Expressions

# ParenthesizedExpressionSyntax

Always contains a single expression, so have this a characteristic of other expressions

# PrefixUnaryExpressionSyntax

Always contains a single operand and an expression – but that expression cannot be just any expression and be legal. Not sure whether to have this as a wrapping expression or a characteristic of other expressions.

For reference, Roslyn does untangle this as valid +”Fred”

# AwaitExpressionSyntax

Keyword and another expression. So, probably have this as a characteristic (although await is more complicated)

# PostfixUnaryExpressionSyntax

Same as PrefixUnaryExpressionSyntax

# MemberAccessExpressionSyntax

This contains an expression and a SimpleNameSyntax name. The expression points to the object that is being accessed, which will often be a TypeSyntax, which includes another SimpleNameSyntax.

This feels different than the wrapped types.

# ConditionalAccessExpressionSyntax

Huh? Not sure which conditional type this is. The new one?

# MemberBindingExpressionSyntax

Huh? Dot operator and token?

# ElementBindingExpressionSyntax

Huh? Bracketed argument list?

# ImplicitElementAccessSyntax

Huh? Bracketed argument list?

# BinaryExpressionSyntax

Feels like a fundamental type, but it would be nice to combine a bit more graciously than continually doing pairs.

# AssignmentExpressionSyntax

Not available in VB

Similar to binary in construction, different purpose

# ConditionalExpressionSyntax

Apparently the ? : one.

# ThisExpressionSyntax

Doesn’t feel like an expression, well, feels more like a literal expression

# BaseExpressionSyntax

Doesn’t feel like an expression, well, feels more like a literal expression

# LiteralExpressionSyntax

Just has a token representing the kind? I don’t see where the value is held.

# MakeRefExpressionSyntax

Huh? Open, Close, MakeRefKeyword, expression

# RefTypeExpressionSyntax

Huh? Open, Close, RefTypeKeyword, expression

# RefValueExpressionSyntax

Huh? Open, Close, RefValueKeyword, expression and TypeSyntax and comma

# CheckedExpressionSyntax

Wraps another expression

# DefaultExpressionSyntax

Fundamental

# TypeOfExpressionSyntax

Fundamental

# SizeOfExpressionSyntax

Wraps another expression, but that’s not what it feels like it does

# InvocationExpressionSyntax

Draft is done. Underlying treats all of the thing called as an expression.

# ElementAccessExpressionSyntax

Expression and bracketed argument list.

# CastExpressionSyntax

Expression and type

# AnonymousMethodExpressionSyntax

ParameterList and BlockSyntax and has AsyncKeyword, DelegateKeyword

# SimpleLambdaExpressionSyntax

Simple in having only one parameter, I think multi-line is a better distinction. Parameter and body, with Async keyword. Parameters here allow attributes and modifiers

# ParenthesizedLambdaExpressionSyntax

Parenthesized parameters, I think multi-line is a better distinction. Parameter and body, with Async keyword. Parameters here allow attributes and modifiers

# InitializerExpressionSyntax

Is this only used for curly bracket initialization? Braces and expression list.

# ObjectCreationExpressionSyntax

Type, arguments and initializer (braces kind)

# AnonymousObjectCreationExpressionSyntax

Uses AnonumousObjectMemberDeclaratorSyntax as initializers

# ArrayCreationExpressionSyntax

Type is ArrayCreationExpressionSyntax, which includes TypeRankSpecifiers and initializer

# ImplicitArrayCreationExpressionSyntax

Seems like it should be a variation of ArrayCreationSyntax. Contains commas not included with the normal array

# StackAllocArrayCreationExpressionSyntax

Huh? StackAllocKeyword and Type

# QueryExpressionSyntax

Uses FromClauseSyntax and QueryBodySyntax, which uses SelectOrGroupClauseSyntax and QueryContinuationSyntax as well as QueryClauseSyntaxes.

This is just long form, which I’d like to be able to interchange with the lambda form

# OmittedArraySizeExpressionSyntax

Hmmm. Do I need this?

# InterpolatedStringExpressionSyntax

New string interpolation