Extended property patterns

Article • 06/23/2023

> **Note**
>
> This article is a feature specification. The specification represents the *proposed* feature specification. There may be some discrepancies between the feature specification and the completed implementation. Those differences are captured in the pertinent **language design meeting (LDM) notes**. Links to pertinent meetings are included at the bottom of the spec. You can learn more about the process for merging feature speclets into the C# language standard in the article on the **specifications**.

Summary

Allow property subpatterns to reference nested members, for instance:

```
C#
if (e is MethodCallExpression { Method.Name: "MethodName" })
```

Instead of:

```
C#
if (e is MethodCallExpression { Method: { Name: "MethodName" } })
```

Motivation

When you want to match a child property, nesting another recursive pattern adds too much noise which will hurt readability with no real advantage.

Detailed design

The *property_pattern* syntax is modified as follow:

```
diff
```

```
property_pattern
  : type? property_pattern_clause simple_designation?
  ;

property_pattern_clause
  : '{' (subpattern (',' subpattern)* ','?)? '}'
  ;

subpattern
- : identifier ':' pattern
+ : subpattern_name ':' pattern
  ;

+subpattern_name
+ : identifier
+ | subpattern_name '.' identifier
+ ;
```

The receiver for each name lookup is the type of the previous member *T0*, starting from the *input type* of the *property_pattern*. if *T* is a nullable type, *T0* is its underlying type, otherwise *T0* is equal to *T*.

For example, a pattern of the form `{ Prop1.Prop2: pattern }` is exactly equivalent to `{ Prop1: { Prop2: pattern } }`.

Note that this will include the null check when *T* is a nullable value type or a reference type. This null check means that the nested properties available will be the properties of *T0*, not of *T*.

Repeated member paths are allowed. The compilation of pattern matching can take advantage of common parts of patterns.