Checkmarx CxEnterprise
CxQuery API Guide

V8.4.1

December, 2016

Contents

1 Preface

The CxQuery API Guide documents the Checkmarx Query Language (CxQL) used in CxAudit to query source code.

CxQL allows us to virtually data-mine any aspect of the source, and to build custom queries.

Checkmarx-provided queries are written using the CxQL.

These queries can be inherited, expanded, or rewritten.

Note: CxQL queries are language-dependent.

2 Introduction

A query written in Checkmarx Query Language allows us to analyze the scanned code and return a list of results.

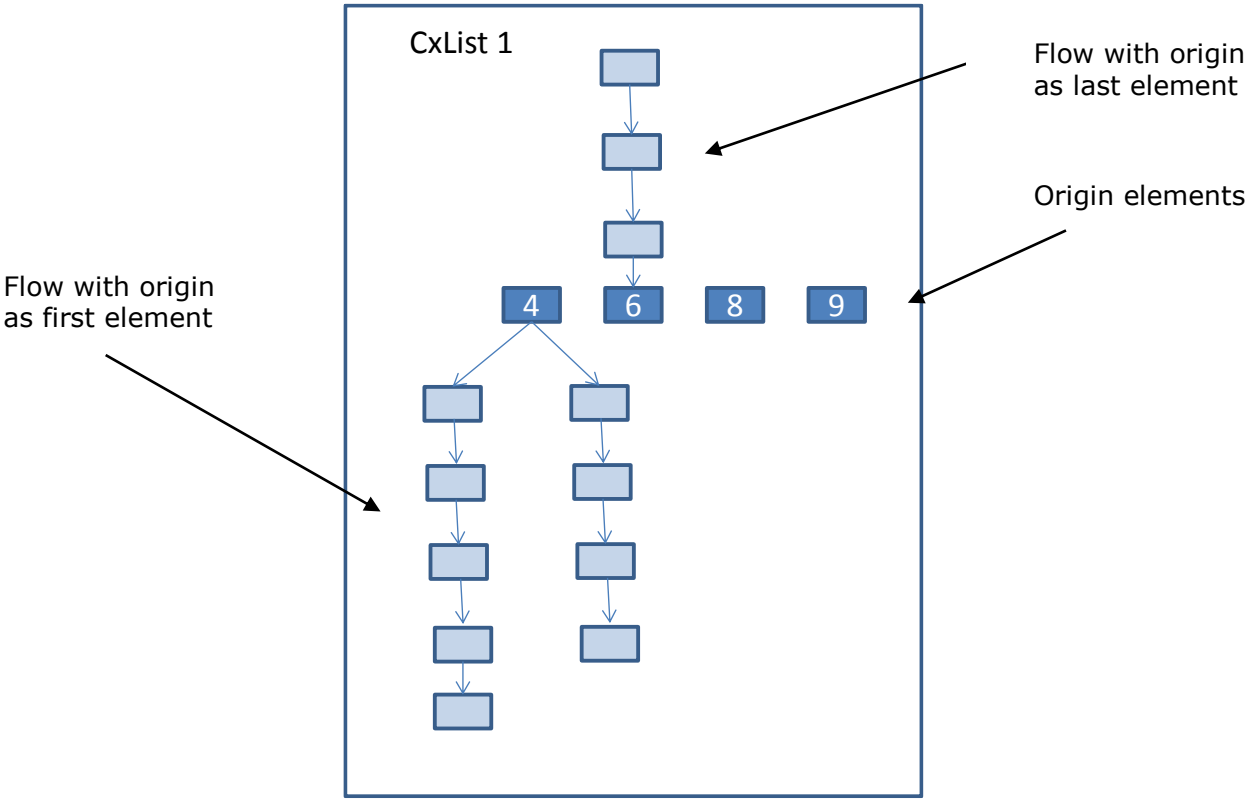
Each result can be an element in the scanned code (e.g. a variable) or a "flow" – a path in the code consisting of an ordered list of these elements.

2.1 Definitions

Basic code element – Code elements such as variables, method invocations and assignments that have representation in the code model.

Data flow (flow) – an ordered list of code elements that represent a possible data change progression in the program from a certain location where the data has changed and the end location where that change had an affect (as a subsequent data change).
Every flow is attached to an origin basic code element. This origin element may be the first or last code element in the flow. The origin element appears as the first element in the flow if that element was queried as to whether it influenced other elements. The origin element appears as the last element in the flow if that element was queried as to whether it was influenced by other elements.

CxList - the central data type in CxQL. The CxList is a list that consists of basic code elements such as variables, method invocations, assignments, and so forth. Each element may have an attached flow, if the element was added to the CxList because it fulfilled a certain flow query.



There are two special CxList objects by default:

All – contains all elements in the scanned code, and

result – the return value of the query.

Note: All contains only basic code elements (without any flow).

2.2 Queries and Commands:

Now we are ready for our first query:

```
CxQL

 This example demonstrates the use of "All" and "result" objects
result = All;

 This would return a list of all objects in the code for a specific
language
```

CxList includes a vast assortment of commands. In the following example, we investigate the CxList FindByName command:

```
CxQL

result = All.FindByName("*MyName*")
 This would return a list of all objects where their name
 contains the string "MyName".

 It is the same as:
CxList cml = All.FindByName("*MyName*");
result = cml;
```

Because the return value of almost every command is also of type CxList, several commands can be executed consecutively as shown in the example below.

It is important to note that most CxList methods return a subset of the original CxList (we can think of the method as a **filter**).

So in the example below, consisting of chained method calls:

```
All.FindByName("*.MyName").FindByType (typeof(MemberAccess))
```

The order of execution is:

1. Return a CxList consisting of a subset of "All" (all elements in code) with name containing MyName .
2. Return a subset of the previous result , only those of type MemberAccess.

```
CxQL

result = All.FindByName("*.MyName").FindByType (typeof(MemberAccess));
 This would return a list of all access data members in the code whose
name contains the string "MyName" (e.g. a = b.MyName ).
```

```
 First we find all objects whose name ends with ".MyName", and on the
result we execute another command that retrieves only access members.
 This is the same as the following:

result = All.FindByType (typeof(MemberAccess)).FindByName("*.MyName");
* The difference is in efficiency. We want to work on the smallest
groups possible, so actually first looking by name and then by type
should be more efficient.
```

While the result in both cases is identical (order of filtering doesn't matter), the choice of execution order can have a noticeable effect on performance.

2.2.1 Data Flow Graph

We have seen in the previous section several commands that can operate on CxList objects. All the commands were "static" since they locate elements in the code, but they do not capture the flow between elements. The Data Flow Graph (DFG) in Source Code Analysis (SCA) describes how data is flowed through the program. Object A is "data influenced by" object B if the value of B flows to A.

In the example below, **d** is "data influenced by" **a** and **b**, but not by **c**. This means that both **a** and **b** are "data influencing on" **d**, but not on **c**.

```
C#

a = 5;
b = 6;
c = 7;
d = a + b;
```

3 Using CxDebug

The CxDebug method is a way to output debug messages from within a query, in the CxAudit environment.

The messages can be seen in the CxAudit bottom window, in the tab named "Debug Messages".

The most common case is when exceptions happen, so that the exception details can be viewed after the query has finished.

For example:

```
CXQL
 foreach (CxList rt in redirectThings)
       {
              try
              {
                     […]
              }
              catch (Exception ex)
              {
                     // in case of an exception in the loop
                     CxDebug(ex);
              }
       }
```

It can also be used for more detailed inspection of the query behavior from within the query itself.

For example:

```
CXQL
       if(hexEquiv != "")
       {
              CxDebug("hexEquiv=" + hexEquiv + ", #finds=" +
finds.Count);
              count++;
       } else {
              CxDebug("hexEquiv=empty" + hexEquiv + ", #finds=" +
finds.Count);
       }
```

Note that CxDebug cannot display CxList data directly. Executing **CxDebug(myCxList)** will yield just an integer value.

However, in many cases (when the CxList does not represent a path), one can retrieve and output the CxList element fields.

For example:

```
CXQL
CxList inputs = Find_Inputs();
foreach (CxList inp in inputs)
{
       CSharpGraph inp_Graph = inp.data.GetByIndex(0) as CSharpGraph;
       String inp_Name = inp_Graph.ShortName;
       CxDebug("Name = " + inp_Name);
}
```

4 Methods documentation

4.1 CxList.Add Method (int, IGraph)

Adds to the current instance the given graph node, indexed by the given id.

Syntax

```
CxQL
public void Add(int id, IGraph node)
```

Parameters

Id

Id of the node to be added to the graph node.

Node

Graph node to be associated to the given Id.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Example

```
CxQL

 This example demonstrates the CxList.Add() method.
 The input source code is:

 int b, a = 5;
 if (a == 33)
      b = 6;

CxList myList = All.FindByName("a");
CSharpGraph nodeGraph = All.FindByName("b").GetFirstGraph();
myList.Add(nodeGraph.NodeId, nodeGraph);
result = myList;
   The resulting list will include the initial two "a"'s and the first
b
```

4.2 CxList.Add Method (CxList)

Add all the elements from the given CxList to the current instance.

Syntax

```
CxQL
public void Add(CxList list)
```

Parameters

list

The CxList to be added to the current CxList instance.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Example

```
CxQL

 This example demonstrates the CxList.Add() method.
 The input source code is:

 int b, a = 5;
 if (a == 33)
      b = 6;

CxList list_a = All.FindByName("a");
CxList list_b = All.FindByName("b");
list_a.Add(list_b);
result = list_a;
 The resulting list will contain 4 elements
```

4.3 CxList.Add Method (KeyValuePair<int, IGraph>)

Add the given pair to the current CxList instance.

Syntax

```
CxQL
public void Add(KeyValuePair<int, IGraph> dictionary)
```

Parameters

dictionary

Pair to be added to the current CxList instance.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Example

```
CxQL

 This example demonstrates the CxList.Add() method.
 The input source code is:

 int b, a = 5;
 if (a == 33)
      b = 6;

CxList myList = All.FindByName("a");
foreach(KeyValuePair<int,IGraph> entry in All.FindByName("b"))
{
      myList.Add(entry);
}
result = myList;
 The resulting list will contain 4 elements
```

4.4 CxList.CallingMethodOfAny Method (CxList)

Returns a CxList which is a subset of "this" instance and are methods or constructors declarations which matches the given CxList elements.

Syntax

```
CxQL
public CxList CallingMethodOfAny(CxList elements)
```

Parameters

elements

The list of elements containing the methods or constructors to look for their declaration.

Return Value

The methods or constructor declarations which matches the given CxList elements.

Example

```
CxQL

 This example demonstrates the CxList.CallingMethodOfAny() method.
 The input source code is:
 void foo()
 {
      int goo = 3;
      int boo = 5;
 }

result = All.CallingMethodOfAny(All.FindByName ("oo"));

 The result would consist of 1 item:
      foo (in void foo())
```

4.5 CxList.Clear Method ()

Clears the information in "this" instance.

Syntax

```
CxQL
public bool Clear()
```

Parameters

None

Return Value

None

Comments

This method removes all the information stored in the List.

Example

```
CxQL

 This example demonstrates the CxList.Clear() method.

CxList MyList = All;
MessageBox.Show(MyList.Count.ToString());

MyList.Clear();
MessageBox.Show(MyList.Count.ToString());
```

4.6 CxList Concatenate Methods

4.6.1 CxList.Concatenate Method (CxList list, bool _testFlow)

Concatenates two nodes into a flow.

Syntax

```
CxQL
public CxList Concatenate (CxList list, bool _testFlow)
```

Parameters

list

A CxList containing one node only. This node will be concatenated to **this** instance

_testFlow

If true, searches for a flow between **this** instance and **list**. Otherwise, connects the two nodes directly (more efficient).

Return Value

A flow that starts with **this** instance node, and ends with the **list** parameter node.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

1. If either **this** instance or **list** parameter contains more than one node or contains flows, the function return value is undefined.
2. This function is deprecated, use **ConcatenatePath** instead.

Example

The following code example shows how you can use the Concatenate method.

```
CxQL

void main()
{
	int a = 1;
	int b = 2;
	int c = a + b;
	printf("%d", c);
}

CxList one = All.FindByName("1");
CxList two = All.FindByName("2");
result = one.Concatenate(two);

 the result would be -
```

```
1 flow found:
    [1] -> [2]
```

Version Information

Supported from **CxAudit** v7.1.2

4.6.2 CxList.Concatenate Method (CxList list)

Concatenates two nodes into a flow.

Syntax

```
CxQL
public CxList Concatenate (CxList list)
```

Parameters

list

A CxList containing one node only. This node will be concatenated to **this** instance

Return Value

A flow that starts with **this** instance node, and ends with the **list** parameter node.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

1. This function calls CxList.Concatenate(list, false).
2. If either this instance or list parameter contains more than one node or contains flows, the function return value is undefined.
3. This function is deprecated, use ConcatenatePath instead.

Version Information

Supported from **CxAudit** v7.1.2

4.6.3 CxList.ConcatenatePath Method (CxList list, bool _testFlow)

Concatenates two flows into one connected flow.

Syntax

```
CxQL
public CxList ConcatenatePath (CxList list, bool _testFlow)
```

Parameters

list

A CxList containing one flow only. This flow will be concatenated to **this** instance

_testFlow

If true, searches for a flow between **this** instance and **list**. Otherwise, connects the two flows directly (more efficient).

Return Value

A flow that starts with **this** instance flow, and ends with the **list** parameter flow.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

Both **this** instance and **list** have to contain only one flow (or one node as a private case), otherwise return value is undefined.

Example

The following code example shows how you can use the ConcatenatePath method.

```
CxQL

void main()
{
      int a = 1;
      int b = 2;
}

CxList one = All.FindByName("1");
CxList a = All.FindByShortName("a").FindByType(typeof(Declarator));
//Declarator is a new type defined in cXQL
CxList flow1 = a.InfluencedBy(one); // [1] -> [a]

CxList two = All.FindByName("2");
CxList b = All.FindByShortName("b").FindByType(typeof(Declarator));
CxList flow2 = b.InfluencedBy(two); // [2] -> [b]

result = flow2.ConcatenatePath(flow1);

 the result would be -
     1 flow found:
           [2] -> [b] -> [1] -> [a]
```

Version Information

Supported from **CxAudit** v7.1.2

4.6.4CxList.ConcatenatePath Method (CxList list)

Concatenates two flows into one connected flow.

Syntax

```
CxQL
```

```
public CxList ConcatenatePath (CxList list)
```

Parameters

list

A CxList containing one flow only. This flow will be concatenated to **this** instance

Return Value

A flow that starts with **this** instance flow, and ends with the **list** parameter flow.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

1. This function calls CxList.ConcatenatePath(list, true).
2. Both this instance and list have to contain only one flow (or one node as a private case), otherwise return value is undefined.

Version Information

Supported from **CxAudit** v7.1.2

4.6.5 CxList.ConcatenateAllPaths Method (CxList list, bool _testFlow)

Concatenates all flows in this instance to all flows in **list**.

Syntax

```
CxQL
public CxList ConcatenateAllPaths (CxList list, bool _testFlow)
```

Parameters

list

A CxList containing flows. These flow will be concatenated to the flows in **this** instance

_testFlow

If true, searches for a flow between **this** instance and **list**. Otherwise, connects the two flows directly (more efficient).

Return Value

A product of all flows in **this** instance with the ones in **list** parameter.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

If **this** instance contains ***n*** flows in it and **list** contains ***m*** flows in it, the return set will contain ***nxm*** flows, where each flow from **this** instance will be concatenated to each flow from **list**.

Example

The following code example shows how you can use the ConcatenateAllPaths method.

```
CxQL

void main()
{
       int a = 1;
       int b = 2;
}

CxList one = All.FindByName("1");
CxList a = All.FindByShortName("a").FindByType(typeof(Declarator));
CxList flow1 = a.InfluencedBy(one); // [1] -> [a]
CxList two = All.FindByName("2");
CxList b = All.FindByShortName("b").FindByType(typeof(Declarator));
CxList flow2 = b.InfluencedBy(two); // [2] -> [b]
CxList flow = flow1 + flow2;
result = flow.ConcatenateAllPaths(flow);
 the result would be -
      4 flow found:
            [1] -> [a] -> [1] -> [a]
             [1] -> [a] -> [2] -> [b]
             [2] -> [b] -> [1] -> [a]
             [2] -> [b] -> [2] -> [b]
```

Version Information

Supported from **CxAudit** v7.1.2

4.6.6 CxList.ConcatenateAllPaths Method (CxList list)

Concatenates all flows in this instance to all flows in **list**.

Syntax

```
CxQL
public CxList ConcatenateAllPaths (CxList list)
```

Parameters

list

A CxList containing flows. These flow will be concatenated to the flows in **this** instance

Return Value

A product of all flows in **this** instance with the ones in **list** parameter.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

1. This function calls CxList.ConcatenateAllPaths(list, true).
2. If this instance contains *n* flows in it and list contains *m* flows in it, the return set will contain *nxm* flows, where each flow from this instance will be concatenated to each flow from list.

Version Information

Supported from **CxAudit** v7.1.2

4.6.7CxList.ConcatenateAllSources Method (CxList list)

Concatenates the node in **list** to each node in **this** instance. Concatenation is node-to-node (doesn't support connecting flows).
Note: Currently is identical to calling ConcatenateAllSources with testFlow = false

Syntax

```
CxQL
public CxList ConcatenateAllSources (CxList list)
```

Parameters

list

A CxList. It will be concatenated to each node in this instance

Return Value

Flows that starts with **this** instance nodes, and end with the **list** parameter node.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

1. If the **list** parameter contains more than one node or contains flows or **this** instance contains flows, the function return value is undefined.
2. The number of the returned items is same as the number of items in **this** instance.
3. This function calls the Concatenate function for each item in **this** instance with **list** as parameter.
4. Currently is identical to calling ConcatenateAllSources with testFlow = false

Example

The following code example shows how you can use the ConcatenateAllSources method.

```
CxQL

void main()
{
	int a = 1;
	int b = 2;
}

CxList a = All.FindByShortName("a").FindByType(typeof(Declarator));
CxList b = All.FindByShortName("b").FindByType(typeof(Declarator));
```

```
CxList main = All.FindByShortName("main");
CxList list = a + b;
result = list.ConcatenateAllSources(main);

 the result would be -
      2 flow found:
            [a] -> [main]
             [b] -> [main]
```

Version Information

Supported from **CxAudit** v7.1.2

4.6.8 CxList.ConcatenateAllSources Method (CxList list, bool testFlow)

Concatenates the node in **list** to each node in **this** instance. Concatenation is node-to-node (doesn't support connecting flows).

Syntax

```
CxQL
public CxList ConcatenateAllSources (CxList list, bool testFlow)
```

Parameters

list

A CxList. It will be concatenated to each node in this instance

testFlow

If this parameter true -> test possible flow , otherwise connect directly

Return Value

Flows that starts with **this** instance nodes, and end with the **list** parameter node.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

1. If the **list** parameter contains more than one node or contains flows or **this** instance contains flows, the function return value is undefined.
2. The number of the returned items is same as the number of items in **this** instance.
3. This function calls the Concatenate function for each item in **this** instance with **list** as parameter.

Example

The following code example shows how you can use the ConcatenateAllSources method.

```
CxQL

void main()
{
      int a = 1;
```

```
      int b = 2;
}

CxList a = All.FindByShortName("a").FindByType(typeof(Declarator));
CxList b = All.FindByShortName("b").FindByType(typeof(Declarator));
CxList main = All.FindByShortName("main");
CxList list = a + b;
result = list.ConcatenateAllSources(main, false);

 the result would be -
     2 flow found:
           [a] -> [main]
            [b] -> [main]
```

Version Information

Supported from **CxAudit** v7.1.2

4.6.9 CxList.ConcatenateAllTargets Method (CxList list)

Concatenates each node in the **list** to the node in **this** instance. Concatenation is node-to-node (doesn't support connecting flows).

Syntax

```
CxQL
public CxList ConcatenateAllTargets (CxList list)
```

Parameters

list

A CxList. It will be concatenated to each node in this instance

Return Value

Flows that start with this instance nodes, and end with the **list** parameter node

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

1. If the "**this**" instance parameter contains more than one node or contains flows or **list** contains flows, the function return value is undefined.
2. The number of the returned items is same as the number of items in **list**.
3. This function calls the Concatenate function for **this** instance with each item in **list** as parameter..
4. Currently is identical to calling ConcatenateAllTargets with testFlow = false

Example

The following code example shows how you can use the ConcatenateAllSources method.

```
CxQL
```

```
void main()
{
      int a = 1;
      int b = 2;
}

CxList a = All.FindByShortName("a").FindByType(typeof(Declarator));
CxList b = All.FindByShortName("b").FindByType(typeof(Declarator));
CxList main = All.FindByShortName("main");
CxList list = a + b;
result = main.ConcatenateAllTargets(list);

 the result would be -
     2 flow found:
           [main] -> [a]
            [main] -> [b]
```

Version Information

Supported from **CxAudit** v7.1.2

4.6.10 CxList.ConcatenateAllTargets Method (CxList list, bool testFlow)

Concatenates each node in the **list** to the node in **this** instance. Concatenation is node-to-node (doesn't support connecting flows).

Syntax

```
CxQL
public CxList ConcatenateAllTargets (CxList list, bool testFlow)
```

Parameters

list

A CxList. It will be concatenated to each node in this instance

testFlow

If this parameter true -> test possible flow , otherwise connect directly

Return Value

Flows that start with this instance nodes, and end with the **list** parameter node

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

1. If the "**this**" instance parameter contains more than one node or contains flows or **list** contains flows, the function return value is undefined.
2. The number of the returned items is same as the number of items in **list**.
3. This function calls the Concatenate function for **this** instance with each item in **list** as parameter..

Example

The following code example shows how you can use the ConcatenateAllSources method.

```
CxQL

void main()
{
       int a = 1;
       int b = 2;
}

CxList a = All.FindByShortName("a").FindByType(typeof(Declarator));
CxList b = All.FindByShortName("b").FindByType(typeof(Declarator));
CxList main = All.FindByShortName("main");
CxList list = a + b;
result = main.ConcatenateAllTargets(list, false);

 the result would be -
      2 flow found:
            [main] -> [a]
             [main] -> [b]
```

Version Information

Supported from **CxAudit** v7.1.2

4.7 CxList.Contained Method (CxList, GetStartEndNodesType)

Returns a subset of "this" instance whose elements are contained in the given list, filtered according to the given nodes type.

Syntax

```
CxQL
public CxList Contained(CxList pathList, GetStartEndNodesType requestedType)
```

Parameters

pathList

The list where the method looks for the requested node type.

requestedType

An enum matching the relevant GetStartEndNodes types, which are:
EndNodesOnly, StartNodesOnly, StartAndEndNodes, AllNodes and AllButNotStartAndEnd

Return Value

A subset of "this" instance with elements from the requested nodes type.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.Contained() method.
 The input source code is:

 void foo()
{
      int b = 2, a = 5, c;
      if (a > b)
            b = 3;
      c = b;
}

result =
All.FindByShortName("b").Contained(All.InfluencedBy(All.FindById(50)),
CxList.GetStartEndNodesType.AllNodes); //Id 50 is "3" in "b = 3;"

 The result would consist of 2 items:
```

```
            b (from b = 3;)
            b (from c = b;)

result =
All.FindByShortName("a").Contained(All.InfluencedBy(All.FindById(50)),
GetStartEndNodesType.EndNodesOnly); //Id 50 is "3" in "b = 3;"

 The result would consist of 0 items
```

4.8 CxList.ExtractFromSOQL Method ()

Extracts the parameters of a SOQL statement into a dictionary.

Syntax

```
CxQL
public Dictionary<String, List<String>> ExtractFromSOQL()
```

Return Value

A dictionary with keys that match SOQL keywords and their relevant parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.ExtractFromSOQL() method.
 The input source code is:

int b = 0;
String a = "select * from table where x=" + b;

result = All.ExtractFromSOQL();

 the result would consist of 3 results:
             { "select" : "*",
               "from"   : "table",
               "where" : "x=" }
```

4.9 CxList.ExtractFromSOQL Method (string)

Extracts the parameters of the given keyword from a SOQL statement into a list.

Syntax

```
CxQL
public List<string> ExtractFromSOQL(string keyword)
```

Parameters

keyword

The SOQL keyword to extract.

Return Value

A list with the parameters of the keyword.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the ExtractFromSOQL method.

```
CxQL

 This example demonstrates the CxList.ExtractFromSOQL() method.
 The input source code is:

int b = 0;
String a = “select * from table where x=“ + b;

result = All.ExtractFromSOQL("select");

 the result would be -
      1 item found:
             [“*”]
```

4.10 CxList.DataInfluencedBy Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are data influenced by the CxList specified in parameter.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- DataInfluencedBy(list, InfluenceAlgorithmCalculation.OldAlgorithm)

Syntax

```
CxQL
public CxList DataInfluencedBy(CxList influencing)
```

Parameters

influencing

CxList data-influencing on "this" instance.

Return Value

A subset of "this" instance data influenced by the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.DataInfluencedBy() method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = a;

CxList five = All.FindByName("5");
result = All.DataInfluencedBy(five);

 the result would be -
      6 items found:
            a (in a = 5),
            a (in a > 3),
            > (in a > 3),
            a (in b = a),
            = (in b = a),
            b (in b = a)
```

4.11 CxList.DataInfluencedBy Method (CxList, InfluenceAlgorithmCalculation)

Returns a CxList which is a subset of this instance and its elements are data influenced by the CxList specified in the first parameter using the influence algorithm specified in the second parameter.

Syntax

```
CxQL
public CxList DataInfluencedBy(CxList influencing,
InfluenceAlgorithmCalculation algorithm)
```

Parameters

influencing

CxList data-influencing on "this" instance.

algorithm

An enum matching the relevant InfluenceAlgorithmCalculation options which are:

OldAlgorithm, NewAlgorithm

Return Value

A subset of "this" instance data influenced by the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.DataInfluencedBy() method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = a;

CxList five = All.FindByName("5");
result = All.DataInfluencedBy(five,
CxList.InfluenceAlgorithmCalculation.NewAlgorithm);

 the result would be -
      6 items found:
            a (in a = 5),
            a (in a > 3),
            > (in a > 3),
```

```
a (in b = a),
= (in b = a),
b (in b = a)
```

4.12 CxList.DataInfluencingOn Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are data influencing on the CxList specified in parameter.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `DataInfluencingOn(list, InfluenceAlgorithmCalculation.OldAlgorithm)`

Syntax

```
CxQL
public CxList DataInfluencingOn(CxList influenced)
```

Parameters

influenced

CxList data-influenced by "this" instance.

Return Value

A subset of "this" instance data influencing on the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.DataInfluencingOn() method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = a;

CxList b = All.FindByName("*.b");
result = All.DataInfluencingOn(b);

 the result would be -
      3 items found:
            a (in b = a),
            a (in a = 5),
            5 (in a = 5)
```

4.13 CxList.DataInfluencingOn Method (CxList, InfluenceAlgorithmCalculation)

Returns a CxList which is a subset of "this" instance and its elements are data influencing on the CxList specified in the first parameter using the influence algorithm specified in the second parameter.

Syntax

```
CxQL
public CxList DataInfluencingOn(CxList influenced,
InfluenceAlgorithmCalculation algorithm)
```

Parameters

influenced

CxList data-influenced by "this" instance.

algorithm

An enum matching the relevant InfluenceAlgorithmCalculation options which are:

OldAlgorithm, NewAlgorithm

Return Value

A subset of "this" instance data influencing on the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.DataInfluencingOn() method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = a;

CxList b = All.FindByName("*.b");
result = All.DataInfluencingOn(b,
              CxList.InfluenceAlgorithmCalculation.NewAlgorithm);

 the result would be -
      3 items found:
            a (in b = a),
            a (in a = 5),
            5 (in a = 5)
```

4.14 CxList.InfluencedBy Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are influenced (either data or control) by the CxList specified in parameter.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `InfluencedBy(list, InfluenceAlgorithmCalculation.OldAlgorithm)`

Syntax

```
CxQL
public CxList InfluencedBy(CxList influencing)
```

Parameters

influencing

CxList data-influencing on "this" instance.

Return Value

A subset of "this" instance influenced by (either data or control) the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList.InfluencedBy() method.
 Notice the difference between ControlInfluencedBy, DataInfluencedBy
and InfluencedBy
 The input source code is:
 int b = 2, a = 5, c;
 if (a > b)
      b = 3;
 c = b;
result = All.InfluencedBy(All.FindById(43)); // Id 43 is 5 from a = 5;
 Notice that among all the results also c (in c = b) appears because c
is
 data-dependant on  b=3, which in turn is control dependant on a > b,
which  itself is data-dependant on a = 5.

result = All.DataInfluencedBy(All.FindById(43)); // 5
 Notice that now c (in c = b) doesn't appear because its value is not
 influenced by 5.
result = All.ControlInfluencedBy(All.FindById(43)); // 5
 Notice that now c (in c = b) doesn't appear because it is not control
 dependant by 5.
```

4.15 CxList.InfluencedBy Method (CxList, InfluenceAlgorithmCalculation)

Returns a CxList which is a subset of "this" instance and its elements are influenced (either data or control) by the CxList specified in the first parameter using the influence algorithm specified in the second parameter.

Syntax

```
CxQL
public CxList InfluencedBy(CxList influencing, InfluenceAlgorithmCalculation
algorithm)
```

Parameters

influencing

CxList data-influencing on "this" instance.

algorithm

An enum matching the relevant InfluenceAlgorithmCalculation options which are:

OldAlgorithm, NewAlgorithm

Return Value

A subset of "this" instance influenced by (either data or control) the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList.InfluencedBy() method.
 Notice the difference between ControlInfluencedBy, DataInfluencedBy
and InfluencedBy
 The input source code is:
 int b = 2, a = 5, c;
 if (a > b)
      b = 3;
 c = b;
result = All.InfluencedBy(All.FindById(43),
       CxList.InfluenceAlgorithmCalculation.NewAlgorithm); // Id 43 is 5
from a = 5;
 Notice that among all the results also c (in c = b) appears because c
is
 data-dependant on  b = 3, which in turn is control dependant on a > b,
which  itself is data-dependant on a = 5.

result = All.DataInfluencedBy(All.FindById(43)); // 5
```

```
 Notice that now c (in c = b) doesn't appear because its value is not
 influenced by 5.
result = All.ControlInfluencedBy(All.FindById(43)); // 5
 Notice that now c (in c = b) doesn't appear because it is not control
 dependant by 5.
```

4.16 CxList.InfluencedByAndNotSanitized Method (CxList, CxList)

Returns a CxList which is a subset of "this" instance and its elements are influenced by the CxList specified in the first parameter, and their influencing path doesn't contain elements from the CxList specified in the second parameter.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `InfluencedByAndNotSanitized(influencing, sanitized, InfluenceAlgorithmCalculation.OldAlgorithm)`

Syntax

```
CxQL
public CxList InfluencedByAndnotSanitized(CxList influencing, CxList
sanitization)
```

Parameters

influencing

CxList influencing on "this" instance.

sanitization

CxList that "cuts" the influencing path.

Return Value

A subset of "this" instance and its elements are influenced by the first specified parameter, and their influencing path doesn't contain element from the second CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

This example demonstrates the CxList.InfluencedByAndNotSanitized()
method.
The input source code is:

          string s = input();
          string s1 = fixSql(s);
          string s2 = s + s1;

          execute(s);      (*)
          execute(s1);
          execute(s2);     (*)
```

```
            s = s1;
            execute(s);
            execute(s1);
            execute(s2);     (*)

            s2 = s;
            execute(s);
            execute(s1);
            execute(s2);

CxList execute = All.FindByName("execute");
CxList input = All.FindByName("input");
CxList fixSql = All.FindByName("fixSql");
result = execute.InfluencedByAndNotSanitized(input, fixSql);

 Notice that only the lines marked with a (*) are returned. These are
the
 only statements that have an influencing path from the input() command,
 without being completely sanitized by fixSql().
```

4.17 CxList.InfluencedByAndNotSanitized Method (CxList, CxList, InfluenceAlgorithmCalculation)

Returns a CxList which is a subset of "this" instance and its elements are influenced by the CxList specified in the first parameter, and their influencing path doesn't contain elements from the CxList specified in the second parameter, using the influence algorithm specified in the third parameter.

Syntax

```
CxQL
public CxList InfluencedByAndnotSanitized(CxList influencing, CxList
sanitization, InfluenceAlgoritmCalculation algorithm)
```

Parameters

influencing

CxList influencing on "this" instance.

sanitization

CxList that "cuts" the influencing path.

algorithm

An enum matching the relevant InfluenceAlgorithmCalculation options which are:

OldAlgorithm, NewAlgorithm

Return Value

A subset of "this" instance and its elements are influenced by the first specified parameter, and their influencing path doesn't contain element from the second CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

This example demonstrates the CxList.InfluencedByAndNotSanitized()
method.
The input source code is:

            string s = input();
            string s1 = fixSql(s);
            string s2 = s + s1;

            execute(s);      (*)
            execute(s1);
            execute(s2);     (*)
```

```
            s = s1;
            execute(s);
            execute(s1);
            execute(s2);     (*)

            s2 = s;
            execute(s);
            execute(s1);
            execute(s2);

CxList execute = All.FindByName("execute");
CxList input = All.FindByName("input");
CxList fixSql = All.FindByName("fixSql");
result = execute.InfluencedByAndNotSanitized(input, fixSql,
          CxList.InfluenceAlgorithmCalculation.NewAlgoritm);

 Notice that only the lines marked with a (*) are returned. These are
the
 only statements that have an influencing path from the input() command,
 without being completely sanitized by fixSql().
```

4.18 CxList.InfluencingOn Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are influencing (data and control) on the CxList specified in parameter.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `InfluencingOn(influenced, InfluenceAlgorithmCalculation.OldAlgorithm)`

Syntax

```
CxQL
public CxList InfluencingOn (CxList influenced)
```

Parameters

influenced

CxList influenced by "this" instance.

Return Value

A subset of "this" instance influencing on the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.InfluencingOn() method.
 The input source code is:

 int a;
 a = 5;
 b = a;

CxList b_var = All.FindByShortName("b");
result = All.InfluencingOn(b_var);

 the result would be -
      3 items found:
            5,
            a,
            a
```

Version Information

CxAudit

Supported from **CxAudit** v1.8.1

4.19 CxList.InfluencingOn Method (CxList, InfluenceAlgorithmCalculation)

Returns a CxList which is a subset of "this" instance and its elements are influencing (data and control) on the CxList specified in the first parameter using the influence algorithm specified in the second parameter.

Syntax

```
CxQL
public CxList InfluencingOn (CxList influenced,
InfluenceAlgorithmCalculation algorithm)
```

Parameters

influenced

CxList influenced by "this" instance.

algorithm

An enum matching the relevant InfluenceAlgorithmCalculation options which are:

OldAlgorithm, NewAlgorithm

Return Value

A subset of "this" instance influencing on the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.InfluencingOn() method.
 The input source code is:

 int a;
 a = 5;
 b = a;

CxList b_var = All.FindByShortName("b");
result = All.InfluencingOn(b_var,
                CxList.InfluenceAlgorithmCalculation.NewAlgorithm);

 the result would be -
      3 items found:
            5,
            a,
            a
```

4.20 CxList.InfluencingOnAndNotSanitized Method (CxList,CxList)

Returns a CxList which is a subset of "this" instance and its elements are influencing on (Data or Control), and an influencing path exists which doesn't contain elements from the sanitization.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `InfluencingOnAndNotSanitized(list, InfluenceAlgorithmCalculation.OldAlgorithm)`

Syntax

```
CxQL
public CxList InfluencingOnAndNotSanitized (CxList influencing, CxList
sanitization)
```

Parameters

influencing

CxList influencing on "this" instance.

sanitization

CxList that "cuts" the influencing path

Return Value

A subset of "this" instance and its elements are influencing on the first specified parameter, and their influencing path doesn't contain elements from the CxList specified in second parameter.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList.InfluencingOnandNotSanitized()
 method.
 The input source code is:
            string s = input();
            string s1 = fixSql(s);
            string s2 = s + s1;

            execute(s);      (*)
            execute(s1);
            execute(s2);     (*)

            s = s1;
```

```
            execute(s);
            execute(s1);
            execute(s2);     (*)

            s2 = s;
            execute(s);
            execute(s1);
            execute(s2);
CxList execute = All.FindByName("execute");
CxList input = All.FindByName("input");
CxList fixSql = All.FindByName("fixSql");
result = input.InfluencingOnAndNotSanitized(execute, fixSql);

 Notice that only the first line is returned (string s = input();)
```

4.21 CxList.InfluencingOnAndNotSanitized Method (CxList,CxList,InfluenceAlgorithmCalculation)

Returns a CxList which is a subset of "this" instance and its elements are influencing on (Data or Control), and an influencing path exists which doesn't contain elements from the sanitization using the influence algorithm specified in the third parameter.

Syntax

```
CxQL
public CxList InfluencingOnAndNotSanitized (CxList influencing, CxList
sanitization, InfluenceAlgorithmCalculation algorithm)
```

Parameters

influencing

CxList influencing on this instance.

sanitization

CxList that "cuts" the influencing path

algorithm

An enum matching the relevant InfluenceAlgorithmCalculation options which are:

OldAlgorithm, NewAlgorithm

Return Value

A subset of "this" instance and its elements are influencing on the first specified parameter, and their influencing path doesn't contain elements from the CxList specified in the second parameter.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList.InfluencingOnandNotSanitized()
 method.
 The input source code is:
            string s = input();
            string s1 = fixSql(s);
            string s2 = s + s1;

            execute(s);      (*)
            execute(s1);
            execute(s2);     (*)
```

```
            s = s1;
            execute(s);
            execute(s1);
            execute(s2);     (*)

            s2 = s;
            execute(s);
            execute(s1);
            execute(s2);
CxList execute = All.FindByName("execute");
CxList input = All.FindByName("input");
CxList fixSql = All.FindByName("fixSql");
result = input.InfluencingOnAndNotSanitized(execute, fixSql,
          CxList.InfluenceAlgorithmCalculation.NewAlgorithm);

 Notice that only the first line is returned (string s = input();)
```

4.22 CxList.NotInfluencedBy Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are not influenced (either data or control) by the CxList specified in parameter.

Syntax

```
CxQL
public CxList NotInfluencedBy(CxList influencing)
```

Parameters

influencing

CxList data on "this" instance.

Return Value

A subset of "this" instance not influenced by (either data or control) the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList.NotInfluencedBy() method.

 The input source code is:
 int b = 2, a = 5, c;
 if (a > b)
      b = 3;
 c = b;

result = All.NotInfluencedBy(All.FindById(43)); // 5
 Returns every object besides a and 5 (from a = 5) and a (from a > b)
 Notice that among all the results also b (in b = 2) appears because it
does not influence the value of a.
```

4.23 CxList.NotInfluencingOn Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are not influencing (data and control) on the CxList specified in parameter.

Syntax

```
CxQL
public CxList NotInfluencingOn (CxList notInfluenced)
```

Parameters

notInfluenced

CxList data in "this" instance.

Return Value

A subset of "this" instance not influencing on the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.InfluencingOn() method.
 The input source code is:

 int a, c = 3;
 a = 5;
 b = a;

CxList b_var = All.FindByShortName("b");
result = All.NotInfluencingOn(b_var);

 the result would be -
      2 items found:
             c,
             3
```

4.24 CxList.FindAllMembers Method (CxList)

Returns a CxList which is a subset of "this" instance, with elements that are members of the classes in the given CxList.

Syntax

```
CxQL
public CxList FindAllMembers(CxList Ids)
```

Parameters

Ids

The list of Classes whose members are to be found.

Return Value

A subset of "this" instance, with elements that are members of the classes in the given CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindAllMembers() method.
 The input source code is:

public class MyClass
{
 public int b, a = 5;
 boolean c=false;
}
result = All.FindAllMembers(All.FindByName("MyClass"));
 The result would consist of 3 items:
             b (public int b, a = 5;),
             a (public int b, a = 5;),
             c (boolean c=false)
```

4.25 CxList.FindAllReferences Method (CxList)

Returns a CxList which is a subset of "this" instance, with elements that are references of the given CxList.

Syntax

```
CxQL
public CxList FindAllReferences(CxList referenced)
```

Parameters

referenced

The CxList whose references are to be found.

Return Value

A subset of "this" instance, with elements that are references of the given CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindAllRefernces() method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = a;

result = All.FindAllReferences(All.FindById(36)); //a in (a = 5)

 the result would consist of 3 items:
             a (in a = 5),
             a (in a > 5),
             a (in b = a)
```

4.26 CxList.FindAllReferences Method (CxList, CxList)

Returns a CxList which is a subset of "this" instance, with elements that are references of the given CxList, excluding elements in the second CxList.

Syntax

```
CxQL
public CxList FindAllReferences(CxList referenced, CxList exclude)
```

Parameters

referenced

The CxList whose references are to be found.

exclude

The CxList whose elements will be ignored and excluded.

Return Value

A subset of "this" instance, with elements that are references of the given CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindAllRefernces() method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = a;

result = All.FindAllReferences(All.FindById(36), All.FindById(30)); //a
in (a = 5), b in (int b)

 the result would consist of 3 items:
             a (in a = 5),
             a (in a > 5),
             a (in b = a)
```

4.27 CxList.FindByAssignmentSide Method (AssignmentSide)

Returns a CxList which is a subset of "this" instance and its elements are being on the given side of an assignment expression.

Syntax

```
CxQL
public CxList FindByAssignmentSide(CxList.AssignmentSide side)
```

Parameters

side

The side of the assignment expression, which can be one of the following values: Left, Right (see Section AssignmentSide).

Return Value

A subset of "this" instance on the specified side of an assignment expression.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByAssignmentSide() method.
 The input source code is:


 a = 3;
 b = a;
 if (a == 4)
      b = a - 1;

result = All.FindByAssignmentSide(CxList.AssignmentSide.Left);

 The result would consist of 3 items:
             a (in a = 3),
             b (in b = a),
             b (in b = a - 1)
```

Version Information

Supported from **CxAudit** v1.8.1

4.28 CxList.FindByCustomAttribute Method (string)

Returns a CxList which is a subset of "this" instance and its elements are custom attributes of the specified name.

Syntax

```
CxQL
public CxList FindByCustomAttribute(string name)
```

Parameters

name

The attribute name.

Return Value

A subset of "this" instance with custom attributes of the specified name.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByCustomAttribute() method.
 The input source code is:

 [WebMethod]
 void foo()
 {

 }

result = All. FindByCustomAttribute("WebMethod");

 the result would consist of 1 item:
            foo
```

4.29 CxList.FindByExtendedType Method (string)

Returns a CxList which is a subset of "this" instance and the type of its elements match the type specified as parameter.

Syntax

```
CxQL
public CxList FindByExtendedType (string extendedType)
```

Parameters

extendedType

The extended type of the objects to be found. Prefix and postfix wildcard (*) are supported.

Return Value

A subset of "this" instance and its elements are those with type specified by the parameter.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByExtendedType() method.
 The input source code is:

MyClass a;
 MyClassExtended b;
 int c;
 a.DataMember = 3;
 c = a.Method();

result = All.FindByExtendedType ("MyClass*");
 the result would consists of 4 items:
             a (in MyClass a)
             b (in MyClassExtended b)
             a (in a.DataMember = 3)
             a (in b = a.Method())
```

4.30 CxList.FindByFathers Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are those that their CxDOM-Fathers are in the specified CxList.

Syntax

CxQL
```
public CxList FindByFathers(CxList fathers)
```

Parameters

fathers

A CxList consisting of the Fathers to be matched.

Return Value

A subset of "this" instance and its elements are those which their CxDOM-Fathers are in the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

CxQL
```
 This example demonstrates the CxList.FindByFather() method.
 First we find the number "3", then we seek for 3's father (which is the
 assignment expression), finally we look for the assignment-expression's
 sons (the "a" and the "3")
 Input source code is:

 a = 3;
 b = a;
 if (a == 4)
      b = a - 1;
CxList three = All.FindByName("*.3");
CxList threesFathers = three.GetFathers();
Result = All.FindByFathers(threesFathers);
 the result would be -
      2 items found:
            a (in a = 3),
            3 (in a = 3)
```

4.31 CxList.FindByFieldAttributes Method (Modifiers)

Returns a CxList which is a subset of "this" instance and its elements are modified by the modifier (private, external, etc).

Syntax

```
CxQL
public CxList FindByFieldAttributes(Modifiers attrib)
```

Parameters

Attrib

Attribute of the fields to be found.

Return Value

A subset of "this" instance and its elements are those with attribute *attrib*.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByAttributes() method.
 Input source code is:
 public class cl1{
          private void foo(){}
          protected void guu(){}
          private int a,b;
          protected int c;
  }

result=All.FindByFieldAttributes(Modifiers.Protected);
 the result would be -
      2 items found:
             guu (in protected void guu(){}),
             c (int c;)
```

Version Information

Supported from **CxAudit**v2.0.5

4.32 CxList.FindByFileName Method (string)

Returns a CxList which is a subset of “this” instance and its elements are in a given source code file.

Syntax

```
CxQL
public CxList FindByFileName(string FileName)
```

Parameters

FileName

String with the file name.

Return Value

A subset of "this” instance with elements from a given file name.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByFileName() method.
 The input source code is:

 //file myCode.cs
 class Cl {
      void foo() {
            int i;
      }
 }

result = All.FindByFileName("*myCode.cs");

 the result consists of 5 items:
             Cl
             void,
             foo,
              int,
              i
```

Version Information

Supported from **CxAudit** v1.8.1

4.33 CxList.FindById Method (int)

Finds all objects with the specified id. This method is mainly used to find all the uses of a code element (e.g. variable, class).

Syntax

```
CxQL
public CxList FindById (int id)
```

Parameters

id

id number to be found.

Return Value

A subset of "this" instance and its elements that have the specified id number.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindById() method.
 The input source code is:

 a = 3;
 b = a;
 if (a == 4)
      b = a - 1;

result = All.FindById(60);

 the result would be -
      1 item found:
            b (in b = a - 1)
```

4.34 CxList.FindByInitialization Method (CxList)

Returns a CxList which is a subset of "this" instance and contains elements initialized by the given CxList.

Syntax

```
CxQL
public CxList FindByInitialization(CxList initializators)
```

Parameters

initializators

A CxList with initializers to search in "this" instance.

Return Value

A subset of "this" instance containing declarators initialized by the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByInitialization() method.
 The input source code is:

 int b = 5;

CxList declarators = All.FindByType(typeof(Declarator));
result= declarators.FindByInitialization(All);

 the result would consist of 1 item:
             b
```

Version Information

Supported from **CxAudit** v1.8.1

4.35 CxList.FindByLanguage Method (string)

Returns a CxList which is a subset of "this" instance whose elements are from the given language.

Syntax

```
CxQL
public CxList FindByLanguage (string languageName)
```

Parameters

languageName

Language name to search.

Return Value

A subset of "this" instance whose elements are from the given language.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByLanguage() method.
 The input source code is:
 //file myCode.cs
 class myCode {

 }

 //file MyCode.java
 class MyCode {

 }

result = All.FindByLanguage ("Java");
 the result would consist of 1 item:
            myCode (class MyCode)
```

4.36 CxList.FindByMemberAccess Method (string)

Returns a CxList which is a subset of "this" instance where its elements are the ones that match the given member being accessed. Notice that this is a case-sensitive search. For a non case-sensitive search, please use the `FindByMemberAccess Method (string, bool)` instead.

Syntax

```
CxQL
public CxList FindByMemberAccess(string memberAccess)
```

Parameters

memberAccess

Contains both the name of the type and the name of the accessed member in the qualified notation (eg. "CheckBoxList.SelectedValue"). Prefix and suffix wild card (*) are permitted.

Return Value

A subset of "this" instance where its elements are the ones which their given member is accessed.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByMemberAccess() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
result = All.FindByMemberAccess("MyClass.DataMember");
 the result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)

result = All.FindByMemberAccess("MyClass.Met*");

 the result would consist of 1 item:
            a.Method (in b = a.Method())
```

4.37 CxList.FindByMemberAccess Method (string,bool)

Returns a CxList which is a subset of "this" instance where its elements are the ones that match the given member being accessed. This search allows both case-sensitive and non case-sensitive searches.

Syntax

CxQL
```
public CxList FindByMemberAccess(string memberAccess, bool caseSensitive)
```

Parameters

memberAccess

Contains both the name of the type and the name of the accessed member in qualified notation (eg. "CheckBoxList.SelectedValue"). Prefix and suffix wild card (*) are permitted.

caseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of "this" instance where its elements are the ones which their specified member is accessed.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

CxQL
```
 This example demonstrates the CxList.FindByMemberAccess() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
result = All.FindByMemberAccess("MyClass.dataMember", true);
 Notice that the result would consist of 0 items because the search is
case-sensitive.

result = All.FindByMemberAccess("MyClass.dataMember", false);
 The result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)
```

```
result = All.FindByMemberAccess("MyClass.met*", true);
 Notice that the result would consist of 0 items because the search is
case-sensitive.


result = All.FindByMemberAccess("MyClass.met*", false);
 the result would consist of 1 item:
            a.Method (in b = a.Method())
```

Version Information

Supported from **CxAudit** v1.8.1

4.38 CxList.FindByMemberAccess Method (string,string)

Returns a CxList which is a subset of "this" instance where its elements are the ones that match the given member being accessed. This is a case-sensitive search by both the name of the type and the name of the accessed member. For a non case-sensitive search please use the `FindByMemberAccess Method(string, string, bool)` instead.

Syntax

```
CxQL
public CxList FindByMemberAccess(string typeName, string memberName)
```

Parameters

typeName

Contains the name of the accessed type (eg. "CheckBoxList");

memberName

Contains the name of the accessed member (eg. "SelectedValue");

Return Value

A subset of "this" instance where its elements are the ones which their specified member is accessed.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByMemberAccess() method.
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
result = All.FindByMemberAccess("MyClass", "DataMember");

 The result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)

result = All.FindByMemberAccess("MyClass", "Met*");

 The result would consist of 1 item:
            a.Method (in b = a.DataMethod())
```

Version Information

Supported from **CxAudit** v7.9.0

4.39 CxList.FindByMemberAccess Method (string, string, bool)

Returns a CxList which is a subset of this instance where its elements are the ones that match the given member being accessed. This search allows both case-sensitive and non case-sensitive searches by the type name and the name of the accessed member.

Syntax

```
CxQL
public CxList FindByMemberAccess(string typeName, string memberName, bool
caseSensitive)
```

Parameters

typeName

Contains the name of the accessed type (eg. "CheckBoxList");

memberName

Contains the name of the accessed member (eg. "SelectedValue");

caseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of "this" instance where its elements are the ones which their specified member is accessed.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByMemberAccess() method.
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
result = All.FindByMemberAccess("MyClass", "dataMember", true);

 the result would consist of 0 item because it is a case-sensitive
search.

result = All.FindByMemberAccess("MyClass", "dataMember", false);
```

```
 the result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)


result = All.FindByMemberAccess("MyClass", "met*", true);



the result would consist of 0 item, because it is a case-sensitive
search.

result = All.FindByMemberAccess("MyClass", "met*", false);

the result would consist of 1 item:
            a.Method (in b = a.DataMethod())
```

4.40 CxList.FindByExactMemberAccess Method (string)

Returns a CxList which is a subset of "this" instance where its elements are the ones that match the given member being accessed. Notice that this is a case-sensitive search. For a non case-sensitive search, please use the `FindByExactMemberAccess Method (string, bool)` instead.

Syntax

```
CxQL
public CxList FindByExactMemberAccess(string memberAccess)
```

Parameters

memberAccess

Contains both the name of the type and the name of the accessed member in the qualified notation (eg. "CheckBoxList.SelectedValue").

Return Value

A subset of "this" instance where its elements are the ones which their given member is accessed.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByMemberAccess() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
result = All.FindByExactMemberAccess("MyClass.DataMember");
 the result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)
```

Version Information

Supported from **CxAudit** v8.2.0

4.41 CxList.FindByExactMemberAccess Method (string,bool)

Returns a CxList which is a subset of "this" instance where its elements are the ones that match the given member being accessed. This search allows both case-sensitive and non case-sensitive searches.

Syntax

```
CxQL
public CxList FindByMemberAccess(string memberAccess, bool caseSensitive)
```

Parameters

memberAccess

Contains both the name of the type and the name of the accessed member in qualified notation (eg. "CheckBoxList.SelectedValue").

caseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of "this" instance where its elements are the ones which their specified member is accessed.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByMemberAccess() method.
 The input source code is:


 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
result = All.FindByExactMemberAccess("MyClass.dataMember", true);
 Notice that the result would consist of 0 items because the search is
case-sensitive.

result = All.FindByExactMemberAccess("MyClass.dataMember", false);
 The result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)
```

Version Information

Supported from **CxAudit** v8.2.0

4.42 CxList.FindByExactMemberAccess Method (string,string)

Returns a CxList which is a subset of "this" instance where its elements are the ones that match the given member being accessed. This is a case-sensitive search by both the name of the type and the name of the accessed member. For a non case-sensitive search please use the `FindByExactMemberAccess Method(string, string, bool)` instead.

Syntax

```
CxQL
public CxList FindByExactMemberAccess(string typeName, string memberName)
```

Parameters

typeName

Contains the name of the accessed type (eg. "CheckBoxList");

memberName

Contains the name of the accessed member (eg. "SelectedValue");

Return Value

A subset of "this" instance where its elements are the ones which their specified member is accessed.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByMemberAccess() method.
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
result = All.FindByExactMemberAccess("MyClass", "DataMember");

 The result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)
```

Version Information

Supported from **CxAudit** v8.2.0

4.43 CxList.FindByExactMemberAccess Method (string, string, bool)

Returns a CxList which is a subset of this instance where its elements are the ones that match the given member being accessed. This search allows both case-sensitive and non case-sensitive searches by the type name and the name of the accessed member.

Syntax

```
CxQL
public CxList FindByExactMemberAccess(string typeName, string memberName,
bool caseSensitive)
```

Parameters

typeName

Contains the name of the accessed type (eg. "CheckBoxList");

memberName

Contains the name of the accessed member (eg. "SelectedValue");

caseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of "this" instance where its elements are the ones which their specified member is accessed.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByMemberAccess() method.
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
result = All.FindByExactMemberAccess("MyClass", "dataMember", true);
  the result would consist of 0 item because it is a case-sensitive
search.
result = All.FindByExactMemberAccess("MyClass", "dataMember", false);

 the result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)
```

Version Information

Supported from **CxAudit** v8.2.0

4.44 CxList.FindByMethodReturnType Method (string)

Returns a CxList which is a subset of "this" instance and its elements are method declarators of a given return type.

Syntax

```
CxQL
public CxList FindByMethodReturnType(string type)
```

Parameters

type

The return type name string.

Return Value

A subset of "this" instance with method declarators of a given return type.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByMethodReturnType() method.
 The input source code is:

 MyType foo() {
       ...
 }

result = All.FindByMethodReturnType("MyType");

 the result would consists of 1 item:
            foo
```

4.45 CxList.FindByName Method (string)

Returns a CxList which is a subset of "this" instance and its elements are the ones which their name is the given parameter.

Syntax

```
CxQL
public CxList FindByName(string name)
```

Parameters

name

The name of the objects to look for. Prefix and postfix wildcard (*) are supported.

Return Value

A subset of "this" instance and its elements are the ones which their name is the given parameter.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByName() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByName("*Member*");

 the result would consist of 1 item:
            a.DataMember (in a.DataMember = 3)
```

4.46 CxList.FindByName Method (string, int, int)

Returns a CxList which is a subset of "this" instance and its elements are the ones which their name is the given parameter (optionally with wildcards) and is not shorter than minLength and not longer than maxLength.

Syntax

```
CxQL
public CxList FindByName(string name, int minLength, int maxLength)
```

Parameters

name

Contains the name of the objects. Prefix and postfix wildcard (*) are supported.

minLength

Minimum length of the searched strings.

maxLength

Maximum length of the searched strings.

Result

A subset of "this" instance and its elements are the ones which their name is the given parameter, according to the given length interval.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByName() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByName("*Me*",3,7);
 the result would consist of 1 item:
      Method (in b = a.Method())
```

Version Information

Supported from **CxAudit** v2.0.5

4.47 CxList.FindByName Method (string, bool)

Returns a CxList which is a subset of "this" instance and its elements are the ones which their name is the given parameter, according to the specified comparison criteria.

Syntax

```
CxQL
public CxList FindByName(string name, bool caseSensitive)
```

Parameters

name

Contains the name of the objects. Prefix and postfix wildcard (*) are supported.

caseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of "this" instance and its elements are the ones which their name is the given parameter, according to the given comparison criteria. The *caseSensitive* boolean value defines the ability to search using case sensitive or case insensitive comparison.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByName() method.
 The input source code is:


 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();


result = All.FindByName("*member*",true);
 the result would consist of 0 items.

result = All.FindByName("*member*", false);
 the result would consist of 1 item:
       a.DataMember (in a.DataMember = 3)
```

Version Information

Supported from **CxAudit** v1.8.1

4.48 CxList.FindByName Method (string, StringComparison)

Returns a CxList which is a subset of "this" instance and its elements are the ones which their name is the given parameter. The comparison method specified in parameter is used for matching.

Syntax

```
CxQL
public CxList FindByName(string name, StringComparison comparisonType)
```

Parameters

name

The name of the objects to look for. Prefix and postfix wildcard (*) are supported.

comparisonType

StringComparison type to be used in name comparison. One of the following values: *CurrentCulture, CurrentCultureIgnoreCase, InvariantCulture, InvariantCultureIgnoreCase, Ordinal, OrdinalIgnoreCase*

Return Value

A subset of "this" instance and its elements are the ones which their name is the given parameter.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByName() method.
 The input source code is:


 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByName("*member*", StringComparison.OrdinalIgnoreCase);
 the result would consist of 1 item:
       DataMember (in a.DataMember = 3)
```

4.49 CxList.FindByName Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are the ones which their names are equal to the given list.

Syntax

```
CxQL
public CxList FindByName(CxList nodesList)
```

Parameters

nodesList

The list of nodes containing the names to be found.

Return Value

A subset of "this" instance and its elements are the ones which the name is contained in the given list.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

This example demonstrates the CxList.FindByName() method.
The input source code is:

MyClass a;
int b;
a.DataMember = 3;
b = a.Method();

result = All.FindByName(All.FindByType(typeof(MemberAccess)));

the result would consist of 3 items:
      a (in MyClass a)
      a (in a.DataMember = 3)
      a (in b = a.Method())
```

4.50 CxList.FindByName Method (CxList, bool)

Returns a CxList which is a subset of "this" instance and its elements are the ones which their names are equal to the list given.

Syntax

```
CxQL
public CxList FindByName(CxList nodesList, bool CaseSensitive)
```

Parameters

nodesList

The list of nodes containing the names to be found.

CaseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of "this" instance and its elements are the ones which their name is contained in the given list, according to the specified case sensitivity comparison criteria.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByName() method.
 The input source code is:

 MyClass A;
 int a;
 A.DataMember = 3;
 a = A.Method();

result = All.FindByName(All.FindByType(typeof(MemberAccess)), false);

 the result would consist of 5 items:
            A (in MyClass A)
            a (in int a)
            A (in A.DataMember = 3)
            a (in a = A.Method())
            A (in a = A.Method())
```

4.51 CxList.FindByParameters Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are methods of the given CxList with the specified parameters.

Syntax

```
CxQL
public CxList FindByParameters (CxList paramList)
```

Parameters

paramList

CxList of method parameters.

Return Value

A subset of "this" instance with methods whose parameters are given in the list.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByParameters() method.
 The input source code is:

 foo("myVar");

CxList var = All.FindByShortName("myVar");
result = All. FindByParameters(var);

 the result would consist of 1 item:
           foo
```

4.52 CxList.FindByParameterValue Method (int, string, BinaryOperator)

Returns a CxList which is a subset of "this" instance with methods where a given parameter number is equal (or not) to the specified value.

Syntax

```
CxQL
public CxList FindByParameterValue(int ParamNo, string ParamValue,
                        BinaryOperator opr)
```

Parameters

ParamNo

Zero-based index of the parameter

ParamValue

The value of the parameter

BinaryOperator

One of the followings values:

```
BinaryOperator.IdentityEquality
BinaryOperator.IdentityInequality
```

Return Value

Returns a CxList which is a subset of "this" instance with methods where a given parameter number is equal (or not) to the specified value.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList.FindByParameterValue() method.
 The input source code is:
 a = Method("Val1", 1);
 a = Method("val2", 2);
result =
All.FindByParameterValue(0,"value1",BinaryOperator.IdentityEquality);
      the result would consist of 1 item:
            Method (in a = Method("Val1", 1))
result=All.FindByParameterValue(0,"val1",BinaryOperator.IdentityInequali
ty);
      the result would consist of 1 item:
            Method (in a = Method("Val2", 2))
result=All.FindByParameterValue(1,"2",BinaryOperator.IdentityEquality);
```

```
      the result would consist of 1 item:
            Method (in a = Method("val2", 2))
```

4.53 CxList.FindByParameterValue Method (int, int, BinaryOperator)

Returns a CxList which is a subset of "this" instance and its elements are methods whose parameters values (referred by their index) are equal (or not).

Syntax

```
CxQL
public CxList FindByParameterValue(int paramNo1, int paramNo2,
BinaryOperator opr)
```

Parameters

paramNo1

Zero-based index of the parameter.

paramNo2

Zero-based index of the parameter.

opr

One of the following values:

`BinaryOperator.IdentityEquality`

`BinaryOperator.IdentityInequality`

Return Value

A subset of "this" instance whose parameter values are equal or not equal (depending on the operator choosen).

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.GetParameters() method.
 The input source code is:

foo(1, i, 1);

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
result = All. FindByParameterValue(0, 2,
BinaryOperator.IdentityEquality);

 the result would consist of 1 item:
            foo (first parameter value is equal to the third one)
```

4.54 CxList.FindByPosition Method (int)

Returns a CxList which is a subset of "this" instance and its elements are in the given line number.

Syntax

```
CxQL
public CxList FindByPosition(int line)
```

Parameters

line

The line number.

Return Value

A subset of "this" instance with elements from the given line.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList. FindByPosition() method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = 6;

result = All.FindByPosition(2);

 the result would consist of 4 items:
            if
            a,
            >,
            3
```

4.55 CxList.FindByPosition Method (int, int)

Returns a CxList which is a subset of "this" instance and its elements are located in the given line and column number.

Syntax

```
CxQL
public CxList FindByPosition(int line, int col)
```

Parameters

Line

Line number in the source code.

Col

Column number in the source code.

Return Value

A subset of "this" instance with elements from the given line and column.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByPosition() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByPosition (3, 16);
 the result would consist of 1 item:
             3 (in a.DataMember = 3)
```

4.56 CxList.FindByPosition Method (int, int, int)

Returns a CxList which is a subset of "this" instance and its elements are in the given line/column and with the given length.

Syntax

```
CxQL
public CxList FindByPosition(int line, int col, int length)
```

Parameters

line

The line number.

col

The column number.

length

The element length.

Return Value

A subset of "this" instance with elements from the given line, column and with the given length.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList. FindByPosition() method.
 The input source code is:

 int b, a = 5;
 if (a == 33)
      b = 6;

result = All.FindByPosition(2, 5, 1);

 the result would consist of 1 item:
            a
```

4.57 CxList.FindByPosition Method (string, int)

Returns a CxList which is a subset of "this" instance and its elements are located in the given file and line number.

Syntax

```
CxQL
public CxList FindByPosition(string file, int line)
```

Parameters

file

File name in the source code.

line

Line number in the source code.

Return Value

A subset of "this" instance which is located in the given file and line.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindByPosition() method.
 The input source code is (file name "Mycode.java"):

 MyClass a;
 int b;
 a.DataMember = 5;
 b = a.Method();

result = All.FindByPosition ("MyCode.java", 3);
 the result would consist of 1 item:
       5 (in a.DataMember = 5)
```

4.58 CxList.FindByPosition Method (string, int, int)

Returns a CxList which is a subset of "this" instance and its elements are located in the given file, line and column.

Syntax

```
CxQL
public CxList FindByPosition(string file, int line, int col)
```

Parameters

file

File name in the source code.

line

Line number in the source code.

col

Column number in the source code.

Return Value

A subset of "this" instance which is located in the given file, line and column.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

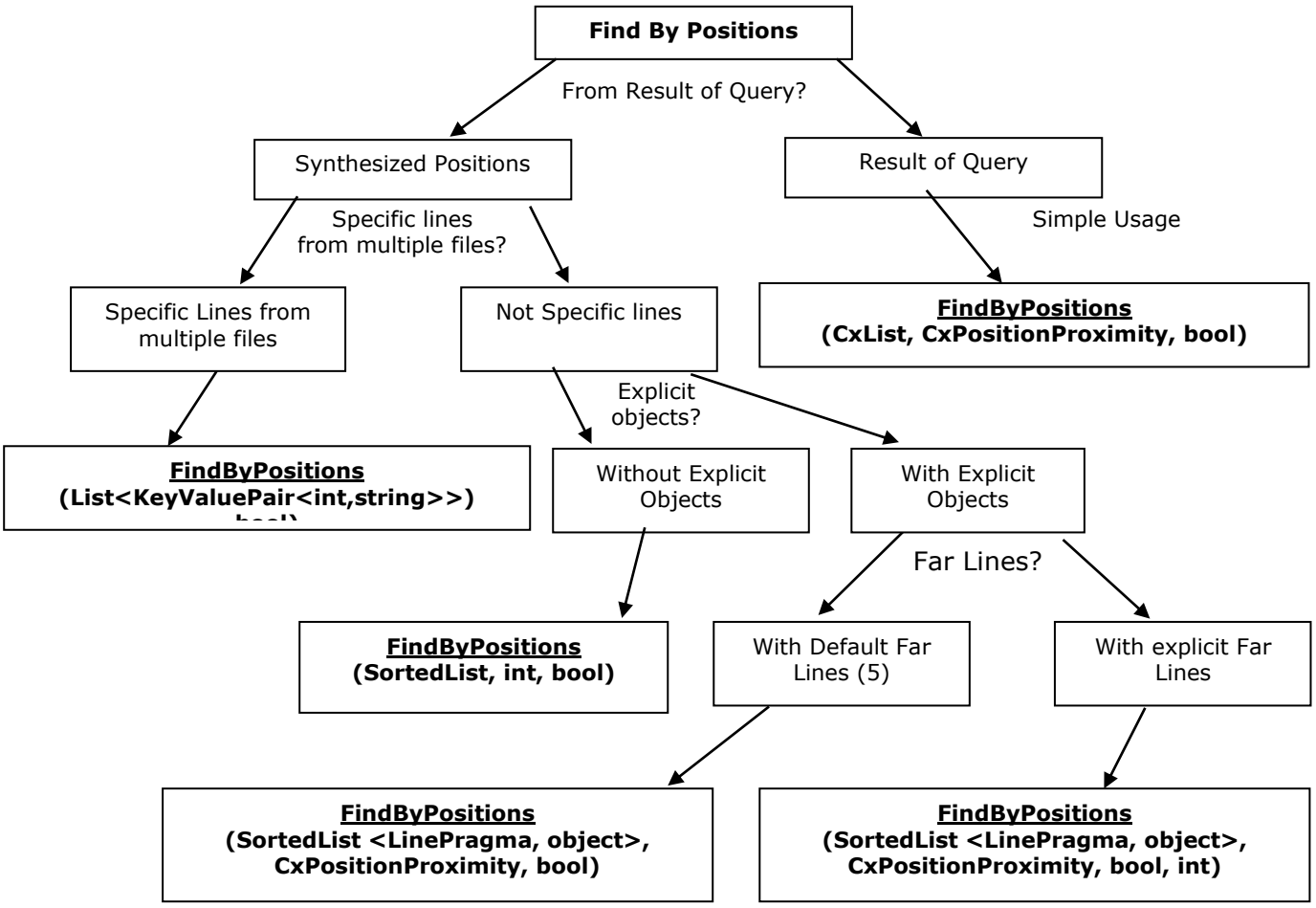
Example

```
CxQL

 This example demonstrates the CxList.FindByPosition() method.
file name "Mycode.java"
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 5;
 b = a.Method();

result = All.FindByPosition ("MyCode.java", 3, 16);
 the result would be -
      1 item found:
             5 (in a.DataMember = 5)
```

4.59 CxList.FindByPositions Methods

There are four methods (atomic queries) for using the "Find By Positions" method CxQL.

The recommended selection between the possible methods should be done according to the following tree:



4.59.1 CxList.FindByPositions Method (SortedList, int, bool)

Finds the elements of "this" instance at positions given in the pragmas list.

Syntax

```
CxQL
public CxList FindByPositions(SortedList pragmas, int extendMatch, bool
oneOnly)
```

Parameters

pragmas

A sorted list containing the pragmas to match.

extendMatch

Defines the closeness of the matching results:

0 => *ExactMatch*: find exact match

1 => *FindInLine*: extend search to objects in closest position within same line

2 => *FindClosestMatch*: extend match to closest position within the same file

oneOnly

If true, it returns one result per position.

Return Value

The elements from "this" instance that are at the required positions.

Exceptions

Exception type	Condition
ArgumentNullException	**First parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList. FindByPositions() method.
 The input source code is:

 int b, a = 5;
 if (a == 33)
      b = 6;

CxList list = All.FindByName("b");
SortedList sorted = new SortedList(new
DataCollections.LinePragmaComparer());
foreach (KeyValuePair<int, IGraph> dic in list.data){
    sorted.Add(dic.Value.LinePragma, null);
}
result = All.FindByPositions(sorted, 1, true);

 the result would consist of 2 items:
         b (in int b)
         b (in b = 6)
```

4.59.2 CxList.FindByPositions Method (CxList, CxPositionProximity, bool)

Finds the elements of "this" instance at positions given in the list using the proximity given in parameter.

Syntax

```
CxQL
public CxList FindByPositions(CxList positions, CxPositionProximity
extendMatch, bool oneOnly)
```

Parameters

positions

A list containing the pragmas to match.

extendMatch

Defines the closeness of the matching results. One of the following values:

ExactMatch: find exact match

FindInLine: extend search to objects in closest position within same line

FindClosestMatch: extend match to closest position within the same file

oneOnly

If true, it returns one result per position.

Return Value

The elements of "this" instance that are at the given positions.

Exceptions

Exception type	Condition
ArgumentNullException	**First parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList. FindByPositions() method.
 The input source code is:
 int b, a = 5;
 if (a == 33)
      b = 6;

CxList list = All.FindByName("b");
result = All.FindByPositions(list, CxPositionProximity.FindInLine,
false);

 the result would be all the elements in the 5 lines closer to lines
that appear variable b -
    2 items found
         b (in int b)
         b (in b = 6)
```

4.59.3 CxList.FindByPositions Method (SortedList<LinePragma,object>, CxPositionProximity, bool)

Finds the elements of "this" instance at positions given in the pragmas list using the proximity from the parameter.

Syntax

```
CxQL
public CxList FindByPositions(SortedList<LinePragma,object> pragmas,
CxPositionProximity extendMatch, bool oneOnly)
```

Parameters

pragmas

A sorted list containing the pragmas to match.

extendMatch

Defines the closeness of the matching results. One of the following values:

FindInLine: extend search to objects in closest position within same line.

FindClosestMatch: extend match to closest position within the same file.

ExactMatch: find exact match.

oneOnly

If true, it returns one result per position.

Return Value

The elements from the current instance that are at the given positions.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList. FindByPositions() method.
 The input source code is:

 int b, a = 5;
 if (a == 33)
      b = 6;

CxList list = All.FindByName("b");
SortedList<LinePragma, object> sorted =
new SortedList<LinePragma, object>(new
DataCollections.LinePragmaComparer());

foreach (KeyValuePair<int, IGraph> dic in list.data) {
```

```
        sorted.Add(dic.Value.LinePragma, null);
}

result = All.FindByPositions(sorted,
CxList.CxPositionProximity.FindInLine, true);

 the result would consist of 2 items:
         b (in int b)
         b (in b = 6)
```

4.59.4 CxList.FindByPositions Method (SortedList<LinePragma,object>, CxPositionProximity, bool, int)

Finds the elements of "this" instance at positions given in the pragmas list using the proximity given in parameter.

Syntax

```
CxQL
public CxList FindByPositions(SortedList<LinePragma,object> pragmas,
CxPositionProximity extendMatch, bool oneOnly, int farLines)
```

Parameters

pragmas

A sorted list containing the pragmas to match.

extendMatch

Defines the closeness of the matching results. One of the following values:

FindInLine: extend search to objects in closest position within same line.

FindClosestMatch: extend match to closest position within the same file.

ExactMatch: find exact match.

oneOnly

If true, it returns one result per position.

farLines

Acceptable line distance to look for (the default recomended setting is 5).

Return Value

The elements from "this" instance that are at the given positions.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
This example demonstrates the CxList. FindByPositions() method.
 The input source code is:
```

```
 int b, a = 5;
 if (a == 33)
      b = 6;

CxList list = All.FindByName("b");
SortedList<LinePragma, object> sorted =
new SortedList<LinePragma, object>(new
DataCollections.LinePragmaComparer());

foreach (KeyValuePair<int, IGraph> dic in list.data) {
       sorted.Add(dic.Value.LinePragma, null);
}

result = All.FindByPositions(sorted,
CxList.CxPositionProximity.FindInLine, true, 5);

 the result would consist of 2 items:
        b (in int b)
        b (in b = 6)
```

4.59.5 CxList.FindByPositions Method (List<KeyValuePair<int, string>>)

Finds the elements of "this" instance at lines of files given in parameter.

Syntax

```
CxQL
public CxList FindByPositions(List<KeyValuePair<int, string>> lines)
```

Parameters

lines

A list of pairs line/filename to search the elements.

Return Value

The subset of elements from "this" instance that are in the files given at the lines requested.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

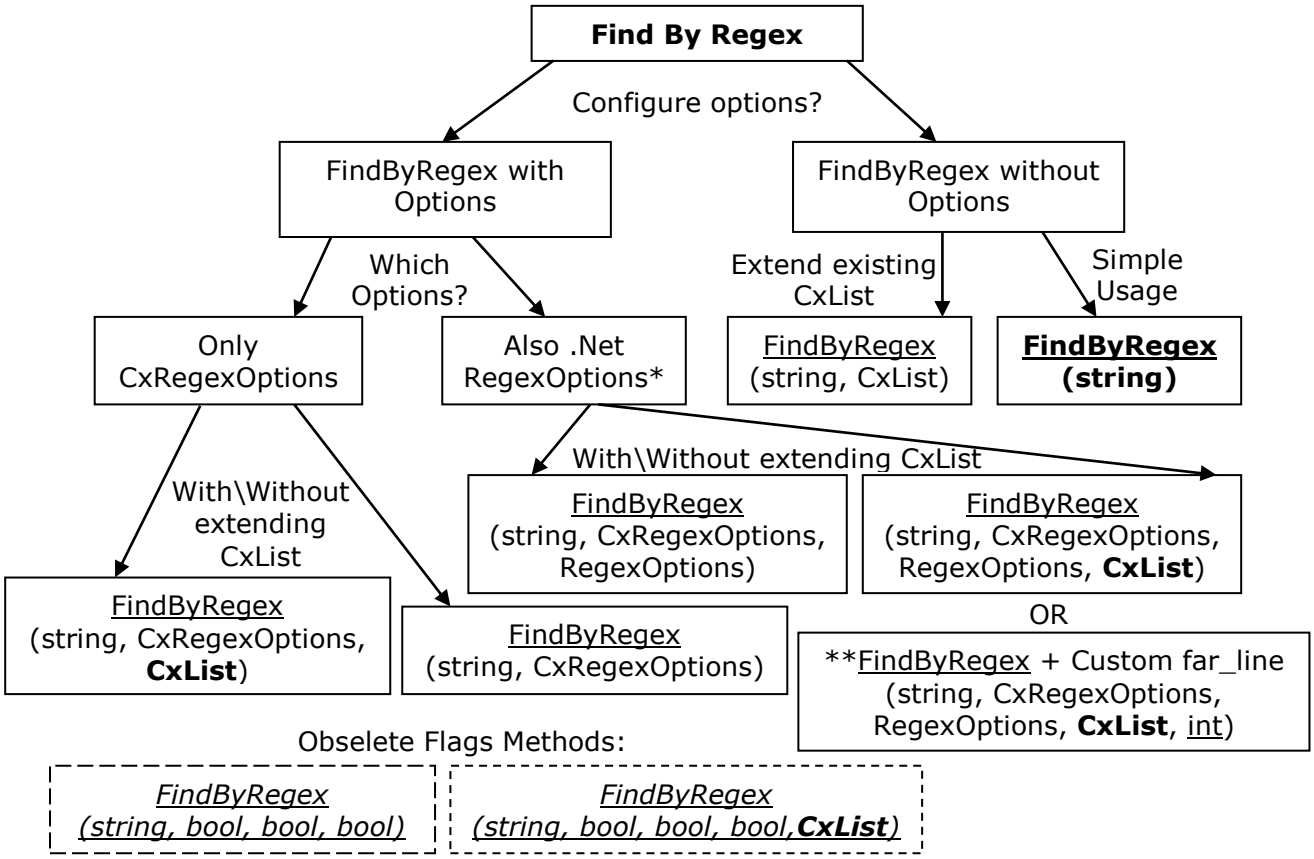
```
CxQL

 This example demonstrates the CxList. FindByPositions() method.
```

```
 The input source code is (file name is “MyCode.cs”:

  int b, a = 5;
 if (a == 33)
     b = 6;

KeyValuePair<int,string> position= new
KeyValuePair<int,string>(3,”path\\MyCode.cs”);
List<KeyValuePair<int,string>> list = new
List<KeyValuePair<int,string>>();
list.Add(position);
result = All.FindByPositions(list);

 the result would consist of 3 items:
           b
           =
           6
```

4.60 CxList.FindByRegex Methods

There are few methods (atomic queries) for using the "Find By Regex" algorithm in CxQL, some of them are obsolete and not recommended, and some of them are more comfortable according to the required parameters scenario.

The recommended selection between the possible methods should be done according to the following tree:



- Even without mentioning it explicitly in the parameter, the RegexOptions.**Multiline**, and RegexOptions.**Singleline** are always enabled in the Find-By-Regex algorithm in these queries.
- Customizing the FAR_LINES parameter is possible using the new method (the default value of this parameter is 5 and it is relevant for searching regex matches in comments).
- The full path (including namespaces) of the CxRegexOptions enum is CxList.CxRegexOptions.
- The full path (including namespaces) of the RegexOptions enum is System.Text.RegularExpressions.RegexOptions.

4.60.1 CxList.FindByRegex Method (string)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `FindByRegex(expression, null)`

- FindByRegex(expression, CxRegexOptions.None)
- FindByRegex(expression, CxRegexOptions.None, RegexOptions.None)
- FindByRegex(expression, CxRegexOptions.None, RegexOptions.None, null)
- FindByRegex(expression, CxRegexOptions.None, RegexOptions.None, null, 5)
- FindByRegex(expression, false, true, false)
- FindByRegex(expression, false, true, false, null)
- FindByRegex(expression, CxRegexOptions.None, null)

Syntax

```
CxQL
public CxList FindByRegex(string expression)
```

Parameters

expression

Regular expression string.

Return Value

A subset of this instance matches the given regular expression.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

 This example demonstrates the CxList.FindByRegex() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);

result = All.FindByRegex(@"(\s)?foo\(");

 the result would be -
      1 item found:
            foo
```

Version Information

Supported from: **CxAudit** v1.8.1

4.60.2 CxList.FindByRegex Method (string, bool, bool, bool)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string, according to specified flag parameters.

This call is equivalent to the following calls and it is highly recommended to use the enum instead of the confusing flags:

- `FindByRegex(expression, searchInComments, searchInStringLiterals, recursive, null)`
- The 3 flags are translated to `CxRegexOptions` enum in the following way (bitmask supported):
  - `(false, false, false) => CxRegexOptions.DoNotSearchInStringLiterals`
  - `(false, false, true) => CxRegexOptions.DoNotSearchInStringLiterals | CxRegexOptions.AllowOverlaps`
  - `(false, true, false) => CxRegexOptions.None`
  - `(false, true, true) => CxRegexOptions.AllowOverlaps`
  - `(true, false, false) => CxRegexOptions.SearchInComments | CxRegexOptions.DoNotSearchInStringLiterals`
  - `(true, false, true) => CxRegexOptions.SearchInComments | CxRegexOptions.DoNotSearchInStringLiterals | CxRegexOptions.AllowOverlaps`
  - `(true, true, false) => CxRegexOptions.SearchInComments`
  - `(true, true, true) => CxRegexOptions.SearchInComments | CxRegexOptions.AllowOverlaps`

After translating the flags to `CxRegexOptions` enum this call is equivalent to the following calls:

- `FindByRegex(expression, cxRegexOptions)`
- `FindByRegex(expression, cxRegexOptions, RegexOptions.None)`
- `FindByRegex(expression, cxRegexOptions, RegexOptions.None, null)`
- `FindByRegex(expression, cxRegexOptions, RegexOptions.None, null, 5)`
- `FindByRegex(expression, cxRegexOptions, null)`

Syntax

CxQL

```
public CxList FindByRegex(string expression , bool searchInComments, bool searchInStringLiterals, bool recursive)
```

Parameters

expression

Regular expression string.

searchInComments

Positive if searching inside comments is desired.

searchInStringLiterals

Positive if searching inside string literals is desired.

recursive

Positive if it is desired to allow regex matches to overlap.

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

 This example demonstrates the CxList.FindByRegex() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);

result = All.FindByRegex(@"(\s)?foo\(", false, true, false);

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.60.3 CxList.FindByRegex Method (string, bool, bool, bool, CxList)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string, according to specified flag parameters and fill the extended results parameter with the strings of the matches.

- The 3 flags are translated to `CxRegexOptions` enum in the following way (bitmask supported):
  - `(false, false, false) => CxRegexOptions.DoNotSearchInStringLiterals`
  - `(false, false, true) =>`

```
                CxRegexOptions.DoNotSearchInStringLiterals |
                CxRegexOptions.AllowOverlaps
```

- `(false, true, false) => CxRegexOptions.None`
- `(false, true, true) => CxRegexOptions.AllowOverlaps`
- `(true, false, false) =>`
  `CxRegexOptions.SearchInComments |`
  `CxRegexOptions.DoNotSearchInStringLiterals`
- `(true, false, true) =>`
  `CxRegexOptions.SearchInComments |`
  `CxRegexOptions.DoNotSearchInStringLiterals |`
  `CxRegexOptions.AllowOverlaps`
- `(true, true, false) => CxRegexOptions.SearchInComments`
- `(true, true, true) =>`
  `CxRegexOptions.SearchInComments | CxRegexOptions.AllowOverlaps`

After translating the flags to `CxRegexOptions` enum this call is equivalent to the following calls:

(It is highly recommended to use the enum instead of the confusing flags)

- `FindByRegex(expression, cxRegexOptions, RegexOptions.None, cxList)`
- `FindByRegex(expression, cxRegexOptions, RegexOptions.None, cxList, 5)`
- `FindByRegex(expression, cxRegexOptions, cxList)`

Syntax

```
CxQL
public CxList FindByRegex(string expression , bool searchInComments, bool
searchInStringLiterals, bool recursive, CxList extendedResults)
```

Parameters

expression

Regular expression string.

searchInComments

Positive if searching inside comments is desired.

searchInStringLiterals

Positive if searching inside string literals is desired.

recursive

Positive if it is desired to allow regex matches to overlap.

extendedResults

extendedResults parameter is filled with the strings of the matches.

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

 This example demonstrates the CxList.FindByRegex() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);

result = All.FindByRegex(@"(\s)?foo\(", false, true, false,
All.NewCxList());

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.60.4 CxList.FindByRegex Method (string, CxList)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string, and fill the extended results parameter with the strings of the matches.

This query search source files with regex, and return the closest same line DOM object to the matches.

If no such object exists, returns the closest object in a successive line.

Search does not include searching inside comments and string literals, and regex matches are not allowed to overlap. The matching strings are returned in the extendedResults paramater.

This call is equivalent to the following calls:

- `FindByRegex(expression, CxRegexOptions.None, RegexOptions.None, cxList)`
- `FindByRegex(expression, CxRegexOptions.None, RegexOptions.None, cxList, 5)`
- `FindByRegex(expression, false, true, false, cxList)`
  - Using the Boolean flags option is not recommended, use the enums instead.
- `FindByRegex(expression, CxRegexOptions.None, cxList)`

Syntax

```
CxQL
public CxList FindByRegex(string expression , CxList extendedResults)
```

Parameters

expression

Regular expression string.

extendedResults

extendedResults parameter is filled with the strings of the matches.

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

 This example demonstrates the CxList.FindByRegex() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);

result = All.FindByRegex(@"(\s)?foo\(", All.NewCxList());

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.60.5 CxList.FindByRegex Method (string, CxRegexOptions)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string, according to specified Checkmarx Regex Options defined in the second parameter.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `FindByRegex(expression, cxRegexOptions, RegexOptions.None)`
- `FindByRegex(expression, cxRegexOptions, RegexOptions.None, null)`
- `FindByRegex(expression, cxRegexOptions, RegexOptions.None, null, 5)`

- `FindByRegex(expression, cxRegexOptions, null)`

Syntax

```
CxQL
public CxList FindByRegex(string expression , CxRegexOptions cxOptions)
```

Parameters

expression

Regular expression string.

cxOptions

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

 This example demonstrates the CxList.FindByRegex() method.
 The input source code is:


 int a = 5;
 if (a > 3)
     foo(a);

result = All.FindByRegex(@"(\s)?foo\(", CxList.CxRegexOptions.None);

 the result would be -
      1 item found:
            foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.60.6 CxList.FindByRegex Method (string, CxRegexOptions, CxList)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string, according to specified Checkmarx Regex Options defined in the second parameter, and also fill the extended results parameter with the strings of the matches.

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `FindByRegex(expression, cxRegexOptions, RegexOptions.None, cxList)`
- `FindByRegex(expression, cxRegexOptions, RegexOptions.None, cxList, 5)`

Syntax

```
CxQL
public CxList FindByRegex(string expression , CxRegexOptions cxOptions,
CxList extendedResults)
```

Parameters

expression

Regular expression string.

cxOptions

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

extendedResults

extendedResults parameter is filled with the strings of the matches.

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

This example demonstrates the CxList.FindByRegex() method.
The input source code is:

int a = 5;
if (a > 3)
    foo(a);
```

```
result = All.FindByRegex(@"(\s)?foo\(", CxList.CxRegexOptions.None,
All.NewCxList());

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.60.7 CxList.FindByRegex Method (string, CxRegexOptions, RegexOptions)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string, according to specified Regex Options defined in the parameters (Checkmarx regex options and standard regex options).

This call is equivalent to the following calls and it is recommended to use the short call format by default:

- `FindByRegex(expression, cxRegexOptions, regexOptions, null)`
- `FindByRegex(expression, cxRegexOptions, regexOptions, null, 5)`

Syntax

```
CxQL
public CxList FindByRegex(string expression , CxRegexOptions cxOptions,
RegexOptions regularOptions)
```

Parameters

expression

Regular expression string.

cxOptions

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

regularOptions

Options to add to the regular expression (case sensitivity, etc.)

In addition to the user-defined regular-expression-options in this arguments, the alogrith also uses the following regex-options by default: `RegexOptions.Multiline`, `RegexOptions.Singleline`.

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

 This example demonstrates the CxList.FindByRegex() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);

result = All.FindByRegex(@"(\s)?foo\(", CxList.CxRegexOptions.None,
System.Text.RegularExpressions.RegexOptions.None);

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.60.8 CxList.FindByRegex Method (string, CxRegexOptions, RegexOptions, CxList)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string, according to specified Regex Options defined in the parameters (Checkmarx regex options and standard regex options), and also fill the extended results parameter with the strings of the matches.

This call is equivalent to the following call and it is recommended to use the short call format by default:

- `FindByRegex(expression, cxRegexOptions, regexOptions, cxList, 5)`

Syntax

```
CxQL
public CxList FindByRegex(string expression , CxRegexOptions cxOptions,
RegexOptions regularOptions, CxList extendedResults)
```

Parameters

expression

Regular expression string.

cxOptions

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

regularOptions

Options to add to the regular expression (case sensitivity, etc.)

In addition to the user-defined regular-expression-options in this arguments, the alogrith also uses the following regex-options by default: `RegexOptions.Multiline`, `RegexOptions.Singleline`.

extendedResults

extendedResults parameter is filled with the strings of the matches.

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

 This example demonstrates the CxList.FindByRegex() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);

result = All.FindByRegex(@"(\s)?foo\(", CxList.CxRegexOptions.None,
System.Text.RegularExpressions.RegexOptions.None, All.NewCxList());

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.60.9 CxList.FindByRegex Method (string, CxRegexOptions, RegexOptions, CxList, int, CxPositionSearchDirection)

Returns a CxList which is a subset of this instance and its elements match the specified regular expression string, according to specified Regex Options defined in the parameters (Checkmarx regex options and standard regex options), and also fill the extended results parameter with the strings of the matches.

Also get a customized far-lines parameter to be considered as acceptable lines distance when looking for regex in comments.

All the other calls to "FindByRegex.." with\without different parameters lead in the end to this specific method.

Syntax

```
CxQL
public CxList FindByRegex(string expression , CxRegexOptions cxOptions,
RegexOptions regularOptions, CxList extendedResults, int farLines,
CxPositionSearchDirection searchDirection )
```

Parameters

expression

Regular expression string.

cxOptions

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

regularOptions

Options to add to the regular expression (case sensitivity, etc.)

In addition to the user-defined regular-expression-options in this arguments, the alogrith also uses the following regex-options by default: `RegexOptions.Multiline`, `RegexOptions.Singleline`.

extendedResults

extendedResults parameter is filled with the strings of the matches.

farLines

Configure the line distance to look for regex matches in comments (it is **5** lines by default).

searchDirection

Determines the search direction that can be one of the following values: Default, Backward, Forward. The Backward and Forward values means that the search is for the CxList which is a subset of the instance that is the last one before the regular expression or just the first one after, respectively. The default search (that is the default value of the parameter) just compares the distance between both (in manner of Line distance and column distance) and chooses the one that is the closest between the two.

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegex method.

```
CxQL

 This example demonstrates the CxList.FindByRegex() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);

result = All.FindByRegex(@"(\s)?foo\(", CxList.CxRegexOptions.None,
System.Text.RegularExpressions.RegexOptions.None, All.NewCxList(), 5);

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.60.10 CxList.FindByRegexSecondOrder Method (string, CxList)

Filters a CxList of Comments DOM objects according to a check of whether a Comment object contain a match to the provided regex expression, and returns closest DOM object to those that pass the filter.

Used in C\C++ MISRA Preset queries in order to validate comments style.

Syntax

```
CxQL
public CxList FindByRegexSecondOrder(string expression , CxList
extendedResults)
```

Parameters

expression

Regular expression search string.

inputList

The comments CxList that's should be filtered.

Return Value

A subset of this instance matches the given regular expression according to the additional parameters.

Exceptions

Exception type	Condition
ArgumentNullException	**Expression parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegexSecondOrder method.

```
CxQL

 This example demonstrates the CxList.FindByRegexSecondOrder() method.
 The input source code is taken from MISRA Code_Commented_Out query:

/* Function comment is compliant. * /
void mc2_0202 ( void )
{
use_int32(0);   // Comment Not Compliant
}
*/

// Find all comments ending with } or ;
CxList extendedResult = All.NewCxList();

// All /* */ comments
CxList res = All.FindByRegex(@"/\*.*?\*/", true, false, false,
extendedResult);

// Search results for } or ; at end of comment
result = All.FindByRegexSecondOrder(@"[;{}]\s*\*/", extendedResult);
The result will be the commented out function which is found out by this
regex
```

Version Information

Supported from **CxAudit** v1.8.1

4.61 CxList.FindByRegexExt Methods

Find by regular expression in all files of the project regardless of DOM and language.

Remarks

The results are not related to DOM so they can't be compared to DOM objects returned by other functions. Results can't be used as parameters to other queries.

4.61.1 CxList.FindByRegexExt Method (string)

Syntax

```
CxQL
public CxList FindByRegexExt(string pattern)
```

Parameters

pattern

Regular expression pattern

Return Value

A list of matches for given regular expression in all project files.

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegexExt method.

```
CxQL

 This example demonstrates the CxList.FindByRegexExt() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);
 else
     FOO(a);

// foo(a)
/* foo */

result = All.FindByRegexExt(@"(\s)?foo");

 the result would be -
      3 items found:
             foo
             // FOO
             /* foo
```

Version Information

Supported from **CxAudit** version 7.1.8 and 7.1.6HF5

4.61.2 CxList.FindByRegexExt Method (string, string)

Syntax

```
CxQL
public CxList FindByRegexExt(string pattern, string fileMask)
```

Parameters

pattern

Regular expression pattern

fileMask

File mask for search. Control characters "*" and "?" are supported.

Return Value

A list of matches for given regular expression in all project files.

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByRegexExt method.

```
CxQL

 This example demonstrates the CxList.FindByRegexExt() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);
 else
     FOO(a);

// foo(a)
/* foo */

result = All.FindByRegexExt(@"(\s)?foo");

 the result would be -
      3 items found:
             foo
             // FOO
             /* foo
```

Version Information

Supported from **CxAudit** version 7.1.8 and 7.1.6HF5

4.61.3 CxList.FindByRegexExt Method (string, string, bool)

Syntax

```
CxQL
public CxList FindByRegexExt(string pattern, string fileMask, bool
searchInComments)
```

Parameters

expression

Regular expression pattern

fileMask – optional

File mask for search. Control characters "*" and "?" are supported.

For example: "*.*" looks in all files and "*.aspx" looks in aspx files.

searchInComments – optional

Allow or not search in comments

Return Value

A list of matches for given regular expression in choosen project files including or excluding results in comments.

Remarks

The return value may be empty (Count = 0).

Default values relevant only from version 7.1.8.

Example

The following code example shows how you can use the FindByRegexExt method.

```
CxQL

 This example demonstrates the CxList.FindByRegexExt() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);
 else
     FOO(a);

// foo(a)
/* foo */

result = All.FindByRegexExt(@"(\s)?foo","*.cs",false,
RegexOptions.IgnoreCase);

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** version 7.1.8 and 7.1.6HF5

4.61.4 CxList.FindByRegexExt Method (string, string, bool, RegexOptions)

Syntax

```
CxQL
public CxList FindByRegexExt(string pattern, string fileMask = "*.*", bool
searchInComments = true, RegexOptions regularOptions = RegexOptions.None)
```

Parameters

expression

Regular expression pattern

fileMask – optional

Default value: "*.*".

File mask for search. Control characters "*" and "?" are supported.

For example: "*.*" looks in all files and "*.aspx" looks in aspx files.

searchInComments – optional

Default value: true.

Allow or not search in comments

regularOptions – optional

Default value: RegexOptions.None.

Options for regular expression build from first parameter - **pattern**

Return Value

A list of matches for given regular expression in choosen project files including or excluding results in comments with regex build with specified options.

Remarks

The return value may be empty (Count = 0).

Default values relevant only from version 7.1.8.

Example

The following code example shows how you can use the FindByRegexExt method.

```
CxQL

 This example demonstrates the CxList.FindByRegexExt() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);
 else
     FOO(a);

// foo(a)
/* foo */

result = All.FindByRegexExt(@"(\s)?foo","*.cs",false,
RegexOptions.IgnoreCase);
```

```
the result would be -
     2 item found:
            foo
            FOO
```

Version Information

Supported from **CxAudit** version 7.1.8 and 7.1.6HF5

4.61.5 CxList.FindByRegexExt Method (string, string, bool, CxRegexOptions ,RegexOptions)

Syntax

```
CxQL
public CxList FindByRegexExt(string pattern, string fileMask = "*.*", bool
searchInComments = true, CxRegexOptions cxOptions =
CxRegexOptions.None,RegexOptions regularOptions = RegexOptions.None)
```

Parameters

expression

Regular expression pattern

fileMask – optional

Default value: "*.*".

File mask for search. Control characters "*" and "?" are supported.

For example: "*.*" looks in all files and "*.aspx" looks in aspx files.

searchInComments – optional

Default value: true.

Allow or not search in comments

cxOptions

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

regularOptions – optional

Default value: RegexOptions.None.

`Options for regular expression build from first parameter - pattern`

Return Value

A list of matches for given regular expression in choosen project files including or excluding results in comments with regex build with specified options.

Remarks

The return value may be empty (Count = 0).

Default values relevant only from version 7.1.8.

Example

The following code example shows how you can use the FindByRegexExt method.

```
CxQL

 This example demonstrates the CxList.FindByRegexExt() method.
```

```
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);
 else
     FOO(a);

// foo(a)
/* foo */

result = All.FindByRegexExt(@"(\s)?foo","*.cs",false,
CxRegexOptions.None, RegexOptions.IgnoreCase);

 the result would be -
      2 items found:
             foo
             FOO
```

Version Information

Supported from **CxAudit** version 7.1.8 and 7.1.6HF5

4.61.6 CxList.FindByRegexExt Method (string, string, CxRegexOptions)

Syntax

```
CxQL
public CxList FindByRegexExt(string pattern, string fileMask, CxRegexOptions
cxOptions)
```

Parameters

pattern

Regular expression pattern

fileMask

File mask for search. Control characters "*" and "?" are supported.

For example: "*.*" looks in all files and "*.aspx" looks in aspx files.

cxOptions

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

Return Value

A list of matches for given regular expression in choosen project files including or excluding results in comments with regex build with specified options.

Remarks

The return value may be empty (Count = 0).

Default values relevant only from version 7.1.8.

Example

The following code example shows how you can use the FindByRegexExt method.

```
CxQL

 This example demonstrates the CxList.FindByRegexExt() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);
 else
     FOO(a);

// foo(a)
/* foo */

result = All.FindByRegexExt(@"(\s)?foo","*.cs",
RegexOptions.IgnoreCase);

 the result would be -
      2 items found:
             foo
             FOO
```

Version Information

Supported from **CxAudit** version 7.1.8 and 7.1.6HF5

4.61.7 CxList.FindByRegexExt Method (string, string, CxRegexOptions, RegexOptions)

Syntax

```
CxQL
public CxList FindByRegexExt(string pattern, string fileMask = "*.*",
CxRegexOptions cxOptions = CxRegexOptions.None, RegexOptions regularOptions
= RegexOptions.None)
```

Parameters

expression

Regular expression pattern

fileMask – optional

Default value: "*.*".

File mask for search. Control characters "*" and "?" are supported.

For example: "*.*" looks in all files and "*.aspx" looks in aspx files.

cxOptions – optional

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

regularOptions – optional

Default value: RegexOptions.None.

Options for regular expression build from first parameter - **pattern**

Return Value

A list of matches for given regular expression in choosen project files including or excluding results in comments with regex build with specified options.

Remarks

The return value may be empty (Count = 0).

Default values relevant only from version 7.1.8.

Example

The following code example shows how you can use the FindByRegexExt method.

```
CxQL

 This example demonstrates the CxList.FindByRegexExt() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);
 else
     FOO(a);

// foo(a)
/* foo */

result = All.FindByRegexExt(@"(\s)?foo","*.cs",false,
RegexOptions.IgnoreCase);

 the result would be -
      2 item found:
             foo
             FOO
```

Version Information

Supported from **CxAudit** version 7.1.8 and 7.1.6HF5

4.61.8 CxList.FindByRegexExt Method (string, List<string>, bool, CxRegexOptions, RegexOptions)

Syntax

```
CxQL
public CxList FindByRegexExt(string expression, List<string> fileMaskList,
bool searchInComments = true, CxRegexOptions cxOptions =
CxRegexOptions.SearchInComments, RegexOptions regularOptions =
RegexOptions.None)
```

Parameters

expression

Regular expression pattern

fileMaskList

List of File masks for search. Control characters "*" and "?" are supported.

For example: "*.*" looks in all files and "*.aspx" looks in aspx files.

searchInComments – optional

Default value: true.

Allow or not search in comments

cxOptions - optional

An enum matching the relevant CxRegexOptions which are:

None, SearchInComments, DoNotSearchInStringLiterals, AllowOverlaps and SearchOnlyInComments

regularOptions – optional

Default value: RegexOptions.None.

`Options for regular expression build from first parameter - pattern`

Return Value

A list of matches for given regular expression in choosen project files including or excluding results in comments with regex build with specified options.

Remarks

The return value may be empty (Count = 0).

Default values relevant only from version 7.1.8.

Example

The following code example shows how you can use the FindByRegexExt method.

```
CxQL

 This example demonstrates the CxList.FindByRegexExt() method.
 The input source code is:

 int a = 5;
 if (a > 3)
     foo(a);
 else
     FOO(a);
// foo(a)
/* foo */

result = All.FindByRegexExt(@"(\s)?foo",new List<string>{"*.cs",
"*.js"}, false, CxRegexOptions.SearchInComments,
RegexOptions.IgnoreCase);

 the result would be -
      2 item found:
             foo
             FOO
```

Version Information

Supported from **CxAudit** version 8.0.0

4.62 CxList.FindByReturnType Method (string)

Returns a CxList which is a subset of this instance and its elements are of the specified type.

Syntax

```
CxQL
public CxList FindByReturnType(String Type)
```

Parameters

Type

The type of the objects to be found

Return Value

A subset of this instance and its elements are of the specified return type.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByReturnType method.

```
CxQL

 This example demonstrates the CxList.FindByReturnType() method.
 The input source code is:
public class a
{
       int bla()
       {
              int b, a = 5;
               if (a == 33)
                      b = 6;
           return b;
       }
}
result = All.FindByReturnType ("int");
 the result would be -
      1 items found:
             bla() (in int bla())
```

Version Information

Supported from **CxAudit** v1.8.1

4.63 CxList.FindByShortName Method (string)

Returns a CxList which is a subset of this instance and its elements are the ones which their short name is the specified string.

Syntax

```
CxQL
public CxList FindByShortName(string Name)
```

Parameters

Name

The short name of the objects to look for. Prefix and postfix wildcard (*) are supported.

Return Value

A subset of this instance and its elements are the ones which their name is the specified string.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByShortName method.

```
CxQL

 This example demonstrates the CxList.FindByShortName() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByShortName("Method");

 the result would be -
      1 item found:
             Method ( in b = a.Method() )
```

Version Information

Supported from **CxAudit** v1.8.1

4.64 CxList.FindByShortName Method (string, bool)

Returns a CxList which is a subset of this instance and its elements are the ones which their short name is the specified string, according to the specified comparison criteria.

Syntax

```
CxQL
public CxList FindByName(string ShortName, bool caseSensitive)
```

Parameters

ShortName

Contains the short name of the objects. Prefix and postfix wildcard (*) are supported.

caseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of this instance and its elements are the ones which their short name is the specified string, according to the specified comparison criteria. Where the caseSensitive value can be true for case sensitive and false for case insensitive.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByShortName method.

```
CxQL

 This example demonstrates the CxList.FindByShortName() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByShortName("method",true);

 the result would be -
      0 items found

result = All.FindByShortName("method", false);
```

```
the result would be -
     1 item found:
            a.Method (in b = a.Method() )
```

Version Information

Supported from **CxAudit** v1.8.1

4.65 CxLis.FindByShortNames Method (List<string>)

Returns a CxList which is a subset of this instance and its elements are the ones which their short name is the specified list of strings.

Syntax

```
CxQL
public CxList FindByShortNames(List<string> nodeNames)
```

Parameters

nodeNames

The short names of the objects to look for. Prefix and postfix wildcard (*) are supported.

Return Value

A subset of this instance and its elements are the ones which their name listed in specified list of strings.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0). Works afficient if wildcard not present.

Example

The following code example shows how you can use the FindByShortNames method.

```
CxQL

 This example demonstrates the CxList.FindByShortNames() method.
 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
 c = a.Method1()

result = All.FindByShortNames(new List<string> {"Method","Method1"});

 the result would be -
      2 item found:
            Method  ( in b = a.Method() )
            Method1 ( in c = a.Method1() )
```

Version Information

Supported from **CxAudit** v7.1.8

4.66 CxList.FindByShortNames Method (List<string>, bool)

Returns a CxList which is a subset of this instance and its elements are the ones which their short name is the specified string, according to the specified comparison criteria.

Syntax

```
CxQL
public CxList FindByNames(List<string> nodeNames, bool caseSensitive)
```

Parameters

nodeNames

Contains the short name of the objects. Prefix and postfix wildcard (*) are supported.

caseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of this instance and its elements are the ones which their short name is the specified string, according to the specified comparison criteria. Where the caseSensitive value can be true for case sensitive and false for case insensitive.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0). Works afficient if wildcard not present.

Example

The following code example shows how you can use the FindByShortName method.

```
CxQL

 This example demonstrates the CxList.FindByShortNames() method.
 The input source code is:


 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
 c = a.method1();


result = All.FindByShortNames(new List<string> {"method",”Method1”},
true);
 the result would be -
      1 items found
            c = a.method1();
```

```
result = All.FindByShortNames(new List<string>
{"method",”Method1”},false);

 the result would be -
      2 item found:
             a.Method  (in b = a.Method() )
             a.method1 (in c = a.method1 () )
```

Version Information

Supported from **CxAudit** v7.1.8

4.67 CxList.FindByShortName Method (CxList)

Returns a CxList which is a subset of this instance and its elements are the ones which their short name is the specified string.

Syntax

```
CxQL
public CxList FindByShortName(CxList nodesList)
```

Parameters

nodesList

The short name of the objects to look for. Prefix and postfix wildcard (*) are supported.

Return Value

A subset of this instance and its elements are the ones which their name is the specified string.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByShortName method.

```
CxQL

 This example demonstrates the CxList.FindByShortName() method.
 The input source code is:

    class Program
    {
        static void Main(string[] args)
        {
             Customer c = new customer();
        }
    }
class Customer{}
class User{}
CxList classes = All.FindByType(typeof(ClassDecl));
CxList types = All.FindByType(typeof(TypeRef));
CxList classesWithInstances = classes - classes.FindByShortName(types);

 the result would be -
      3 item found:
            Customer ( in class Customer{})
             Program ( in class Program)
              User ( in class User{})
```

Version Information

Supported from **CxAudit** v1.8.1

4.68 CxList.FindByShortName Method (CxList, bool)

Returns a CxList which is a subset of this instance and its elements are the ones which their short name is the specified string, according to the specified comparison criteria.

Syntax

```
CxQL
public CxList FindByName(CxList nodesList, bool caseSensitive)
```

Parameters

nodesList

Contains the short name of the objects. Prefix and postfix wildcard (*) are supported.

caseSensitive

Boolean which indicates to the search to be (or not) case sensitive.

Return Value

A subset of this instance and its elements are the ones which their short name is the specified string, according to the specified comparison criteria. Where the csaeSensitive value can be true for case sensitive and false for case insensitive.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByShortName method.

```
CxQL

 This example demonstrates the CxList.FindByShortName() method.
 The input source code is:


    class Program
    {
        static void Main(string[] args)
        {
             Customer c = new customer();
        }
    }
class Customer{}
class User{}

CxList classes = All.FindByType(typeof(ClassDecl));
CxList types = All.FindByType(typeof(TypeRef));
```

```
CxList classesWithInstances = classes - classes.FindByShortName(types,
true);

 the result would be -
      the same as FindByShortName(CxList nodesList)

CxList classes = All.FindByType(typeof(ClassDecl));
CxList types = All.FindByType(typeof(TypeRef));
CxList classesWithInstances = classes - classes.FindByShortName(types,
false);

 the result would be -
      2 item found:
            Program ( in class Program)
             User ( in class User{})
```

Version Information

Supported from **CxAudit** v1.8.1

4.69 CxList.FindByType Method (Type)

Returns a CxList which is a subset of this instance and its elements are of the specified type of code element.

Syntax

```
CxQL
public CxList FindByType(Type TypeName)
```

Parameters

TypeName

The type of the objects to be found

Return Value

A subset of this instance and its elements are of the specified type of code element.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByType method.

```
CxQL

 This example demonstrates the CxList.FindByType() method.
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByType (typeof(MemberAccess));
 the result would be -
      2 items found:
             a.DataMember (in a.DataMember = 3)
             a.Method (in b = a.Method())
```

Version Information

Supported from **CxAudit** v1.8.1

4.70 CxList.FindByType Method (string)

Returns a CxList which is a subset of this instance and its elements are of the specified type.

Syntax

```
CxQL
public CxList FindByType(String Type)
```

Parameters

Type

The type of the objects to be found

Return Value

A subset of this instance and its elements are of the specified type.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByType method.

```
CxQL

 This example demonstrates the CxList.FindByType() method.
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByType ("MyClass");
 the result would be -
      3 items found:
             a (in MyClass a)
             a (in a.DataMember = 3)
             a (in b = a.Method())
```

Version Information

Supported from **CxAudit** v1.8.1

4.71 CxList.FindByType Method (string, bool)

Returns a CxList which is a subset of this instance and its elements are of the specified type.

Syntax

```
CxQL
public CxList FindByType(String Type, bool CaseSensitive)
```

Parameters

Type

The type of the objects to be found

CaseSensitive

Ignore case true/false

Return Value

A subset of this instance and its elements are of the specified type.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByType method.

```
CxQL

 This example demonstrates the CxList.FindByType() method.
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

result = All.FindByType ("MyClass",true);
 the result would be -
      3 items found:
            a (in MyClass a)
            a (in a.DataMember = 3)
            a (in b = a.Method())
```

Version Information

Supported from **CxAudit** v1.8.1

4.72 CxList.FindByTypes Method (string[])

Returns a CxList which is a subset of this instance and its elements are of the specified type.

Syntax

```
CxQL
public CxList FindByType(String[] Types)
```

Parameters

Types

The types of the objects to be found

Return Value

A subset of this instance and its elements are of the specified types.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByType method.

```
CxQL

 This example demonstrates the CxList.FindByType() method.
 The input source code is:
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();

String[] arr = new String[]{"MyClass","int"};
result = All.FindByTypes(arr);
the result would be -
      6 items found:
            a (in MyClass a)
            a (in a.DataMember = 3)
            a (in b = a.Method())
            b (in int b)
             b (in b = a.Method())
            MyClass (in MyClass a)
```

Version Information

Supported from **CxAudit** v1.8.1

4.73 CxList.FindByTypes Method (string[], bool)

Returns a CxList which is a subset of this instance and its elements are of the specified type.

Syntax

```
CxQL
public CxList FindByType(String[] Types, bool caseSensitive)
```

Parameters

Types

The types of the objects to be found

CaseSensitive

Ignore case true/false

Return Value

A subset of this instance and its elements are of the specified types.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the FindByType method.

```
CxQL

 This example demonstrates the CxList.FindByType() method.

 The input source code is:

 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
String[] arr = new String[]{"MyClass","int"};
result = All.FindByTypes(arr,false);
the result would be -
      6 items found:
            a (in MyClass a)
            a (in a.DataMember = 3)
            a (in b = a.Method())
            b (in int b)
             b (in b = a.Method())
            MyClass (in MyClass a)
```

Version Information

Supported from **CxAudit** v7.1.8

4.74 CxList.FindDefinition Method (CxList)

Returns a CxList which is a subset of "this" instance, with elements that are the definition locations of the first element in the given CxList.

Syntax

```
CxQL
public CxList FindDefinition(CxList items)
```

Parameters

Items

Items whose definition to be found.

Return Value

A subset of "this" instance, with elements that are the definition locations of the first element in the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindDefinition() method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = a;

result = All.FindDefinition(All.FindByName("*b*"));

 The result would consist of 1 item:
             b (in int b, a = 5)
```

Version Information

Supported from **CxAudit** v1.8.1

4.75 CxList.FindInitialization Method (CxList)

Returns a CxList which is a subset of "this" instance and the elements are the initialization values of the elements from the given CxList.

Syntax

```
CxQL
public CxList FindInitialization(CxList declarators)
```

Parameters

Declarators

A CxList of declarators.

Return Value

A subset of "this" instance whose elements are the initialization values of the given CxList elements.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.FindInitialization() method.
 The input source code is:

 int b = 5;

CxList declarators = All.FindByType(typeof(Declarator));
result = All.FindInitialization(declarators);

 The result would consist of 1 item:
             5
```

Version Information

Supported from **CxAudit** v1.8.1

4.76 CxList.GetAncOfType Method (Type)

Returns a CxList with all the elements that are CxDOM first ancestor of the calling CxList and which are of type t. First ancestor means that it searches upward in the CxDOM graph until the first ancestor matching the condition (type t), and NOT that it searches only for fathers

Syntax

```
CxQL
public CxList GetAncOfType(Type t)
```

Parameters

The type of DOM objetct the methods looks for

Return Value

Returns a CxList with all the CxDOM elements of type t, which are first ancestor, of some element in the calling CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).
This command does not return a subset of the CxList, but a subset of All.

Example

The following code example shows how you can use the GetAncOfType method.

```
CxQL

 This example demonstrates the CxList.GetAncOfType() method.
 The input source code is:
 if (a>b)
 {
      c=100;
 }
 else
 {
      if(a<100)
      {
       d=200;
      }
 }


result = All.FindByName("d"). GetAncOfType(typeof(IfStmt));
 the result would be -
      1 item found:
      if (in  if(a<100))
```

Version Information

Supported from **CxAudit** v2.0.5

4.77 CxList.GetArrayOfNodeIds Method ()

Returns a ArrayList which is a set of all elements IDs All this CxList.

Syntax

```
CxQL
public ArrayList GetArrayOfNodeIds()
```

Parameters

None

Return Value

ArrayList which is a set of all elements IDs All this CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Example

The following code example shows how you can use the FindByReturnType method.

```
CxQL

 This example demonstrates the CxList.FindByReturnType() method.
 The input source code is:
public class a
{
void foo(){
 MyClass a;
 int b;
 a.DataMember = 3;
 b = a.Method();
 }
}
CxList ls = All;
foreach(int NodeId in ls.GetArrayOfNodeIds())
{
	if(NodeId !=1)
	{
		result = All.FindById(NodeId);
	}
}
```

Version Information

Supported from **CxAudit** : v1.8.1

4.78 CxList.GetByAncs Method (CxList)

Returns all elements in this instance that is a CxDOM descendant of an element of the parameter.

Syntax

```
CxQL
public CxList GetByAncs(CxList ancs)
```

Parameters

ancs

The Ancestors whose descendants are to be returned

Return Value

Returns all elements in this instance that descends any of the elements in the parameter

Example

The following code example shows how you can use the GetByAncs method.

```
CxQL

 This example demonstrates the CxList.GetByAncs() method.
 The input source code is:
 public notmuch (boolean tf)
{
      boolean localboolean = tf;
}

result = All.GetByAncs(All.FindByName("notmuch"));

 6 items found:
notmuch
boolean (in Boolean tf)
tf
boolean
localboolean
=
tf (in localoolean=tf)
```

Version Information

Supported from **CxAudit** v2.0.5

4.79 CxList.GetByBinaryOperator Method (BinaryOperator)

Returns a CxList which is a subset of this instance and its elements are binary expressions with a given binary operator.

Syntax

```
CxQL
public CxList GetByBinaryOperator(BinaryOperator opr)
```

Parameters

opt

Enum type of binary operators.

Return Value

A subset of this instance with binary expressions which have a given binary operator.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the GetByBinaryOperator method.

```
CxQL

 This example demonstrates the CxList.GetByBinaryOperator() method.
 The input source code is:

 int i;
 if(i < 1)
       ...

result = All.GetByBinaryOperator(BinaryOperator.LessThan);

 the result would be -
      1 item found:
             <
```

Version Information

Supported from **CxAudit** v1.8.1

4.80 CxList.GetByClass Method (CxList)

Returns all elements in “this” instance that belong to any of the classes in the parameter.

Syntax

```
CxQL
public int GetByClass(CxList classes)
```

Parameters

classes

The classes whose elements to be returned

Return Value

Returns all elements in this instance that belong to any of the classes in the parameter

Example

```
CxQL

 This example demonstrates the CxList.GetByClass() method.
 The input source code is:
 class cl1
 {
      void foo()
      {
             int a = 3;
             int b = 5;
      }
 }
 class cl2
 {
      void foo2()
      {
             int c = 3;
      }
 }

result = All.GetByClass(All.FindByName("*.cl1")).FindByName("3");

 The result would consist of 1 item found:
      3 (in int a = 3)
 Notice that 3 (in int c = 3) doesn't appear in the results, since it is
not  in the "cl1" class
```

4.81 CxList.GetByMethod Method (CxList)

Returns all elements in this instance that belong to any of the methods in the parameter

Syntax

```
CxQL
public int GetByMethod(CxList methods)
```

Parameters

methods

The methods whose elemets to be returned

Return Value

Returns all elements in this instance that belong to any of the methods in the parameter

Example

The following code example shows how you can use the GetByMethod method.

```
CxQL

 This example demonstrates the CxList.GetByMethod() method.
 The input source code is:
 class cl1
 {
      void foo()
      {
             int a = 3;
             int b = 5;
      }
      void foo2()
      {
             int c = 3;
      }
 }

result = All.GetByClass(All.FindByName("foo2")).FindByName("3");

 1 item found:
      3 (in int c = 3)
 Notice that 3 (in int a = 3) doesn't appear in the results, since it is
not  in the "foo2" method
```

Version Information

Supported from **CxAudit** v2.0.5

4.82 CxList.GetClass Method (CxList)

Returns the classes of this instance containing the objects in the parameter.

Syntax

```
CxQL
public CxList GetClass(CxList elements)
```

Parameters

elements

The elements whose classes to be returned

Return Value

Returns the classes of this instance containing the objects in the parameter.

Example

The following code example shows how you can use the GetClass method.

```
CxQL

 This example demonstrates the CxList.GetClass() method.
 The input source code is:
 class cl1
 {
      void foo()
      {
             int a = 3;
             int b = 5;
      }
 }

result = All.GetClass(All.FindByName ("5"));

 1 item found:
      cl1 (in class cl1)
```

Version Information

Supported from **CxAudit** v2.0.5

4.83 CxList.GetCxListByPath Method ()

Create enumerator on CxList that enumerate on all existing paths.

Syntax

CxQL

public IEnumerable<CxList> GetCxListByPath()

Parameters

No parameters

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

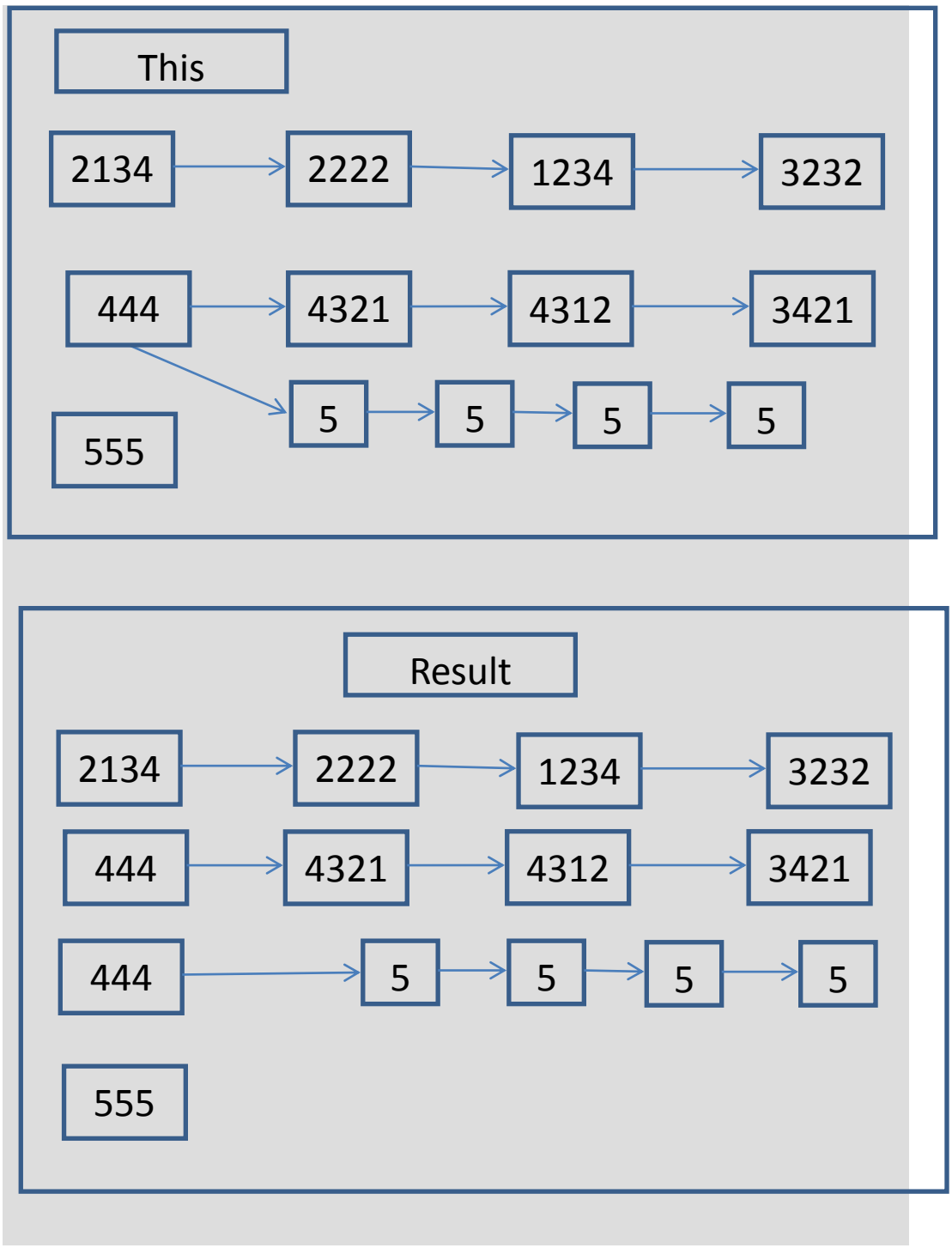
Remarks

None

Example

CxQL

```
 This example demonstrates the IEnumerable<CxList> GetCxListByPath()
method.

foreach (CxList thisCxList in this.GetCxListByPath())
{
   // thisCxList shall include one node and one path. If in “this”
exists nodes without
   // pathes than thisCxList will have only one node.
}
```



Version Information

Supported from **CxAudit** v7.1.3

4.84 CxList.GetEnumerator Method ()

Return IEnumerator of CxList.Data

Syntax

CxQL

```
public IEnumerator GetEnumerator()
```

Parameters

none

Return Value

Enumerator of CxList.Data.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

Not in use (deprecated). A simpler implementation is by:

```
foreach  (CxList cxItem in resultList)
{
  :
}
```

Example

CxQL

```
 This example demonstrates the GetEnumerator, CxDebug and GetFirstGraph
method.
 The input source code is:

class cl1
 {
      void foo()
      {
            int a = 3;
            int b = 5;
      }
 }

IEnumerator ieNum = All.GetEnumerator();
bool finish = false;
int i = 1;
while (!finish)
{
```

```
      if (!ieNum.MoveNext())
      {
            finish = true;
      }
      else
      {
            CxList curr = (CxList) ieNum.Current;
            if (curr.GetFirstGraph() != null)
            {
                  CxDebug("#=" + i.ToString() +
                  "  curr name = " + curr.GetName() + "   type = " +
                 curr.GetFirstGraph().GraphType.ToString());
                  i++;
            }
      }
}
```

the result would be on DebugMessage tab in CxAudit program

Query | Results | Comments | Debug Messages

Query Name	Debug Message
CxDefaultQuery	#=1 curr name = DefaultNamespace type = NamespaceDecl
CxDefaultQuery	#=2 curr name = cl1 type = ClassDecl
CxDefaultQuery	#=3 curr name = type = MemberDeclCollection
CxDefaultQuery	#=4 curr name = foo type = MethodDecl
CxDefaultQuery	#=5 curr name = void type = TypeRef
CxDefaultQuery	#=6 curr name = type = StatementCollection
CxDefaultQuery	#=7 curr name = type = VariableDeclStmt
CxDefaultQuery	#=8 curr name = int type = TypeRef
CxDefaultQuery	#=9 curr name = a type = Declarator
CxDefaultQuery	#=10 curr name = 3 type = IntegerLiteral
CxDefaultQuery	#=11 curr name = type = VariableDeclStmt
CxDefaultQuery	#=12 curr name = int type = TypeRef
CxDefaultQuery	#=13 curr name = b type = Declarator
CxDefaultQuery	#=14 curr name = 5 type = IntegerLiteral

Version Information

Supported from **CxAudit** v1.8.1

4.85 CxList.GetFathers Method ()

Returns a CxList which contains the direct fathers of the elements of "this" instance.

Syntax

```
CxQL
public CxList GetFathers ()
```

Return Value

A CxList which contains the direct fathers of the element of "this" instance.

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.GetFathers () method.
 The input source code is:

 int b, a = 5;
 if (a > 3)
      b = 6;

CxList six = All.FindByName("6");
result = six. GetFathers();

 the result would consist of 1 item found:
             =
```

4.86 CxList.GetFinallyClause Method (CxList)

Returns a CxList which is a subset of this instance and its elements are finally clauses of the specified CxList of try statements.

Syntax

```
CxQL
public CxList GetFinallyClause (CxList TryList)
```

Parameters

TryList

CxList of try statements.

Return Value

A subset of this instance with finally clauses.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the GetFinallyClause method.

```
CxQL

 This example demonstrates the CxList.GetFinallyClause() method.
 The input source code is:

int j;
 try
 {
      int i = 0;
      j = 1 / i;
 }
 finally
 {
      j = 1;
 }

CxList Try = All.FindByType(typeof(TryCatchFinallyStmt));
result = All.GetFinallyClause(Try);

 the result would be -
     1 item found:
          finally
```

Version Information

Supported from **CxAudit** v1.8.1

4.87 CxList.GetFirstGraph Method ()

Returns a first data element in requested CxList. Using to get internal data of first object in requested CxList

Syntax

CxQL

```
public CSharpGraph GetFirstGraph()
```

Parameters

none

Return Value

A first element in Data. If CxList empty return null.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

N/A

Example

CxQL

```
 This example demonstrates the CxList.GetFirstGraph() method.
 The input source code is:

class cl1
 {
      void foo()
      {
            int a = 3;
            int b = 5;
      }
 }
result = All.FindByShortName("foo");
if (result.Count > 0)
      CxDebug(result.GetFirstGraph().ShortName);

 the result would be on DebugMessage tab in CxAudit program
       foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.88 CxList.GetMembersOfTarget Method ()

Returns a CxList with all found members of the specified target.

Syntax

```
CxQL
public CxList GetMembersOfTarget()
```

Return Value

A CxList with members of a given target.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the GetMembersOfTarget method.

```
CxQL

 This example demonstrates the CxList. GetMembersOfTarget() method.
 The input source code is:

 StreamWriter sw = new StreamWriter();
 sw.Write("");

CxList sWriter = All.FindByType("StreamWriter");
result = sWriter. GetMembersOfTarget();

 the result would be -
      1 item found:
            Write
```

Version Information

Supported from **CxAudit** v1.8.1

4.89 CxList GetRightmostMember()

Returns a CxList with the rightmost members of the specified target.

Syntax

```
CxQL
public CxList GetRightmostMember()
```

Return Value

A CxList with the rightmost members of a given target.

Exceptions

Exception type	Condition

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the GetRightmostMember method.

```
CxQL

 This example demonstrates the CxList. GetRightmostMember() method.
 The input source code is:

int i = foo().Bar().a.b;

CxList foo = All.FindByName("foo");
result = foo. GetRightmostMember();

 the result would be -
      1 item found:
             b
```

Version Information

Supported from **CxAudit** v8.0

4.90 CxList GetLeftmostTarget()

Returns a CxList with leftmost target of the specified member.

Syntax

```
CxQL
public CxList GetLeftmostTarget()
```

Return Value

A CxList with the leftmost target of a given member.

Exceptions

Exception type	Condition

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the GetRightmostMember method.

```
CxQL

 This example demonstrates the CxList. GetLeftmostTarget() method.
 The input source code is:

int i = foo().Bar().a.b;

CxList b = All.FindByName("b");
result = b.LeftmostTarget();

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v8.0

4.91 CxList.GetMembersWithTargets Method ()

Returns a CxList which is a subset of "this" instance with nodes that are part of a member/target pair (typical example: **target.member**) and have a direct target (i.e. they are the member).

Syntax

```
CxQL
public CxList GetMembersWithTargets()
```

Return Value

Returns a CxList which is a subset of "this" instance with nodes that have a direct target.

Exceptions

Exception type	Condition

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the GetMembersWithTargets method.

```
CxQL

 This example demonstrates the CxList.GetMembersWithTargets() method.
 The input source code is:

 int num = 55;
 string Str = num.ToString().ToUpper().PadLeft(5, ' ');

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
result = methods.GetMembersWithTargets();

 the result would be -
      3 items found:
            PadLeft, ToUpper, ToString in
num.ToString().ToUpper().PadLeft(5, ' ');
```

Version Information

Supported from **CxAudit** v1.8.1

4.92 CxList.GetMembersWithTargets Method (CxList)

Returns a CxList which is a subset of “this” instance with nodes that are part of a member/target pair (typical example: **target.member**) and have a direct target in the CxList paramemeter "targets".

Syntax

```
CxQL
public CxList GetMembersWithTargets(CxList targets)
```

Parameters

targets - CxList of DOM objects which might be the target(s) of elements in "this"

Return Value

Returns a CxList which is a subset of “this” instance with nodes that have a direct target in "**targets**" parameter

Exceptions

Exception type	Condition

Remarks

The return value may be empty (Count = 0).

If targets is null – returns an empty CxList

Example

The following code example shows how you can use the GetMembersWithTargets method.

```
CxQL

 This example demonstrates the CxList.GetMembersWithTargets() method.
 The input source code is:

 int num = 55;
 string Str = num.ToString().ToUpper().PadLeft(5, ' ');

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
CxList num = All.FindByShortName("num");
result = methods.GetMembersWithTargets(num);

 the result would be -
      1 item found:
            ToString in num.ToString().ToUpper().PadLeft(5, ' ');
```

Version Information

Supported from **CxAudit** v1.8.1

4.93 CxList.GetMembersWithTargets Method (CxList, int)

Returns a CxList which is a subset of “this” instance with nodes that are part of a member/target pair (typical example: **target.member**) and have a direct target in "targets" parameter, or a target of target, a target of a target of a target …. Up to depthLimit depth

Syntax

```
CxQL
public CxList GetMembersWithTargets (CxList targets, int depthLimit)
```

Parameters

targets - CxList of DOM objects which might be the target(s) of elements in "this", or the target of a target of this…
depthLimit – the number of iterations to look for targets

Return Value

Returns a CxList which is a subset of “this” instance with nodes that have a direct target.

Exceptions

Exception type	Condition

Remarks

The return value may be empty (Count = 0).

If targets is null – returns an empty CxList

Example

The following code example shows how you can use the GetMembersWithTargets method.

```
CxQL

 This example demonstrates the CxList. GetMembersWithTargets() method.
 The input source code is:

 int num = 55;
 string Str = num.ToString().ToUpper().PadLeft(5, ' ');

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
CxList num = All.FindByShortName("num");
result = methods.GetMembersWithTargets(num, 2);

 the result would be -
      2 items found:
           ToString, ToUpper in num.ToString().ToUpper().PadLeft(5, '
');
```

Version Information

Supported from **CxAudit** v1.8.1

4.94 CxList.GetMethod Method (CxList)

Returns CxList which is a subset of this instance and its elements are methods of the specified CxList.

Syntax

```
CxQL
public CxList GetMethod(CxList list)
```

Parameters

List

CxList of any DOM objects.

Return Value

A subset of this instance which contains methods of the specified CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the GetMethod method.

```
CxQL

 This example demonstrates the CxList. GetMethod() method.
 The input source code is:

 Class Cl
 {
       void foo()
       {
             int i = 1;
             i++;
       }
 }

CxList I_var = All.FindByShortName("i");
result = All. GetMethod(I_var);

 the result would be -
      1 item found:
             foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.95 CxList.GetName Method ()

Returns a first data element name in requested CxList. Using to get internal data of first object in requested CxList

Syntax

CxQL

```
public string GetName()
```

Parameters

none

Return Value

A name of the first element in Data. If CxList empty return null.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

None

Example

CxQL

```
 This example demonstrates the CxList.GetName() method.
 The input source code is:

class cl1
 {
      void foo()
      {
             int a = 3;
             int b = 5;
      }
 }
result = All.FindByShortName("foo");
if (result.Count > 0)
       CxDebug(result.GetName());

 the result would be on DebugMessage tab in CxAudit program
       foo
```

Version Information

Supported from **CxAudit** v1.8.1

4.96 CxList.GetParameters Method (CxList)

Returns a CxList which is a subset of this instance and its elements are parameters of methods elements provided in CxList.

Syntax

```
CxQL
public CxList GetParameters (CxList MethodsList)
```

Parameters

MethodList

CxList of methods.

Return Value

Returns a CxList with all the parameters, from instance CxList, of the methods in MethodsLis.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

This example demonstrates the CxList.GetParameters(MethodsList) method.
The input source code is:

 foo(1, 3, i);

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
result = All.GetParameters(methods);

 the result would consist of 3 items:
             1,
             3,
             i
```

Version Information

Supported from **CxAudit** v1.8.1

4.97 CxList.GetParameters Method (CxList, int)

Returns a CxList which is a subset of instance CxList and its elements are parameters of methods elements provided in CxList.

Syntax

```
CxQL
public CxList GetParameters (CxList MethodsList, int paramNo)
```

Parameters

MethodList

CxList of methods.

paramNo

The number of parameter to return (begins with 0)

Return Value

Returns a CxList with paramNo parameters, from instance CxList, of the methods in MethodsList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

This example demonstrates the CxList.GetParameters(MethodsList,paramNo)
method.
The input source code is:

 foo(1, 3, i);

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
result = All.GetParameters(methods, 1);

 the result would consist of 1 items:
            3
```

Version Information

Supported from **CxAudit** v1.8.1

4.98 CxList.GetPathsOrigins Method ()

Returns a CxList which is a subset of instance CxList and contains end nodes of paths.

Syntax

```
CxQL
public CxList GetPathsOrigins ()
```

Return Value

Returns CxList that contains end nodes of paths

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.GetPathsOrigins() method.
 The input source code is:

 public void setString (String str){
      if (str.length >0){
            lst.add(str);
      }
 }
CxList paths = All.DataInfluencingOn(All.FindByShortName("add"));
result = paths.GetPathsOrigins();

the result would consist of 3 items:
           lst     (in lst.add(str);)
            str    (in lst.add(str);)
            str    (in (String str);)
```

4.99 CxList.GetStartAndEndNodes Method (GetStartEndNodesType)

Returns CxList which is a subset of instance CxList and contains start nodes or end nodes or both start and nodes of path or all nodes in path.

Syntax

```
CxQL
public CxList GetStartAndEndNodes (GetStartEndNodesType type)
```

Parameters

Type

The type of nodes to be returned:

CxList.GetStartEndNodesType.StartNodesOnly

CxList.GetStartEndNodesType.EndNodesOnly

CxList.GetStartEndNodesType.StartAndEndNodes

CxList.GetStartEndNodesType.AllNodes

Return Value

Returns CxList which is a start nodes or end nodes or both start and nodes of path or all nodes in path.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList.GetStartsAndEndNodes (type)
method.
 The input source code is:

public void setString (String str){
      lst.add(str);
}

CxList paths = All.DataInfluencingOn(All.FindByShortName("add"));

1. result =
paths.GetStartAndEndNodes(CxList.GetStartEndNodesTypeStartNodesOnly)
;
the result would consist of 2 items:
```

```
            lst     (in lst.add(str);)
            str     (in (String str);)
2. result = paths.GetStartsAndEndNodes(CxList.GetStartEndNodesType
EndNodesOnly);
the result would consist of 1 items:
             add    (in lst.add(str);)
3. result=
paths.GetStartAndEndNodes(CxList.GetStartEndNodesType.StartAndEndNod
es);
the result would consist of 3 items:
            lst     (in lst.add(str);)
            str     (in (String str);)
             add    (in lst.add(str);)
4. result =
paths.GetStartAndEndNodes(CxList.GetStartEndNodesType.AllNodes);
the result would consist of 4 items:
            lst     (in lst.add(str);)
            str     (in (String str);)
             add    (in lst.add(str);)
             str    (in lst.add(str);)
5. result =
paths.GetStartAndEndNodes(CxList.GetStartEndNodesType.AllButNotStart
AndEnd);
the result would consist of 2 items:
            lst     (in lst.add(str);)
             str    (in lst.add(str);)
```

Version Information

Supported from **CxAudit** v7.1.2

4.100 CxList.GetTargetOfMembers Method ()

Returns the list of elements which are the targets from the members of "this" instance.

Syntax

```
CxQL
public CxList GetTargetOfMembers()
```

Parameters

none

Return Value

A list of objects from which "this" instance elements are member of.

Example

```
CxQL

 This example demonstrates the CxList.GetTargetOfMembers() method.
 The input source code is:
 class cl1
 {
      void foo()
      {
            int a = obj.func();
      }
 }

result = All.FindByName ("*.func").GetTargetOfMembers();

 The result would consist of 1 item:
      obj (in int a = obj.func())
```

4.101 CxList.GetTargetsWithMembers Method ()

Returns a CxList which is a subset of "this" instance with nodes that are part of a member/target pair (typical example: **target.member**) and have a direct member (i.e. they are the target).

Syntax

```
CxQL
public CxList GetTargetsWithMembers()
```

Return Value

Returns a CxList which is a subset of "this" instance with nodes that have a direct member.

Exceptions

Exception type	Condition

Remarks

The return value may be empty (Count = 0).

Example

The following code example shows how you can use the GetTargetsWithMembers method.

```
CxQL

 This example demonstrates the CxList.GetTargetsWithMembers() method.
 The input source code is:

 int num = 55;
 string Str = num.ToString().ToUpper().PadLeft(5, ' ');

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
result = methods.GetTargetsWithMembers();

 the result would be -
      2 items found:
            ToUpper, ToString in num.ToString().ToUpper().PadLeft(5, '
');
```

Version Information

Supported from **CxAudit** v1.8.1

4.102 CxList.GetTargetsWithMembers Method (CxList)

Returns a CxList which is a subset of “this” instance with nodes that are part of a member/target pair (typical example: **target.member**) and have a direct member in the CxList paramemeter "members".

Syntax

```
CxQL
public CxList GetTargetsWithMembers(CxList members)
```

Parameters

members - CxList of DOM objects which might be the member(s) of elements in "this"

Return Value

Returns a CxList which is a subset of “this” instance with nodes that have a direct member in members parameter.

Exceptions

Exception type	Condition

Remarks

The return value may be empty (Count = 0).

If members is null – returns an empty CxList

Example

The following code example shows how you can use the GetTargetsWithMembers method.

```
CxQL

 This example demonstrates the CxList.GetTargetsWithMembers() method.
 The input source code is:

 int num = 55;
 string Str = num.ToString().ToUpper().PadLeft(5, ' ');

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
CxList member = All.FindByShortName("PadLeft");
result = methods.GetTargetsWithMembers(member);

 the result would be -
      1 item found:
            ToUpper in num.ToString().ToUpper().PadLeft(5, ' ');
```

Version Information

Supported from **CxAudit** v1.8.1

4.103 CxList.GetTargetsWithMembers Method (CxList, int)

Returns a CxList which is a subset of “this” instance with nodes that are part of a member/target pair (typical example: **target.member**), and have a direct member in CxList paramemeter "members", or a member of a member… up to depth depthLimit

Syntax

```
CxQL
public CxList GetTargetsWithMembers(CxList targets, int depthLimit)
```

Parameters

members - CxList of DOM objects which might be the member(s) of elements in "this", or the memebr of member of this, …

depthLimit – the number of iterations to look for members

Return Value

Returns a CxList which is a subset of “this” instance with nodes that have a direct /chain memebr.

Exceptions

Exception type	Condition

Remarks

The return value may be empty (Count = 0).

If members is null – returns an empty CxList

Example

The following code example shows how you can use the GetTargetsWithMembers method.

```
CxQL

 This example demonstrates the CxList.GetTargetsWithMembers() method.
 The input source code is:

string Str = "sample".ToString().ToUpper().PadLeft(5, ' ');

CxList methods = All.FindByType(typeof(MethodInvokeExpr));
CxList member = All.FindByShortName("PadLeft");
result = methods.GetTargetsWithMembers(member, 2);

 the result would be -
      2 items found:
           ToString, ToUpper in num.ToString().ToUpper().PadLeft(5, '
');
```

Version Information

Supported from **CxAudit** v1.8.1

4.104 CxList.InheritsFrom Method (string)

Returns a CxList which is a subset of "this" instance and its elements are inherited from the given class name.

Syntax

```
CxQL
public CxList InheritsFrom(string baseClassName)
```

Parameters

baseClassName

The name of the base class.

Return Value

A subset of "this" instance which elements are inherited from the given base class name.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.InheritsFrom() method.
 The input source code is:

 class BClass
 {

 }

 class CClass : BClass
 {

 }

result = All.InheritsFrom("BClass");

 The result would consist of 1 item:
           CClass
```

Version Information

Supported from **CxAudit** v1.8.1

4.105 CxList.InheritsFrom Method (CxList)

Returns a CxList which is a subset of "this" instance and its elements are inherited from the given CxList of classes.

Syntax

```
CxQL
public CxList InheritsFrom(CxList baseClassList)
```

Parameters

baseClassList

The CxList of base classes.

Return Value

A subset of "this" instance which elements are inherited from the given base classes.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Comments

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList.InheritsFrom() method.
 The input source code is:

 class BClass
 {

 }

 class CClass : BClass
 {

 }

CxList cl = All.FindByName("BClass");
result = All.InheritsFrom(cl);

 The result would consist of 1 item:
           CClass
```

Version Information

Supported from **CxAudit** v1.8.1

4.106 CxList.IntersectWithNodes Method (CxList)

Returns a CxList which is a subset of paths, which are the instance CxList, that includes elements of intersected CxList.

Syntax

```
CxQL
public CxList IntersectWithNodes (CxList intersect)
```

Parameters

intersect

intersected CxList elements

Return Value

Returns a CxList which is a subset of paths , that includes elements of intersected CxList.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList. FindByPosition() method.
 The input source code is:

 public void setString (String str){
       if (str.length >0){
             lst.add(str);
       }
       else{
             String otherStr ="string is empty";
             lst.add(otherStr);
       }
}

CxList intersect = All.FindByShortName("otherStr");
CxList paths = All.DataInfluencingOn(All.FindByShortName("add"));
result = paths.IntersectWithNodes(intersect);

the result would consist of 3 items:
       all ending at add   (in lst.add(otherStr);)
       starting
            otherStr             (in lst.add(otherStr);)
```

```
            otherStr            (in String otherStr ="string is
empty";)
            "string is empty"   (in String otherStr ="string is
empty";)
```

Version Information

Supported from **CxAudit** 7.1.2

4.107 CxList.ReduceFlow Method (CxList.ReduceFlowType)

Returns CxList which is a subset of instance CxList and consists of longest paths to/from destination element for CxList.ReduceFlowType.ReduceSmallFlow parameter or shortest paths to/from destination element for CxList.ReduceFlowType.ReduceBigFlow parameter.

Syntax

```
CxQL
public CxList ReduceFlow (CxList.ReduceFlowType)
```

Parameters

Type

The type of flow for reduce:

CxList.ReduceFlowType.ReduceBigFlow

CxList.ReduceFlowType.ReduceSmallFlow

Return Value

Returns CxList which is a subset of paths that consists of longest paths or shortest paths to/from destination element, depending on ReduceFlow methods parameter.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL
 This example demonstrates the CxList. ReduceFlow () method.
 The input source code is:

ArrayList<String> lst = new ArrayList<String>();
public void setString (String str){
      if (str.length >0){
            lst.add(str);
      }
      else{
            String otherStr ="string is empty";
            lst.add(otherStr);
      }
}

CxList paths = All.DataInfluencingOn(All.FindByShortName("add"));
```

```
1.result =  paths.ReduceFlow(CxList.ReduceFlowType.ReduceBigFlow);
the result would consist of 4 items:
      all ending at add   (in lst.add(otherStr);)
      starting
            lst                (in lst.add(str))
            str                (in lst.add(str))
            lst                (in lst.add(otherStr);)
           otherStr            (in lst.add(otherStr);)

2. result = paths.ReduceFlow(CxList.ReduceFlowType.ReduceSmallFlow);



the result would consist of 4 items:
      all ending at add   (in lst.add(otherStr);)
starting     lst    (in ArrayList<String> lst = new
ArrayList<String>();)
ending add          (in lst.add(str);)

starting     lst          (in lst.add(str))
ending add          (in lst.add(otherStr);)

starting     str          (in (String str))
ending add          (in lst.add(str);)

starting     "string is empty"         (in String otherStr ="string is
empty";)
ending add                       (in lst.add(otherStr);)
```

Version Information

Supported from **CxAudit** 7.1.2

4.108 CxList.ReduceFlowByPragma Method ()

Returns a CxList which is a subset of instance CxList and consists of shortest paths from path starting line to path end line.

Syntax

```
CxQL
public CxList ReduceFlowByPragma ()
```

Parameters

Return Value

Returns a CxList which are shortest paths from path starting line to path end line.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList. ReduceFlowByPragma () method.
 The input source code is:

 public void setString (){
       String otherStr = otherStr;
       lst.add(otherStr);
}

CxList paths = All.DataInfluencedBy(All.FindByShortName("otherStr"));
result = paths.ReduceFlowByPragma();

the result would consist of 4 items:
  starts in otherStr of (String otherStr) ends in otherStr of
(lst.add(otherStr);)
  starts in otherStr of (= otherStr;)     ends in otherStr of
(String otherStr)
  starts in otherStr of (String otherStr) ends in add of
(lst.add(otherStr);)
  starts in otherStr of (lst.add(otherStr);) ends in add of
(lst.add(otherStr);)
```

Version Information

Supported from **CxAudit** 7.1.2

4.109 CxList.SanitizeCxList Method (CxList sanitizeNodes)

Returns a CxList which is a subset of paths, which are the instance CxList, that doesn't include sanitize nodes.

Syntax

```
CxQL
public CxList SanitizeCxList (CxList sanitizeNodes)
```

Parameters

SanitizeNodes

CxList of sanitizer nodes.

Return Value

Returns a CxList which is a subset of paths that doesn't include sanitize nodes.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

The return value may be empty (Count = 0).

Example

```
CxQL

 This example demonstrates the CxList. SanitizeCxList () method.
 The input source code is:

public void setString (String input){
      String otherStr = input;
      lst.add(otherStr);
}
CxList paths = All.DataInfluencingOn(All.FindByShortName("add"));
CxList sanitizeNodes = All.FindByShortName("input");
result = paths.SanitizeCxList(sanitizeNodes);

the result would consist of 3 items:
      all ends with add in lst.add(otherStr);
      starts:
      otherStr     (in String otherStr = input;)
      lst          (in lst.add(otherStr);)
      otherStr     (in lst.add(otherStr);)
```

Version Information

Supported from **CxAudit** 7.1.2

4.110 CxList.FillGraphsList Method (CxList)

Fills graphs for the list of roots given.

Syntax

```
CxQL
public void FillGraphsList (CxList graphRoots)
```

Parameters

graphRoots

List of roots to be filled with the graphs.

Return Value

None.

Exceptions

Exception type	Condition
NullReferenceException	**parameter is a null reference**

Example

```
CxQL

 This example demonstrates the CxList. FillGraphsList () method.
 With any Input source code, the method can be called after a Query.
result=All;
FillGraphsList(result);
 At this point, the result list is filled with the Graphs.
```

Version Information

Supported from **CxAudit** 7.1.2

4.111 CxList.FillGraphsList Method (CSharpGraph)

Fill graphs from one root element.

Syntax

```
CxQL
public void FillGraphsList (CSharpGraph graphRoot)
```

Parameters

graphRoot

CSharpGraph instance to be filled with Graphs.

Return Value

None.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Example

```
CxQL

 This example demonstrates the CxList. FillGraphsList () method.
 With any Input source code, the method can be called after a Query.
first=All.GetFirstGraph();
FillGraphsList(first);
 At this point, the first is filled with the Graphs.
```

4.112 CxList.GetIndexOfParameter Method ()

For a single Param or ParamDecl returns the index of the parameter 0 based.

Syntax

```
CxQL
public int GetIndexOfParameter ()
```

Parameters

Return Value

Integer containing the index of the parameter zero based, or -1 if not a parameter or list empty or contains multiple nodes. Note that the CxList must contain exactly one node, and the node must be of type Param or ParamDecl.

Exceptions

Exception type	Condition

Example

```
CxQL

 This example demonstrates the CxList.GetIndexOfParameterMethod()
method.
 It prints out to the log the index of each of the parameters.
result = All.FindByType(typeof(Param));
foreach (CxList list in result)
{
       CxDebug("Parameter index = " + list. GetIndexOfParameterMethod());
}
```

4.113 CxList.FindSQLInjections Method (CxList, CxList, CxList)

Returns flow for SQL Injection from input to db that is not sanitized

Syntax

```
CxQL
public CxList FindSQLInjections (CxList inputs, CxList db, CxList sanitize)
```

Parameters

inputs

CxList containing input elements

db

CxList containing output elements (eg. database)

sanitize

CxList containing sanitizing elements (cast to integer etc).

Return Value

CxList containing flow of SQL injection from input to output which is not flowing through a sanitizer

Exceptions

Exception type	Condition

Remarks

Actually uses inputs.InfluencingOnAndNotSanitized(db, sanitize).

Example

```
CxQL
```

4.114 CxList.FindXSS Method (CxList, CxList, CxList)

Returns flow for XSS from input to output that is not sanitized

Syntax

```
CxQL
public CxList FindXSS (CxList inputs, CxList outputs, CxList sanitize)
```

Parameters

inputs

CxList containing input elements

outputs

CxList containing output elements for xss.

sanitize

CxList containing sanitizing elements (cast to integer etc).

Return Value

CxList containing flow of XSS from input to output which is not flowing through a sanitizer

Exceptions

Exception type	Condition

Remarks

Actually uses inputs.InfluencingOnAndNotSanitized(db, sanitize).

Example

```
CxQL
```

4.115 CxList.Clone Method ()

Clones the current (this) CxList

Syntax

```
CxQL
public CxList Clone ()
```

Parameters

Return Value

CxList containing a clone of the current (this) CxList

Exceptions

Exception type	Condition

Example

```
CxQL

CxList A = All.FindByType(typeof(UnknownReference));
CxList B = A; //B points to same elements as A
CxList B = A.Clone(); //B has a copy (clone) of the elements in A
```

4.116 CxList.TryGetCSharpGraph<T> Method () where T : CSharpGraph

Try to extract the DOM object from the first node in 'this' CxList and cast it ti type 'T'. Returns null if the CxList is empty, or if the casting fails.

Syntax

```
CxQL
public CxList TryGetCSharpGraph <T>()
```

Parameters

<T> the type to cast the DOM object to (must inherit from CSharpGraph)

Return Value

The DOM object after casting

Exceptions

Exception type	Condition

Example

```
CxQL

CxList A = All.FindById(10);
CSharpGraph cs = A.TryGetCSharpGraph<CSharpGraph>();
// If A contains at least 1 node, cs will contain its DOM object
```

Version Information

Supported from **CxAudit** 8.4.0

4.117 CxList.GetQueryParam Method (string paramName)

Try to get a value for a query parameter using the key paramName. Returns an empty string if the key was not found and on errors.

Syntax

```
CxQL
public string GetQueryParam(string paramName)
```

Parameters

paramName

The parameter name (key)

Return Value

The value of the received key, or an empty string if the key was not found.

Exceptions

Exception type	Condition

Example

```
CxQL

string val = All.GetQueryParam("Param");
// If the key "Param" was found in the configuration, than val now holds
its value, otherwise, val is an empty string
if (string.IsNullOrEmpty(val))
{
      // Use val
}
```

Version Information

Supported from **CxAudit** 8.4.0

4.118 CxList.GetQueryParam<T> Method (string paramName, T defaultVal = default(T))

Try to get a value for a query parameter using the key paramName and parse the returnd string value to type T. Returns defaultVal if the key was not found and on errors.

Syntax

```
CxQL
public string GetQueryParam<T>(string paramName, T defaultVal = default(T))
```

Parameters

<T>

The type of defaultVal and the returned value

paramName

The parameter name (key)

defaultVal

The value to return on errors

Return Value

The value for the received key parset to type T, or defaultVal if the key was not found or if the value returned cannot be parsed to type T..

Exceptions

Exception type	Condition

Example

```
CxQL

int val = All.GetQueryParam<int>("Param", 0);
// If the key "Param" was found in the configuration, than val now holds
its value, otherwise, val is 0
if (val > 0)
{
	// Use val
}
```

Version Information

Supported from **CxAudit** 8.4.0

4.119 CxList.FindByFiles Method (CxList source)

Return a subset of 'this' instance, where its DOM objects are on the same file(s) as the DOM objects in the 'source' CxList.

Syntax

```
CxQL
public CxList FindByFiles(CxList source)
```

Parameters

source

A Cxlist that have DOM objects in the required files.

Return Value

Rerurns Return a subset of 'this' instance, where its DOM objects are on the same file(s) as the DOM objects in the 'source' CxList..

Exceptions

Exception type	Condition

Example

```
CxQL

CxList a = All.FindByFileName("*method.js*");
CxList b = All.FindByFileName("*method.json");
Result = a.FindByFiles(b);
// Return a subset of 'a' where the objects are of the same file as the
objects of b.
```

Version Information

Supported from **CxAudit** 8.4.0

4.120 CxList.FindRegexMatches Method (CxList comments)

Return a subset of 'this' instance which objects are of type Comment, and are equivalent to objects of type Comment in the comments CxList.

Syntax

```
CxQL
public CxList FindRegexMatches(CxList comments)
```

Parameters

A CxList of Comment type objects to find matches against the Comment objects in 'this'

Return Value

Return a subset of 'this' instance which objects are of type Comment, and are equivalent to objects of type Comment in the comments CxList..

Exceptions

Exception type	Condition

Example

```
CxQL

// Each time a FindByRegexExt or FindByRegex generate Comment type
objects, they are get a different NodeId, even if the represent the same
string in the project code.

CxList a = All.FindByRegexExt("http://");
a.Add(All.FindByRegexExt("https://"));
CxList b = All.FindByRegexExt("http://"); // The strings that starts
with http:// in 'a', now exist in 'b' but with a different NodId number.
Result = a. FindRegexMatches(b);
// Return a subset of 'b' where the ibjects returned are rquivalent to
othe objects in 'a'.
```

Version Information

Supported from **CxAudit** 8.4.0

4.121 CxList.GetAssigner Method (CxList others = null)

For each DOM object in 'this' which is on the left side of an assignment, return the right side of the assignment, which are in the others CxList.

Syntax

```
CxQL
public CxList GetAssigner(CxList others = null)
```

Parameters

others

CxList containing the right side of the assignment. If null – treat it as if it was All.

Return Value

For each DOM object in 'this' which is on the left side of an assignment, return the right side of the assignment, which are in the others CxList

Exceptions

Exception type	Condition

Example

```
CxQL

The input source code is:
 int a = 0;
 int b = a;
 b = 2;
 int c = b > 1 ? 3 : a;
CxList a = All.FindByShortName("a");
CxList b = All.FindByShortName("b");
CxList c = All.FindByShortName("c");

result = b.GetAssigner();
// result now holds 'a' in int b = a; and 2 in b = 2;
result = c.GetAssigner();
// result now holds 3 and 'a' in int c = b > 1 ? 3 : a;
result = c.GetAssigner(a);
// result now holds 'a' in int c = b > 1 ? 3 : a;
```

Version Information

Supported from **CxAudit** 8.4.0

4.122 CxList.GetAssignee Method (CxList others = null)

For each DOM object in 'this' which is on the right side of an assignment, return the left side of the assignment, which are in the others CxList.

Syntax

```
CxQL
public CxList GetAssignee(CxList others = null)
```

Parameters

others

CxList containing the left side of the assignment. If null – treat it as if it was All.

Return Value

For each DOM object in 'this' which is on the right side of an assignment, return the left side of the assignment, which are in the others CxList

Exceptions

Exception type	Condition

Example

```
CxQL

The input source code is:
 int a = 0;
 int b = a;
 b = 2;
 int c = b > 1 ? 3 : a;
CxList a = All.FindByShortName("a");
CxList b = All.FindByShortName("b");
CxList c = All.FindByShortName("c");

result = a.GetAssignee();
// result now holds 'b' in int b = a; and c in int c = b > 1 ? 3 : a;
result = a.GetAssignee(b);
// result now holds 'b' in int b = a;
```

Version Information

Supported from **CxAudit** 8.4.0

5 Method documentation (for internal use only)

5.1 CxList.SetQueryProperty Method (QueryProperties.propertyEnum, QueryProperties.flowDirectionEnum)

Adds/modifies a query property of the current query

Syntax

```
CxQL
public static void SetQueryProperty (QueryProperties.propertyEnum
queryProperty, QueryProperties.flowDirectionEnum directionValue)
```

Parameters

queryProperty

enum of query properties: `FLOW_DIRECTION`

directionValue

enum of flow direction: `From_Small_To_Large = 1, From_Large_To_Small, From_Start, From_End`

Exceptions

Exception type	Condition

Example

```
CxQL

SetQueryProperty(QueryProperties.propertyEnum.FLOW_DIRECTION,
QueryProperties.flowDirectionEnum.From_Start); //makes the calculation
of flows be from start node to end node.
```

Version Information

Supported from **CxAudit** 8.0.0

5.2 CxList.GetSanitizerByMethodInCondition Method (CxList)

For each input method, finds all the calls inside a "if" condition and returns all the references, of the methods parameters, that are inside the "if" statement.

Syntax

```
CxQL
public CxList GetSanitizerByMethodInCondition (CxList
MethodCallsInCondition)
```

Parameters

MethodCallsInCondition

method call list inside "if" condition (must be of type MethodInvoke)

Return Value

all references of a method call parameter in the scope of the if statement

Example

```
This example demonstrates the
CxList.GetSanitizerByMethodInCondition(CxList MethodCallsInCondition)
method.
 The input source code is:
      string a = getInput();
      If(Valid(a)){
            Print(a);
      }

CxList valid = All.FindByShortName("Valid");
result = All.GetSanitizerByMethodInCondition(valid);

the result would consist of 1 item:
           a       (in Print(a))
The purpose of the query is to mark 'a' as a sanitizer, because the flow
doesn't pass through the condition.
```

Remarks

Calls GetSanitizerByMethodInCondition(MethodCallsInCondition, IfBlock.Both).

Version Information

Supported from **CxAudit** 7.1.2

5.3 CxList.GetSanitizerByMethodInCondition Method (CxList, IfBlock)

For each method, finds all the calls inside a "if" condition and returns all the references, of the methods parameters, that are inside the "if" block (the IfBlock input parameter) statement.

Syntax

```
CxQL
public CxList GetSanitizerByMethodInCondition (CxList
MethodCallsInCondition, IfBlock block)
```

Parameters

MethodCallsInCondition

method call list inside "if" condition (must be of type MethodInvoke)

block

select only "true", only "false" or both scopes.

Return Value

all references of a method call parameter in the scope of the if statement

Example

```
This example demonstrates the
CxList.GetSanitizerByMethodInCondition(CxList MethodCallsInCondition,
IfBlock block) method.
 The input source code is:
      string a = getInput();
      If(Valid(a)){
            Print(a);
      }

CxList valid = All.FindByShortName("Valid");
result = All.GetSanitizerByMethodInCondition(valid,CxList.IfBlock.True);

the result would consist of 1 item:
           a       (in Print(a))
The purpose of the query is to mark 'a' as a sanitizer, because the flow
doesn't pass through the condition.
```

Version Information

Supported from **CxAudit** 7.1.2

6 CxList operators

The operators of the **CxList** class are listed here.

Public Operators

== Operator	Determines whether two specified CxList objects have the same values.
!= Operator	Determines whether two specified CxList objects do not have the same values (they differ by a least one value).
+ Operator	Merges two specified CxList objects (same as | operator)
- Operator	Removes the values of second specified CxList object from the first one.
& Operators	Intersects the values of the two specified CxList objects (same as * operator)
| Operator	Merges two specified CxList object (same as + operator)
* Operator	Intersects the values of the two specified CxList objects (same as & operator)
< Operator	Return true if the first specified CxList is completely contained within the second one.
> Operator	Return true if the second specified CxList is completely contained within the first one.
<= Operator	Return true if the first specified CxList is completely contained within the second one or they are equal.
=> Operator	Return true if the second specified CxList is completely contained within the first one or they are equal.

7 CxQuery Miscellaneous Methods

7.1 CxDebug Method (string)

Display string to DebugMessages tab in CxAudit program

Syntax

```
CxQL
public void CxDebug(string)
```

Parameters

String to be displayed.

Return Value

none.

Exceptions

Exception type	Condition
ArgumentNullException	**parameter is a null reference**

Remarks

All calling to CxDebug should be removed from production version!!!

In debug version the results will appears in log of CxAudit program too.

Example

The following code example shows how you can use the CxDebug method.

```
CxQL

 This example demonstrates the CxDebug method.
 The input source code is:

class cl1
 {
      void foo()
      {
            int a = 3;
            int b = 5;
      }
 }
result = All.FindByShortName("foo");
if (result.Count > 0)
       CxDebug(result.GetFirstGraph().ShortName);
CxDebug("number of DOM elements =" + All.Count);

 the result would be – on DebugMessage tab in CxAudit program
       foo
       number of DOM elements = 14
```

Version Information

Supported from **CxAudit** v1.8.1

8 CxDOM Types

The built-in types in CxDOM are listed here:

Types	Types (cont.)	Types(Cont.)
AccessorDecl	DestructorDecl	ParamDecl
ArgumentRef	EnumDecl	PostfixExpr
ArrayCreateExpr	EnumMemberDecl	PrimitiveExpr
ArrayElementRef	EventDecl	PropertyDecl
ArrayInitializer	EventRef	PropertyRef
AssemblyReference	Expression	PropertySetValueRef
AssignExpr	ExprStmt	RankSpecifier
AttachDelegateStmt	FieldDecl	RealLiteral
BaseRef	FieldRef	Reference
BinaryExpr	ForEachStmt	RemoveDelegateStmt
BooleanLiteral	GotoStmt	ReturnStmt
BreakStmt	IfStmt	Statement
BuiltInType	Import	StaticRef
Case	IndexerDecl	StringLiteral
CastExpr	IndexerRef	StructDecl
Catch	IntegerLiteral	SubExpr
CharLiteral	InterfaceDecl	SwitchStmt
CheckedStmt	IterationStmt	TernaryExpr
ClassDecl	LabeledStmt	ThisRef
Comment	LinePragma	ThrowStmt
CommentStmt	LocalRef	TryCatchFinallyStmt
CompileUnit	LockStmt	TypeDecl
ConstantDecl	MemberAccess	TypeOfExpr
ConstantDeclStmt	MemberDecl	TypeRef
ConstructorDecl	MethodDecl	UnaryExpr
ContinueStmt	MethodInvokeExpr	UncheckedStmt
CreateDelegateExpr	MethodRef	UnknownReference
CSharpGraph	NamespaceDecl	UsingStmt

Types	Types (cont.)	Types(Cont.)
CustomAttribute	NullLiteral	VariableDecl
Declarator	ObjectCreateExpr	VariableDeclStmt
DelegateDecl	OperatorDecl	
DelegateInvokeExpr	Param	

Example

In order to better understand each of these types, try the following query:

```
CxQL

result = All.FindByType(typeof(IfStmt));
Change "IfStmt" to one of the above types.
```

9 CSharpGraph methods

In the previous sections, we explained how to manipulate CxList, which are basically collections of basic code elements. Those basic elements are represented as instances of the CSharpGraph class or a class which inherits from CSharpGraph.

In some situations, it may be useful to manipulate directely basic code elements. It will give you the ability to implement home-made fine-grained queries in an optimized way.

This section describes the inheritance hierarchy of the CSharpGraph and its decendants.

Own methods:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.1 MemberDecl : CSharpGraph

Own methods:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.1 TypeDecl : MemberDecl

Own methods:

- public abstract int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public Checkmarx.Dom.TypeRefCollection BaseTypes { set; get; }
- public SymbolTable.IDefinition Definition { get; }
- public Checkmarx.Dom.TypeRefCollection Generics { set; get; }
- public bool HasGenerics { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public Checkmarx.Dom.MemberDeclCollection Members { get; }
- public string Name { set; get; }
- public override string Text { get; }
- public Checkmarx.Dom.TypeModifiers TypeAttributes { set; get; }
- public abstract Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId

- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.1.1 ClassDecl : TypeDecl

Own methods:

- public override int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public Checkmarx.Dom.ClassDecl BaseClass { get; }
- public SymbolTable.OverloadableDefinition Constructors { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public new SymbolTable.Scope Scope { set; get; }
- public override Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from TypeDecl:

- public abstract int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public Checkmarx.Dom.TypeRefCollection BaseTypes { set; get; }
- public SymbolTable.IDefinition Definition { get; }
- public Checkmarx.Dom.TypeRefCollection Generics { set; get; }
- public bool HasGenerics { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public Checkmarx.Dom.MemberDeclCollection Members { get; }
- public string Name { set; get; }
- public override string Text { get; }
- public Checkmarx.Dom.TypeModifiers TypeAttributes { set; get; }
- public abstract Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }

- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.1.2 DelegateDecl : TypeDecl

Own methods:

- public override int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.ParamDeclCollection Parameters { get; }
- public Checkmarx.Dom.TypeRef ReturnType { set; get; }
- public override Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from TypeDecl:

- public abstract int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public Checkmarx.Dom.TypeRefCollection BaseTypes { set; get; }
- public SymbolTable.IDefinition Definition { get; }
- public Checkmarx.Dom.TypeRefCollection Generics { set; get; }
- public bool HasGenerics { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public Checkmarx.Dom.MemberDeclCollection Members { get; }
- public string Name { set; get; }
- public override string Text { get; }
- public Checkmarx.Dom.TypeModifiers TypeAttributes { set; get; }
- public abstract Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.1.3 EnumDecl : TypeDecl

Own methods:

- public override int **DistanceToType**(SymbolTable.IDefinition baseType, int distance)
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public new SymbolTable.Scope **Scope** { set; get; }
- public override Checkmarx.Dom.TypeKind **TypeKind** { get; }

Methods from TypeDecl:

- public abstract int **DistanceToType**(SymbolTable.IDefinition baseType, int distance)
- public Checkmarx.Dom.TypeRefCollection **BaseTypes** { set; get; }
- public SymbolTable.IDefinition **Definition** { get; }
- public Checkmarx.Dom.TypeRefCollection **Generics** { set; get; }
- public bool **HasGenerics** { get; }

- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public Checkmarx.Dom.MemberDeclCollection Members { get; }
- public string Name { set; get; }
- public override string Text { get; }
- public Checkmarx.Dom.TypeModifiers TypeAttributes { set; get; }
- public abstract Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.1.4 InterfaceDecl : TypeDecl

Own methods:

- public override int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public new SymbolTable.Scope Scope { set; get; }
- public override Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from TypeDecl:

- public abstract int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public Checkmarx.Dom.TypeRefCollection BaseTypes { set; get; }
- public SymbolTable.IDefinition Definition { get; }
- public Checkmarx.Dom.TypeRefCollection Generics { set; get; }
- public bool HasGenerics { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public Checkmarx.Dom.MemberDeclCollection Members { get; }
- public string Name { set; get; }
- public override string Text { get; }
- public Checkmarx.Dom.TypeModifiers TypeAttributes { set; get; }
- public abstract Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }

- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.1.5 StructDecl : TypeDecl

Own methods:

- public override int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public SymbolTable.OverloadableDefinition Constructors { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public new SymbolTable.Scope Scope { set; get; }
- public override Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from TypeDecl:

- public abstract int DistanceToType(SymbolTable.IDefinition baseType, int distance)
- public Checkmarx.Dom.TypeRefCollection BaseTypes { set; get; }
- public SymbolTable.IDefinition Definition { get; }
- public Checkmarx.Dom.TypeRefCollection Generics { set; get; }
- public bool HasGenerics { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public Checkmarx.Dom.MemberDeclCollection Members { get; }
- public string Name { set; get; }
- public override string Text { get; }
- public Checkmarx.Dom.TypeModifiers TypeAttributes { set; get; }
- public abstract Checkmarx.Dom.TypeKind TypeKind { get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)

- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.2 AccessorDecl : MemberDecl

Own methods:

- public Checkmarx.Dom.AccessorModifiers AccessorModifier { set; get; }
- public virtual SymbolTable.IDefinition Definition { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public new SymbolTable.Scope Scope { set; get; }
- public Checkmarx.Dom.StatementCollection Statements { get; }
- public SymbolTable.Definition ValueDefinition { set; get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.3 ConstantDecl : MemberDecl

Own methods:

- public Checkmarx.Dom.DeclaratorCollection Declarators { get; }
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }
- public Checkmarx.Dom.TypeRef Type { set; get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.4 ConstructorDecl : MemberDecl

Own methods:

- public SymbolTable.Definition **BaseDefinition** { set; get; }
- public Checkmarx.Dom.ParamCollection **BaseParameters** { get; }
- public SymbolTable.Definition **ChainDefinition** { set; get; }
- public Checkmarx.Dom.ParamCollection **ChainParameters** { get; }
- public SymbolTable.IDefinition **Definition** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public string **HashText** { get; }
- public bool **InvokeBase** { set; get; }
- public bool **InvokeChain** { set; get; }
- public override Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public string **Name** { set; get; }
- public Checkmarx.Dom.ParamDeclCollection **Parameters** { get; }
- public new SymbolTable.Scope **Scope** { set; get; }
- public Checkmarx.Dom.StatementCollection **Statements** { get; }

- public override string Text { get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.5 DestructorDecl : MemberDecl

Own methods:

- public SymbolTable.IDefinition Definition { get; }

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public string Name { set; get; }
- public new SymbolTable.Scope Scope { set; get; }
- public Checkmarx.Dom.StatementCollection Statements { get; }
- public override string Text { get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.6 EnumMemberDecl : MemberDecl

Own methods:

- public SymbolTable.IDefinition **Definition** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public override Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public string **Name** { set; get; }
- public override string **Text** { get; }
- public Checkmarx.Dom.Expression **Value** { set; get; }

Methods from MemberDecl:

- public bool **IsCommentsExists**()
- public bool **IsCustomAttributesExists**()
- public Checkmarx.Dom.Modifiers **Attributes** { set; get; }
- public Checkmarx.Dom.CommentStmtCollection **Comments** { get; }
- public Checkmarx.Dom.CustomAttributeCollection **CustomAttributes** { get; }
- public abstract Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId

- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.7 EventDecl : MemberDecl

Own methods:

- public Checkmarx.Dom.AccessorDecl **AddAccessor** { set; get; }
- public Checkmarx.Dom.DeclaratorCollection **Declarators** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public override Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public Checkmarx.Dom.AccessorDecl **RemoveAccessor** { set; get; }
- public override string **Text** { get; }
- public Checkmarx.Dom.TypeRef **Type** { set; get; }
- public bool **UsesAccessors** { set; get; }

Methods from MemberDecl:

- public bool **IsCommentsExists**()
- public bool **IsCustomAttributesExists**()
- public Checkmarx.Dom.Modifiers **Attributes** { set; get; }
- public Checkmarx.Dom.CommentStmtCollection **Comments** { get; }
- public Checkmarx.Dom.CustomAttributeCollection **CustomAttributes** { get; }
- public abstract Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }

- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.8 FieldDecl : MemberDecl

Own methods:

- public Checkmarx.Dom.DeclaratorCollection **Declarators** { get; }
- public override string **FullName** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public override Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public override string **Text** { get; }
- public Checkmarx.Dom.TypeRef **Type** { set; get; }

Methods from MemberDecl:

- public bool **IsCommentsExists**()
- public bool **IsCustomAttributesExists**()
- public Checkmarx.Dom.Modifiers **Attributes** { set; get; }
- public Checkmarx.Dom.CommentStmtCollection **Comments** { get; }
- public Checkmarx.Dom.CustomAttributeCollection **CustomAttributes** { get; }
- public abstract Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }

- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.9 IndexerDecl : MemberDecl

Own methods:

- public SymbolTable.IDefinition Definition { get; }
- public Checkmarx.Dom.AccessorDecl GetAccessor { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public bool HasGet { set; get; }
- public string HashText { get; }
- public bool HasSet { set; get; }
- public Checkmarx.Dom.TypeRef InterfaceType { set; get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public string Name { set; get; }
- public Checkmarx.Dom.ParamDeclCollection Parameters { get; }
- public new SymbolTable.Scope Scope { set; get; }
- public Checkmarx.Dom.AccessorDecl SetAccessor { set; get; }
- public Checkmarx.Dom.TypeRef Type { set; get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }

- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.10 MethodDecl : MemberDecl

Own methods:

- public bool IsReturnTypeCustomAttributesExists()
- public SymbolTable.IDefinition Definition { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public string HashText { get; }
- public bool HasTypeParameters { get; }
- public override Checkmarx.Dom.MemberKind MemberKind { get; }
- public string Name { set; get; }
- public Checkmarx.Dom.ParamDeclCollection Parameters { get; }
- public Checkmarx.Dom.TypeRef ReturnType { set; get; }
- public Checkmarx.Dom.CustomAttributeCollection ReturnTypeCustomAttributes { get; }
- public new SymbolTable.Scope Scope { set; get; }
- public Checkmarx.Dom.StatementCollection Statements { get; }
- public override string Text { get; }
- public Checkmarx.Dom.TypeRefCollection TypeParameters { set; get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.11 OperatorDecl : MemberDecl

Own methods:

- public static Checkmarx.Dom.OverloadableOperator **OperatorFromString**(string sop)
- public static string **StringFromOperator**(Checkmarx.Dom.OverloadableOperator sop)
- public string **ConversionName** { set; get; }
- public SymbolTable.IDefinition **Definition** { get; }
- public Checkmarx.Dom.ParamDecl **FirstParameter** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public string **HashText** { get; }
- public override Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public string **Name** { get; }
- public Checkmarx.Dom.OverloadableOperator **Operator** { set; get; }
- public Checkmarx.Dom.ParamDeclCollection **Parameters** { get; }
- public new SymbolTable.Scope **Scope** { set; get; }
- public Checkmarx.Dom.ParamDecl **SecondParameter** { set; get; }
- public Checkmarx.Dom.StatementCollection **Statements** { get; }

- public override string Text { get; }
- public Checkmarx.Dom.TypeRef Type { set; get; }

Methods from MemberDecl:

- public bool IsCommentsExists()
- public bool IsCustomAttributesExists()
- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.CommentStmtCollection Comments { get; }
- public Checkmarx.Dom.CustomAttributeCollection CustomAttributes { get; }
- public abstract Checkmarx.Dom.MemberKind MemberKind { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.1.12 PropertyDecl : MemberDecl

Own methods:

- public SymbolTable.IDefinition **Definition** { get; }
- public Checkmarx.Dom.AccessorDecl **GetAccessor** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public bool **HasGet** { set; get; }
- public bool **HasSet** { set; get; }
- public override Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public string **Name** { set; get; }
- public Checkmarx.Dom.AccessorDecl **SetAccessor** { set; get; }
- public override string **Text** { get; }
- public Checkmarx.Dom.TypeRef **Type** { set; get; }

Methods from MemberDecl:

- public bool **IsCommentsExists**()
- public bool **IsCustomAttributesExists**()
- public Checkmarx.Dom.Modifiers **Attributes** { set; get; }
- public Checkmarx.Dom.CommentStmtCollection **Comments** { get; }
- public Checkmarx.Dom.CustomAttributeCollection **CustomAttributes** { get; }
- public abstract Checkmarx.Dom.MemberKind **MemberKind** { get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father

- `public int p_no`

9.2 Expression : CSharpGraph

Own methods:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1 Reference : Expression

Own methods:

- public Checkmarx.Dom.IDeclaration **DeclaredType** { set; get; }
- public virtual SymbolTable.IDefinition **Definition** { set; get; }

- public override string **Text** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.1 ArgumentRef : Reference

Own methods:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public string **ParameterName** { set; get; }
- public override string **Text** { get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration DeclaredType { set; get; }
- public virtual SymbolTable.IDefinition Definition { set; get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.2 ArrayElementRef : Reference

Own methods:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.ExpressionCollection **Indices** { get; }
- public Checkmarx.Dom.Expression **Target** { set; get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration **DeclaredType** { set; get; }
- public virtual SymbolTable.IDefinition **Definition** { set; get; }
- public override string **Text** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.3 BaseRef : Reference

Own methods:

- public override SymbolTable.IDefinition **Definition** { get; }
- public override string **FullName** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public override string **Text** { get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration **DeclaredType** { set; get; }
- public virtual SymbolTable.IDefinition **Definition** { set; get; }
- public override string **Text** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father

- public int p_no

9.2.1.4 BuiltInType : Reference

Own methods:

- public static bool isBuiltinType(string fullName)
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }
- public override string TypeName { set; get; }
- public static int falseCount
- public static int trueCount

Methods from Reference:

- public Checkmarx.Dom.IDeclaration DeclaredType { set; get; }
- public virtual SymbolTable.IDefinition Definition { set; get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }

- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.5 EventRef : Reference

Own methods:

- public string **EventName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Target** { set; get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration **DeclaredType** { set; get; }
- public virtual SymbolTable.IDefinition **Definition** { set; get; }
- public override string **Text** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }

- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.6 FieldRef : Reference

Own methods:

- public string FieldName { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Target { set; get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration DeclaredType { set; get; }
- public virtual SymbolTable.IDefinition Definition { set; get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }

- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.7 IndexerRef : Reference

Own methods:

- public override SymbolTable.IDefinition Definition { get; }
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.ExpressionCollection Indices { get; }
- public Checkmarx.Dom.Expression Target { set; get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration DeclaredType { set; get; }
- public virtual SymbolTable.IDefinition Definition { set; get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }

- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.8 LocalRef : Reference

Own methods:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public override string **Text** { get; }
- public string **VariableName** { set; get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration **DeclaredType** { set; get; }
- public virtual SymbolTable.IDefinition **Definition** { set; get; }
- public override string **Text** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)

- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.9 MemberAccess : Reference

Own methods:

- public Checkmarx.Dom.CSharpGraph Declarator { get; }
- public Checkmarx.Dom.TypeRef DeclaratorType { get; }
- public string Def { get; }
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public string MemberName { set; get; }
- public Checkmarx.Dom.Expression Target { set; get; }
- public override string Text { get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration DeclaredType { set; get; }
- public virtual SymbolTable.IDefinition Definition { set; get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.10 MethodRef : Reference

Own methods:

- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public string **MethodName** { set; get; }
- public Checkmarx.Dom.Expression **Target** { set; get; }
- public override string **Text** { get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration **DeclaredType** { set; get; }
- public virtual SymbolTable.IDefinition **Definition** { set; get; }
- public override string **Text** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.11 PropertyRef : Reference

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public string PropertyName { set; get; }
- public Checkmarx.Dom.Expression Target { set; get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration DeclaredType { set; get; }
- public virtual SymbolTable.IDefinition Definition { set; get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.12 PropertySetValueRef : Reference

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration DeclaredType { set; get; }

- public virtual SymbolTable.IDefinition **Definition** { set; get; }
- public override string **Text** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.13 ThisRef : Reference

Own methods:

- public override SymbolTable.IDefinition **Definition** { get; }
- public override string **FullName** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public override string **Text** { get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration DeclaredType { set; get; }
- public virtual SymbolTable.IDefinition Definition { set; get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.1.14 UnknownReference : Reference

Own methods:

- public Checkmarx.Dom.CSharpGraph **Declarator** { get; }
- public Checkmarx.Dom.TypeRef **DeclaratorType** { get; }
- public override string **FullName** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public override string **Text** { get; }
- public string **VariableName** { set; get; }

Methods from Reference:

- public Checkmarx.Dom.IDeclaration **DeclaredType** { set; get; }
- public virtual SymbolTable.IDefinition **Definition** { set; get; }
- public override string **Text** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father

- public int p_no

9.2.2 PrimitiveExpr : Expression

Own methods:

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.2.1 BooleanLiteral : PrimitiveExpr

Own methods:

- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }
- public bool Value { set; get; }

Methods from PrimitiveExpr:

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.2.2 CharLiteral : PrimitiveExpr

Own methods:

- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }
- public char Value { set; get; }

Methods from PrimitiveExpr:

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.2.3 IntegerLiteral : PrimitiveExpr

Own methods:

- public string ExtendedType { set; get; }
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }
- public long Value { set; get; }

Methods from PrimitiveExpr:

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.2.4 NullLiteral : PrimitiveExpr

Own methods:

- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }
- public string Value { set; get; }

Methods from PrimitiveExpr:

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.2.5 RealLiteral : PrimitiveExpr

Own methods:

- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }
- public double Value { set; get; }

Methods from PrimitiveExpr:

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.2.6 StringLiteral : PrimitiveExpr

Own methods:

- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }
- public string Value { set; get; }

Methods from PrimitiveExpr:

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.3 ArrayCreateExpr : Expression

Own methods:

- public Checkmarx.Dom.TypeRef **CreateType** { set; get; }
- public int **DimensionCount** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.ArrayInitializer **Initializer** { set; get; }
- public Checkmarx.Dom.RankSpecifierCollection **RankSpecifiers** { get; }
- public Checkmarx.Dom.ExpressionCollection **Sizes** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.4 ArrayInitializer : Expression

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.ExpressionCollection InitialValues { get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.5 AssignExpr : Expression

Own methods:

- public static Checkmarx.Dom.AssignOperator OperatorFromString(string op)
- public static string StringFromOperator(Checkmarx.Dom.AssignOperator op)

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Left { set; get; }
- public Checkmarx.Dom.AssignOperator Operator { set; get; }
- public Checkmarx.Dom.Expression Right { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.6 BinaryExpr : Expression

Own methods:

- public static Checkmarx.Dom.BinaryOperator OperatorFromString(string txt)

- public static string StringFromOperator(Checkmarx.Dom.BinaryOperator bopEnum)
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Left { set; get; }
- public Checkmarx.Dom.BinaryOperator Operator { set; get; }
- public Checkmarx.Dom.Expression Right { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.7 CastExpr : Expression

Own methods:

- public Checkmarx.Dom.Expression **Expression** { set; get; }
- public override string **FullName** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public new Checkmarx.Dom.IDeclaration **ResultType** { get; }
- public Checkmarx.Dom.TypeRef **TargetType** { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.8 CreateDelegateExpr : Expression

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public string MethodName { set; get; }
- public Checkmarx.Dom.Expression Target { set; get; }
- public Checkmarx.Dom.TypeRef Type { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.9 DelegateInvokeExpr : Expression

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }

- public Checkmarx.Dom.ParamCollection **Parameters** { get; }
- public Checkmarx.Dom.Expression **Target** { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.10 MethodInvokeExpr : Expression

Own methods:

- public void **computeDefinition**()
- public SymbolTable.IDefinition **Definition** { get; }
- public override string **FullName** { set; get; }

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.ParamCollection Parameters { get; }
- public Checkmarx.Dom.MethodRef Target { set; get; }
- public override string Text { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.11 ObjectCreateExpr : Expression

Own methods:

- public Checkmarx.Dom.MemberDeclCollection AnonymousBody { set; get; }

- public SymbolTable.IDefinition **ConstructorDefinition** { set; get; }
- public Checkmarx.Dom.TypeRef **CreateType** { set; get; }
- public override string **FullName** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.ParamCollection **Parameters** { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.12 PostfixExpr : Expression

Own methods:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Left** { set; get; }
- public Checkmarx.Dom.PostfixOperator **Operator** { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.13 SubExpr : Expression

Own methods:

- public Checkmarx.Dom.Expression **Expression** { set; get; }
- public override string **FullName** { get; }

- public override Checkmarx.Dom.GraphTypes GraphType { get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.14 TernaryExpr : Expression

Own methods:

- public Checkmarx.Dom.Expression False { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Test { set; get; }
- public Checkmarx.Dom.Expression True { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Member** { set; get; }
- public SymbolTable.IDefinition **ResultType** { set; get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.15 TypeOfExpr : Expression

Own methods:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.TypeRef **Type** { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.2.16 UnaryExpr : Expression

Own methods:

- public static Checkmarx.Dom.UnaryOperator OperatorFromString(string txt)
- public static string StringFromOperator(Checkmarx.Dom.UnaryOperator uOpEnum)
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.UnaryOperator Operator { set; get; }
- public Checkmarx.Dom.Expression Right { set; get; }

Methods from Expression:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression Member { set; get; }
- public SymbolTable.IDefinition ResultType { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3 Statement : CSharpGraph

Own methods:

- `public override string Text { get; }`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.3.1 AttachDelegateStmt : Statement

Own methods:

- `public Checkmarx.Dom.TypeRef Event { set; get; }`
- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public Checkmarx.Dom.Expression Listener { set; get; }`

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.2 BreakStmt : Statement

Own methods:

- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)

- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.3 CheckedStmt : Statement

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.StatementCollection Statements { get; }

Methods from Statement:

- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)

- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.4 CommentStmt : Statement

Own methods:

- public Checkmarx.Dom.Comment Comment { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }

Methods from Statement:

- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }

- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.5 ConstantDeclStmt : Statement

Own methods:

- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.DeclaratorCollection Declarators { get; }
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.TypeRef Type { set; get; }

Methods from Statement:

- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }

- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.6 ContinueStmt : Statement

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }

Methods from Statement:

- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }

- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.7 ExprStmt : Statement

Own methods:

- public Checkmarx.Dom.Expression **Expression** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved

- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.8 ForEachStmt : Statement

Own methods:

- public Checkmarx.Dom.Expression **Collection** { set; get; }
- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.IterationType **IterationType** { get; }
- public string **Name** { set; get; }
- public Checkmarx.Dom.StatementCollection **Statements** { get; }
- public Checkmarx.Dom.TypeRef **Type** { set; get; }
- public Checkmarx.Dom.VariableDeclStmt **VariableDeclarator** { set; get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved

- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.9 GotoStmt : Statement

Own methods:

- public Checkmarx.Dom.Expression **CaseLabel** { set; get; }
- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public string **Label** { set; get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.10 IfStmt : Statement

Own methods:

- public Checkmarx.Dom.Expression **Condition** { set; get; }
- public Checkmarx.Dom.StatementCollection **FalseStatements** { get; }
- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.StatementCollection **TrueStatements** { get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.11 IterationStmt : Statement

Own methods:

- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.StatementCollection **Increment** { get; }
- public Checkmarx.Dom.StatementCollection **Init** { get; }
- public Checkmarx.Dom.IterationType **IterationType** { set; get; }
- public Checkmarx.Dom.StatementCollection **Statements** { get; }
- public Checkmarx.Dom.Expression **Test** { set; get; }
- public bool **TestFirst** { get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.12 LabeledStmt : Statement

Own methods:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public string **Label** { set; get; }
- public Checkmarx.Dom.Statement **Statement** { set; get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.13 LockStmt : Statement

Own methods:

- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.StatementCollection **Statements** { get; }
- public Checkmarx.Dom.Expression **Target** { set; get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.14 RemoveDelegateStmt : Statement

Own methods:

- public Checkmarx.Dom.TypeRef **Event** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.Expression **Listener** { set; get; }

Methods from Statement:

- `public override string Text { get; }`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.3.15 ReturnStmt : Statement

Own methods:

- `public Checkmarx.Dom.Expression Expression { set; get; }`
- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`

Methods from Statement:

- `public override string Text { get; }`

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.16 SwitchStmt : Statement

Own methods:

- public Checkmarx.Dom.CaseCollection Cases { get; }
- public Checkmarx.Dom.Expression Condition { set; get; }
- public override string FullName { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }

Methods from Statement:

- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)

- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.17 ThrowStmt : Statement

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.Expression ToThrow { set; get; }

Methods from Statement:

- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }

- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.18 TryCatchFinallyStmt : Statement

Own methods:

- public Checkmarx.Dom.CatchCollection **CatchClauses** { set; get; }
- public Checkmarx.Dom.StatementCollection **Finally** { get; }
- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.StatementCollection **Try** { get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }

- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.19 UncheckedStmt : Statement

Own methods:

- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.StatementCollection Statements { get; }

Methods from Statement:

- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }

- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.20 UsingStmt : Statement

Own methods:

- public Checkmarx.Dom.VariableDeclStmt **Declaration** { set; get; }
- public bool **DeclaresResource** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.StatementCollection **Statements** { get; }
- public Checkmarx.Dom.Expression **Target** { set; get; }

Methods from Statement:

- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }

- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.3.21 VariableDeclStmt : Statement

Own methods:

- public Checkmarx.Dom.Modifiers Attributes { set; get; }
- public Checkmarx.Dom.DeclaratorCollection Declarators { get; }
- public int DimensionCount { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public Checkmarx.Dom.TypeRef Type { set; get; }

Methods from Statement:

- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }

- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.4 AssemblyReference : CSharpGraph

Own methods:

- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public string HintPath { set; get; }`
- `public string Name { set; get; }`
- `public override string Text { get; }`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.5 Case : CSharpGraph

Own methods:

- `public Checkmarx.Dom.Expression Condition { set; get; }`
- `public override string FullName { set; get; }`
- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public bool IsDefault { set; get; }`
- `public Checkmarx.Dom.StatementCollection Statements { get; }`
- `public override string Text { get; }`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.6 Catch : CSharpGraph

Own methods:

- public Checkmarx.Dom.TypeRef **CatchExceptionType** { set; get; }
- public SymbolTable.IDefinition **Definition** { get; }
- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public string **LocalName** { set; get; }
- public Checkmarx.Dom.StatementCollection **Statements** { get; }
- public override string **Text** { get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.7 Comment : CSharpGraph

Own methods:

- public string CommentText { set; get; }
- public bool DocComment { set; get; }
- public override string FullName { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.8 CompileUnit : CSharpGraph

Own methods:

- `public CompileUnit()`
- `public bool IsCustomAttributesExists()`
- `public void resetLanguage()`
- `public void resetLanguage(bool correctNSLanguage)`
- `public Checkmarx.Dom.CustomAttributeCollection AssemblyCustomAttributes { get; }`
- `public System.Collections.ArrayList FileName { get; }`
- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public Checkmarx.Dom.ImportCollection Imports { get; }`
- `public System.DateTime Modified { set; get; }`
- `public System.Collections.ArrayList Name { get; }`
- `public Checkmarx.Dom.NamespaceDeclCollection Namespaces { get; }`
- `public Checkmarx.Dom.AssemblyReferenceCollection ReferencedAssemblies { get; }`
- `public new SymbolTable.Scope Scope { set; get; }`
- `public SymbolTable.ScopeCollection ScopeTable { get; }`
- `public override string Text { get; }`
- `public bool IsAttributed`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`

- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.9 CustomAttribute : CSharpGraph

Own methods:

- `public static string StringFromAttributeTarget(Checkmarx.Dom.AttributeTarget at)`
- `public Checkmarx.Dom.AttributeTarget AttributeTarget { set; get; }`
- `public override string FullName { get; }`
- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public string Name { set; get; }`
- `public Checkmarx.Dom.ExpressionCollection Parameters { get; }`
- `public override string Text { get; }`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.10 Declarator : CSharpGraph

Own methods:

- `public SymbolTable.IDefinition Definition { get; }`
- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public Checkmarx.Dom.Expression InitExpression { set; get; }`
- `public string Name { set; get; }`
- `public override string Text { get; }`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.11 Import : CSharpGraph

Own methods:

- `public string Alias { set; get; }`
- `public override string FullName { set; get; }`
- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public new string Namespace { set; get; }`
- `public override string Text { get; }`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

9.12 NamespaceDecl : CSharpGraph

Own methods:

- public SymbolTable.IDefinition **Definition** { get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public Checkmarx.Dom.ImportCollection **Imports** { get; }
- public string **Name** { set; get; }
- public Checkmarx.Dom.NamespaceDeclCollection **Namespaces** { get; }
- public new SymbolTable.Scope **Scope** { set; get; }
- public override string **Text** { get; }
- public Checkmarx.Dom.TypeDeclCollection **Types** { set; get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigV alue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.13 Param : CSharpGraph

Own methods:

- public Checkmarx.Dom.ParamDirection **Direction** { set; get; }
- public override string **FullName** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public string **Name** { set; get; }
- public override string **Text** { get; }
- public Checkmarx.Dom.Expression **Value** { set; get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.14 ParamDecl : CSharpGraph

Own methods:

- public bool **IsCustomAttributesExists**()
- public Checkmarx.Dom.Modifiers **Attributes** { set; get; }
- public Checkmarx.Dom.CustomAttributeCollection **CustomAttributes** { set; get; }
- public SymbolTable.IDefinition **Definition** { get; }
- public Checkmarx.Dom.ParamDirection **Direction** { set; get; }
- public override Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public bool **IsOptional** { set; get; }
- public bool **IsParams** { set; get; }
- public string **Name** { set; get; }
- public override string **Text** { get; }
- public Checkmarx.Dom.TypeRef **Type** { set; get; }

Methods from CSharpGraph:

- public void **AddUserData**(string key, object data)
- public void **AddUserData**(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph **GetAncOfType**(System.Type objType)
- public static System.Type **GetTypeFromString**(string s)
- public object **GetUserData**(string key)
- public bool **IsAncOf**(Checkmarx.Dom.CSharpGraph child)
- public virtual bool **isSubTypeOf**(string subTypeName)
- public int **BlockId** { set; get; }
- public System.Collections.Generic.HashSet<int> **BlockIdArr** { get; }
- public Checkmarx.Dom.TypeDecl **Class** { get; }
- public int **DOMDepth** { get; }
- public virtual Checkmarx.Dom.IGraph **_father** { set; get; }
- public string **Father** { get; }
- public virtual string **FullName** { set; get; }
- public virtual Checkmarx.Dom.GraphTypes **GraphType** { get; }
- public int **language** { set; get; }
- public Checkmarx.Dom.LinePragma **LinePragma** { set; get; }
- public Checkmarx.Dom.MemberDecl **Method** { get; }
- public Checkmarx.Dom.NamespaceDecl **Namespace** { get; }
- public virtual int **_no** { set; get; }
- public string **Scope** { get; }
- public virtual string **ShortName** { get; }
- public abstract string **Text** { get; }
- public string **TokenPos** { get; }
- public virtual string **TypeName** { set; get; }
- public System.Collections.IDictionary **UserData** { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.15 Project : CSharpGraph

Own methods:

- public Project()
- public Checkmarx.Dom.CompileUnitCollection CompileUnits { get; }
- public string FileName { set; get; }
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public string Name { set; get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.16 RankSpecifier : CSharpGraph

Own methods:

- public int Dimensions { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.17 Solution : CSharpGraph

Own methods:

- public int getNumberOfMethods()
- public Checkmarx.Dom.Solution Merge(Checkmarx.Dom.Solution s)
- public Solution()
- public static Checkmarx.Dom.Solution operator +(Checkmarx.Dom.Solution s1, Checkmarx.Dom.Solution s2)
- public string FileName { set; get; }
- public override string FullName { get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public string Name { set; get; }
- public Checkmarx.Dom.ProjectsCollection Projects { get; }
- public override string Text { get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.18 TypeRef : CSharpGraph

Own methods:

- public bool IsRankSpecifierCollectionExists()
- public Checkmarx.Dom.TypeRef ArrayElementType { set; get; }
- public Checkmarx.Dom.RankSpecifierCollection ArrayRanks { set; get; }
- public SymbolTable.IDefinition Definition { set; get; }
- public string ExtendedTypeName { set; get; }
- public Checkmarx.Dom.TypeRefCollection Generics { set; get; }
- public override Checkmarx.Dom.GraphTypes GraphType { get; }
- public bool HasGenerics { get; }
- public override string Text { get; }
- public override string TypeName { set; get; }

Methods from CSharpGraph:

- public void AddUserData(string key, object data)
- public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)
- public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)
- public static System.Type GetTypeFromString(string s)
- public object GetUserData(string key)
- public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)
- public virtual bool isSubTypeOf(string subTypeName)
- public int BlockId { set; get; }
- public System.Collections.Generic.HashSet<int> BlockIdArr { get; }
- public Checkmarx.Dom.TypeDecl Class { get; }
- public int DOMDepth { get; }
- public virtual Checkmarx.Dom.IGraph _father { set; get; }
- public string Father { get; }
- public virtual string FullName { set; get; }
- public virtual Checkmarx.Dom.GraphTypes GraphType { get; }
- public int language { set; get; }
- public Checkmarx.Dom.LinePragma LinePragma { set; get; }
- public Checkmarx.Dom.MemberDecl Method { get; }
- public Checkmarx.Dom.NamespaceDecl Namespace { get; }
- public virtual int _no { set; get; }
- public string Scope { get; }
- public virtual string ShortName { get; }
- public abstract string Text { get; }
- public string TokenPos { get; }
- public virtual string TypeName { set; get; }
- public System.Collections.IDictionary UserData { set; get; }
- public System.Collections.Generic.List Children
- public static bool FullnameResolved
- public static int nodeId
- public int NodeId
- public Checkmarx.Dom.IGraph p_father
- public int p_no

9.19 VariableDecl : CSharpGraph

Own methods:

- `public override Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public override string Text { get; }`

Methods from CSharpGraph:

- `public void AddUserData(string key, object data)`
- `public void AddUserData(string key, object data, string boolConfigKey,bool defaultConfigValue)`
- `public Checkmarx.Dom.CSharpGraph GetAncOfType(System.Type objType)`
- `public static System.Type GetTypeFromString(string s)`
- `public object GetUserData(string key)`
- `public bool IsAncOf(Checkmarx.Dom.CSharpGraph child)`
- `public virtual bool isSubTypeOf(string subTypeName)`
- `public int BlockId { set; get; }`
- `public System.Collections.Generic.HashSet<int> BlockIdArr { get; }`
- `public Checkmarx.Dom.TypeDecl Class { get; }`
- `public int DOMDepth { get; }`
- `public virtual Checkmarx.Dom.IGraph _father { set; get; }`
- `public string Father { get; }`
- `public virtual string FullName { set; get; }`
- `public virtual Checkmarx.Dom.GraphTypes GraphType { get; }`
- `public int language { set; get; }`
- `public Checkmarx.Dom.LinePragma LinePragma { set; get; }`
- `public Checkmarx.Dom.MemberDecl Method { get; }`
- `public Checkmarx.Dom.NamespaceDecl Namespace { get; }`
- `public virtual int _no { set; get; }`
- `public string Scope { get; }`
- `public virtual string ShortName { get; }`
- `public abstract string Text { get; }`
- `public string TokenPos { get; }`
- `public virtual string TypeName { set; get; }`
- `public System.Collections.IDictionary UserData { set; get; }`
- `public System.Collections.Generic.List Children`
- `public static bool FullnameResolved`
- `public static int nodeId`
- `public int NodeId`
- `public Checkmarx.Dom.IGraph p_father`
- `public int p_no`

10CSharpGraph Examples

This section some common usages of the CSharpGraph API.

Precise Name Filtering

The methods CxList.FindByName and CxList.FindByShortName can be used for name filtering. However, if you need more flexibility and want to express a more precise filter, you can do this by looking at each code element, using the CSharpGraph API.

```
CxQL

 This example demonstrates the use of CSharpGraph.ShortName.
 The input source code is:

 public class sumClass{
       public int sumAll (int parameter, int parameterparameter){
              int hello = 0, hellohello = 1;
              int hellohellohello = 2, hellohellohellohello = 3;
              return parameter + hello + hellohellohellohello
                     + hellohellohello + hellohello + parameterparameter;
       }
 }

CxList vars = All.FindByType(typeof(UnknownReference));
foreach(CxList v in vars){
       // Get the CSharpGraph of the only code element in v
       CSharpGraph g = v.data.GetByIndex(0) as CSharpGraph;
       // Check a condition on the ShortName
       if(g != null && g.ShortName != null && g.ShortName.Length < 13){
              // Add only elements which meet the condition
              result.Add(v);
       }
}

 The result would consist of 3 items:
 - parameter
 - hello
 - hellohello
```

Filtering String Content

Another example of how to use CSharpGraph.ShortName.

```
CxQL

 This example demonstrates the use of CSharpGraph.ShortName.
 The input source code is:

 public class DbClass{
       public int myQueries (int query_id){
              switch(query_id){
              case 0: return "UPDATE table SET comment = ''";
```

```
            case 1: return "SELECT * FROM table WHERE id = 1";
            case 2: return "  insert into table set select = 'all'  ";
            case 3: return "  select name from table where id = 3  ";
            default: return "DELETE from table";
            }
      }
 }

CxList strings = All.FindByType(typeof(StringLiteral));
char[] trimChars = new char[6] {' ', '\t', '"', '(', '\r', '\n'};
CxList SQL = strings.FindByName("*select *", false);
foreach (CxList sql in SQL)
{
      // Get the CSharpGraph of the only code element in sql
      CSharpGraph gr = sql.data.GetByIndex(0) as CSharpGraph;
      string name = gr.ShortName.TrimStart(trimChars);
      // Check a condition on name
      if (name.ToLower().StartsWith("select"))
      {
            // Add only elements which meet the condition
            result.Add(sql);
      }
}

 The result would consist of 2 items:
 - case 1
 - case 3
```

Find Variables at the same Line

This example shows how to use CSharpGraph.LinePragma in order to find all the variables present at the same line as one of the element of a given CxList.

```
CxQL

 This example demonstrates the use of CSharpGraph.ShortName.
 The input source code is:

 public class myClass{
      public int myMethod (int a, int b){
            int x = a + b;
            int y = a * b;
            int z = x * x + b * b;
            return x + y + z;
      }
 }

CxList x = All.FindByShortName("x");
CxList variables = All.FindByType(typeof(UnknownReference));
foreach (KeyValuePair<int, IGraph> dic in x.data)
{
      if (dic.Value.LinePragma != null)
      {
```

```
      result.Add(variables.FindByPosition(dic.Value.LinePragma.Line));
      }
}

 The result would consist of 9 items:
 - 2 results at line 5
 - 4 retults at line 7
 - 3 results at line 8
```

11 IDeclaration and IDefinition

11.1 public interface IDeclaration

```
IDefinition Definition { get;}
GraphTypes GraphType { get;}
```

11.2 public interface IDefinition

```
int Id { get; set; }
string Name { get; set; }
string FullName { get; }
Scope Scope { get; set; }
IDeclaration SourceGraph { get; set; }
IDeclaration Type { get; set; }
```

11.3 public interface ISymbol : IDefinition

```
IDefinition DeclaredType { get; set; }
IDefinition ActualType { get; set; }
```

Signatures from IDefinition

```
int Id { get; set; }
string Name { get; set; }
string FullName { get; }
Scope Scope { get; set; }
IDeclaration SourceGraph { get; set; }
IDeclaration Type { get; set; }
```

11.4 public class Definition : IDefinition

```
ArrayList AddMemberDefinitionRange(ArrayList defId);
ArrayList AddMemberInstanceRange(ArrayList instId);
bool AddMemberRange(ArrayList defIds,ArrayList instIds);
int AddNewDefinitionID(int defId);
int AddNewInstanceID(int instId);
bool AddNewMember(int defId, int instId);
void computeFullName();
void ConvertToOldMembersDS();
ArrayList Definition_Id { get; set; }
string FullName { get; }
int GetHashCode();
int GetMemberInstByDef(int defId);
int Id { get; set; }
bool InsertMember(int pos, int defId, int instId);
ArrayList InsertMemberDefinitionRange(ArrayList defId);
ArrayList InsertMemberInstanceRange(ArrayList instId);
bool InsertMemberRange(.ArrayList defIds, .ArrayList instIds);
ArrayList Instance_Id { get; set; }
Definition.CxArrayList MembersDefinition_id { get; set; }
```

```
Definition.CxArrayList MembersInstance_id { get; set; }
MembersDefinition MembersList { get; set; }
string Name { get; set; }
Scope Scope { get; set; }
void SetFullName(string fullName);
IDeclaration SourceGraph { get; set; }
string ToString();
IDeclaration Type { get; set; }
```

11.5 public class OverloadableDefinition : Definition

```
public DefinitionCollection Definitions
```

Methods from Definition:

```
ArrayList AddMemberDefinitionRange(ArrayList defId);
ArrayList AddMemberInstanceRange(ArrayList instId);
bool AddMemberRange(ArrayList defIds,ArrayList instIds);
int AddNewDefinitionID(int defId);
int AddNewInstanceID(int instId);
bool AddNewMember(int defId, int instId);
void computeFullName();
void ConvertToOldMembersDS();
ArrayList Definition_Id { get; set; }
string FullName { get; }
int GetHashCode();
int GetMemberInstByDef(int defId);
int Id { get; set; }
bool InsertMember(int pos, int defId, int instId);
ArrayList InsertMemberDefinitionRange(ArrayList defId);
ArrayList InsertMemberInstanceRange(ArrayList instId);
bool InsertMemberRange(.ArrayList defIds, .ArrayList instIds);
ArrayList Instance_Id { get; set; }
Definition.CxArrayList MembersDefinition_id { get; set; }
Definition.CxArrayList MembersInstance_id { get; set; }
MembersDefinition MembersList { get; set; }
string Name { get; set; }
Scope Scope { get; set; }
void SetFullName(string fullName);
IDeclaration SourceGraph { get; set; }
string ToString();
IDeclaration Type { get; set; }
```