CppDepend Graphs

Table of contents

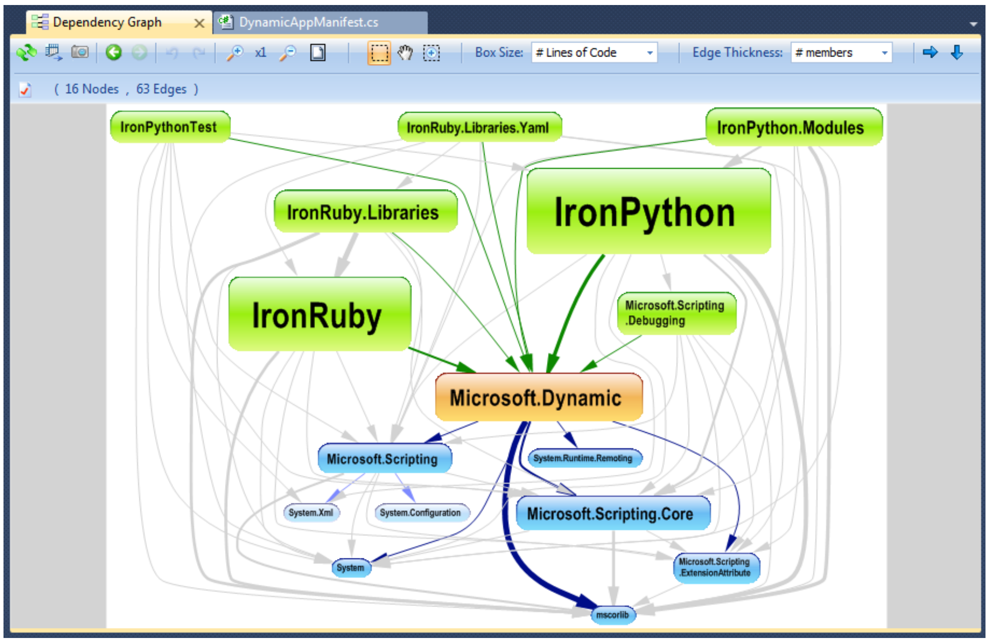
1. Introduction

CppDepend capabilities to help user exploring an existing Code Architecture are endless. In this document you'll learn how to benefit from these features in order to achieve most popular Code Exploration scenarios:

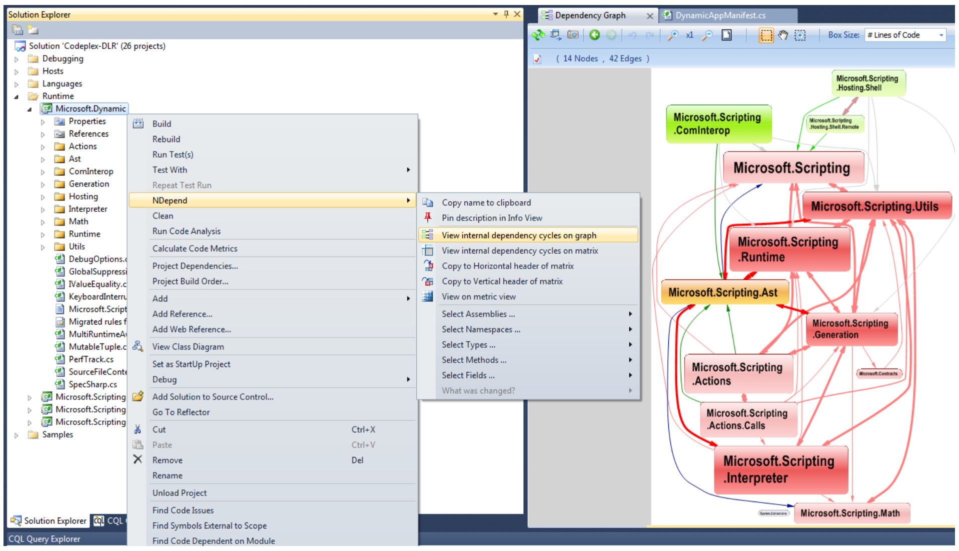
Dependency Graph	**Call Graph**
Coupling Graph	**All Paths Graph**
Cycle Graph	**Large Graph visualized with Dependency Structure Matrix**
Path Graph	**Class Inheritance Graph**

2. Dependency graph

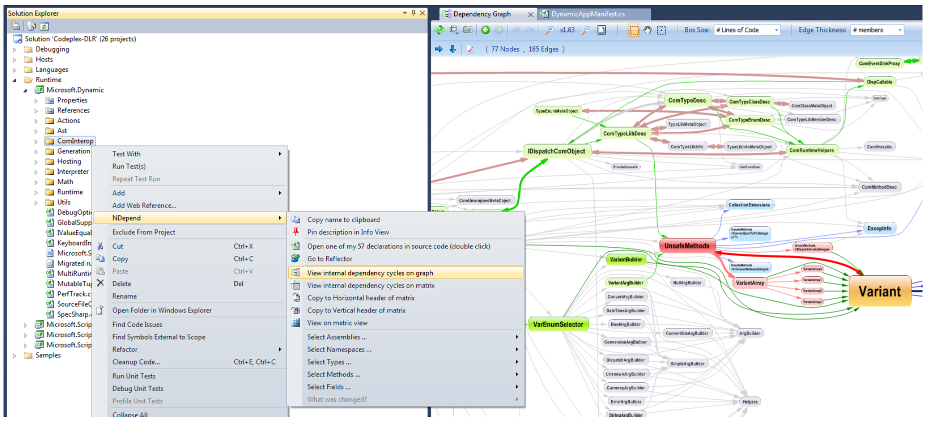
By default, the CppDepend dependency graph panel displays the graph of dependencies between C\C++ projects:



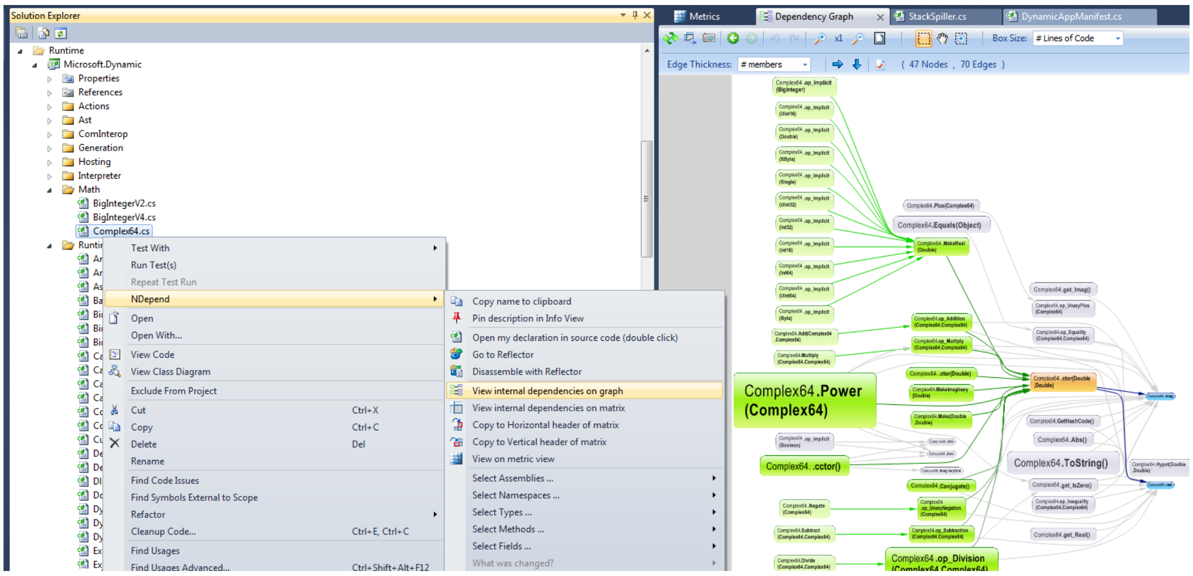
In the Solution Explorer right-click menu, CppDepend proposes to explore the graph of dependencies between namespaces.



In the Solution Explorer (or Code Editor window) right-click menu, CppDepend proposes to explore the graph of dependencies between types of a namespace. Notice that CppDepend comes with an heuristic to try to infer a namespace from a folder in Solution Explorer.



In the Solution Explorer (or Code Editor window) right-click menu, CppDepend proposes to explore the graph of dependencies between members (methods + fields) of a type. Notice that CppDepend comes with an heuristic to try to infer a type from a source file in Solution Explorer.



3. Call graph

CppDepend can generate any call graph you might need with a two steps procedure.

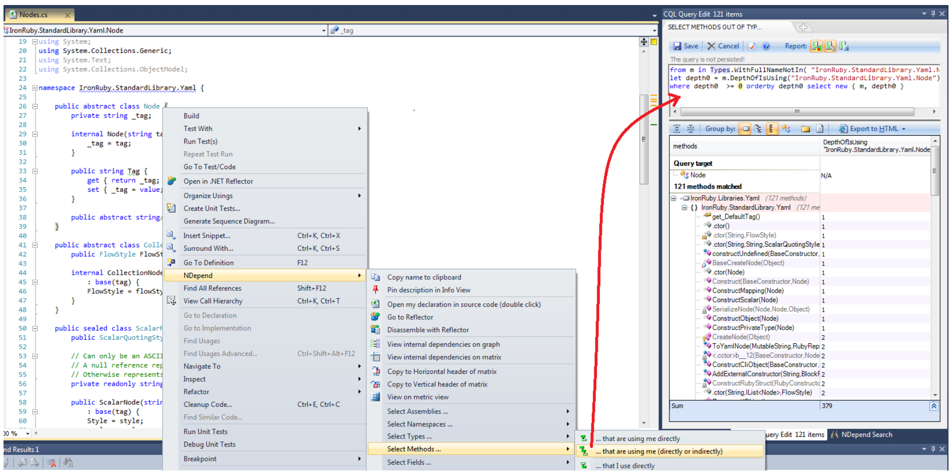
First: Ask for direct and indirect callers/callees of a type, a field, a method, a namespace or an assembly. The effect is that the following CQLinq query is generated to match all callers or callees asked.

```
from m in Types.WithFullNameNotIn( "IronRuby.StandardLibrary.Yaml.Nod
e").ChildMethods()
let depth0 = m.DepthOfIsUsing("IronRuby.StandardLibrary.Yaml.Node")
where depth0  >= 0 orderby depth0
select new { m, depth0 }
```

Notice that, in the CQLinq query result,

The metric *DepthOfIsUsing/DepthOfIsUsedBy* shows depth of usage (1 means direct, 2 means using a direct user etc...). The CQLinq query can easily be modified to only match indirect callers/callees with a certain condition on depth of usage.

Notice also that callers/callees asked are not necessarily of the same kind of the concerned code element. For example here we ask for methods that are using directly or indirectly a type.



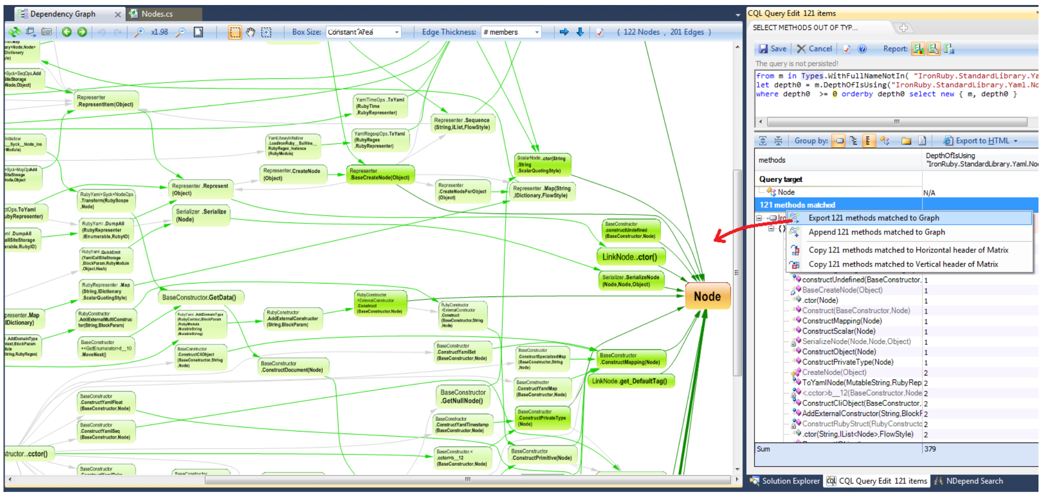
Nodes.cs
IronRuby.StandardLibrary.Yaml.Node
_tag
Build
Test With
Run Test(s)
Repeat Test Run
Go To Test/Code
Open in .NET Reflector
Organize Usings
Create Unit Tests...
Generate Sequence Diagram...
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Go To Definition F12
NDepend
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Go to Declaration
Go to Implementation
Find Usages
Find Usages Advanced... Ctrl+Shift+Alt+F12
Navigate To
Inspect
Refactor
Cleanup Code... Ctrl+E, Ctrl+C
Find Similar Code...
Run Unit Tests
Debug Unit Tests
Breakpoint
Copy name to clipboard
Pin description in Info View
Open my declaration in source code (double click)
Go to Reflector
Disassemble with Reflector
View internal dependencies on graph
View internal dependencies on matrix
Copy to Horizontal header of matrix
Copy to Vertical header of matrix
View on metric view
Select Assemblies ...
Select Namespaces ...
Select Types ...
Select Methods ...
Select Fields ...
... that are using me directly
... that are using me (directly or indirectly)
... that I use directly
CQL Query Edit 121 items
SELECT METHODS OUT OF TYP...
Save
Cancel
Report:
The query is not persisted!
Group by:
Export to HTML
Query Edit 121 items
NDepend Search
nd Results 1
00 %

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections.ObjectModel;

namespace IronRuby.StandardLibrary.Yaml {

    public abstract class Node {
        private string _tag;

        internal Node(string ta
            _tag = tag;
        }

        public string Tag {
            get { return _tag;
            set { _tag = value;
        }

        public abstract string
    }

    public abstract class Colle
        public FlowStyle FlowSt

        internal CollectionNode
            : base(tag) {
            FlowStyle = flowSty
        }
    }

    public sealed class ScalarN
        public ScalarQuotingSty

        // Can only be an ASCII
        // A null reference rep
        // Otherwise represents
        private readonly string

        public ScalarNode(strin
            : base(tag) {
            Style = style;
```

```
from m in Types.WithFullNameNotIn( "IronRuby.StandardLibrary.Yaml.N
let depth0 = m.DepthOfIsUsing("IronRuby.StandardLibrary.Yaml.Node")
where depth0  >= 0 orderby depth0 select new { m, depth0 }
```

methods	DepthOfIsUsing "IronRuby.StandardLibrary.Yaml.Node
Query target	
Node	N/A
121 methods matched	
IronRuby.Libraries.Yaml *(121 methods)*	
IronRuby.StandardLibrary.Yaml *(121 me*	
get_DefaultTag()	1
.ctor()	1
.ctor(String,FlowStyle)	1
.ctor(String,String,ScalarQuotingStyle	1
constructUndefined(BaseConstructor,	1
BaseCreateNode(Object)	1
.ctor(Node)	1
Construct(BaseConstructor,Node)	1
ConstructMapping(Node)	1
ConstructScalar(Node)	1
SerializeNode(Node,Node,Object)	1
ConstructObject(Node)	1
ConstructPrivateType(Node)	1
CreateNode(Object)	2
ToYamlNode(MutableString,RubyRep	2
<.cctor>b__12(BaseConstructor,Node	2
ConstructCliObject(BaseConstructor,	2
AddExternalConstructor(String,BlockF	2
ConstructRubyStruct(RubyConstructo	2
.ctor(String,IList<Node>,FlowStyle)	2
Sum	379

Second: Once the CQLinq query matches the set of callers/callees that the user wishes, the set of matches result can be exported to the Dependency Graph. This has for effect to show the call graph wished.



```
from m in Types.WithFullNameNotIn( "IronRuby.StandardLibrary.Ya
let depth0 = m.DepthOfIsUsing("IronRuby.StandardLibrary.Yaml.No
where depth0  >= 0 orderby depth0 select new { m, depth0 }
```

methods	DepthOfIsUsing "IronRuby.StandardLibrary.Yaml.No
Query target	
Node	N/A
121 methods matched	
constructUndefined(BaseConstructor,	1
BaseCreateNode(Object)	1
.ctor(Node)	1
Construct(BaseConstructor,Node)	1
ConstructMapping(Node)	1
ConstructScalar(Node)	1
SerializeNode(Node,Node,Object)	1
ConstructObject(Node)	1
ConstructPrivateType(Node)	1
CreateNode(Object)	2
ToYamlNode(MutableString,RubyRep	2
<.cctor>b__12(BaseConstructor,Node	2
ConstructCliObject(BaseConstructor,	2
AddExternalConstructor(String,BlockF	2
ConstructRubyStruct(RubyConstructo	2
.ctor(String,IList<Node>,FlowStyle)	2
Sum	379

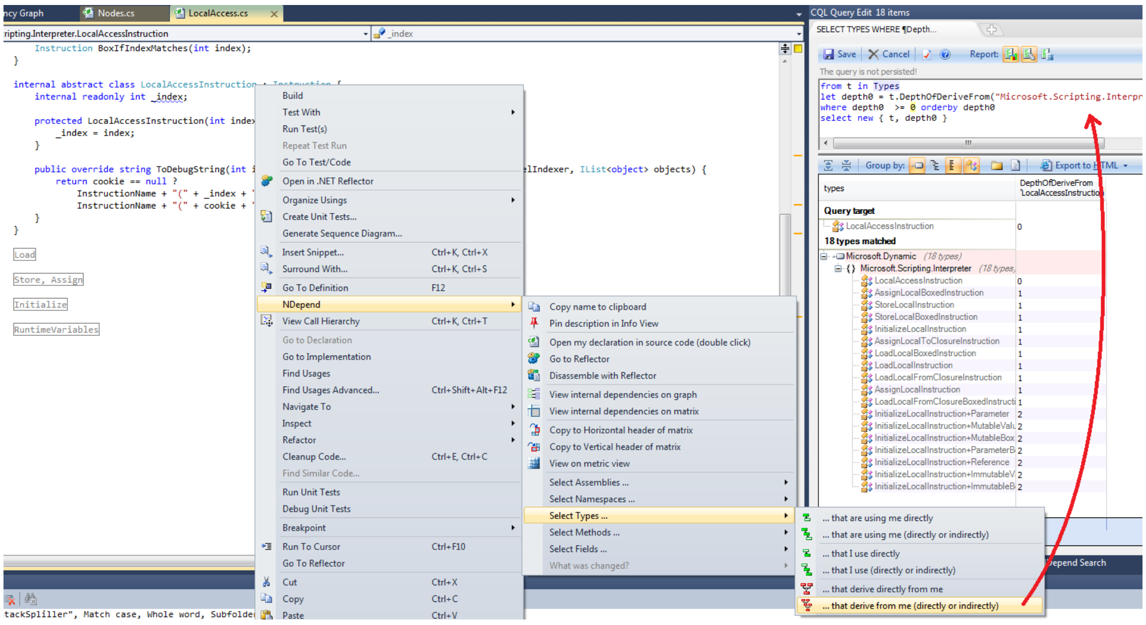
Dependency Graph
Nodes.cs
Box Size: Constant Area
Edge Thickness: # members
(122 Nodes , 201 Edges)
Node
LinkNode..ctor()
LinkNode.get_DefaultTag()
BaseConstructor.GetData()
CQL Query Edit 121 items
SELECT METHODS OUT OF TYP...
Save
Cancel
Report:
The query is not persisted!
Group by:
Export to HTML
Export 121 methods matched to Graph
Append 121 methods matched to Graph
Copy 121 methods matched to Horizontal header of Matrix
Copy 121 methods matched to Vertical header of Matrix
Solution Explorer
CQL Query Edit 121 items
NDepend Search

4. Class inheritance graph

To display a Class of Inheritance Graph, the same two steps procedure shown in the precedent section (on generating a Call Graph) must be applied.

First: Generate a CQLinq query asking for the set of classes that inherits from a particular class (or that implement a particular interface). Here, the following CQLinq query is generated:

```
from t in Types
let depth0 = t.DepthOfDeriveFrom("Microsoft.Scripting.Interpreter.LocalAccessInstruction")
where depth0  >= 0 orderby depth0
select new { t, depth0 }
```



ncy Graph
Nodes.cs
LocalAccess.cs
ripting.Interpreter.LocalAccessInstruction
_index
Build
Test With
Run Test(s)
Repeat Test Run
Go To Test/Code
Open in .NET Reflector
Organize Usings
Create Unit Tests...
Generate Sequence Diagram...
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Go To Definition F12
NDepend
View Call Hierarchy Ctrl+K, Ctrl+T
Go to Declaration
Go to Implementation
Find Usages
Find Usages Advanced... Ctrl+Shift+Alt+F12
Navigate To
Inspect
Refactor
Cleanup Code... Ctrl+E, Ctrl+C
Find Similar Code...
Run Unit Tests
Debug Unit Tests
Breakpoint
Run To Cursor Ctrl+F10
Go To Reflector
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy name to clipboard
Pin description in Info View
Open my declaration in source code (double click)
Go to Reflector
Disassemble with Reflector
View internal dependencies on graph
View internal dependencies on matrix
Copy to Horizontal header of matrix
Copy to Vertical header of matrix
View on metric view
Select Assemblies ...
Select Namespaces ...
Select Types ...
Select Methods ...
Select Fields ...
What was changed?
... that are using me directly
... that are using me (directly or indirectly)
... that I use directly
... that I use (directly or indirectly)
... that derive directly from me
... that derive from me (directly or indirectly)
CQL Query Edit 18 items
SELECT TYPES WHERE ¶Depth...
Save
Cancel
Report:
The query is not persisted!
Group by:
Export to HTML
Load
Store, Assign
Initialize
RuntimeVariables
pend Search

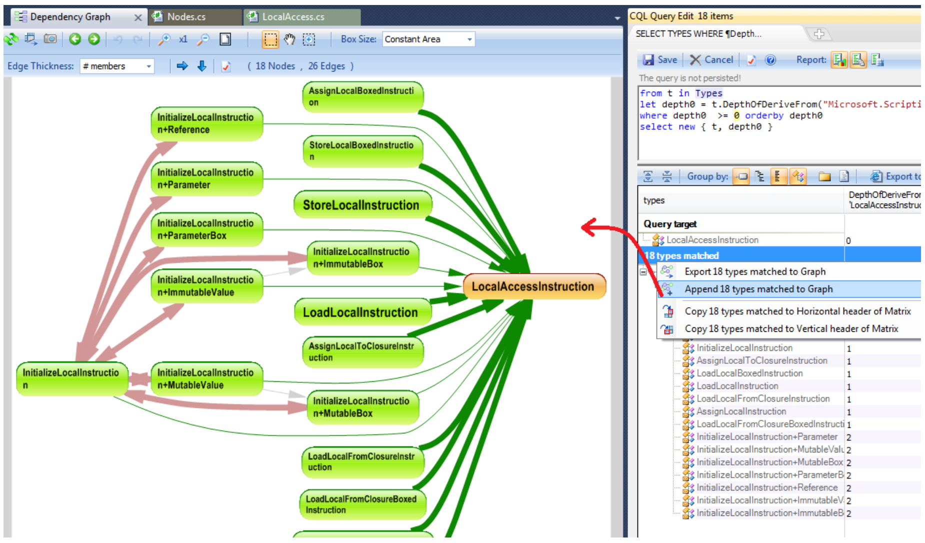
```
    Instruction BoxIfIndexMatches(int index);
}

internal abstract class LocalAccessInstruction : Instruction {
    internal readonly int _index;

    protected LocalAccessInstruction(int index
        _index = index;
    }

    public override string ToDebugString(int i                                  lIndexer, IList<object> objects) {
        return cookie == null ?
            InstructionName + "(" + _index + '
            InstructionName + "(" + cookie + '
    }
}
```

```
tackSplliler", Match case, Whole word, Subfolde
```

```
from t in Types
let depth0 = t.DepthOfDeriveFrom("Microsoft.Scripting.Interpr
where depth0  >= 0 orderby depth0
select new { t, depth0 }
```

types	DepthOfDeriveFrom 'LocalAccessInstructio
Query target	
LocalAccessInstruction	0
18 types matched	
Microsoft.Dynamic (18 types)	
Microsoft.Scripting.Interpreter (18 types,	
LocalAccessInstruction	0
AssignLocalBoxedInstruction	1
StoreLocalInstruction	1
StoreLocalBoxedInstruction	1
InitializeLocalInstruction	1
AssignLocalToClosureInstruction	1
LoadLocalBoxedInstruction	1
LoadLocalInstruction	1
LoadLocalFromClosureInstruction	1
AssignLocalInstruction	1
LoadLocalFromClosureBoxedInstructi	1
InitializeLocalInstruction+Parameter	2
InitializeLocalInstruction+MutableValu	2
InitializeLocalInstruction+MutableBox	2
InitializeLocalInstruction+ParameterB	2
InitializeLocalInstruction+Reference	2
InitializeLocalInstruction+ImmutableV	2
InitializeLocalInstruction+ImmutableB	2

Second: Export the result of the CQLinq query to the Dependency Graph to show the inheritance graph wished.


Dependency Graph
Nodes.cs
LocalAccess.cs
Box Size: Constant Area
Edge Thickness: # members
(18 Nodes , 26 Edges)
AssignLocalBoxedInstruction
InitializeLocalInstruction+Reference
StoreLocalBoxedInstruction
InitializeLocalInstruction+Parameter
StoreLocalInstruction
InitializeLocalInstruction+ParameterBox
InitializeLocalInstruction+ImmutableBox
InitializeLocalInstruction+ImmutableValue
LocalAccessInstruction
LoadLocalInstruction
AssignLocalToClosureInstruction
InitializeLocalInstruction
InitializeLocalInstruction+MutableValue
InitializeLocalInstruction+MutableBox
LoadLocalFromClosureInstruction
LoadLocalFromClosureBoxedInstruction
CQL Query Edit 18 items
SELECT TYPES WHERE ¶Depth...
Save
Cancel
Report:
Group by:
Export to
Export 18 types matched to Graph
Append 18 types matched to Graph
Copy 18 types matched to Horizontal header of Matrix
Copy 18 types matched to Vertical header of Matrix

The query is not persisted!

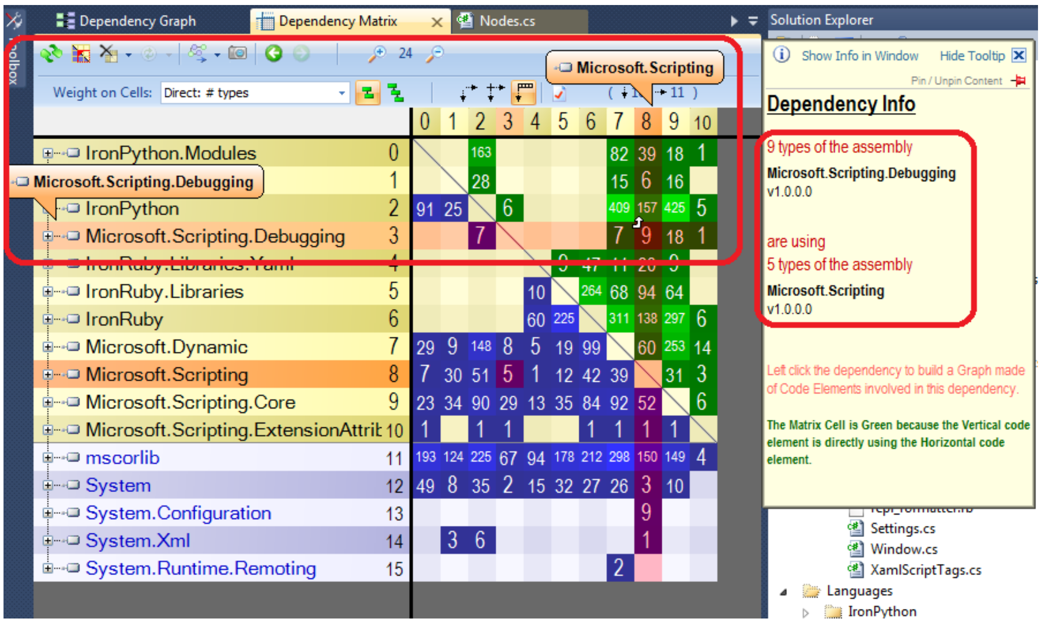
```
from t in Types
let depth0 = t.DepthOfDeriveFrom("Microsoft.Scripti
where depth0  >= 0 orderby depth0
select new { t, depth0 }
```

types	DepthOfDeriveFro 'LocalAccessInstru
Query target	
LocalAccessInstruction	0
18 types matched	
InitializeLocalInstruction	1
AssignLocalToClosureInstruction	1
LoadLocalBoxedInstruction	1
LoadLocalInstruction	1
LoadLocalFromClosureInstruction	1
AssignLocalInstruction	1
LoadLocalFromClosureBoxedInstructi	1
InitializeLocalInstruction+Parameter	2
InitializeLocalInstruction+MutableValu	2
InitializeLocalInstruction+MutableBox	2
InitializeLocalInstruction+ParameterB	2
InitializeLocalInstruction+Reference	2
InitializeLocalInstruction+ImmutableV	2
InitializeLocalInstruction+ImmutableB	2

5. Coupling graph

It might be needed to know which code elements exactly are involved in a particular dependency. Especially when one needs to anticipate the impact of a structural change. In the screenshoot below, the CppDepend Info panel describes a coupling between 2 assemblies.

From pointing a cell in the dependency matrix, it says that X types of an assembly A are using Y types of an assembly B. Notice that you can change the option *Weight* on *Cell* to # *methods*, # *members* or # *namespaces*, if you need to know the coupling with something else than types.



Weight on Cells: Direct: # types

		0	1	2	3	4	5	6	7	8	9	10
IronPython.Modules	0			163					82	39	18	1
Microsoft.Scripting.Debugging	1			28					15	6	16	
IronPython	2	91	25		6				409	157	425	5
Microsoft.Scripting.Debugging	3			7					7	9	18	1
IronRuby.Libraries.Yaml	4						9	47	11	20	9	
IronRuby.Libraries	5					10		264	68	94	64	
IronRuby	6					60	225		311	138	297	6
Microsoft.Dynamic	7	29	9	148	8	5	19	99		60	253	14
Microsoft.Scripting	8	7	30	51	5	1	12	42	39		31	3
Microsoft.Scripting.Core	9	23	34	90	29	13	35	84	92	52		6
Microsoft.Scripting.ExtensionAttrit	10	1		1	1			1	1	1	1	
mscorlib	11	193	124	225	67	94	178	212	298	150	149	4
System	12	49	8	35	2	15	32	27	26	3	10	
System.Configuration	13									9		
System.Xml	14		3	6						1		
System.Runtime.Remoting	15								2			

Dependency Info

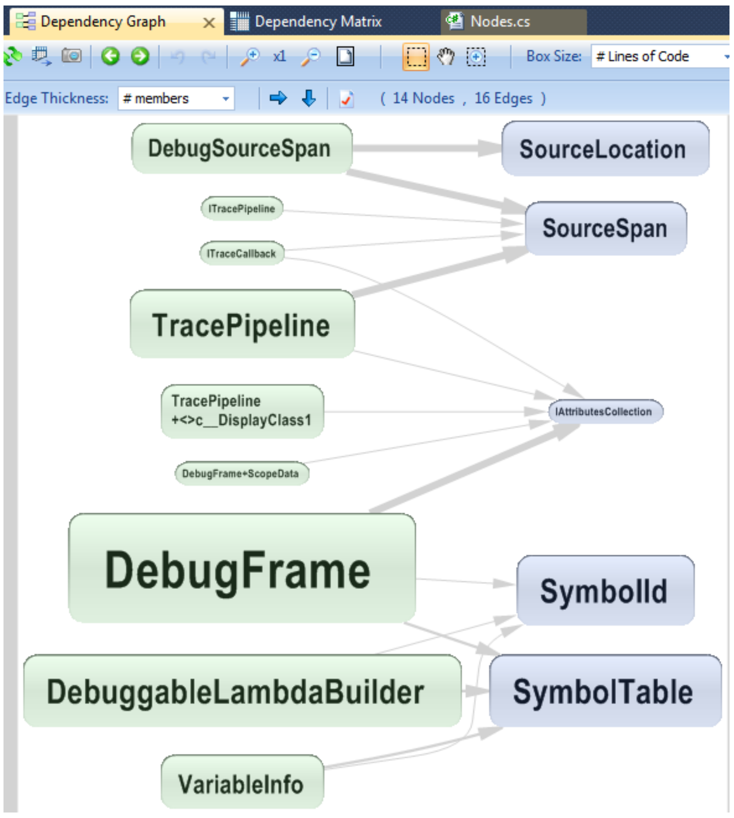
9 types of the assembly
Microsoft.Scripting.Debugging
v1.0.0.0

are using
5 types of the assembly
Microsoft.Scripting
v1.0.0.0

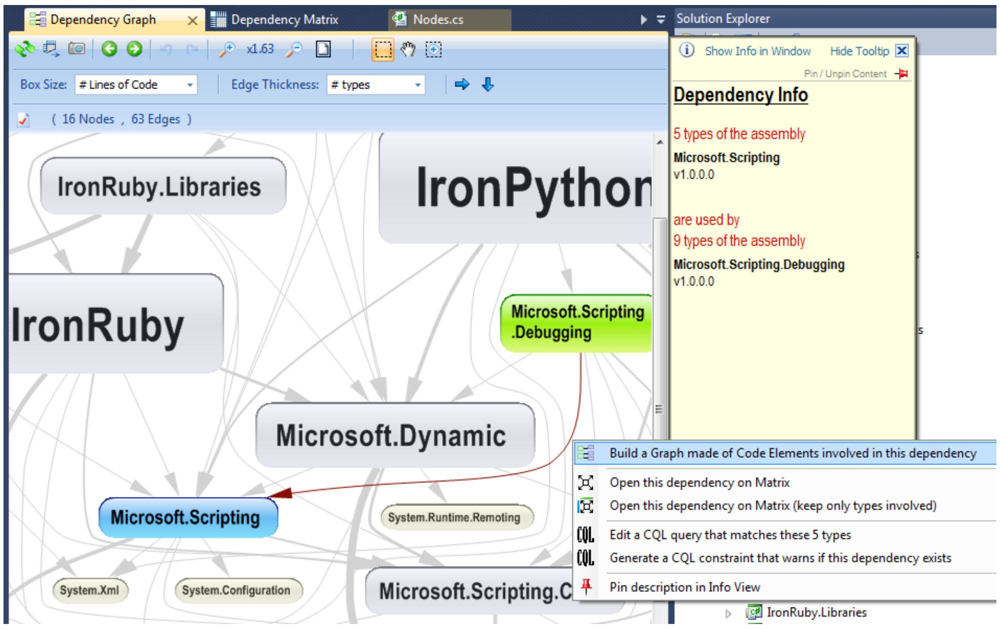
Left click the dependency to build a Graph made of Code Elements involved in this dependency.

The Matrix Cell is Green because the Vertical code element is directly using the Horizontal code element.

Just left clicking the matrix cell shows the coupling graph below.



A coupling graph can also be generated from an edge in the dependency graph. Here, you can adjust the option *Edge Thickness* to something else than # *type*.


Dependency Graph
Dependency Matrix
Nodes.cs
Solution Explorer
x1.63
Show Info in Window
Hide Tooltip
Pin / Unpin Content
IronRuby.Libraries
IronPython
IronRuby
Microsoft.Scripting.Debugging
Microsoft.Dynamic
Microsoft.Scripting
System.Runtime.Remoting
System.Xml
System.Configuration
Microsoft.Scripting.C
Build a Graph made of Code Elements involved in this dependency
Open this dependency on Matrix
Open this dependency on Matrix (keep only types involved)
Edit a CQL query that matches these 5 types
Generate a CQL constraint that warns if this dependency exists
Pin description in Info View
IronRuby.Libraries

Box Size: # Lines of Code   Edge Thickness: # types

(16 Nodes , 63 Edges)

Dependency Info

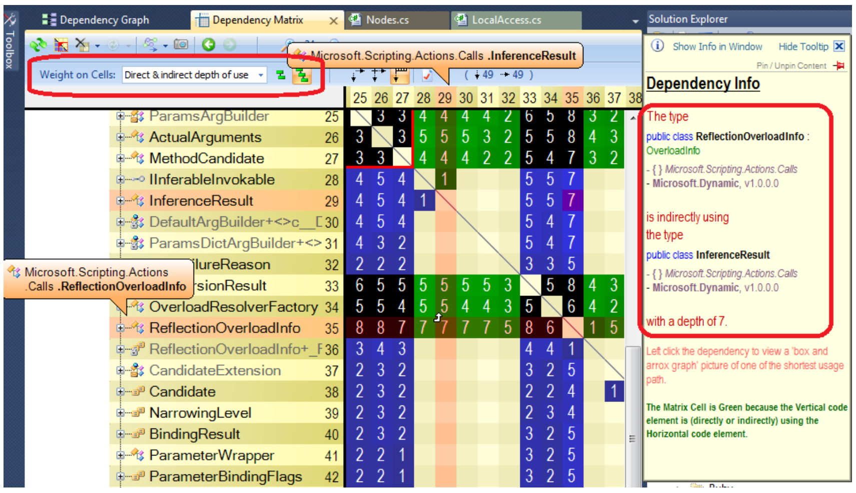
5 types of the assembly
Microsoft.Scripting
v1.0.0.0

are used by
9 types of the assembly
Microsoft.Scripting.Debugging
v1.0.0.0

6. Path graph

If you wish to dig into a path or a dependency cycle between 2 code elements, the first thing to do is to show the dependency matrix with the option *Weight* on *Cells*: **Direct & indirect depth of use**.

Matrix blue and green cells will represent *paths* while black cells will represent *dependency cycles*. For example, here, the Info panel tells us that there is a path of minimal length 7 between the 2 types involved.



Weight on Cells: Direct & indirect depth of use

		25	26	27	28	29	30	31	32	33	34	35	36	37	38
ParamsArgBuilder	25		3	3	4	4	4	4	2	6	5	8	3	2	
ActualArguments	26	3		3	5	5	5	3	2	5	5	8	4	3	
MethodCandidate	27	3	3		4	4	4	2	2	5	4	7	3	2	
IInferableInvokable	28	4	5	4		1				5	5	7			
InferenceResult	29	4	5	4	1					5	5	7			
DefaultArgBuilder+<>c__[	30	4	5	4						5	4	7			
ParamsDictArgBuilder+<>	31	4	3	2						5	4	7			
lureReason	32	2	2	2						3	3	5			
sionResult	33	6	5	5	5	5	5	5	3		5	8	4	3	
OverloadResolverFactory	34	5	5	4	5	5	4	4	3	5		6	4	2	
ReflectionOverloadInfo	35	8	8	7	7	7	7	7	5	8	6		1	5	
ReflectionOverloadInfo+_F	36	3	4	3						4	4	1			
CandidateExtension	37	2	3	2						3	2	5			
Candidate	38	2	3	2						2	2	4		1	
NarrowingLevel	39	2	3	2						2	3	4			
BindingResult	40	2	3	2						3	2	5			
ParameterWrapper	41	2	2	1						3	2	5			
ParameterBindingFlags	42	2	2	1						3	2	5			

Dependency Info

The type
public class **ReflectionOverloadInfo** : OverloadInfo
- { } *Microsoft.Scripting.Actions.Calls*
- **Microsoft.Dynamic**, v1.0.0.0

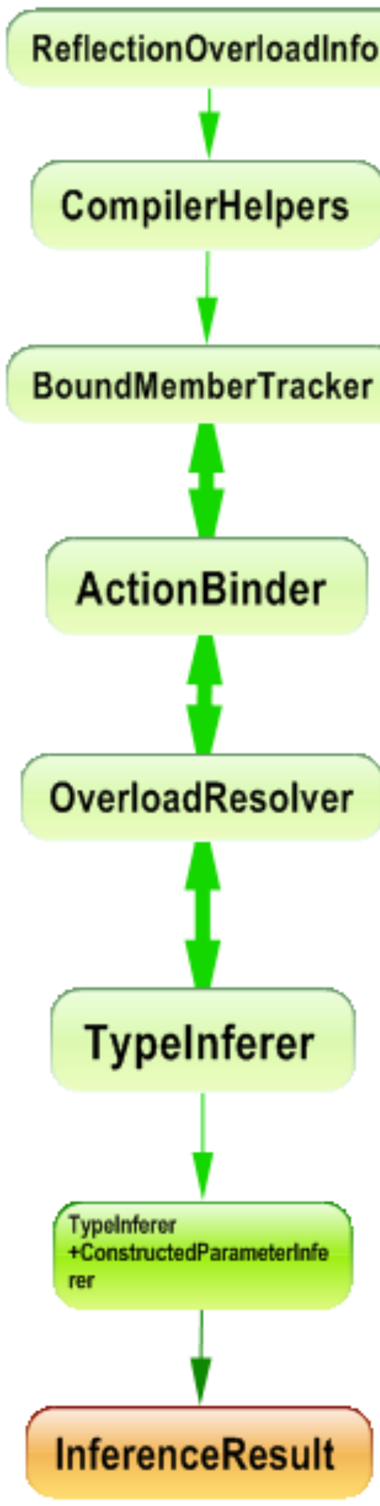
is indirectly using
the type
public class **InferenceResult**
- { } *Microsoft.Scripting.Actions.Calls*
- **Microsoft.Dynamic**, v1.0.0.0

with a depth of 7.

Left click the dependency to view a 'box and arrox graph' picture of one of the shortest usage path.

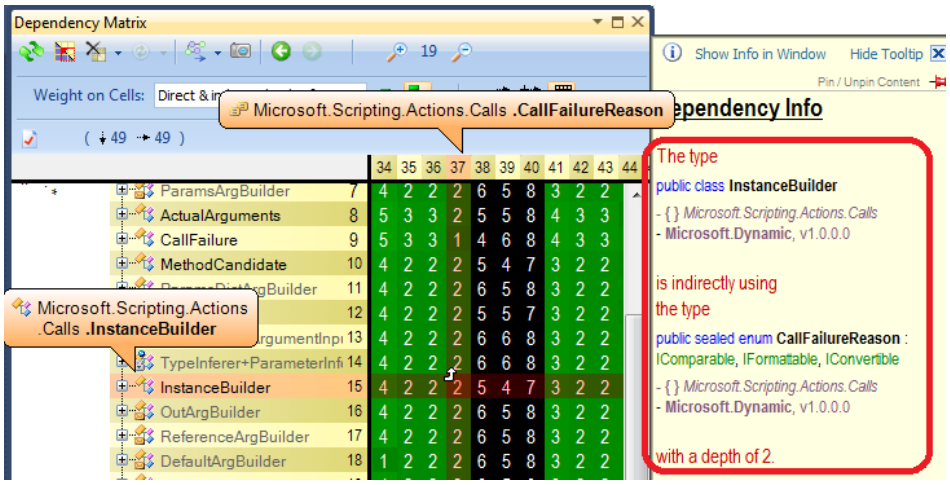
The Matrix Cell is Green because the Vertical code element is (directly or indirectly) using the Horizontal code element.

Just left clicking the cell shows the path graph below.



7. All Paths graph

In certain situations, you'll need to know about all paths from a code element A to a code element B. For example, here, the Info panel tells us that there is a path of minimal length 2 between the 2 types involved.


Dependency Matrix
19
Weight on Cells: Direct & in
(49 49)
Show Info in Window
Hide Tooltip
Pin / Unpin Content
Microsoft.Scripting.Actions.Calls .CallFailureReason
Microsoft.Scripting.Actions .Calls .InstanceBuilder
ependency Info

		34	35	36	37	38	39	40	41	42	43	44
ParamsArgBuilder	7	4	2	2	2	6	5	8	3	2	2	
ActualArguments	8	5	3	3	2	5	5	8	4	3	3	
CallFailure	9	5	3	3	1	4	6	8	4	3	3	
MethodCandidate	10	4	2	2	2	5	4	7	3	2	2	
gBuilder	11	4	2	2	2	6	5	8	3	2	2	
	12	4	2	2	2	5	5	7	3	2	2	
rgumentInpu	13	4	2	2	2	6	6	8	3	2	2	
TypeInferer+ParameterInf	14	4	2	2	2	6	6	8	3	2	2	
InstanceBuilder	15	4	2	2	2	5	4	7	3	2	2	
OutArgBuilder	16	4	2	2	2	6	5	8	3	2	2	
ReferenceArgBuilder	17	4	2	2	2	6	5	8	3	2	2	
DefaultArgBuilder	18	1	2	2	2	6	5	8	3	2	2	

The type
public class **InstanceBuilder**
- { } *Microsoft.Scripting.Actions.Calls*
- **Microsoft.Dynamic**, v1.0.0.0

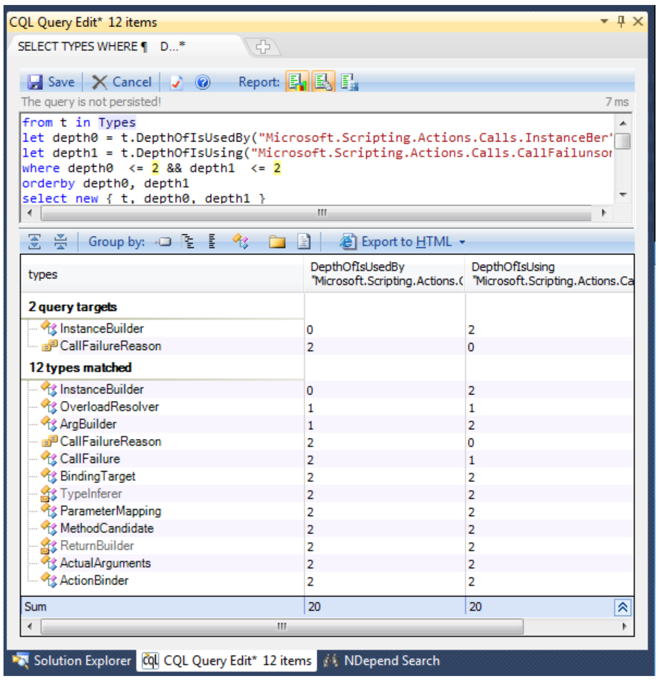
is indirectly using
the type
public sealed enum **CallFailureReason** :
IComparable, IFormattable, IConvertible
- { } *Microsoft.Scripting.Actions.Calls*
- **Microsoft.Dynamic**, v1.0.0.0

with a depth of 2.

Right clicking the matrix's cell and selecting the option **Edit a code query that matches paths** generates the following CQLinq query that matchs all types involved in all paths from type A to type B.

```
from t in Types
let depth0 = t.DepthOfIsUsedBy("Microsoft.Scripting.Actions.Calls.Ins
tanceBuilder")
let depth1 = t.DepthOfIsUsing("Microsoft.Scripting.Actions.Calls.Call
FailureReason")
where depth0  <= 2 && depth1  <= 2
orderby depth0, depth1
select new { t, depth0, depth1 }
//-------------------------------------------------------------------
-
// The type
// public class InstanceBuilder
// - { } Microsoft.Scripting.Actions.Calls
// - Microsoft.Dynamic, v1.0.0.0
//
// is indirectly using
// the type
// public sealed enum CallFailureReason : IComparable, IFormattable
, IConvertible
```

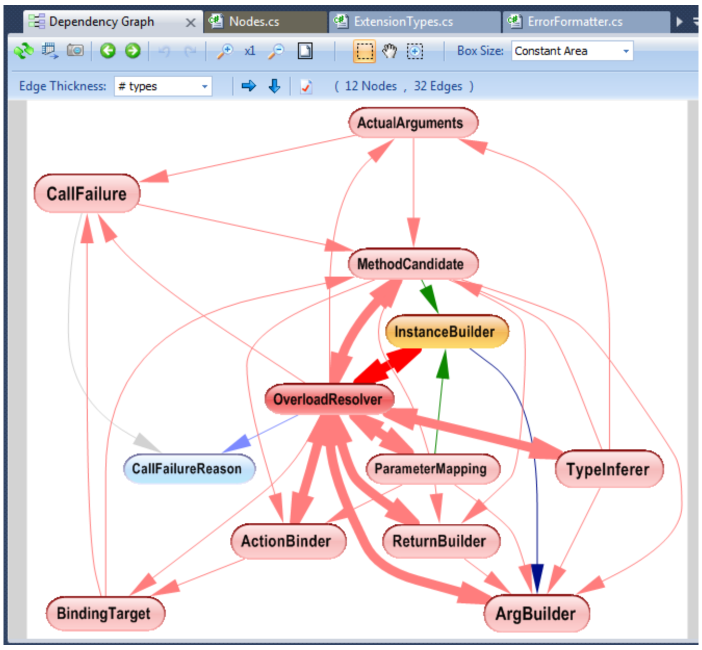
```
// - { } Microsoft.Scripting.Actions.Calls
// - Microsoft.Dynamic, v1.0.0.0
//
// with a depth of 2.
```



```
from t in Types
let depth0 = t.DepthOfIsUsedBy("Microsoft.Scripting.Actions.Calls.InstanceBer'
let depth1 = t.DepthOfIsUsing("Microsoft.Scripting.Actions.Calls.CallFailunsor
where depth0  <= 2 && depth1  <= 2
orderby depth0, depth1
select new { t, depth0, depth1 }
```

types	DepthOfIsUsedBy "Microsoft.Scripting.Actions.(	DepthOfIsUsing "Microsoft.Scripting.Actions.Ca
2 query targets		
InstanceBuilder	0	2
CallFailureReason	2	0
12 types matched		
InstanceBuilder	0	2
OverloadResolver	1	1
ArgBuilder	1	2
CallFailureReason	2	0
CallFailure	2	1
BindingTarget	2	2
TypeInferer	2	2
ParameterMapping	2	2
MethodCandidate	2	2
ReturnBuilder	2	2
ActualArguments	2	2
ActionBinder	2	2
Sum	20	20

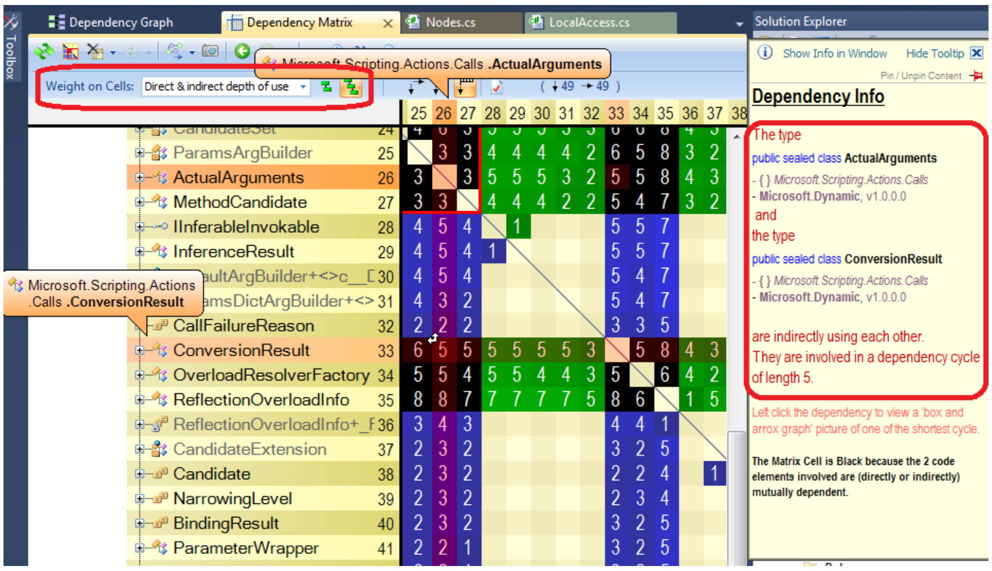
Finally exporting to the graph the 12 types matched by the CQLinq query shows all paths from A to B.



8. Cycle graph

As we explained in the previous section, to deal with dependency cycle graphs, the first thing to do is to show the dependency matrix with the option *Weight* on *Cells*: **Direct & indirect depth of use**. Black cells then represent cycles.

For example, here, the Info panel tells us that there is a dependency cycle of minimal length 5 between the 2 types involved.



Weight on Cells: Direct & indirect depth of use

		25	26	27	28	29	30	31	32	33	34	35	36	37
ParamsArgBuilder	25		3	3	4	4	4	4	2	6	5	8	3	2
ActualArguments	26	3		3	5	5	5	3	2	5	5	8	4	3
MethodCandidate	27	3	3		4	4	4	2	2	5	4	7	3	2
IInferableInvokable	28	4	5	4		1				5	5	7		
InferenceResult	29	4	5	4	1					5	5	7		
ultArgBuilder+<>c__[	30	4	5	4						5	4	7		
amsDictArgBuilder+<>	31	4	3	2						5	4	7		
CallFailureReason	32	2	2	2						3	3	5		
ConversionResult	33	6	5	5	5	5	5	5	3		5	8	4	3
OverloadResolverFactory	34	5	5	4	5	5	4	4	3	5		6	4	2
ReflectionOverloadInfo	35	8	8	7	7	7	7	7	5	8	6		1	5
ReflectionOverloadInfo+_F	36	3	4	3						4	4	1		
CandidateExtension	37	2	3	2						3	2	5		
Candidate	38	2	3	2						2	2	4		1
NarrowingLevel	39	2	3	2						2	3	4		
BindingResult	40	2	3	2						3	2	5		
ParameterWrapper	41	2	2	1						3	2	5		

Dependency Info

The type
public sealed class **ActualArguments**
- { } *Microsoft.Scripting.Actions.Calls*
- **Microsoft.Dynamic**, v1.0.0.0

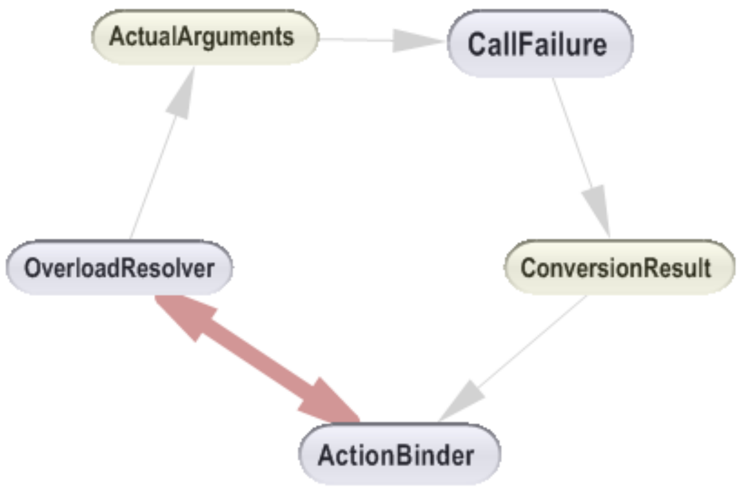
and
the type
public sealed class **ConversionResult**
- { } *Microsoft.Scripting.Actions.Calls*
- **Microsoft.Dynamic**, v1.0.0.0

are indirectly using each other.
They are involved in a dependency cycle of length 5.

Left click the dependency to view a 'box and arrox graph' picture of one of the shortest cycle.

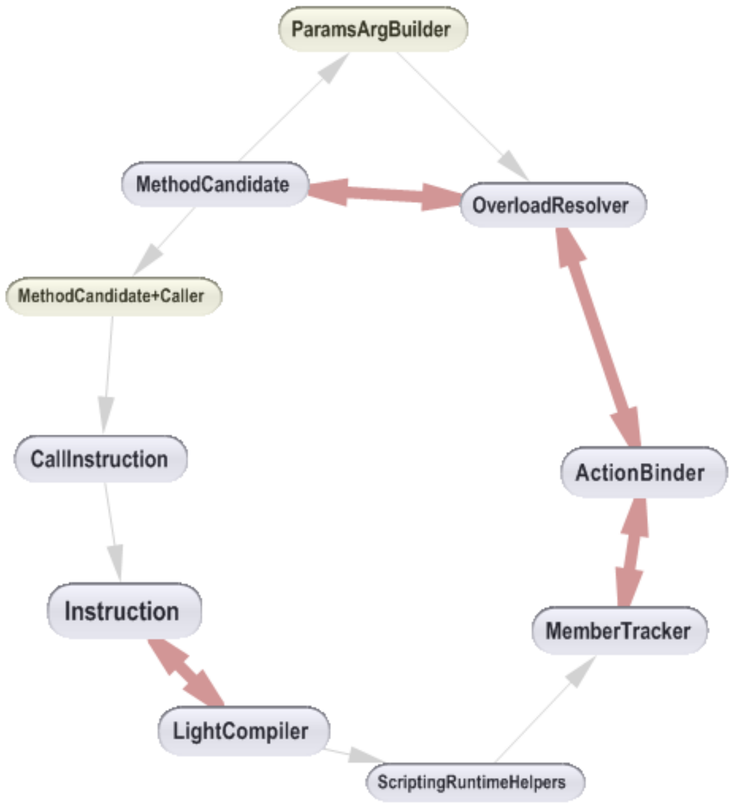
The Matrix Cell is Black because the 2 code elements involved are (directly or indirectly) mutually dependent.

Just left clicking the cell shows the cycle graph below:



We'd like to warn that obtaining a clean 'rounded' dependency cycle as the one shown above, is actually more an exceptional situation than a rule.

Often, exhibiting a cycle will end up in a not 'rounded' graph as the one shown below. In this example, the minimal length of a cycle between the 2 types involved (in yellow) is 12. Count the number of edges crossed from one yellow type to the other one, and you'll get 12. You'll see that some edges will be counted more than once.



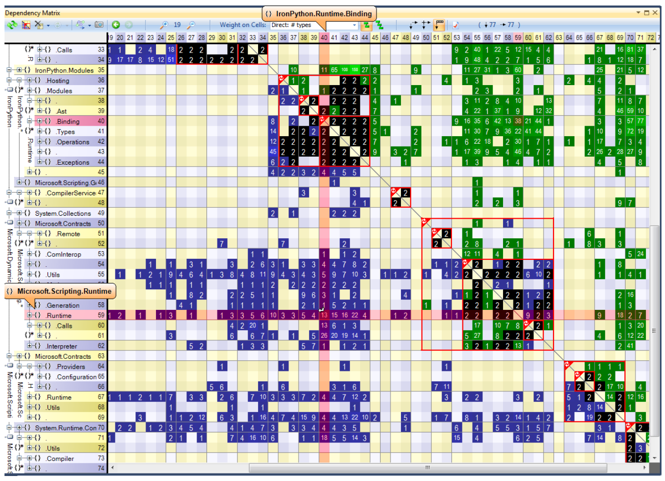
9. Large graph visualized with Dependency Structure Matrix

Here, we'd like to underline the fact that when the dependency Graph becomes unreadable, it is worth switching to the dependency Matrix.

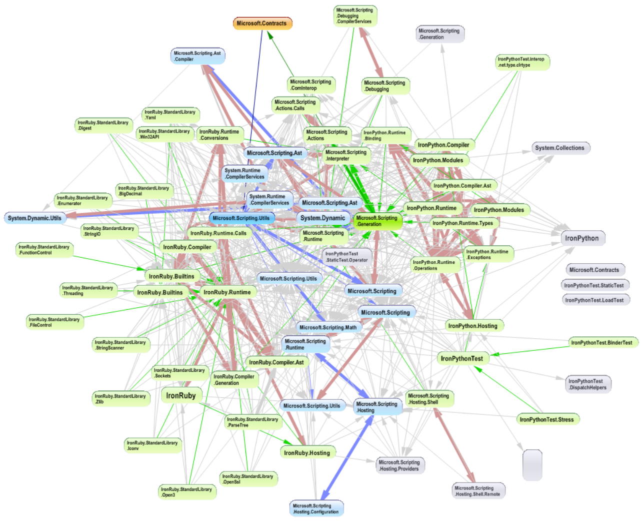
Both dependency Graph and dependency Matrix co-exist because:

- Dependency Graph is intuitive but becomes unreadable as soon as there are too many edges between nodes.
- Dependency Matrix requires time to be understood, but once mastered, you'll see that the Dependency Matrix is much more efficient than **the Dependency Graph to explore an existing architecture**.

To illustrate the point, find below the same dependencies between 77 namespaces shown through Dependency Graph and Dependency Matrix.



Dependency Matrix
IronPython.Runtime.Binding
Microsoft.Scripting.Runtime



Microsoft.Contracts
System.Dynamic.Utils
System.Dynamic
System.Collections
IronPython
IronPython.Compiler
IronPython.Modules
IronPython.Runtime
IronPython.Hosting
IronPythonTest
IronRuby
IronRuby.Compiler
IronRuby.Builtins
IronRuby.Runtime
IronRuby.Hosting
Microsoft.Scripting
Microsoft.Scripting.Ast
Microsoft.Scripting.Utils