Introduction

Attention! This is a paid package, you can not use it in your project if you have not purchased it through Unity Asset Store.

This package provides the software interface for parsing and expression execution written in C#. It is specially designed to work with the Unity on various platforms. Since it is written in C# 3.5, it should work with any version of Unity.

It is tested to work on: * IOS * Android * WebGL * PC/Mac

It should work on any other platforms.

API * CSharpExpression * Evaluate * Parse * AotCompilation * AotRuntime * RegisterForFastCall

Example

Parsing C# expression:

```
CSharpExpression.Parse<double, double, double>("Math.Max(value1,value2)", arg1Name:
"value1", arg2Name: "value2")
// -> Expression<Func<double, double, double>>
```

Evaluating C# expression:

```
CSharpExpression.Evaluate<int>("2 * (2 + 3) << 1 + 1 & 7 | 25 ^ 10");
// -> 19
```

Parser

The parser recognizes the C# 4 grammar only. It includes:

- Arithmetic operations
- Bitwise operations
- Logical operations
- Conditional operator
- Null-coalescing operator
- Method/Delegate/Constructor call
- Property/Field access
- Indexers
- Casting and Conversion
- Is Operator
- As Operator
- TypeOf Operator

- Default Operator
- Expression grouping with parentheses
- Checked/Unchecked scopes
- Aliases for Built-In Types
- "true", "false", "null"

Nullable types are **partially** supported (see roadmap).

Enumerations are supported.

Generics are **not** supported (see roadmap).

Known Types

For security reasons the parser does not provide access to static types, except: * argument types * primitive types * Math class

To access other types your should pass **knownTypes** parameter in **Parse** or **Evaluate** method:

```
CSharpExpression.Evaluate<int>("Mathf.Clamp(Time.time, 1.0F, 3.0F)", knownTypes:
new[]{ typeof(Mathf), typeof(Time) });
```

AOT Execution

You can compile expression created by **System.Linq.Expression** and execute it in AOT environment where it is usually impossible. Any expression can be compiled, even those that are currently not supported by the parser. For example, expression with type constructor is not supported by Parser, but it can be compiled and executed on IOS (and other AOT environment).

```
Expression<Func<Vector3>> expression = () => new Vector3(1.0f, 1.0f, 1.0f);
Func<Vector3> compiledExpression = expression.CompileAot();

compiledExpression();
// -> Vector3(1.0f, 1.0f, 1.0f)
```

IOS, WebGL and most consoles use AOT compilation which imposes following restrictions on the dynamic code execution:

- only **Expression<Func<...>>** could be used with **CompileAot()**
- only static methods using primitives (int, float, string, object ...) are optimized for fast calls
- all used classes/methods/properties should be visible to Unity's static analyser

See Also * AOT Exception Patterns and Hacks * Ahead of Time Compilation (AOT)

WebGL

Since WebGL has problems with unsigned types and *long* type, the use of these types in expressions are disabled with *#if !UNITY_WEBGL* define. Once the problems are solved, this "define" will be removed.

Building under WebGL bears same limitations and recommendations as building under IOS.

IOS

- Lambda Expressions are not supported (see roadmap)
- Instance methods invocation performs slowly due reflection
- Moderate boxing for value types (see roadmap)

Improving methods invocation performance

You can improve the performance of methods invocation by registering their signatures in **AotCompilation.RegisterForFastCall()**.

```
// Supports up to 3 arguments.
// First generic argument is your class type.
// Last generic argument is return type.

AotCompilation.RegisterForFastCall<InstanceT, ResultT>()
AotCompilation.RegisterForFastCall<InstanceT, Arg1T, ResultT>()
AotCompilation.RegisterForFastCall<InstanceT, Arg1T, Arg2T, ResultT>()
AotCompilation.RegisterForFastCall<InstanceT, Arg1T, Arg2T, Arg3T, ResultT>()
```

Example code:

```
public class MyVectorMath
{
    public Vector4 Dot(Vector4 vector, Vector4 vector);
    public Vector4 Cross(Vector4 vector, Vector4 vector);
    public Vector4 Scale(Vector4 vector, float scale);
}

// register Dot and Cross method signatures
AotCompilation.RegisterForFastCall<MyVectorMath, Vector4, Vector4, Vector4>();
// register Scale method signature
AotCompilation.RegisterForFastCall<MyVectorMath, Vector4, float, Vector4>();
```

Roadmap

You can send suggestions to email support@gamedevware.com. The roadmap depends entirely on how many copies of the package will be sold:

- >50
    - Parser: Specifying generic arguments for types and methods
    - Parser: Nullable types specification (like Int32?)

- >100
  - Expression serialization
  - Void expressions (System.Action delegates)
- >200
  - AOT: Lambda expressions
  - Parser: Delegate construction from method reference
  - Parser: Type inference for generics
- >300
  - Parser: C#6 syntax
  - Parser: Extension methods
  - Parser: Type initializers, List initializers
  - Custom editor with auto-completion for Unity

License