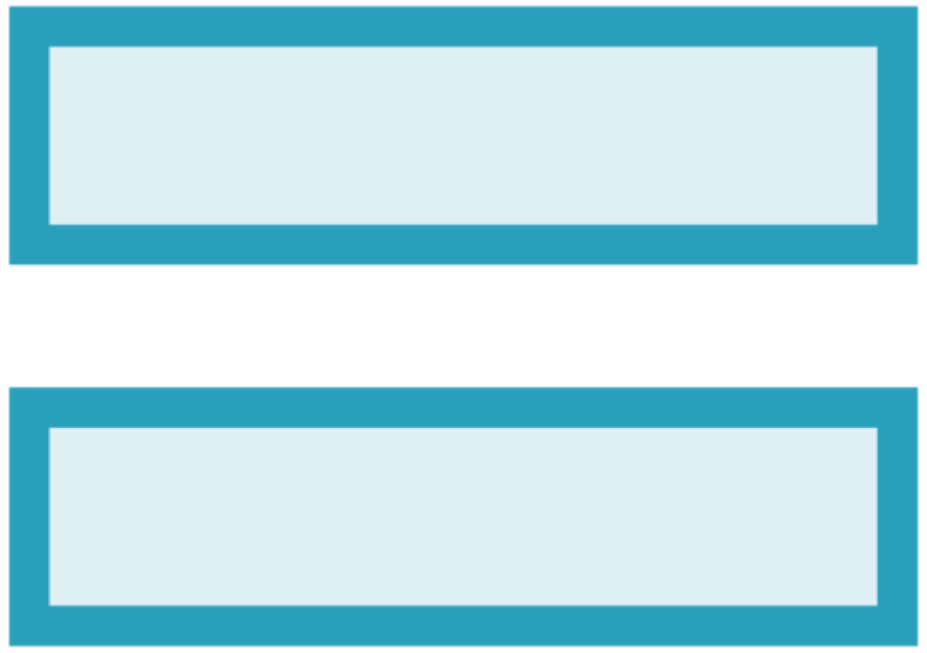
```
// Definition:
public bool Equals (string? value);


// Use:
var myString = "A string literal";
bool result = myString.Equals("Comparison string"); // false
```

## Equals

**Determines whether this instance and another specified String object have the same value.**

# Ordinal Comparisons



**Checks the code point of each character**

**Comparison continues while the code points match**

**Strings are equal if all character code points are equal**

**Comparisons can take a fast path**

- When the two strings have reference equality, they are equal
- If the string lengths differ, they are not equal

**Ordinal matches are case-sensitive**

# Character Code Points

a

**Decimal: 97**
**Hex: 0x61**

A

**Decimal: 65**
**Hex: 0x41**

## Ordinal Comparisons

**Culture insensitive**

**Fastest comparison**

- Linguistic cultural rules are ignored

**Comparison occurs byte-for-byte**

**Not suited to user supplied or UI data which should apply cultural rules during comparison**

## Demo

**Learn more about string equality**

**Apply equality operators**

**Use the StringComparison enum**

- Perform case insensitive equality checks

## Requirements

- Calculate total sales for electrical engineering products in the historical sales data.
- Calculate using any occurrences of the electrical engineering code, regardless of the casing.
- Ensure unit tests pass

# StringComparison

Specifies the culture, case, and sort rules to be used by certain overloads of the Compare and Equals methods.

| Value | Description |
|---|---|
| **CurrentCulture** | Performs a case-sensitive comparison using the current culture. |
| **CurrentCultureIgnoreCase** | Performs a case-insensitive comparison using the current culture. |
| **InvariantCulture** | Performs a case-sensitive comparison using the invariant culture. |
| **InvariantCultureIgnoreCase** | Performs a case-insensitive comparison using the invariant culture. |
| **Ordinal** | Performs an ordinal comparison of code points. |
| **OrdinalIgnoreCase** | Performs a case-insensitive ordinal comparison of code points. |

# CurrentCulture vs. Invariant vs. Ordinal

**Ordinal comparisons check the code point of each character**

- Fastest comparison of strings

**Use ordinal rules when the comparison is linguistically irrelevant**

**Prefer CurrentCulture for user input and displaying data to a user**

**InvariantCulture is more applicable to persisted, linguistically relevant data and when applying a fixed sort order**

## Demo

Learn how to extract multiple matches using regex

## Requirements

- Process and parse customer data.
- Extract the customer code, identifier and country.

```
// Namespace:

System.Text.RegularExpressions

// Definition:

public static MatchCollection Matches (string input, string pattern);

// Use:

MatchCollection result = Regex.Matches("Az Az","Az");
```

## Regex.Matches

**Searches the specified input string for all occurrences of a specified regular expression.**

## Demo

Learn about TryParseExact

Apply TryParseExtract when parsing GUIDs

Several built-in types include a TryParseExact method.

```
// Definition:

public static bool TryParseExact (string? input, string? format,
    out Guid result);

// Use:

var myGuid = "c0fb150f-6bf3-44df-984a-3a0611ae5e4a";

bool result = Guid.TryParseExact(myGuid, "D", out var parsedGuid);
```

## Guid.TryParseExact

**Converts the string representation of a GUID to the equivalent Guid structure, provided that the string is in the specified format.**

# Format Specifiers

Short strings which correspond to a particular format of valid string representations of a data type.

| Specifier | Example |
|---|---|
| N | 00000000000000000000000000000000 |
| D | 00000000-0000-0000-0000-000000000000 |
| B | {00000000-0000-0000-0000-000000000000} |
| P | (00000000-0000-0000-0000-000000000000) |
| X | {0x00000000,0x0000,0x0000,{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}} |

Demo

**Compare two strings**

- Identify the relative position of two strings

```
// Definition:
public static int Compare (string? strA, string? strB);
// Use:
var apple = "apple";
var apple = "carrot";
int result = string.Compare(apple, carrot); // -1
```

## String.Compare

**Compares two specified string objects and returns an integer that indicates their relative position in the sort order.**

# Compare Return Values

| Value | Description |
|---|---|
| **Less than zero** | The first string precedes the second. |
| **Zero** | Both strings occur in the same sort position. |
| **Greater than zero** | The second string precedes the firsts. |

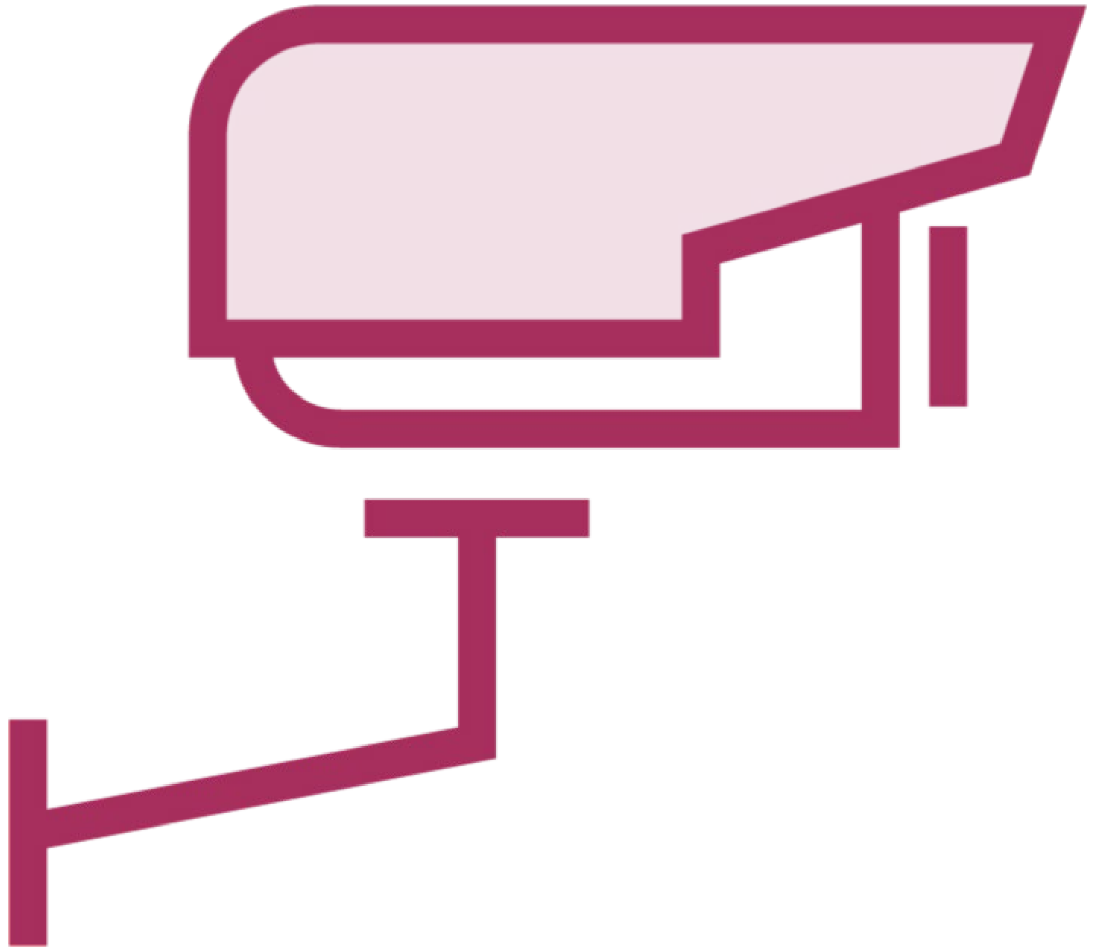
By default, **Compare** performs a culture sensitive comparison using the current thread culture.

## Demo

Optimise runtime performance of regex

Security considerations

## Security Considerations



Generally regex should perform reasonably quickly

Characteristics such as backtracking can cause execution to take much longer

A malicious actor could attempt to cause a denial-of-service attack

## Security Precautions

For trusted source data and patterns ensure you test your regex

Unconstrained, untrusted input requires more care

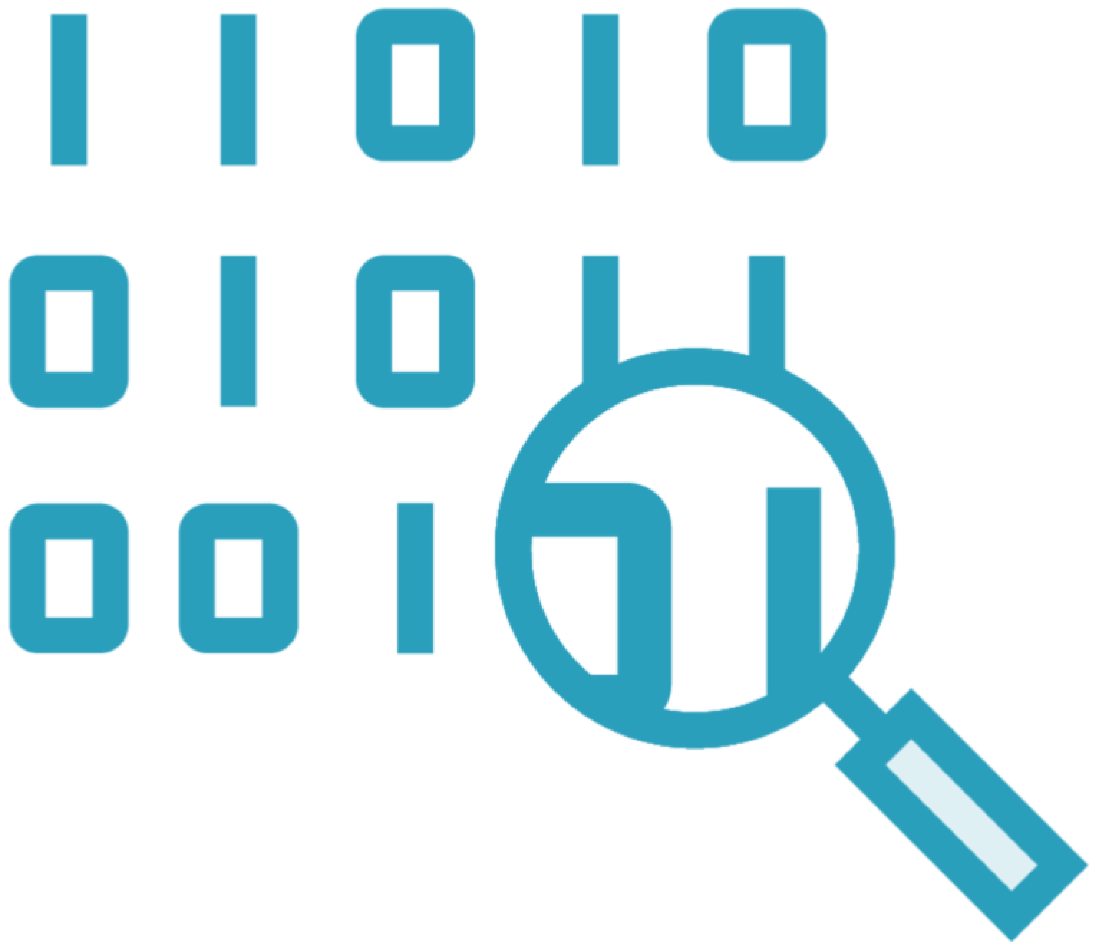
Mitigate risks by providing a timeout

Timeouts define the maximum runtime allowed to execute a matching operation

As a best practice, always include a timeout

Timeouts can prevent attacks and accidental misuse

# Interpreted Regex



**By default, regex is interpreted at runtime**

- The engine converts expressions to operation codes at instantiation
- During execution the codes are interpreted

**Static regex methods cache the op codes to avoid repeatedly reparsing patterns**

**Interpreted regex reduces startup time at the cost of a slower execution time**

# Regex Compilation

Regex can also be compiled into MSIL

Compiling incurs a higher startup cost, but reduces the runtime of executing regex

Consider compiling when an expression will be reused often

Ensure you reuse the same compiled instance

Demo

Sorting strings

# Sorting

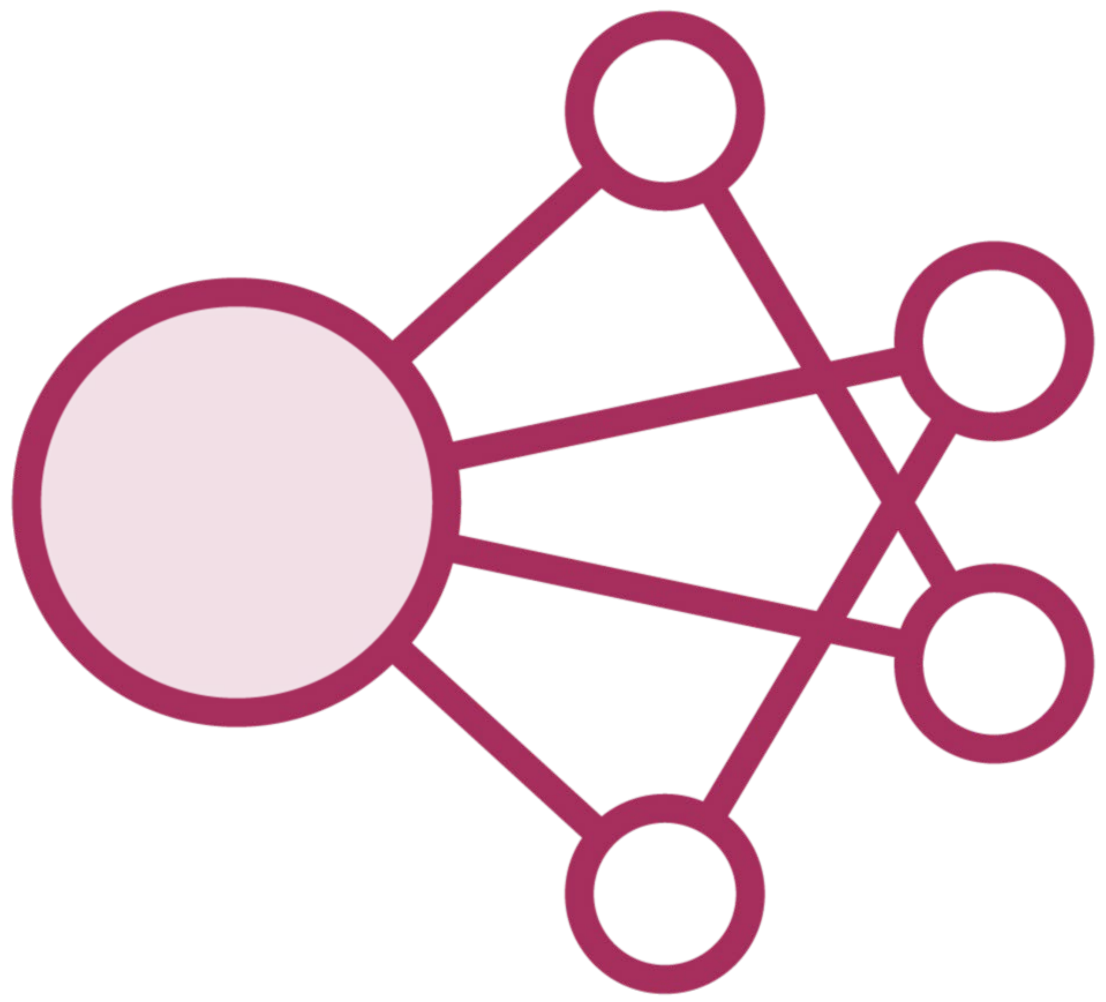
Orders a collection of items according to a set of rules

For strings, sorting is often alphabetical

We may consider casing when sorting

We may consider cultural and linguistic rules to sort data for a given language and country

# LINQ



The default comparison applied by LINQ methods is case insensitive

Many LINQ methods accept a StringComparer

It's a good practice to explicitly include a comparer rather than relying on defaults

This helps avoid bugs and incorrect assumptions

## Requirements

- Produce an output file containing countries where priority customers reside.
- The output should be a list of unique countries, sorted alphabetically.

## Demo

Use a sorted set of strings

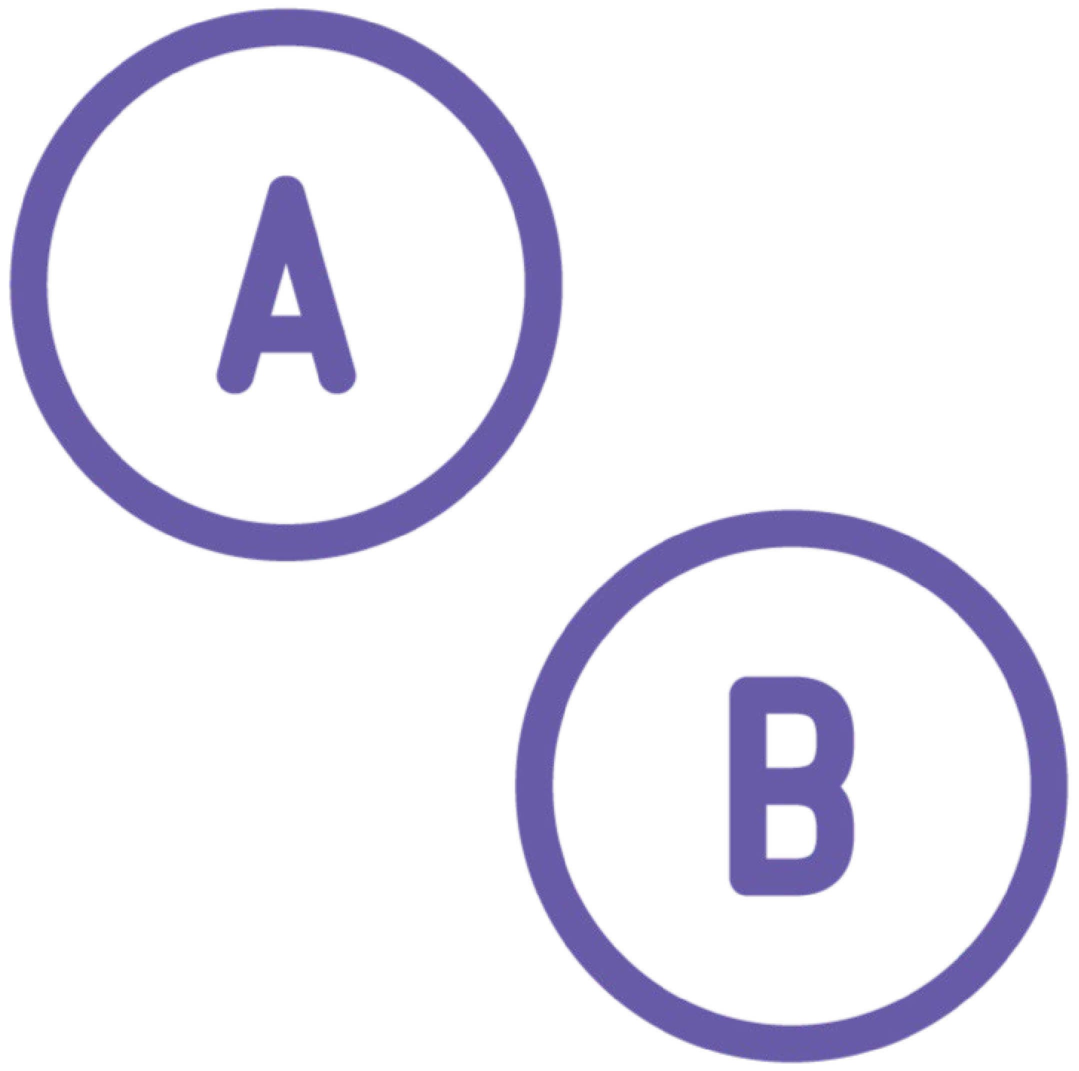
Culture-aware sorting

## Sets

When not specified, sets use the default comparer for sorting

The default comparer expects the type to implement IComparible<T>

Strings implement this interface and use CurrentCulture for sorting



Types and methods in .NET may use different defaults for equality and comparison.

## Avoid Surprises

Explicitly provide a StringComparer to functions which accept one, even if it aligns with the default behavior.

## Up Next:
## Applying Techniques for Searching Strings