Hydrogen

5.0

Generated by Doxygen 1.8.15

Fri Jul 5 2019 13:36:25

1 Hydrogen: MVICFG Generator

Table of Contents

Quick Start Guide:

It is advised to go through `Docs` to get an understanding of the project. If you are in a hurry, this will get you set up.

Building Hydrogen

1) Before building the project, make sure the dependencies are met. You can also make use of the docker image, where the environment is already set up for you. 2) Clone Hydrogen from GitLab. If you are using the Docker, you can clone it into /home/Hydrogen/MVICFG folder. 3) Compile Hydrogen with the help of CMakeLists.txt. You can also use GNU Make, if that is the preferred method. 4) Assuming you are using the Docker and Ninja, the steps would be like below. But first install Docker using the recommended method for your system.

```
# Download and run the Docker from your system.
$ docker run -it -name Hydrogen_Env ashwinkj/hydrogen_env
# The above command will put you inside the Docker Container.
$$ git clone https://git.linux.iastate.edu/HydrogenGroup/Hydrogen /home/Hydrogen/MVICFG
$$ cd /home/Hydrogen/MVICFG
$$ mkdir BuildNinja
$$ cmake -B BuildNinja -G Ninja .
$$ cd BuildNinja
$$ ninja
```

Using Hydrogen

1) Hydrogen needs both the source code and LLVM IR code to generate MVICFG and output it as MVICFG.dot for visualization. 2) To compile a single file program into LLVM IR code necessary for Hydrogen invoke clang with -O0 -Xclang -disable-O0-optnone -g -emit-llvm -S flag. 3) To generate MVICFG, call Hydrogen with both the LLVM IR and paths to their source files. Hydrogen will generate the diff from the source files to generate the MVICFG. 4) Assuming that you have two versions of Prog.c in two folder Buggy and Correct, the tentative steps to generate MVICFG is shown below.

```
# In folder Buggy, compile Prog.c into LLVM IR (ProgV1.bc)
$ cd TestPrograms/Buggy
$ clang -c -O0 -Xclang -disable-O0-optnone -g -emit-llvm -S Prog.c -o ProgV1.bc
# Similary in folder Correct, compile Prog.c into LLVM IR (ProgV2.bc)
$ cd ../Correct
$ clang -c -O0 -Xclang -disable-O0-optnone -g -emit-llvm -S Prog.c -o ProgV2.bc
```

5) Once the LLVM IR are generated, then use Hydrogen to generate the MVICFG.

```
# Generic Command
$ Hydrogen.out <Path-to-LLVMIR_1> <Path-to-LLVMIR_2> .. <Path-to-LLVMIR_N> :: <Path-to-file1-for-Prog_V1>
        ..\
 <Path-to-fileN-for-Prog_V1> :: <Path-to-file1-for-Prog_V2> .. <Path-to-fileN-for-Prog_V2> ..
# Command for the above example from BuildNinja folder
$ ./Hydrogen.out ../TestPrograms/Buggy/ProgV1.bc ../TestPrograms/Correct/ProgV2.bc ::
        ../TestPrograms/Buggy/Prog.c ::\
 ../TestPrograms/Correct/Prog.c
```

6) A python script BuildSystem.py is provided to ease the process of invoking the Hydrogen executable. It will also rebuild Hydrogen (if necessary) and transfer the resulting MVICFG.dot file into the parent directory.

Dependencies

Hydrogen depends on the LLVM Framework and Boost Libraries. Roughly, the following are required for Hydrogen to build properly

While slightly older versions for Cmake and Ninja can be used without any problem, using older versions of LLVM Framework and Boost can have unwanted consequences and may even result in build failure.

Documentation

Comments and more details for the program including the class structure with their supporting functions and their purpose can be found in Doc folder. Follow the README.txt inside the folder for more information.

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

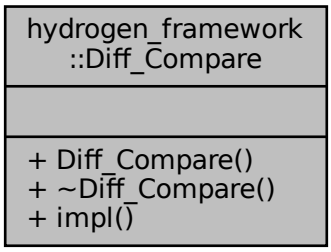
Here is a list of all documented files with brief descriptions:

5 Data Structure Documentation

5.1 hydrogen_framework::Diff_Compare Class Reference

`#include <Diff_Util.hpp>`

Collaboration diagram for hydrogen_framework::Diff_Compare:



Public Member Functions

- Diff_Compare ()
- virtual ~Diff_Compare ()
- virtual bool impl (const std::string &e1, const std::string &e2) const

5.1.1 Detailed Description

DiffCompare Functor class

Definition at line 16 of file Diff_Util.hpp.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Diff_Compare()

`hydrogen_framework::Diff_Compare::Diff_Compare ( ) [inline]`

Constructor

Definition at line 21 of file Diff_Util.hpp.

5.1.2.2 ~Diff_Compare()

```
virtual hydrogen_framework::Diff_Compare::~Diff_Compare ( )  [inline], [virtual]
```

Virtual Destructor

Definition at line 25 of file Diff_Util.hpp.

5.1.3 Member Function Documentation

5.1.3.1 impl()

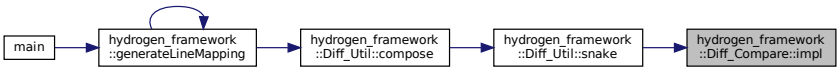
```
virtual bool hydrogen_framework::Diff_Compare::impl (
            const std::string & e1,
            const std::string & e2 ) const  [inline], [virtual]
```

Comparison function Return TRUE if equal

Definition at line 30 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Util::snake().

Here is the caller graph for this function:



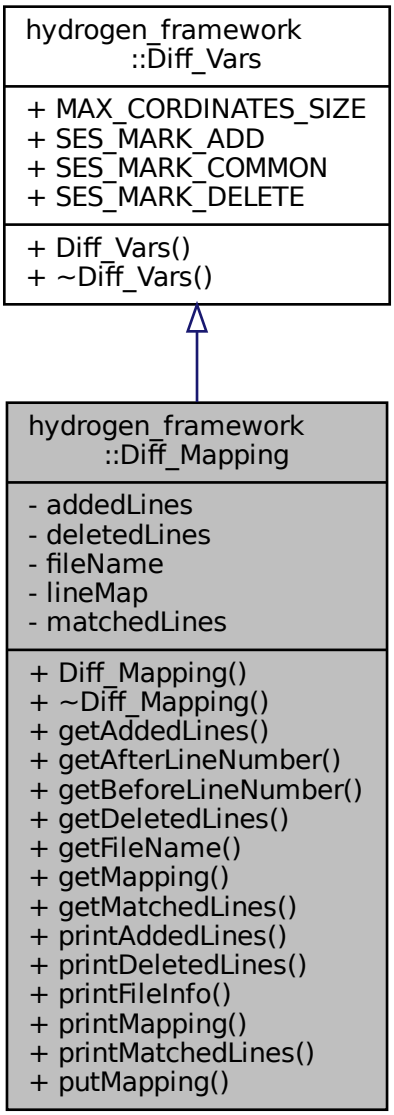
The documentation for this class was generated from the following file:

- Diff_Util.hpp

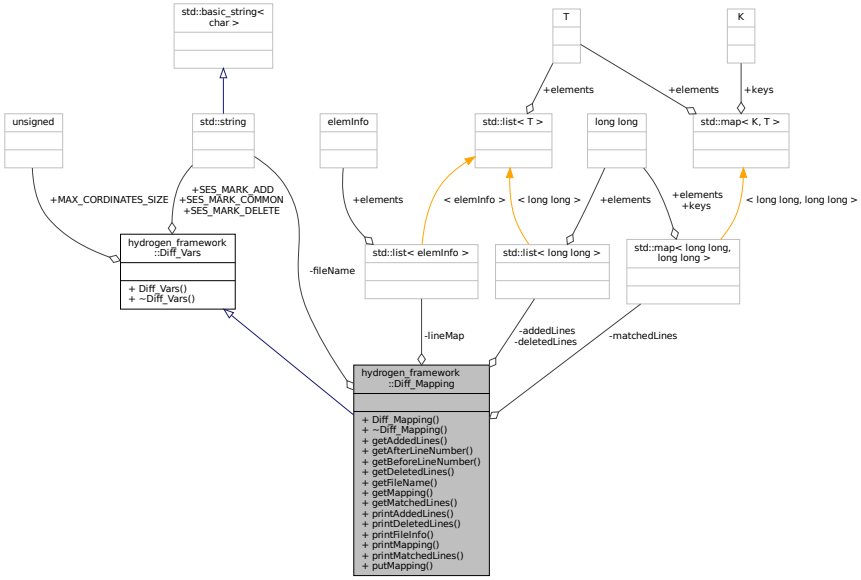
5.2 hydrogen_framework::Diff_Mapping Class Reference

```
#include <Diff_Mapping.hpp>
```

Inheritance diagram for hydrogen_framework::Diff_Mapping:



Collaboration diagram for hydrogen_framework::Diff_Mapping:



Public Member Functions

- Diff_Mapping (std::string name)
- ~Diff_Mapping ()
- std::list< long long > getAddedLines ()
- long long getAfterLineNumber (long long currLine)
- long long getBeforeLineNumber (long long currLine)
- std::list< long long > getDeletedLines ()
- std::string getFileName ()
- std::list< elemInfo > getMapping ()
- std::map< long long, long long > getMatchedLines ()
- void printAddedLines ()
- void printDeletedLines ()
- void printFileInfo ()
- void printMapping ()
- void printMatchedLines ()
- void putMapping (std::vector< sesElem > seqVector)

Private Attributes

- std::list< long long > addedLines
- std::list< long long > deletedLines
- std::string fileName
- std::list< elemInfo > lineMap
- std::map< long long, long long > matchedLines

Additional Inherited Members

5.2.1 Detailed Description

Diff_Mapping Class: Container for storing diff mapping details

Definition at line 18 of file Diff_Mapping.hpp.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Diff_Mapping()

```
hydrogen_framework::Diff_Mapping::Diff_Mapping (
            std::string name )  [inline]
```

Constructor

Definition at line 23 of file Diff_Mapping.hpp.

5.2.2.2 ~Diff_Mapping()

```
hydrogen_framework::Diff_Mapping::~Diff_Mapping ( )  [inline]
```

Destructor

Definition at line 28 of file Diff_Mapping.hpp.

References lineMap.

5.2.3 Member Function Documentation

5.2.3.1 getAddedLines()

```
std::list<long long> hydrogen_framework::Diff_Mapping::getAddedLines ( )  [inline]
```

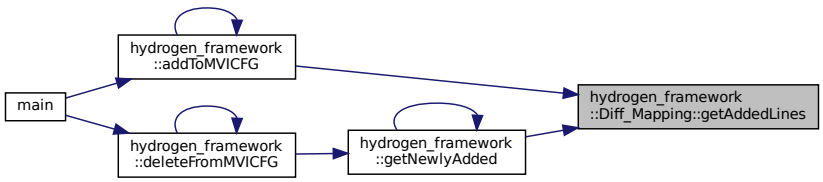
Return addedLines

Definition at line 48 of file Diff_Mapping.hpp.

References addedLines.

Referenced by hydrogen_framework::addToMVICFG(), and hydrogen_framework::getNewlyAdded().

Here is the caller graph for this function:



5.2.3.2 getAfterLineNumber()

```
long long hydrogen_framework::Diff_Mapping::getAfterLineNumber (
            long long currLine )
```

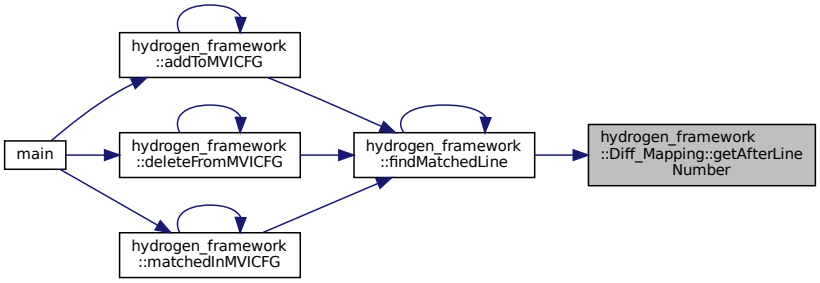
Get the afterIdx line number given the beforeIdx line number Return unsigned MAX if line not found

Definition at line 79 of file Diff_Mapping.cpp.

References lineMap.

Referenced by hydrogen_framework::findMatchedLine().

Here is the caller graph for this function:



5.2.3.3 getBeforeLineNumber()

```
long long hydrogen_framework::Diff_Mapping::getBeforeLineNumber (
            long long currLine )
```

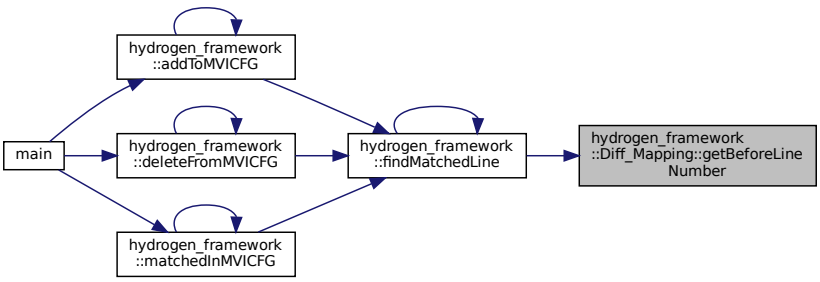
Get the beforeIdx line number given the afterIdx line number Return unsigned MAX if line not found

Definition at line 88 of file Diff_Mapping.cpp.

References lineMap.

Referenced by hydrogen_framework::findMatchedLine().

Here is the caller graph for this function:



5.2.3.4 getDeletedLines()

```
std::list<long long> hydrogen_framework::Diff_Mapping::getDeletedLines ( )  [inline]
```

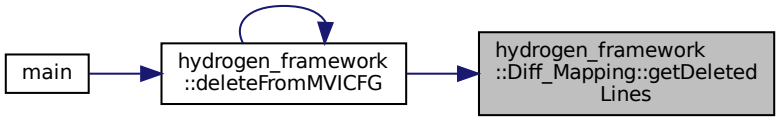
Return deletedLines

Definition at line 53 of file Diff_Mapping.hpp.

References deletedLines.

Referenced by hydrogen_framework::deleteFromMVICFG().

Here is the caller graph for this function:



5.2.3.5 getFileName()

```
std::string hydrogen_framework::Diff_Mapping::getFileName ( )  [inline]
```

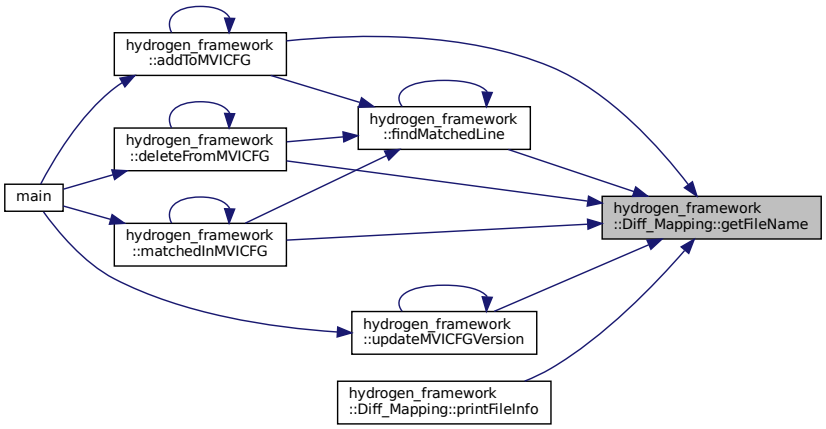
Return fileName

Definition at line 43 of file Diff_Mapping.hpp.

References fileName.

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::findMatch
hydrogen_framework::matchedInMVICFG(), printFileInfo(), and hydrogen_framework::updateMVICFGVersion().

Here is the caller graph for this function:



5.2.3.6 getMapping()

```
std::list<elemInfo> hydrogen_framework::Diff_Mapping::getMapping ( )  [inline]
```

Return lineMap

Definition at line 38 of file Diff_Mapping.hpp.

References lineMap.

5.2.3.7 getMatchedLines()

```
std::map<long long, long long> hydrogen_framework::Diff_Mapping::getMatchedLines ( )  [inline]
```

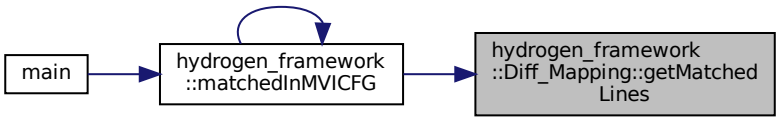
Return matchedLines

Definition at line 58 of file Diff_Mapping.hpp.

References matchedLines.

Referenced by hydrogen_framework::matchedInMVICFG().

Here is the caller graph for this function:



5.2.3.8 printAddedLines()

```
void hydrogen_framework::Diff_Mapping::printAddedLines ( )
```

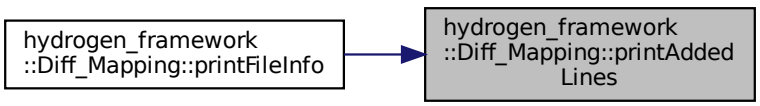
Print addedLines

Definition at line 53 of file Diff_Mapping.cpp.

References addedLines, and hydrogen_framework::Diff_Vars::SES_MARK_ADD.

Referenced by printFileInfo().

Here is the caller graph for this function:



5.2.3.9 printDeletedLines()

```
void hydrogen_framework::Diff_Mapping::printDeletedLines ( )
```

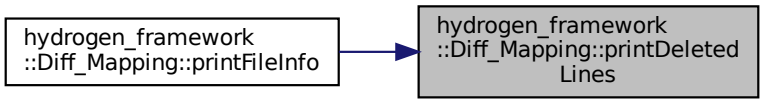
Print deletedLines

Definition at line 59 of file Diff_Mapping.cpp.

References deletedLines, and hydrogen_framework::Diff_Vars::SES_MARK_DELETE.

Referenced by printFileInfo().

Here is the caller graph for this function:



5.2.3.10 printFileInfo()

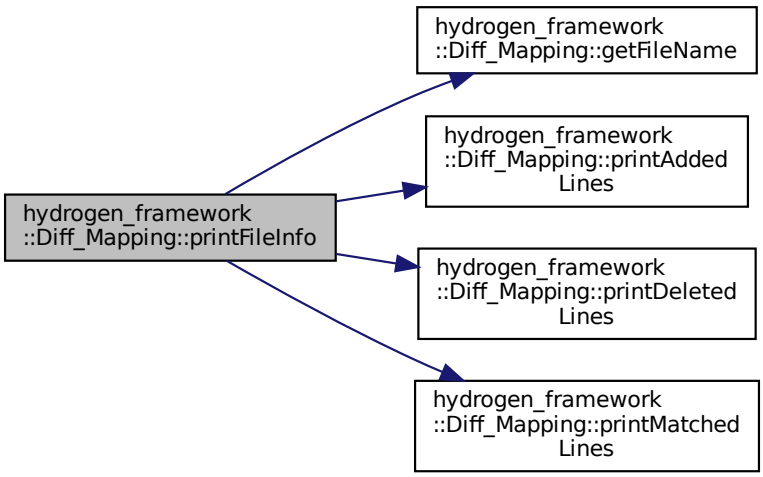
```
void hydrogen_framework::Diff_Mapping::printFileInfo ( )
```

Print File Related Info

Definition at line 71 of file Diff_Mapping.cpp.

References getFileName(), printAddedLines(), printDeletedLines(), and printMatchedLines().

Here is the call graph for this function:



5.2.3.11 printMapping()

```
void hydrogen_framework::Diff_Mapping::printMapping ( )
```

Print lineMap

Definition at line 35 of file Diff_Mapping.cpp.

References fileName, and lineMap.

5.2.3.12 printMatchedLines()

```
void hydrogen_framework::Diff_Mapping::printMatchedLines ( )
```

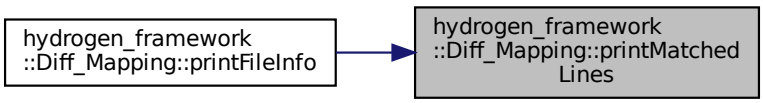
Print matchedLines

Definition at line 65 of file Diff_Mapping.cpp.

References matchedLines.

Referenced by printFileInfo().

Here is the caller graph for this function:



5.2.3.13 putMapping()

```
void hydrogen_framework::Diff_Mapping::putMapping (
            std::vector< sesElem > seqVector )
```

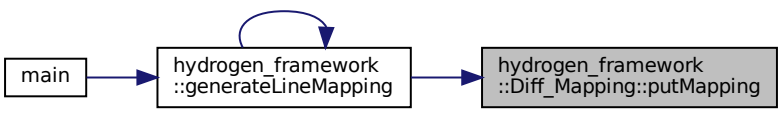
Populate line mapping

Definition at line 8 of file Diff_Mapping.cpp.

References addedLines, deletedLines, lineMap, and matchedLines.

Referenced by hydrogen_framework::generateLineMapping().

Here is the caller graph for this function:



5.2.4 Field Documentation

5.2.4.1 addedLines

```
std::list<long long> hydrogen_framework::Diff_Mapping::addedLines  [private]
```

Container for added line numbers

Definition at line 100 of file Diff_Mapping.hpp.

Referenced by getAddedLines(), printAddedLines(), and putMapping().

5.2.4.2 deletedLines

```
std::list<long long> hydrogen_framework::Diff_Mapping::deletedLines  [private]
```

Container for deleted line numbers

Definition at line 101 of file Diff_Mapping.hpp.

Referenced by getDeletedLines(), printDeletedLines(), and putMapping().

5.2.4.3 fileName

```
std::string hydrogen_framework::Diff_Mapping::fileName  [private]
```

File Name

Definition at line 98 of file Diff_Mapping.hpp.

Referenced by getFileName(), and printMapping().

5.2.4.4 lineMap

```
std::list<elemInfo> hydrogen_framework::Diff_Mapping::lineMap  [private]
```

Container for line mapping

Definition at line 99 of file Diff_Mapping.hpp.

Referenced by getAfterLineNumber(), getBeforeLineNumber(), getMapping(), printMapping(), putMapping(), and ~Diff_Mapping().

5.2.4.5 matchedLines

```
std::map<long long, long long> hydrogen_framework::Diff_Mapping::matchedLines  [private]
```

Container for matched line numbers mapping from before to after lines

Definition at line 103 of file Diff_Mapping.hpp.

Referenced by getMatchedLines(), printMatchedLines(), and putMapping().

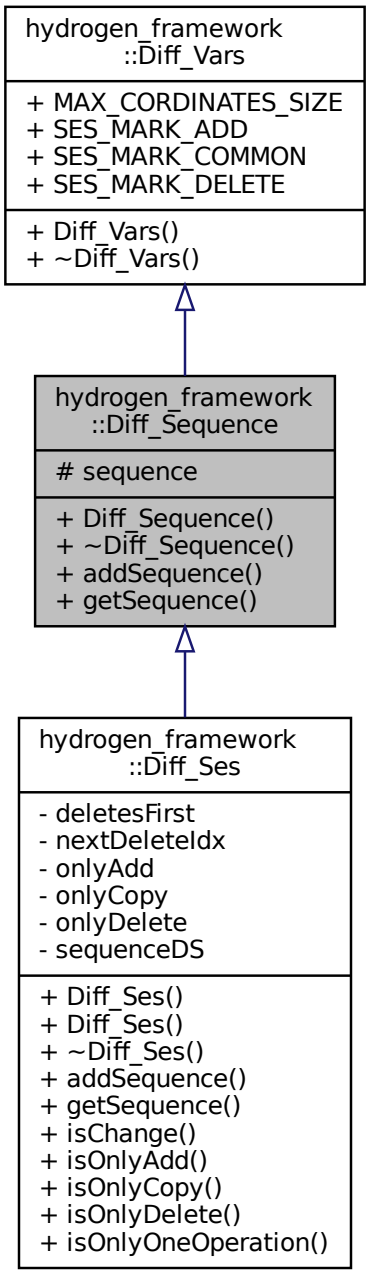
The documentation for this class was generated from the following files:

- Diff_Mapping.hpp
- Diff_Mapping.cpp

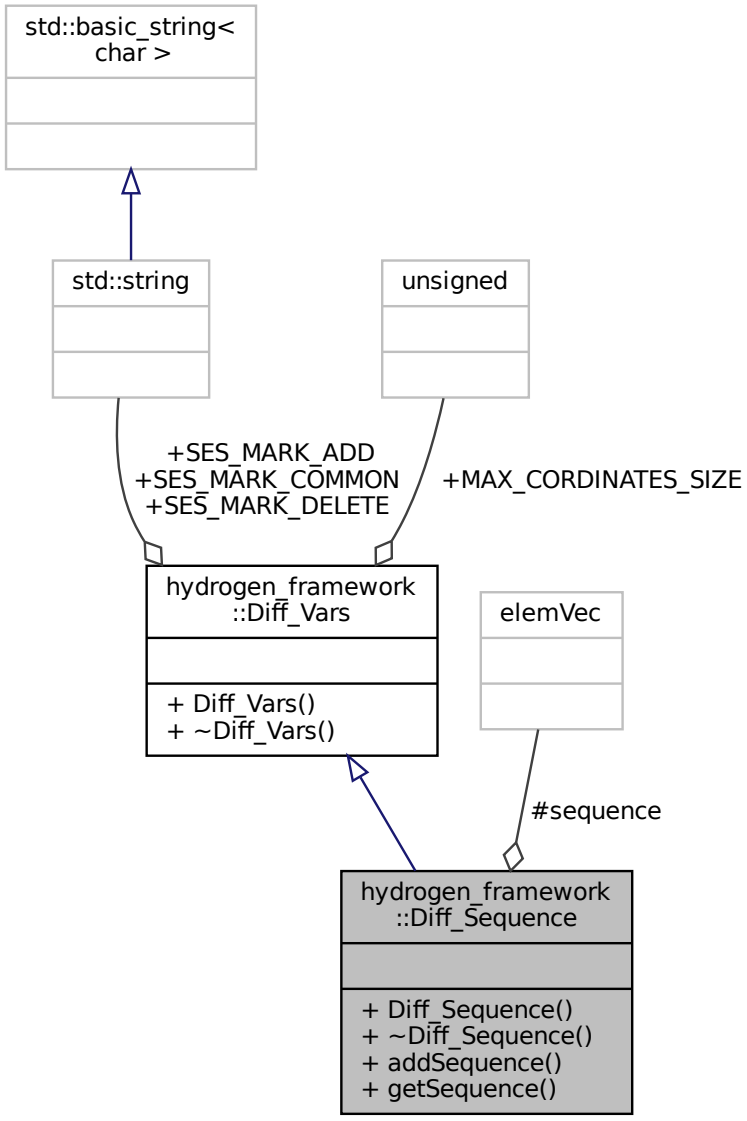
5.3 hydrogen_framework::Diff_Sequence Class Reference

```
#include <Diff_Util.hpp>
```

Inheritance diagram for hydrogen_framework::Diff_Sequence:



Collaboration diagram for hydrogen_framework::Diff_Sequence:



Public Member Functions

- Diff_Sequence ()
- virtual ~Diff_Sequence ()
- void addSequence (elem e)
- elemVec getSequence () const

Protected Attributes

- elemVec sequence

Additional Inherited Members

5.3.1 Detailed Description

Class to store sequence of elements

Definition at line 100 of file Diff_Util.hpp.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Diff_Sequence()

```
hydrogen_framework::Diff_Sequence::Diff_Sequence ( )  [inline]
```

Constructor

Definition at line 105 of file Diff_Util.hpp.

5.3.2.2 ~Diff_Sequence()

```
virtual hydrogen_framework::Diff_Sequence::~Diff_Sequence ( )  [inline], [virtual]
```

Virtual Destructor

Definition at line 109 of file Diff_Util.hpp.

5.3.3 Member Function Documentation

5.3.3.1 addSequence()

```
void hydrogen_framework::Diff_Sequence::addSequence (
            elem e )  [inline]
```

Add to sequence

Definition at line 120 of file Diff_Util.hpp.

5.3.3.2 getSequence()

```
elemVec hydrogen_framework::Diff_Sequence::getSequence ( ) const  [inline]
```

Return sequence

Definition at line 114 of file Diff_Util.hpp.

References sequence.

5.3.4 Field Documentation

5.3.4.1 sequence

```
elemVec hydrogen_framework::Diff_Sequence::sequence  [protected]
```

Store sequence of elems as vector

Definition at line 123 of file Diff_Util.hpp.

Referenced by getSequence().

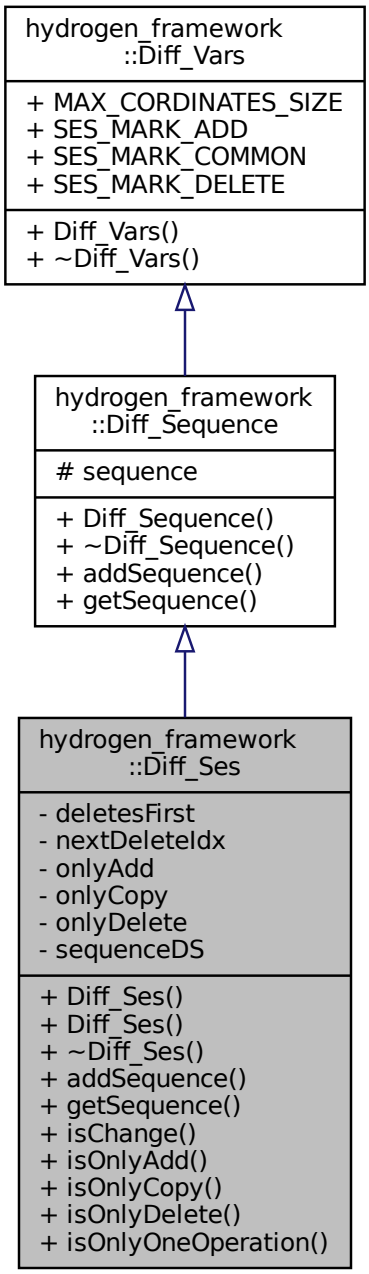
The documentation for this class was generated from the following file:

- Diff_Util.hpp

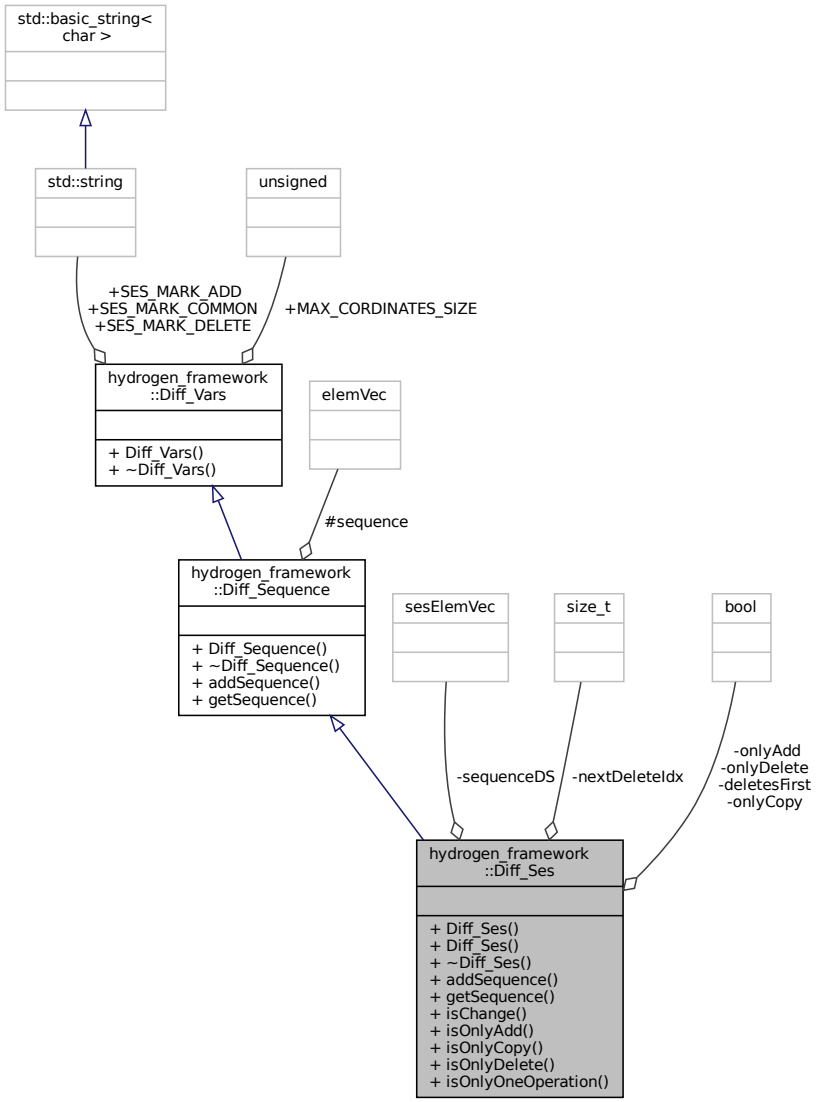
5.4 hydrogen_framework::Diff_Ses Class Reference

```
#include <Diff_Util.hpp>
```

Inheritance diagram for hydrogen_framework::Diff_Ses:



Collaboration diagram for hydrogen_framework::Diff_Ses:



Public Member Functions

- Diff_Ses ()
- Diff_Ses (bool moveDel)
- ~Diff_Ses ()
- void addSequence (elem e, long long beforeIdx, long long afterIdx, const int type)
- sesElemVec getSequence () const
- bool isChange () const
- bool isOnlyAdd () const
- bool isOnlyCopy () const
- bool isOnlyDelete () const
- bool isOnlyOneOperation () const

Private Attributes

- bool deletesFirst
- size_t nextDeleteIdx
- bool onlyAdd
- bool onlyCopy
- bool onlyDelete
- sesElemVec sequenceDS

Additional Inherited Members

5.4.1 Detailed Description

Class to compute Shortest Edit Distance

Definition at line 129 of file Diff_Util.hpp.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Diff_Ses() [1/2]

```
hydrogen_framework::Diff_Ses::Diff_Ses ( )  [inline]
```

Constructor with no argument

Definition at line 134 of file Diff_Util.hpp.

References nextDeleteIdx.

5.4.2.2 Diff_Ses() [2/2]

```
hydrogen_framework::Diff_Ses::Diff_Ses (
            bool moveDel )  [inline]
```

Constructor with one argument

Definition at line 139 of file Diff_Util.hpp.

References nextDeleteIdx.

5.4.2.3 ~Diff_Ses()

```
hydrogen_framework::Diff_Ses::~Diff_Ses ( )  [inline]
```

Destructor

Definition at line 144 of file Diff_Util.hpp.

5.4.3 Member Function Documentation

5.4.3.1 addSequence()

```
void hydrogen_framework::Diff_Ses::addSequence (
            elem e,
            long long beforeIdx,
            long long afterIdx,
            const int type )
```

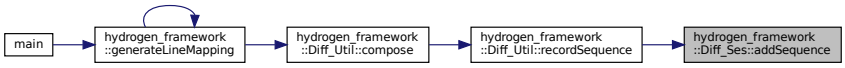
Add sequence

Definition at line 8 of file Diff_Util.cpp.

References hydrogen_framework::Diff_Vars::eleminfo::afterIdx, hydrogen_framework::Diff_Vars::eleminfo::beforeIdx, deletesFirst, nextDeleteIdx, onlyAdd, onlyCopy, onlyDelete, sequenceDS, and hydrogen_framework::Diff_Vars::eleminfo::type.

Referenced by hydrogen_framework::Diff_Util::recordSequence().

Here is the caller graph for this function:



5.4.3.2 getSequence()

```
sesElemVec hydrogen_framework::Diff_Ses::getSequence ( ) const  [inline]
```

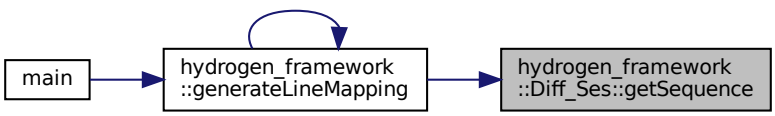
Return sequence

Definition at line 179 of file Diff_Util.hpp.

References sequenceDS.

Referenced by hydrogen_framework::generateLineMapping().

Here is the caller graph for this function:



5.4.3.3 isChange()

```
bool hydrogen_framework::Diff_Ses::isChange ( ) const  [inline]
```

Return TRUE if onlyCopy is FALSE

Definition at line 169 of file Diff_Util.hpp.

References onlyCopy.

5.4.3.4 isOnlyAdd()

```
bool hydrogen_framework::Diff_Ses::isOnlyAdd ( ) const  [inline]
```

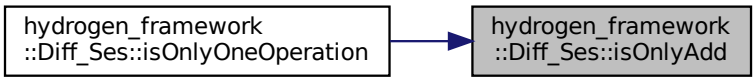
Return onlyAdd

Definition at line 149 of file Diff_Util.hpp.

References onlyAdd.

Referenced by isOnlyOneOperation().

Here is the caller graph for this function:



5.4.3.5 isOnlyCopy()

```
bool hydrogen_framework::Diff_Ses::isOnlyCopy ( ) const  [inline]
```

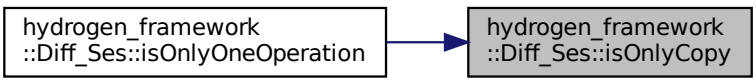
Return onlyCopy

Definition at line 159 of file Diff_Util.hpp.

References onlyCopy.

Referenced by isOnlyOneOperation().

Here is the caller graph for this function:



5.4.3.6 isOnlyDelete()

```
bool hydrogen_framework::Diff_Ses::isOnlyDelete ( ) const  [inline]
```

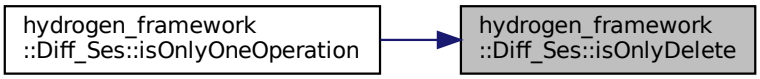
Return onlyDelete

Definition at line 154 of file Diff_Util.hpp.

References onlyDelete.

Referenced by isOnlyOneOperation().

Here is the caller graph for this function:



5.4.3.7 isOnlyOneOperation()

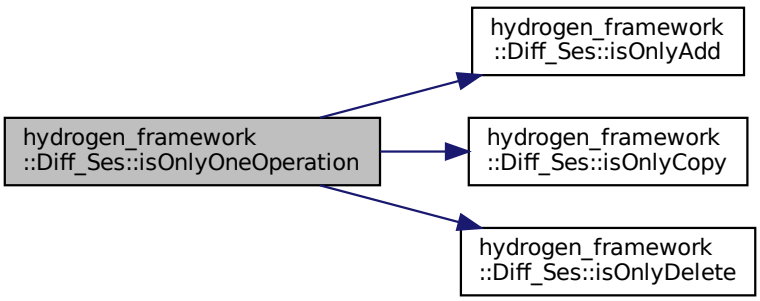
```
bool hydrogen_framework::Diff_Ses::isOnlyOneOperation ( ) const  [inline]
```

Return TRUE if any of onlyAdd, onlyDelete or onlyCopy is TRUE

Definition at line 164 of file Diff_Util.hpp.

References isOnlyAdd(), isOnlyCopy(), and isOnlyDelete().

Here is the call graph for this function:



5.4.4 Field Documentation

5.4.4.1 deletesFirst

```
bool hydrogen_framework::Diff_Ses::deletesFirst  [private]
```

Flag to indicate if deletion is required first

Definition at line 186 of file Diff_Util.hpp.

Referenced by addSequence().

5.4.4.2 nextDeleteIdx

```
size_t hydrogen_framework::Diff_Ses::nextDeleteIdx  [private]
```

Point towards next deletion ID

Definition at line 187 of file Diff_Util.hpp.

Referenced by addSequence(), and Diff_Ses().

5.4.4.3 onlyAdd

```
bool hydrogen_framework::Diff_Ses::onlyAdd  [private]
```

Flag to indicate add operation

Definition at line 183 of file Diff_Util.hpp.

Referenced by addSequence(), and isOnlyAdd().

5.4.4.4 onlyCopy

```
bool hydrogen_framework::Diff_Ses::onlyCopy  [private]
```

Flag to indicate change operation

Definition at line 185 of file Diff_Util.hpp.

Referenced by addSequence(), isChange(), and isOnlyCopy().

5.4.4.5 onlyDelete

```
bool hydrogen_framework::Diff_Ses::onlyDelete  [private]
```

Flag to indicate deletion operation

Definition at line 184 of file Diff_Util.hpp.

Referenced by addSequence(), and isOnlyDelete().

5.4.4.6 sequenceDS

```
sesElemVec hydrogen_framework::Diff_Ses::sequenceDS  [private]
```

SES sequence

Definition at line 182 of file Diff_Util.hpp.

Referenced by addSequence(), and getSequence().

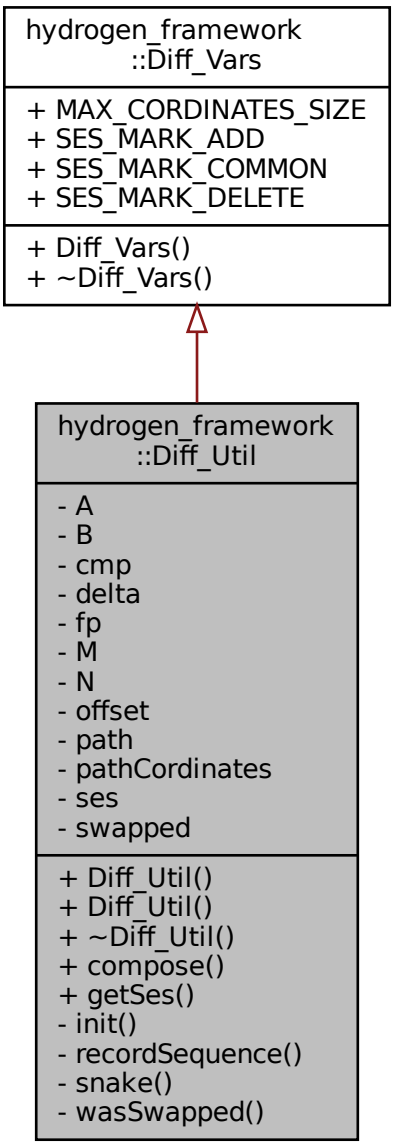
The documentation for this class was generated from the following files:

- Diff_Util.hpp
- Diff_Util.cpp

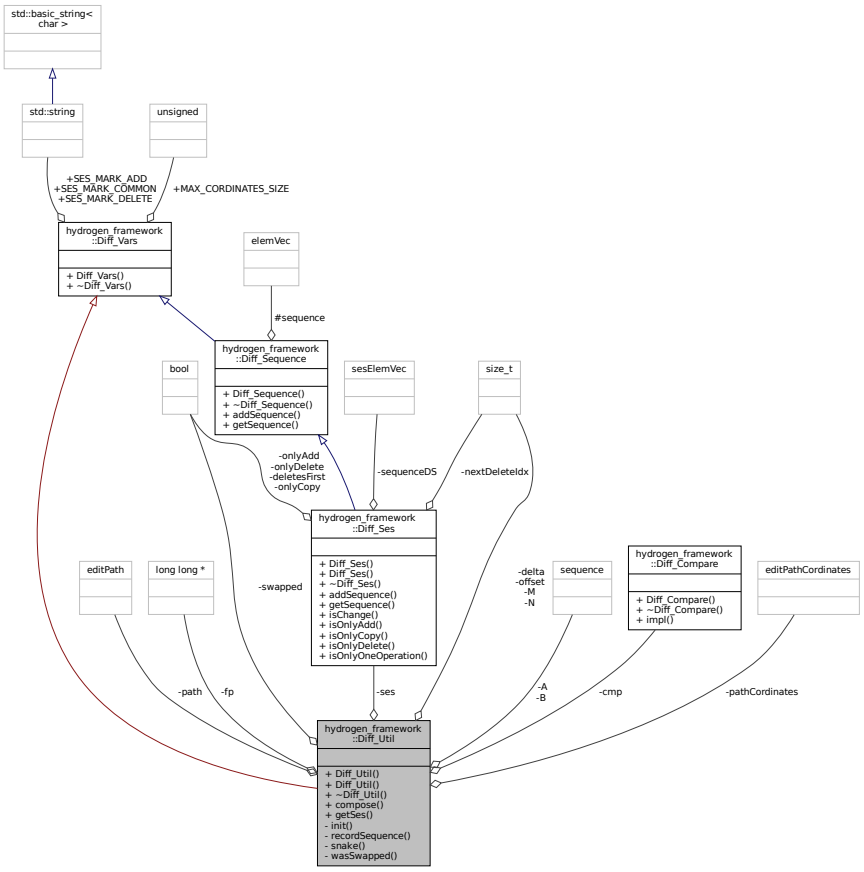
5.5 hydrogen_framework::Diff_Util Class Reference

```
#include <Diff_Util.hpp>
```

Inheritance diagram for hydrogen_framework::Diff_Util:



Collaboration diagram for hydrogen_framework::Diff_Util:



Public Member Functions

- Diff_Util ()
- Diff_Util (const sequence &a, const sequence &b)
- ~Diff_Util ()
- void compose ()
- Diff_Ses getSes () const

Private Member Functions

- void init ()
- bool recordSequence (const editPathCordinates &v)
- long long snake (const long long &k, const long long &above, const long long &below)
- bool wasSwapped () const

Private Attributes

- sequence A
- sequence B
- Diff_Compare cmp
- size_t delta

- long long * fp
- size_t M
- size_t N
- size_t offset
- editPath path
- editPathCordinates pathCordinates
- Diff_Ses ses
- bool swapped

Additional Inherited Members

5.5.1 Detailed Description

Class to compute Diff between two files

Definition at line 193 of file Diff_Util.hpp.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Diff_Util() [1/2]

```
hydrogen_framework::Diff_Util::Diff_Util ( )  [inline]
```

Constructor with no arguments

Definition at line 198 of file Diff_Util.hpp.

5.5.2.2 Diff_Util() [2/2]

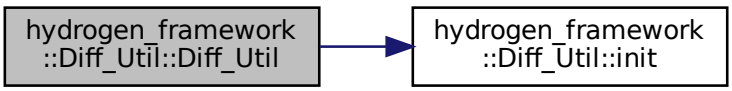
```
hydrogen_framework::Diff_Util::Diff_Util (
            const sequence & a,
            const sequence & b )  [inline]
```

Constructor with two arguments

Definition at line 203 of file Diff_Util.hpp.

References init().

Here is the call graph for this function:



5.5.2.3 ~Diff_Util()

```
hydrogen_framework::Diff_Util::~Diff_Util ( )  [inline]
```

Destructor

Definition at line 208 of file Diff_Util.hpp.

5.5.3 Member Function Documentation

5.5.3.1 compose()

```
void hydrogen_framework::Diff_Util::compose ( )
```

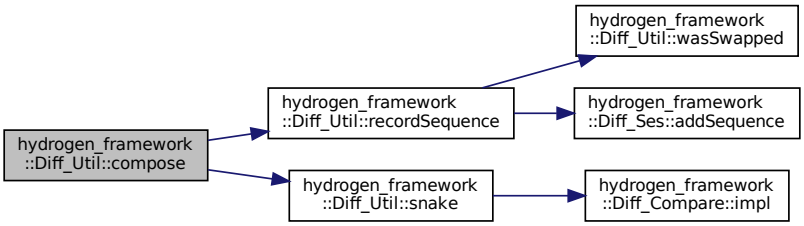
Compose Longest Common Subsequence and Shortest Edit Script. The algorithm implemented here is based on "An O(NP) Sequence Comparison Algorithm" described by Sun Wu, Udi Manber and Gene Myers

Definition at line 44 of file Diff_Util.cpp.

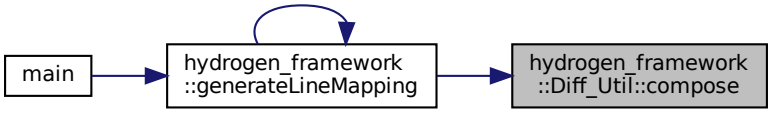
References delta, fp, M, hydrogen_framework::Diff_Vars::MAX_CORDINATES_SIZE, N, offset, path, pathCordinates, recordSequence(), snake(), hydrogen_framework::Diff_Vars::Point::x, and hydrogen_framework::Diff_Vars::Point::y.

Referenced by hydrogen_framework::generateLineMapping().

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.3.2 getSes()

```
Diff_Ses hydrogen_framework::Diff_Util::getSes ( ) const  [inline]
```

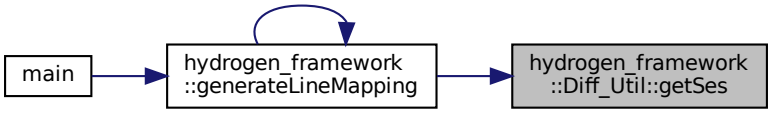
Return ses

Definition at line 213 of file Diff_Util.hpp.

References ses.

Referenced by hydrogen_framework::generateLineMapping().

Here is the caller graph for this function:



5.5.3.3 init()

```
void hydrogen_framework::Diff_Util::init ( )  [private]
```

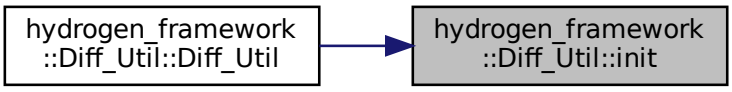
Initialize

Definition at line 84 of file Diff_Util.cpp.

References A, B, delta, fp, M, N, offset, and swapped.

Referenced by Diff_Util().

Here is the caller graph for this function:



5.5.3.4 recordSequence()

```
bool hydrogen_framework::Diff_Util::recordSequence (
            const editPathCordinates & v )  [private]
```

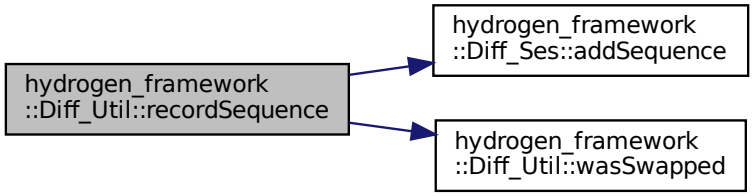
Record SES

Definition at line 118 of file Diff_Util.cpp.

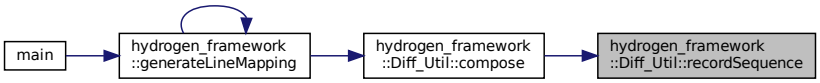
References A, hydrogen_framework::Diff_Ses::addSequence(), B, delta, fp, M, N, offset, path, ses, and wasSwapped().

Referenced by compose().

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.3.5 snake()

```
long long hydrogen_framework::Diff_Util::snake (
            const long long & k,
            const long long & above,
            const long long & below )  [private]
```

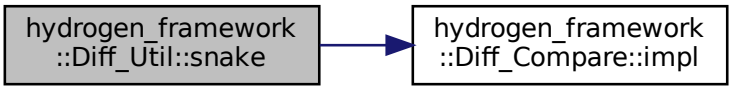
Search shortest path and record the path

Definition at line 99 of file Diff_Util.cpp.

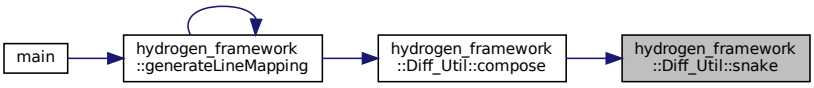
References A, B, cmp, hydrogen_framework::Diff_Compare::impl(), M, N, offset, path, pathCordinates, swapped, and hydrogen_framework::Diff_Vars::Point::x.

Referenced by compose().

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.3.6 wasSwapped()

```
bool hydrogen_framework::Diff_Util::wasSwapped ( ) const  [inline], [private]
```

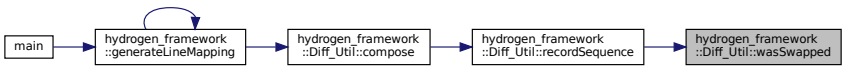
Check if the sequences have been swapped

Definition at line 253 of file Diff_Util.hpp.

References swapped.

Referenced by recordSequence().

Here is the caller graph for this function:



5.5.4 Field Documentation

5.5.4.1 A

```
sequence hydrogen_framework::Diff_Util::A  [private]
```

First sequence

Definition at line 223 of file Diff_Util.hpp.

Referenced by init(), recordSequence(), and snake().

5.5.4.2 B

```
sequence hydrogen_framework::Diff_Util::B  [private]
```

Second sequence

Definition at line 224 of file Diff_Util.hpp.

Referenced by init(), recordSequence(), and snake().

5.5.4.3 cmp

```
Diff_Compare hydrogen_framework::Diff_Util::cmp  [private]
```

Comparison Functor

Definition at line 234 of file Diff_Util.hpp.

Referenced by snake().

5.5.4.4 delta

```
size_t hydrogen_framework::Diff_Util::delta  [private]
```

Delta

Definition at line 227 of file Diff_Util.hpp.

Referenced by compose(), init(), and recordSequence().

5.5.4.5 fp

```
long long* hydrogen_framework::Diff_Util::fp  [private]
```

Pointer to elem

Definition at line 229 of file Diff_Util.hpp.

Referenced by compose(), init(), and recordSequence().

5.5.4.6 M

```
size_t hydrogen_framework::Diff_Util::M  [private]
```

M value

Definition at line 225 of file Diff_Util.hpp.

Referenced by compose(), init(), recordSequence(), and snake().

5.5.4.7 N

```
size_t hydrogen_framework::Diff_Util::N  [private]
```

N value

Definition at line 226 of file Diff_Util.hpp.

Referenced by compose(), init(), recordSequence(), and snake().

5.5.4.8 offset

```
size_t hydrogen_framework::Diff_Util::offset  [private]
```

offset

Definition at line 228 of file Diff_Util.hpp.

Referenced by compose(), init(), recordSequence(), and snake().

5.5.4.9 path

```
editPath hydrogen_framework::Diff_Util::path  [private]
```

Edit path

Definition at line 231 of file Diff_Util.hpp.

Referenced by compose(), recordSequence(), and snake().

5.5.4.10 pathCordinates

```
editPathCordinates hydrogen_framework::Diff_Util::pathCordinates  [private]
```

Edit path coordinates

Definition at line 232 of file Diff_Util.hpp.

Referenced by compose(), and snake().

5.5.4.11 ses

```
Diff_Ses hydrogen_framework::Diff_Util::ses  [private]
```

Shortest edit scrit

Definition at line 230 of file Diff_Util.hpp.

Referenced by getSes(), and recordSequence().

5.5.4.12 swapped

```
bool hydrogen_framework::Diff_Util::swapped  [private]
```

Flag to check if sequence are swapped

Definition at line 233 of file Diff_Util.hpp.

Referenced by init(), snake(), and wasSwapped().

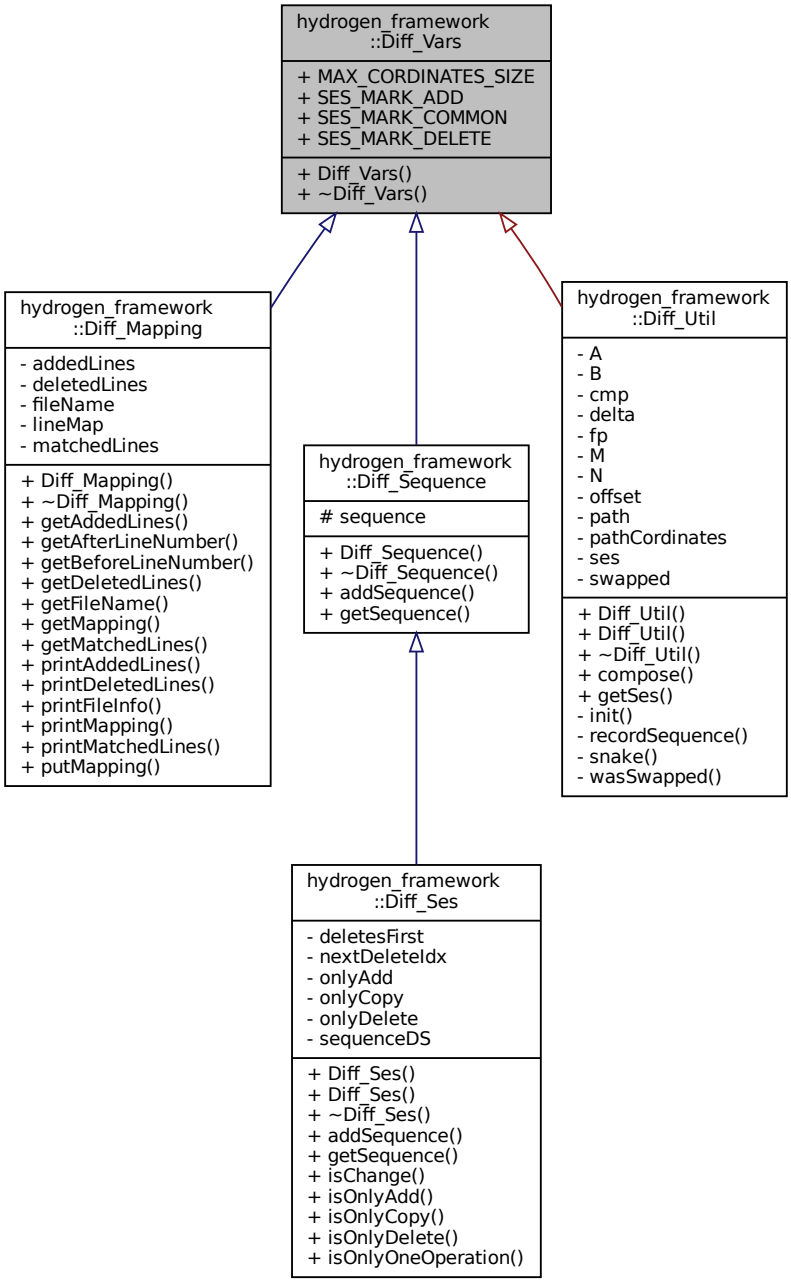
The documentation for this class was generated from the following files:

- Diff_Util.hpp
- Diff_Util.cpp

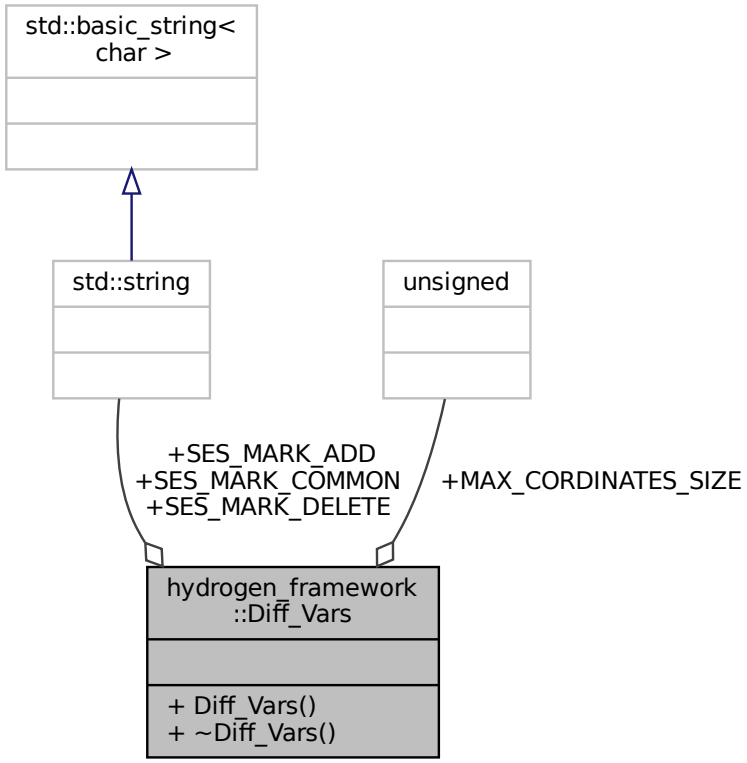
5.6 hydrogen_framework::Diff_Vars Class Reference

```
#include <Diff_Util.hpp>
```

Inheritance diagram for hydrogen_framework::Diff_Vars:



Collaboration diagram for hydrogen_framework::Diff_Vars:



Data Structures

- struct eleminfo
- struct Point

Public Types

- typedef std::vector< long long > editPath
- typedef std::vector< P > editPathCordinates
- typedef std::string elem
- typedef struct hydrogen_framework::Diff_Vars::eleminfo elemInfo
- typedef std::list< elem > elemList
- typedef elemList::iterator elemList_iter
- typedef std::vector< elem > elemVec
- typedef elemVec::iterator elemVec_iter
- typedef struct hydrogen_framework::Diff_Vars::Point P
- typedef std::vector< elem > sequence
- typedef sequence::const_iterator sequence_const_iter
- typedef sequence::iterator sequence_iter
- enum SES_TYPE { **SES_DELETE** = -1, **SES_COMMON** = 0, **SES_ADD** = 1 }
- typedef std::pair< elem, elemInfo > sesElem
- typedef std::vector< sesElem > sesElemVec
- typedef sesElemVec::iterator sesElemVec_iter

Public Member Functions

- Diff_Vars ()
- virtual ~Diff_Vars ()

Data Fields

- const unsigned long long MAX_CORDINATES_SIZE = 2000000
- std::string SES_MARK_ADD = "+"
- std::string SES_MARK_COMMON = " "
- std::string SES_MARK_DELETE = "-"

5.6.1 Detailed Description

Class to hold common/shared type definitions and variables

Definition at line 36 of file Diff_Util.hpp.

5.6.2 Member Typedef Documentation

5.6.2.1 editPath

```
typedef std::vector<long long> hydrogen_framework::Diff_Vars::editPath
```

Type definition for editPath

Definition at line 82 of file Diff_Util.hpp.

5.6.2.2 editPathCordinates

```
typedef std::vector<P> hydrogen_framework::Diff_Vars::editPathCordinates
```

Type definition for editPathCordinates

Definition at line 83 of file Diff_Util.hpp.

5.6.2.3 elem

```
typedef std::string hydrogen_framework::Diff_Vars::elem
```

Type definition for elem

Definition at line 84 of file Diff_Util.hpp.

5.6.2.4 elemInfo

```
typedef struct hydrogen_framework::Diff_Vars::eleminfo hydrogen_framework::Diff_Vars::elemInfo
```

Structure for storing element information

5.6.2.5 elemList

```
typedef std::list<elem> hydrogen_framework::Diff_Vars::elemList
```

Type definition for elemList

Definition at line 88 of file Diff_Util.hpp.

5.6.2.6 elemList_iter

```
typedef elemList::iterator hydrogen_framework::Diff_Vars::elemList_iter
```

Type definition for elemList_iter

Definition at line 91 of file Diff_Util.hpp.

5.6.2.7 elemVec

```
typedef std::vector<elem> hydrogen_framework::Diff_Vars::elemVec
```

Type definition for elemVec

Definition at line 89 of file Diff_Util.hpp.

5.6.2.8 elemVec_iter

```
typedef elemVec::iterator hydrogen_framework::Diff_Vars::elemVec_iter
```

Type definition for elemVec_iter

Definition at line 94 of file Diff_Util.hpp.

5.6.2.9 P

```
typedef struct hydrogen_framework::Diff_Vars::Point hydrogen_framework::Diff_Vars::P
```

Coordinate for registering route

5.6.2.10 sequence

```
typedef std::vector<elem> hydrogen_framework::Diff_Vars::sequence
```

Type definition for sequence

Definition at line 85 of file Diff_Util.hpp.

5.6.2.11 sequence_const_iter

```
typedef sequence::const_iterator hydrogen_framework::Diff_Vars::sequence_const_iter
```

Type definition for sequence_const_iter

Definition at line 93 of file Diff_Util.hpp.

5.6.2.12 sequence_iter

```
typedef sequence::iterator hydrogen_framework::Diff_Vars::sequence_iter
```

Type definition for sequence_iter

Definition at line 92 of file Diff_Util.hpp.

5.6.2.13 sesElem

```
typedef std::pair<elem, elemInfo> hydrogen_framework::Diff_Vars::sesElem
```

Type definition for sesElem

Definition at line 86 of file Diff_Util.hpp.

5.6.2.14 sesElemVec

```
typedef std::vector<sesElem> hydrogen_framework::Diff_Vars::sesElemVec
```

Type definition for sesElemVec

Definition at line 87 of file Diff_Util.hpp.

5.6.2.15 sesElemVec_iter

```
typedef sesElemVec::iterator hydrogen_framework::Diff_Vars::sesElemVec_iter
```

Type definition for sesElemVec_iter

Definition at line 90 of file Diff_Util.hpp.

5.6.3 Member Enumeration Documentation

5.6.3.1 SES_TYPE

```
enum hydrogen_framework::Diff_Vars::SES_TYPE
```

Type of edit for SES

Definition at line 51 of file Diff_Util.hpp.

5.6.4 Constructor & Destructor Documentation

5.6.4.1 Diff_Vars()

```
hydrogen_framework::Diff_Vars::Diff_Vars ( )  [inline]
```

Constructor

Definition at line 41 of file Diff_Util.hpp.

5.6.4.2 ~Diff_Vars()

```
virtual hydrogen_framework::Diff_Vars::~Diff_Vars ( )  [inline], [virtual]
```

Virtual Destructor

Definition at line 46 of file Diff_Util.hpp.

5.6.5 Field Documentation

5.6.5.1 MAX_CORDINATES_SIZE

```
const unsigned long long hydrogen_framework::Diff_Vars::MAX_CORDINATES_SIZE = 2000000
```

Limit of coordinate size

Definition at line 81 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Util::compose().

5.6.5.2 SES_MARK_ADD

```
std::string hydrogen_framework::Diff_Vars::SES_MARK_ADD = "+"
```

Setting SES_MARK_ADD

Definition at line 55 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Mapping::printAddedLines().

5.6.5.3 SES_MARK_COMMON

```
std::string hydrogen_framework::Diff_Vars::SES_MARK_COMMON = " "
```

Setting SES_MARK_COMMON

Definition at line 54 of file Diff_Util.hpp.

5.6.5.4 SES_MARK_DELETE

```
std::string hydrogen_framework::Diff_Vars::SES_MARK_DELETE = "-"
```

Setting SES_MARK_DELETE

Definition at line 53 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Mapping::printDeletedLines().

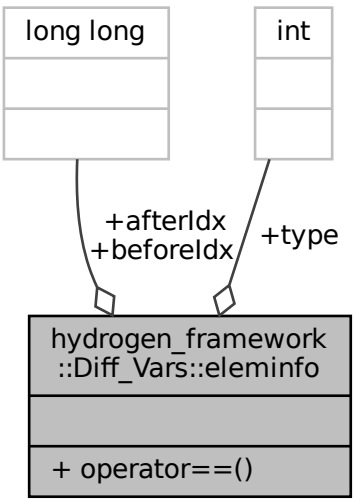
The documentation for this class was generated from the following file:

- Diff_Util.hpp

5.7 hydrogen_framework::Diff_Vars::eleminfo Struct Reference

```
#include <Diff_Util.hpp>
```

Collaboration diagram for hydrogen_framework::Diff_Vars::eleminfo:



Public Member Functions

- bool operator== (const eleminfo &other) const

Data Fields

- long long afterIdx
- long long beforeIdx
- int type

5.7.1 Detailed Description

Structure for storing element information

Definition at line 60 of file Diff_Util.hpp.

5.7.2 Member Function Documentation

5.7.2.1 operator==()

```
bool hydrogen_framework::Diff_Vars::eleminfo::operator== (
            const eleminfo & other ) const  [inline]
```

Overriding equal operation

Definition at line 67 of file Diff_Util.hpp.

References afterIdx, beforeIdx, and type.

5.7.3 Field Documentation

5.7.3.1 afterIdx

```
long long hydrogen_framework::Diff_Vars::eleminfo::afterIdx
```

Index of after sequence

Definition at line 62 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Ses::addSequence(), and operator==().

5.7.3.2 beforeIdx

```
long long hydrogen_framework::Diff_Vars::eleminfo::beforeIdx
```

Index of prev sequence

Definition at line 61 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Ses::addSequence(), and operator==().

5.7.3.3 type

```
int hydrogen_framework::Diff_Vars::eleminfo::type
```

Type of edit(Add, Delete, Common)

Definition at line 63 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Ses::addSequence(), and operator==().

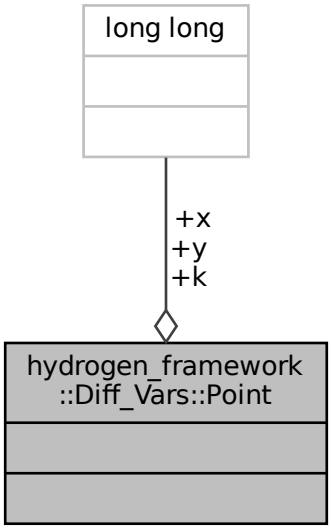
The documentation for this struct was generated from the following file:

- Diff_Util.hpp

5.8 hydrogen_framework::Diff_Vars::Point Struct Reference

```
#include <Diff_Util.hpp>
```

Collaboration diagram for hydrogen_framework::Diff_Vars::Point:



Data Fields

- long long k
- long long x
- long long y

5.8.1 Detailed Description

Coordinate for registering route

Definition at line 75 of file Diff_Util.hpp.

5.8.2 Field Documentation

5.8.2.1 k

```
long long hydrogen_framework::Diff_Vars::Point::k
```

vertex

Definition at line 78 of file Diff_Util.hpp.

5.8.2.2 x

```
long long hydrogen_framework::Diff_Vars::Point::x
```

X coordinate

Definition at line 76 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Util::compose(), and hydrogen_framework::Diff_Util::snake().

5.8.2.3 y

```
long long hydrogen_framework::Diff_Vars::Point::y
```

Y coordinate

Definition at line 77 of file Diff_Util.hpp.

Referenced by hydrogen_framework::Diff_Util::compose().

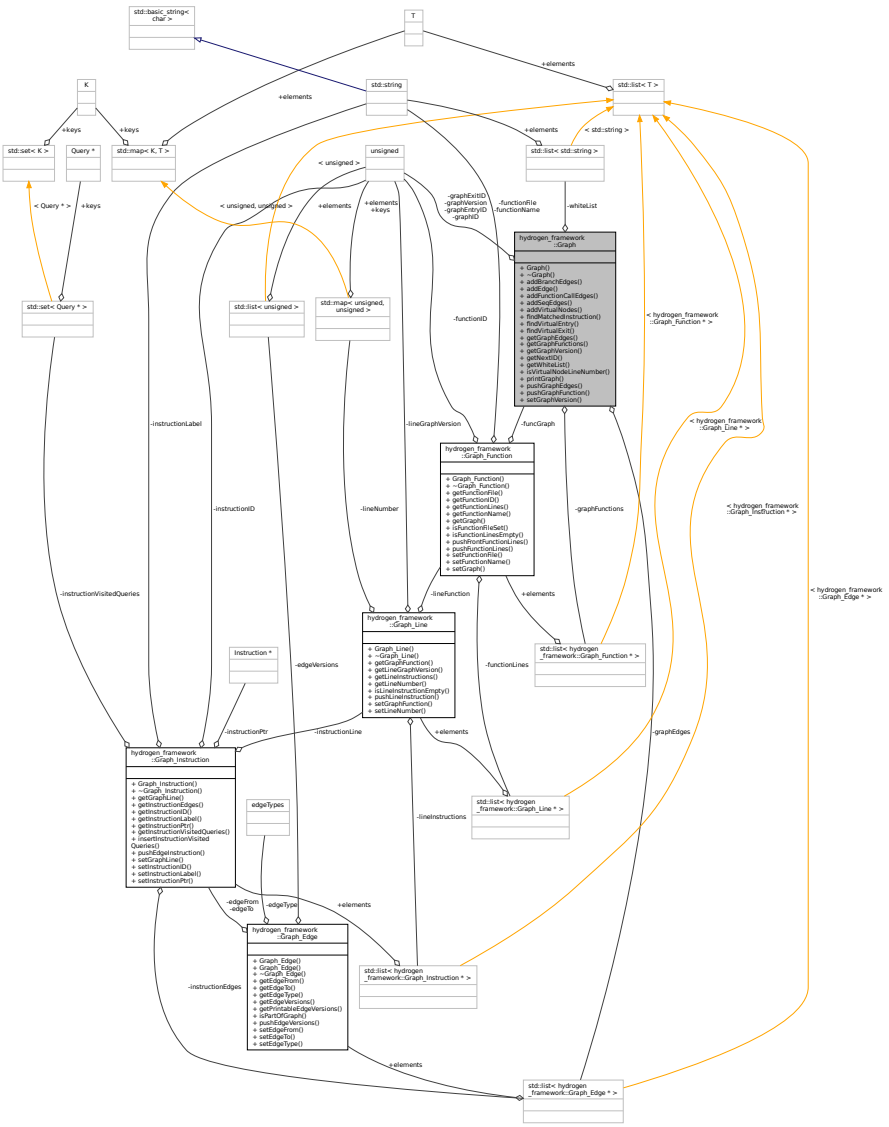
The documentation for this struct was generated from the following file:

- Diff_Util.hpp

5.9 hydrogen_framework::Graph Class Reference

`#include <Graph.hpp>`

Collaboration diagram for hydrogen_framework::Graph:



Public Member Functions

- Graph (unsigned ver)
- ~Graph ()
- void addBranchEdges ()
- void addEdge (Graph_Instruction *from, Graph_Instruction *to, Graph_Edge *edge)
- void addFunctionCallEdges ()
- void addSeqEdges (Graph_Line *line)
- void addVirtualNodes (Graph_Function *func)
- Graph_Instruction * findMatchedInstruction (llvm::Instruction *matchInst)
- Graph_Instruction * findVirtualEntry (std::string funcName)

- Graph_Instruction * findVirtualExit (std::string funcName)
- std::list< Graph_Edge * > getGraphEdges ()
- std::list< Graph_Function * > getGraphFunctions ()
- unsigned getGraphVersion ()
- unsigned getNextID ()
- std::list< std::string > getWhiteList ()
- bool isVirtualNodeLineNumber (unsigned lineNumber)
- void printGraph (std::string graphName)
- void pushGraphEdges (Graph_Edge *edge)
- void pushGraphFunction (Graph_Function *func)
- void setGraphVersion (unsigned ver)

Private Attributes

- std::list< Graph_Edge * > graphEdges
- unsigned graphEntryID
- unsigned graphExitID
- std::list< Graph_Function * > graphFunctions
- unsigned graphID
- unsigned graphVersion
- std::list< std::string > whiteList

5.9.1 Detailed Description

Graph Class: Class for generating Graphs

Definition at line 28 of file Graph.hpp.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Graph()

```
hydrogen_framework::Graph::Graph (
            unsigned ver )  [inline]
```

Constructor Initialize ID to zero

Definition at line 34 of file Graph.hpp.

References whiteList.

5.9.2.2 ~Graph()

```
hydrogen_framework::Graph::~Graph ( )  [inline]
```

Destructor

Definition at line 73 of file Graph.hpp.

References graphEdges, and graphFunctions.

5.9.3 Member Function Documentation

5.9.3.1 addBranchEdges()

```
void hydrogen_framework::Graph::addBranchEdges ( )
```

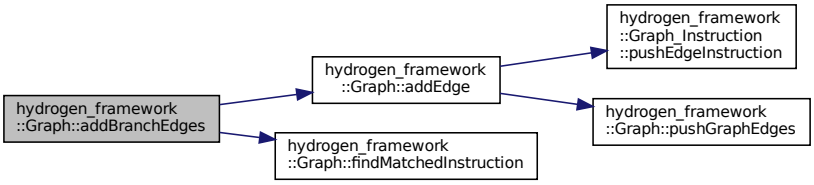
Add branch type edges for ICFG

Definition at line 84 of file Graph.cpp.

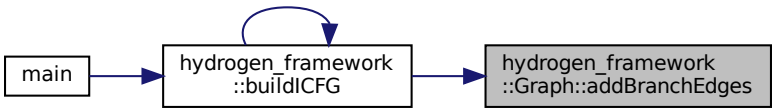
References addEdge(), findMatchedInstruction(), graphFunctions, and graphVersion.

Referenced by hydrogen_framework::buildICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.2 addEdge()

```
void hydrogen_framework::Graph::addEdge (
            Graph_Instruction * from,
            Graph_Instruction * to,
            Graph_Edge * edge )
```

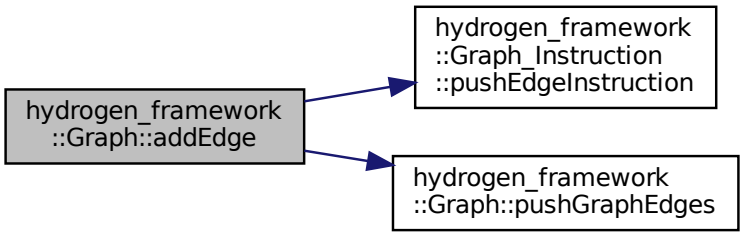
Function to add Graph_Edge to both graphEdges and corresponding Graph_Instruction

Definition at line 17 of file Graph.cpp.

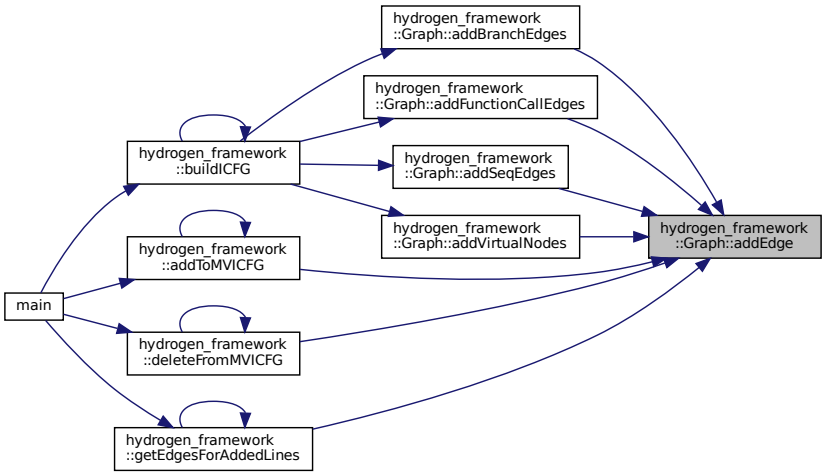
References hydrogen_framework::Graph_Instruction::pushEdgeInstruction(), and pushGraphEdges().

Referenced by addBranchEdges(), addFunctionCallEdges(), addSeqEdges(), hydrogen_framework::addToMVICFG(), addVirtualNodes(), hydrogen_framework::deleteFromMVICFG(), and hydrogen_framework::getEdgesForAddedLines().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.3 addFunctionCallEdges()

```
void hydrogen_framework::Graph::addFunctionCallEdges ( )
```

Add function call edges Call only after addVirtualNodes

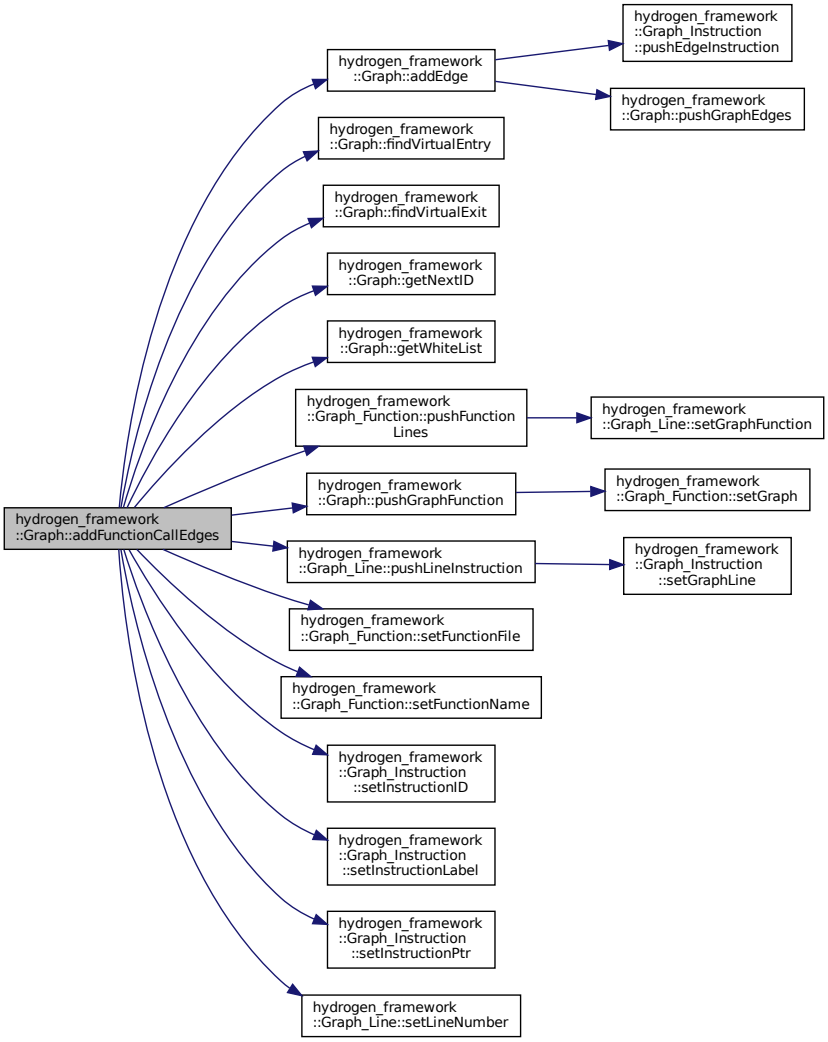
Definition at line 124 of file Graph.cpp.

References addEdge(), findVirtualEntry(), findVirtualExit(), getNextID(), getWhiteList(), graphEntryID, graphFunctions, graphVersion, hydrogen_framework::Graph_Function::pushFunctionLines(), pushGraphFunction(), hydrogen_framework::Graph_Line

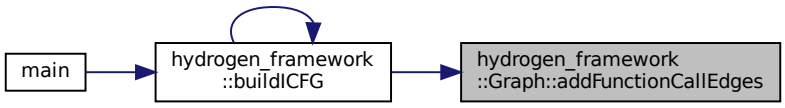
hydrogen_framework::Graph_Function::setFunctionFile(), hydrogen_framework::Graph_Function::setFunctionName(), hydrogen_framework::Graph_Instruction::setInstructionID(), hydrogen_framework::Graph_Instruction::setInstructionLabel(), hydrogen_framework::Graph_Instruction::setInstructionPtr(), and hydrogen_framework::Graph_Line::setLineNumber().

Referenced by hydrogen_framework::buildICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.4 addSeqEdges()

```
void hydrogen_framework::Graph::addSeqEdges (
            Graph_Line * line )
```

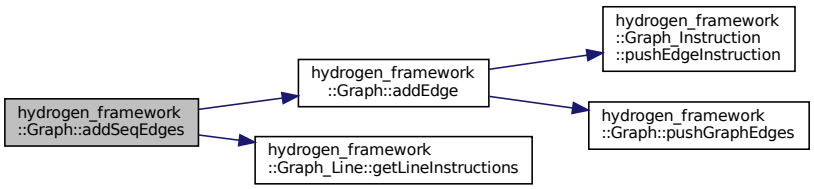
Add sequential edges for the instructions in a Graph_Line

Definition at line 23 of file Graph.cpp.

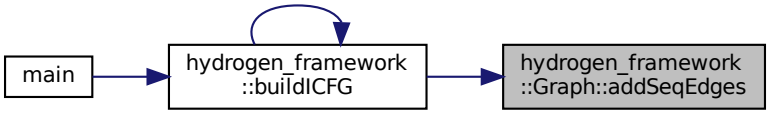
References addEdge(), hydrogen_framework::Graph_Line::getLineInstructions(), and graphVersion.

Referenced by hydrogen_framework::buildICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.5 addVirtualNodes()

```
void hydrogen_framework::Graph::addVirtualNodes (
            Graph_Function * func )
```

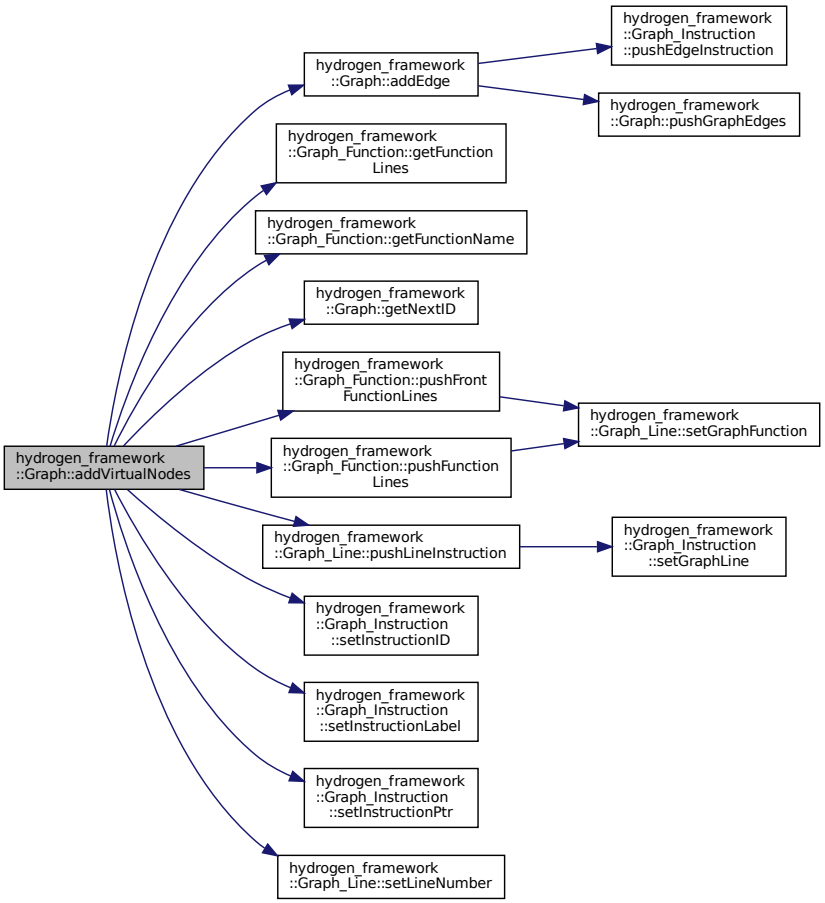
Add virtual nodes and corresponding edges to the Graph_Function

Definition at line 198 of file Graph.cpp.

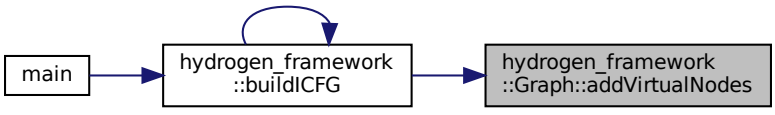
References addEdge(), hydrogen_framework::Graph_Function::getFunctionLines(), hydrogen_framework::Graph_Function::getFuncti
getNextID(), graphEntryID, graphExitID, graphVersion, hydrogen_framework::Graph_Function::pushFrontFunctionLines(),
hydrogen_framework::Graph_Function::pushFunctionLines(), hydrogen_framework::Graph_Line::pushLineInstruction(),
hydrogen_framework::Graph_Instruction::setInstructionID(), hydrogen_framework::Graph_Instruction::setInstructionLabel(),
hydrogen_framework::Graph_Instruction::setInstructionPtr(), and hydrogen_framework::Graph_Line::setLineNumber().

Referenced by hydrogen_framework::buildICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.6 findMatchedInstruction()

```
Graph_Instruction * hydrogen_framework::Graph::findMatchedInstruction (
            llvm::Instruction * matchInst )
```

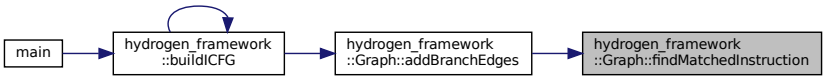
Find matching instruction in the ICFG Can return NULL if no match is found

Definition at line 41 of file Graph.cpp.

References graphFunctions.

Referenced by addBranchEdges().

Here is the caller graph for this function:



5.9.3.7 findVirtualEntry()

```
Graph_Instruction * hydrogen_framework::Graph::findVirtualEntry (
            std::string funcName )
```

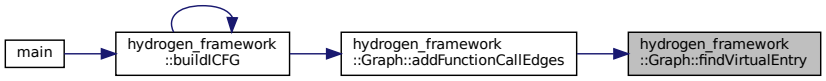
Find virtual entry for the given function name Can return NULL if no match is found

Definition at line 54 of file Graph.cpp.

References graphFunctions.

Referenced by addFunctionCallEdges().

Here is the caller graph for this function:



5.9.3.8 findVirtualExit()

```
Graph_Instruction * hydrogen_framework::Graph::findVirtualExit (
            std::string funcName )
```

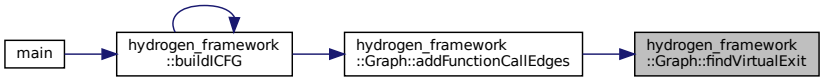
Find virtual entry for the given function name Can return NULL if no match is found

Definition at line 69 of file Graph.cpp.

References graphFunctions.

Referenced by addFunctionCallEdges().

Here is the caller graph for this function:



5.9.3.9 getGraphEdges()

```
std::list<Graph_Edge *> hydrogen_framework::Graph::getGraphEdges ( )  [inline]
```

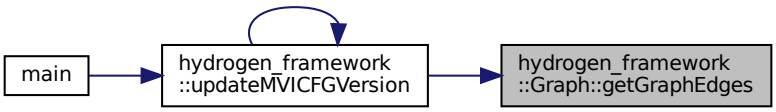
Return graphEdges

Definition at line 165 of file Graph.hpp.

References graphEdges.

Referenced by hydrogen_framework::updateMVICFGVersion().

Here is the caller graph for this function:



5.9.3.10 getGraphFunctions()

```
std::list<Graph_Function *> hydrogen_framework::Graph::getGraphFunctions ( )  [inline]
```

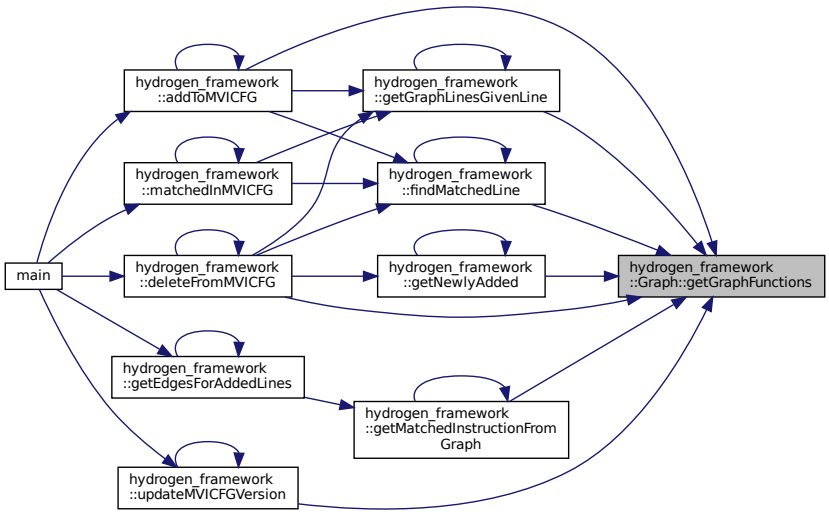
Return graphFunctions

Definition at line 155 of file Graph.hpp.

References graphFunctions.

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::findMatch
hydrogen_framework::getGraphLinesGivenLine(), hydrogen_framework::getMatchedInstructionFromGraph(),
hydrogen_framework::getNewlyAdded(), and hydrogen_framework::updateMVICFGVersion().

Here is the caller graph for this function:



5.9.3.11 getGraphVersion()

```
unsigned hydrogen_framework::Graph::getGraphVersion ( )  [inline]
```

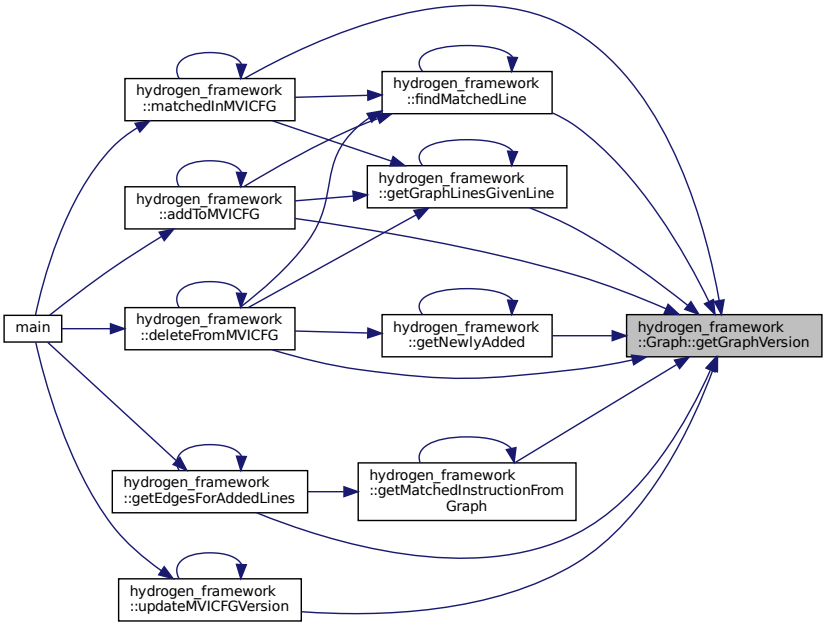
Return graphVersion

Definition at line 86 of file Graph.hpp.

References graphVersion.

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::findMatch
hydrogen_framework::getEdgesForAddedLines(), hydrogen_framework::getGraphLinesGivenLine(), hydrogen_framework::getMatche
hydrogen_framework::getNewlyAdded(), hydrogen_framework::matchedInMVICFG(), and hydrogen_framework::updateMVICFGVersi

Here is the caller graph for this function:



5.9.3.12 getNextID()

```
unsigned hydrogen_framework::Graph::getNextID ( )  [inline]
```

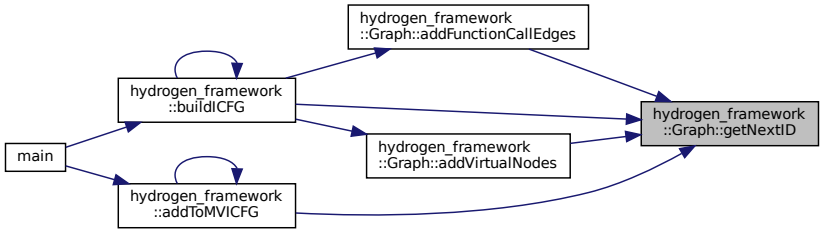
Get next ID

Definition at line 81 of file Graph.hpp.

References graphID.

Referenced by addFunctionCallEdges(), hydrogen_framework::addToMVICFG(), addVirtualNodes(), and hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.9.3.13 getWhiteList()

```
std::list<std::string> hydrogen_framework::Graph::getWhiteList ( )  [inline]
```

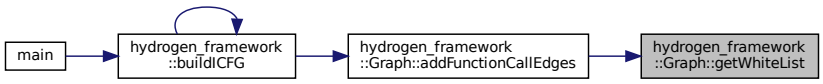
Return whiteList

Definition at line 170 of file Graph.hpp.

References whiteList.

Referenced by addFunctionCallEdges().

Here is the caller graph for this function:



5.9.3.14 isVirtualNodeLineNumber()

```
bool hydrogen_framework::Graph::isVirtualNodeLineNumber (
            unsigned lineNumber )
```

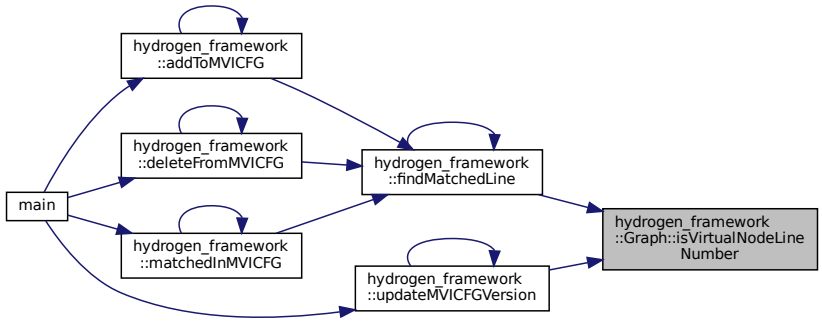
Return TRUE if it is a virtual node

Definition at line 340 of file Graph.cpp.

References graphEntryID, and graphExitID.

Referenced by hydrogen_framework::findMatchedLine(), and hydrogen_framework::updateMVICFGVersion().

Here is the caller graph for this function:



5.9.3.15 printGraph()

```
void hydrogen_framework::Graph::printGraph (
            std::string graphName )
```

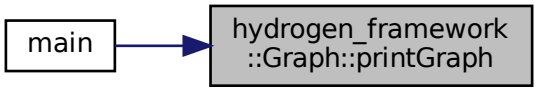
Print the graph in DOT format

Definition at line 226 of file Graph.cpp.

References graphEdges, graphFunctions, and graphVersion.

Referenced by main().

Here is the caller graph for this function:



5.9.3.16 pushGraphEdges()

```
void hydrogen_framework::Graph::pushGraphEdges (
            Graph_Edge * edge )  [inline]
```

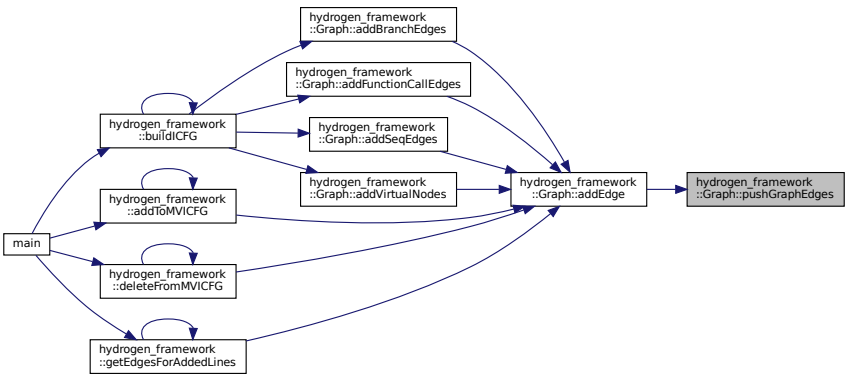
Push Graph_Edge into graphEdges

Definition at line 96 of file Graph.hpp.

References graphEdges.

Referenced by addEdge().

Here is the caller graph for this function:



5.9.3.17 pushGraphFunction()

```
void hydrogen_framework::Graph::pushGraphFunction (
            Graph_Function * func )
```

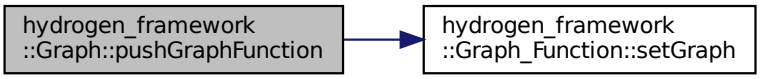
Push Graph_Function into graphFunctions

Definition at line 12 of file Graph.cpp.

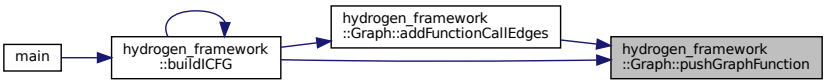
References graphFunctions, and hydrogen_framework::Graph_Function::setGraph().

Referenced by addFunctionCallEdges(), and hydrogen_framework::buildICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.18 setGraphVersion()

```
void hydrogen_framework::Graph::setGraphVersion (
            unsigned ver )  [inline]
```

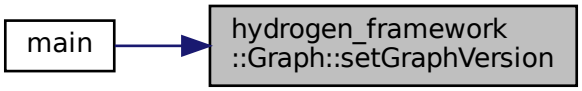
Set graphVersion

Definition at line 91 of file Graph.hpp.

References graphVersion.

Referenced by main().

Here is the caller graph for this function:



5.9.4 Field Documentation

5.9.4.1 graphEdges

```
std::list<Graph_Edge *> hydrogen_framework::Graph::graphEdges  [private]
```

Container for Edges in the graph

Definition at line 177 of file Graph.hpp.

Referenced by getGraphEdges(), printGraph(), pushGraphEdges(), and ~Graph().

5.9.4.2 graphEntryID

```
unsigned hydrogen_framework::Graph::graphEntryID  [private]
```

ID for all virtual entry Node. Set to max -1

Definition at line 175 of file Graph.hpp.

Referenced by addFunctionCallEdges(), addVirtualNodes(), and isVirtualNodeLineNumber().

5.9.4.3 graphExitID

```
unsigned hydrogen_framework::Graph::graphExitID  [private]
```

ID for all virtual exit Node. Set to max -2

Definition at line 176 of file Graph.hpp.

Referenced by addVirtualNodes(), and isVirtualNodeLineNumber().

5.9.4.4 graphFunctions

```
std::list<Graph_Function *> hydrogen_framework::Graph::graphFunctions  [private]
```

Container for function containers

Definition at line 178 of file Graph.hpp.

Referenced by addBranchEdges(), addFunctionCallEdges(), findMatchedInstruction(), findVirtualEntry(), findVirtualExit(), getGraphFunctions(), printGraph(), pushGraphFunction(), and ~Graph().

5.9.4.5 graphID

```
unsigned hydrogen_framework::Graph::graphID  [private]
```

Unique Graph ID

Definition at line 173 of file Graph.hpp.

Referenced by getNextID().

5.9.4.6 graphVersion

```
unsigned hydrogen_framework::Graph::graphVersion  [private]
```

Version of graph.

Definition at line 174 of file Graph.hpp.

Referenced by addBranchEdges(), addFunctionCallEdges(), addSeqEdges(), addVirtualNodes(), getGraphVersion(), printGraph(), and setGraphVersion().

5.9.4.7 whiteList

```
std::list<std::string> hydrogen_framework::Graph::whiteList  [private]
```

Container for white-listed functions

Definition at line 179 of file Graph.hpp.

Referenced by getWhiteList(), and Graph().

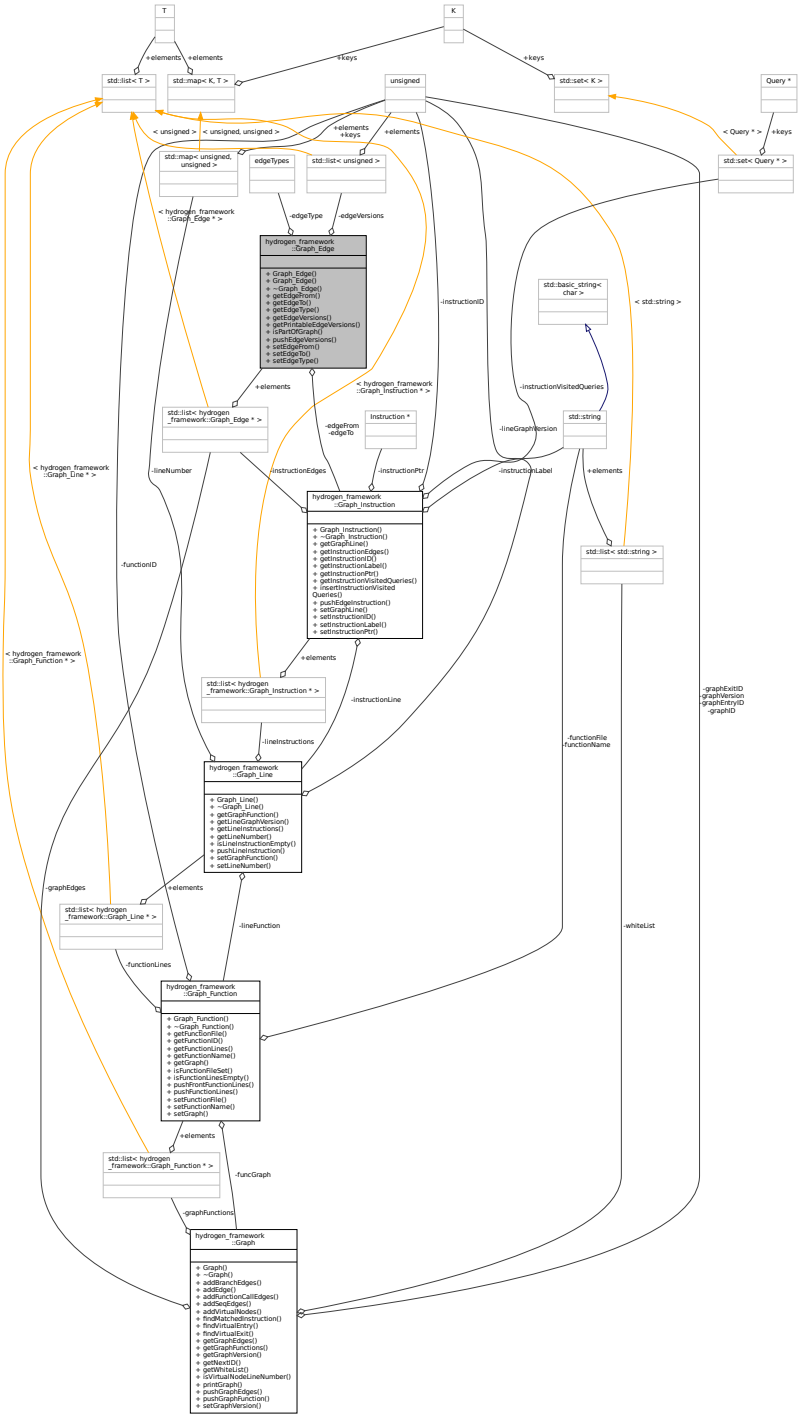
The documentation for this class was generated from the following files:

- Graph.hpp
- Graph.cpp

5.10 hydrogen_framework::Graph_Edge Class Reference

`#include <Graph_Edge.hpp>`

Collaboration diagram for hydrogen_framework::Graph_Edge:



Public Types

- enum edgeTypes {
  SEQUENTIAL, **BRANCH**, **CALL**, **EXTERNAL_CALL**,
  VIRTUAL, **MVICFG_ADD**, **MVICFG_DEL**, **ANY** }

Public Member Functions

- Graph_Edge ()
- Graph_Edge (Graph_Instruction *from, Graph_Instruction *to, edgeTypes type, unsigned ver)
- ~Graph_Edge ()
- Graph_Instruction * getEdgeFrom ()
- Graph_Instruction * getEdgeTo ()
- edgeTypes getEdgeType ()
- std::list< unsigned > getEdgeVersions ()
- std::string getPrintableEdgeVersions ()
- bool isPartOfGraph (unsigned graphVersion)
- void pushEdgeVersions (int ver)
- void setEdgeFrom (Graph_Instruction *I)
- void setEdgeTo (Graph_Instruction *I)
- void setEdgeType (edgeTypes type)

Private Attributes

- Graph_Instruction * edgeFrom
- Graph_Instruction * edgeTo
- edgeTypes edgeType
- std::list< unsigned > edgeVersions

5.10.1 Detailed Description

Graph_Edge Class: Class for storing edge information

Definition at line 18 of file Graph_Edge.hpp.

5.10.2 Member Enumeration Documentation

5.10.2.1 edgeTypes

```
enum hydrogen_framework::Graph_Edge::edgeTypes
```

Enumeration for type of edges

Definition at line 28 of file Graph_Edge.hpp.

5.10.3 Constructor & Destructor Documentation

5.10.3.1 Graph_Edge() [1/2]

```
hydrogen_framework::Graph_Edge::Graph_Edge ( )  [inline]
```

Constructor

Definition at line 23 of file Graph_Edge.hpp.

5.10.3.2 Graph_Edge() [2/2]

```
hydrogen_framework::Graph_Edge::Graph_Edge (
            Graph_Instruction * from,
            Graph_Instruction * to,
            edgeTypes type,
            unsigned ver )  [inline]
```

Alternate constructor

Definition at line 33 of file Graph_Edge.hpp.

References edgeVersions.

5.10.3.3 ~Graph_Edge()

```
hydrogen_framework::Graph_Edge::~Graph_Edge ( )  [inline]
```

Destructor

Definition at line 41 of file Graph_Edge.hpp.

References edgeVersions.

5.10.4 Member Function Documentation

5.10.4.1 getEdgeFrom()

```
Graph_Instruction* hydrogen_framework::Graph_Edge::getEdgeFrom ( )  [inline]
```

Return edgeFrom

Definition at line 66 of file Graph_Edge.hpp.

References edgeFrom.

5.10.4.2 getEdgeTo()

```
Graph_Instruction* hydrogen_framework::Graph_Edge::getEdgeTo ( )  [inline]
```

Return edgeTo

Definition at line 71 of file Graph_Edge.hpp.

References edgeTo.

5.10.4.3 getEdgeType()

```
edgeTypes hydrogen_framework::Graph_Edge::getEdgeType ( )  [inline]
```

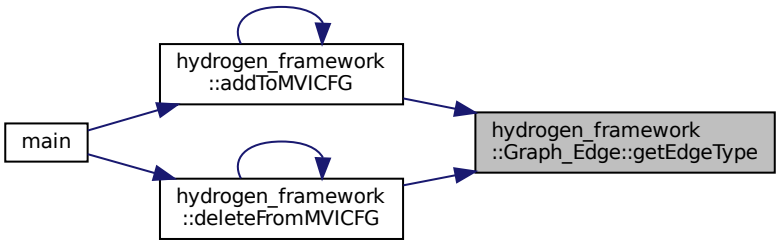
Return edgeType

Definition at line 76 of file Graph_Edge.hpp.

References edgeType.

Referenced by hydrogen_framework::addToMVICFG(), and hydrogen_framework::deleteFromMVICFG().

Here is the caller graph for this function:



5.10.4.4 getEdgeVersions()

```
std::list<unsigned> hydrogen_framework::Graph_Edge::getEdgeVersions ( )  [inline]
```

Return edgeVersions

Definition at line 81 of file Graph_Edge.hpp.

References edgeVersions.

5.10.4.5 getPrintableEdgeVersions()

```
std::string hydrogen_framework::Graph_Edge::getPrintableEdgeVersions ( )
```

Get printable edgeVersions

Definition at line 9 of file Graph_Edge.cpp.

References edgeVersions.

5.10.4.6 isPartOfGraph()

```
bool hydrogen_framework::Graph_Edge::isPartOfGraph (
            unsigned graphVersion )
```

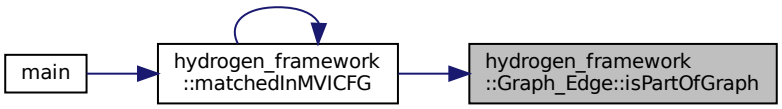
Check if the edge is already part of a given graph Version Return TRUE only if the given graphVersion is contained in edgeVersions

Definition at line 18 of file Graph_Edge.cpp.

References edgeVersions.

Referenced by hydrogen_framework::matchedInMVICFG().

Here is the caller graph for this function:



5.10.4.7 pushEdgeVersions()

```
void hydrogen_framework::Graph_Edge::pushEdgeVersions (
            int ver )  [inline]
```

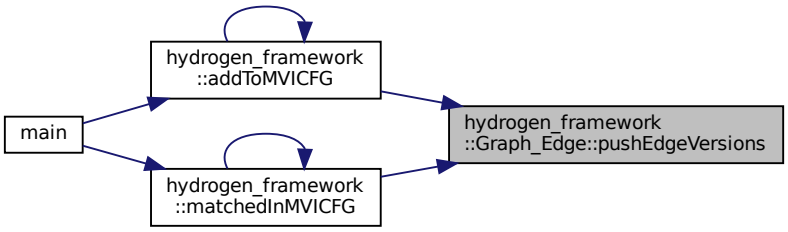
Push version to back of edgeVersions

Definition at line 61 of file Graph_Edge.hpp.

References edgeVersions.

Referenced by hydrogen_framework::addToMVICFG(), and hydrogen_framework::matchedInMVICFG().

Here is the caller graph for this function:



5.10.4.8 setEdgeFrom()

```
void hydrogen_framework::Graph_Edge::setEdgeFrom (
            Graph_Instruction * I )  [inline]
```

Set edgeFrom

Definition at line 46 of file Graph_Edge.hpp.

References edgeFrom.

5.10.4.9 setEdgeTo()

```
void hydrogen_framework::Graph_Edge::setEdgeTo (
            Graph_Instruction * I )  [inline]
```

Set edgeTo

Definition at line 51 of file Graph_Edge.hpp.

References edgeTo.

5.10.4.10 setEdgeType()

```
void hydrogen_framework::Graph_Edge::setEdgeType (
            edgeTypes type )  [inline]
```

Set edgeType

Definition at line 56 of file Graph_Edge.hpp.

References edgeType.

5.10.5 Field Documentation

5.10.5.1 edgeFrom

```
Graph_Instruction* hydrogen_framework::Graph_Edge::edgeFrom  [private]
```

From Instruction

Definition at line 95 of file Graph_Edge.hpp.

Referenced by getEdgeFrom(), and setEdgeFrom().

5.10.5.2 edgeTo

```
Graph_Instruction* hydrogen_framework::Graph_Edge::edgeTo  [private]
```

To Instruction

Definition at line 96 of file Graph_Edge.hpp.

Referenced by getEdgeTo(), and setEdgeTo().

5.10.5.3 edgeType

```
edgeTypes hydrogen_framework::Graph_Edge::edgeType  [private]
```

Edge Type

Definition at line 97 of file Graph_Edge.hpp.

Referenced by getEdgeType(), and setEdgeType().

5.10.5.4 edgeVersions

```
std::list<unsigned> hydrogen_framework::Graph_Edge::edgeVersions  [private]
```

Container to store edge's versions

Definition at line 98 of file Graph_Edge.hpp.

Referenced by getEdgeVersions(), getPrintableEdgeVersions(), Graph_Edge(), isPartOfGraph(), pushEdgeVersions(), and ~Graph_Edge().

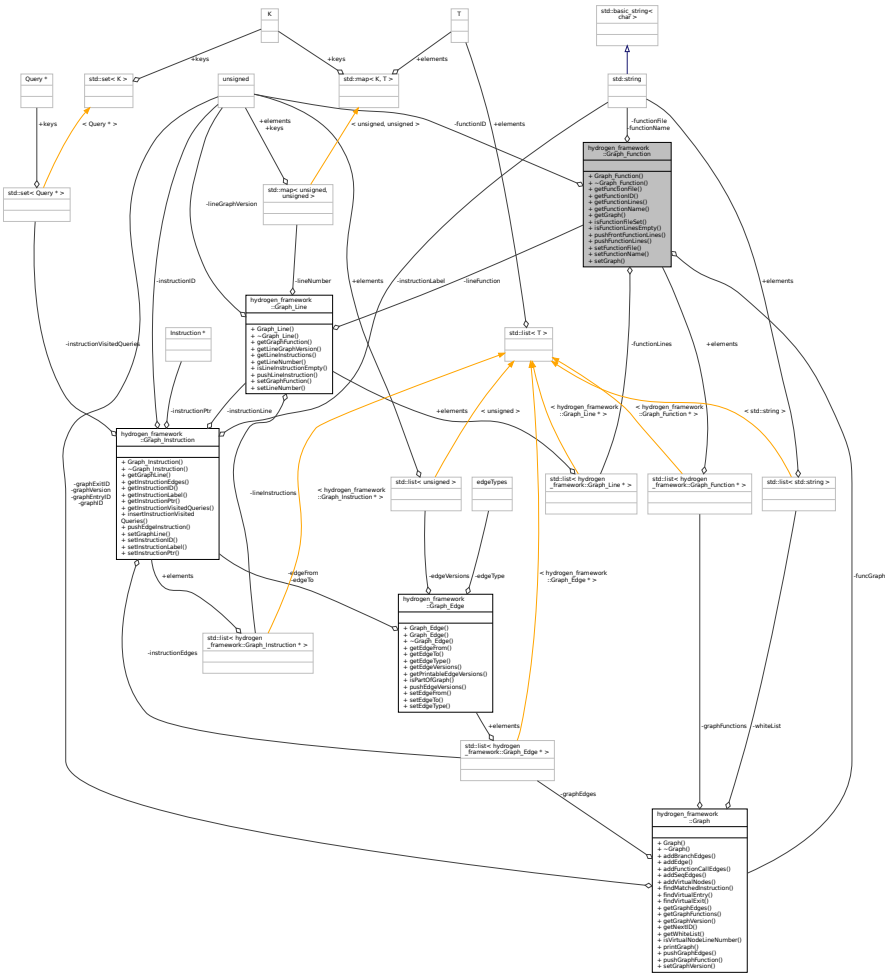
The documentation for this class was generated from the following files:

- Graph_Edge.hpp
- Graph_Edge.cpp

5.11 hydrogen_framework::Graph_Function Class Reference

`#include <Graph_Function.hpp>`

Collaboration diagram for hydrogen_framework::Graph_Function:



Public Member Functions

- Graph_Function (unsigned id)
- ~Graph_Function ()
- std::string getFunctionFile ()
- unsigned getFunctionID ()
- std::list< Graph_Line * > getFunctionLines ()
- std::string getFunctionName ()
- Graph * getGraph ()
- bool isFunctionFileSet ()
- bool isFunctionLinesEmpty ()
- void pushFrontFunctionLines (Graph_Line *line)
- void pushFunctionLines (Graph_Line *line)
- void setFunctionFile (std::string name)
- void setFunctionName (std::string name)
- void setGraph (Graph *graph)

Private Attributes

- Graph * funcGraph
- std::string functionFile
- unsigned functionID
- std::list< Graph_Line * > functionLines
- std::string functionName

5.11.1 Detailed Description

Graph_Function Class: Container for storing source line belonging to a function

Definition at line 19 of file Graph_Function.hpp.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Graph_Function()

```
hydrogen_framework::Graph_Function::Graph_Function (
            unsigned id )  [inline]
```

Constructor

Definition at line 24 of file Graph_Function.hpp.

5.11.2.2 ~Graph_Function()

```
hydrogen_framework::Graph_Function::~Graph_Function ( )  [inline]
```

Destructor

Definition at line 29 of file Graph_Function.hpp.

References functionLines.

5.11.3 Member Function Documentation

5.11.3.1 getFunctionFile()

```
std::string hydrogen_framework::Graph_Function::getFunctionFile ( )  [inline]
```

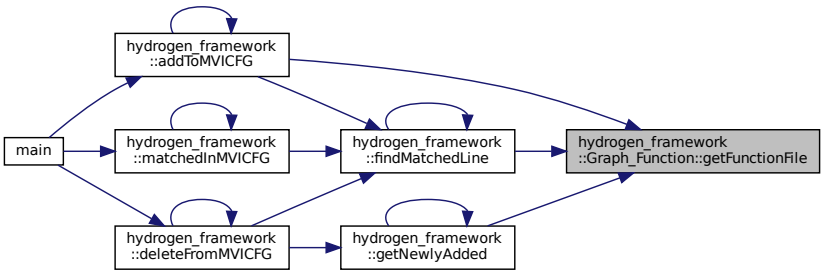
Return functionFile

Definition at line 79 of file Graph_Function.hpp.

References functionFile.

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::findMatchedLine(), and hydrogen_framework::getNewly

Here is the caller graph for this function:



5.11.3.2 getFunctionID()

```
unsigned hydrogen_framework::Graph_Function::getFunctionID ( )  [inline]
```

Return functionID

Definition at line 74 of file Graph_Function.hpp.

References functionID.

5.11.3.3 getFunctionLines()

```
std::list<Graph_Line *> hydrogen_framework::Graph_Function::getFunctionLines ( )  [inline]
```

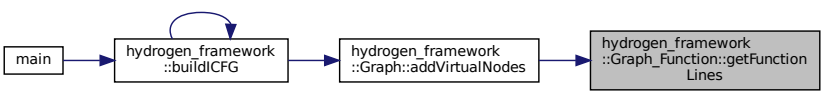
Return functionLines

Definition at line 64 of file Graph_Function.hpp.

References functionLines.

Referenced by hydrogen_framework::Graph::addVirtualNodes().

Here is the caller graph for this function:



5.11.3.4 getFunctionName()

```
std::string hydrogen_framework::Graph_Function::getFunctionName ( )  [inline]
```

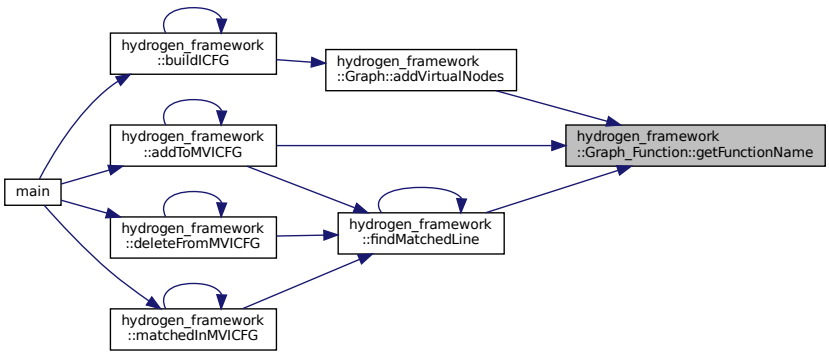
Return funcName

Definition at line 69 of file Graph_Function.hpp.

References functionName.

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::Graph::addVirtualNodes(), and hydrogen_framework::findMatchedLine().

Here is the caller graph for this function:



5.11.3.5 getGraph()

```
Graph* hydrogen_framework::Graph_Function::getGraph ( )  [inline]
```

Return pointer to encompassing Graph

Definition at line 89 of file Graph_Function.hpp.

References funcGraph.

5.11.3.6 isFunctionFileSet()

```
bool hydrogen_framework::Graph_Function::isFunctionFileSet ( )  [inline]
```

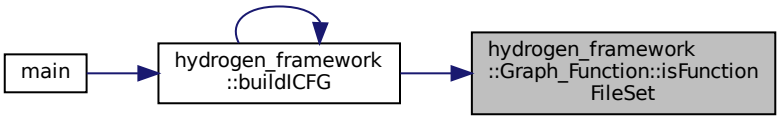
Return true if functionFile is not empty

Definition at line 44 of file Graph_Function.hpp.

References functionFile.

Referenced by hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.11.3.7 isFunctionLinesEmpty()

```
bool hydrogen_framework::Graph_Function::isFunctionLinesEmpty ( )  [inline]
```

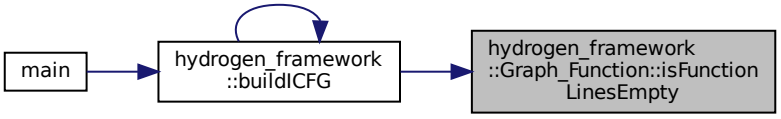
Return true if functionLines is empty

Definition at line 59 of file Graph_Function.hpp.

References functionLines.

Referenced by hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.11.3.8 pushFrontFunctionLines()

```
void hydrogen_framework::Graph_Function::pushFrontFunctionLines (
            Graph_Line * line )
```

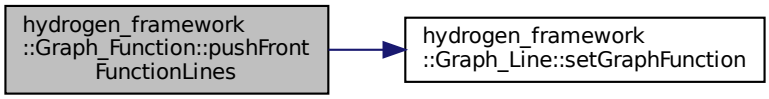
Push Graph_Line at the front of the functionLines list. Only used for Virtual node

Definition at line 15 of file Graph_Function.cpp.

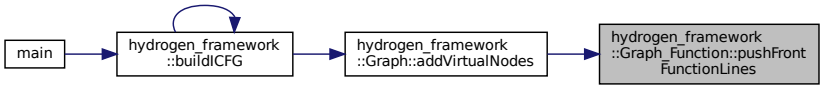
References functionLines, and hydrogen_framework::Graph_Line::setGraphFunction().

Referenced by hydrogen_framework::Graph::addVirtualNodes().

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.9 pushFunctionLines()

```
void hydrogen_framework::Graph_Function::pushFunctionLines (
            Graph_Line * line )
```

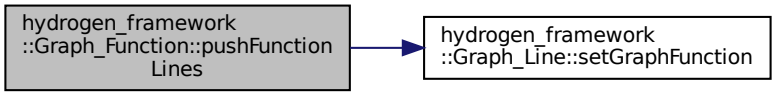
Push Graph_Line at the back of the functionLines list

Definition at line 10 of file Graph_Function.cpp.

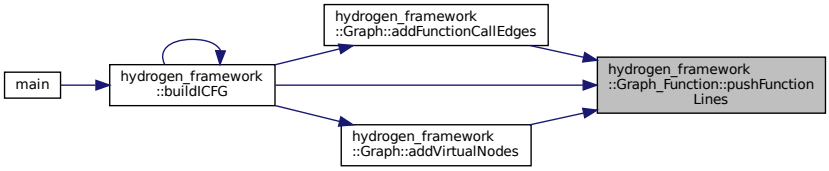
References functionLines, and hydrogen_framework::Graph_Line::setGraphFunction().

Referenced by hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::Graph::addVirtualNodes(), and hydrogen_framework::buildICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.10 setFunctionFile()

```
void hydrogen_framework::Graph_Function::setFunctionFile (
            std::string name )  [inline]
```

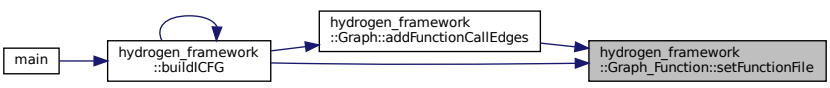
Set functionFile

Definition at line 39 of file Graph_Function.hpp.

References functionFile.

Referenced by hydrogen_framework::Graph::addFunctionCallEdges(), and hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.11.3.11 setFunctionName()

```
void hydrogen_framework::Graph_Function::setFunctionName (
            std::string name )  [inline]
```

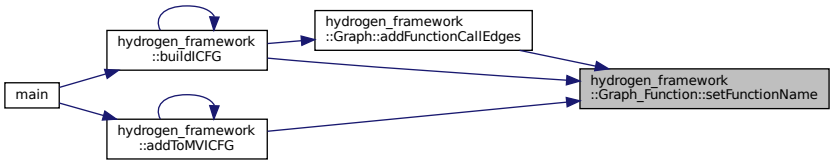
Set functionName

Definition at line 34 of file Graph_Function.hpp.

References functionName.

Referenced by hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::addToMVICFG(), and hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.11.3.12 setGraph()

```
void hydrogen_framework::Graph_Function::setGraph (
            Graph * graph )  [inline]
```

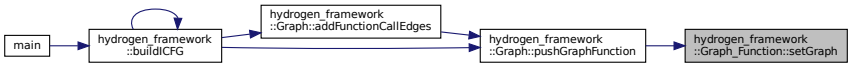
Set pointer to encompassing Graph

Definition at line 84 of file Graph_Function.hpp.

References funcGraph.

Referenced by hydrogen_framework::Graph::pushGraphFunction().

Here is the caller graph for this function:



5.11.4 Field Documentation

5.11.4.1 funcGraph

```
Graph* hydrogen_framework::Graph_Function::funcGraph  [private]
```

Points to the Graph that encompasses this

Definition at line 96 of file Graph_Function.hpp.

Referenced by getGraph(), and setGraph().

5.11.4.2 functionFile

```
std::string hydrogen_framework::Graph_Function::functionFile  [private]
```

Name of the file in which the function resides

Definition at line 94 of file Graph_Function.hpp.

Referenced by getFunctionFile(), isFunctionFileSet(), and setFunctionFile().

5.11.4.3 functionID

```
unsigned hydrogen_framework::Graph_Function::functionID  [private]
```

Function Container ID

Definition at line 92 of file Graph_Function.hpp.

Referenced by getFunctionID().

5.11.4.4 functionLines

```
std::list<Graph_Line *> hydrogen_framework::Graph_Function::functionLines  [private]
```

Container for lines in the function

Definition at line 95 of file Graph_Function.hpp.

Referenced by getFunctionLines(), isFunctionLinesEmpty(), pushFrontFunctionLines(), pushFunctionLines(), and ~Graph_Function().

5.11.4.5 functionName

```
std::string hydrogen_framework::Graph_Function::functionName  [private]
```

Name of the function

Definition at line 93 of file Graph_Function.hpp.

Referenced by getFunctionName(), and setFunctionName().

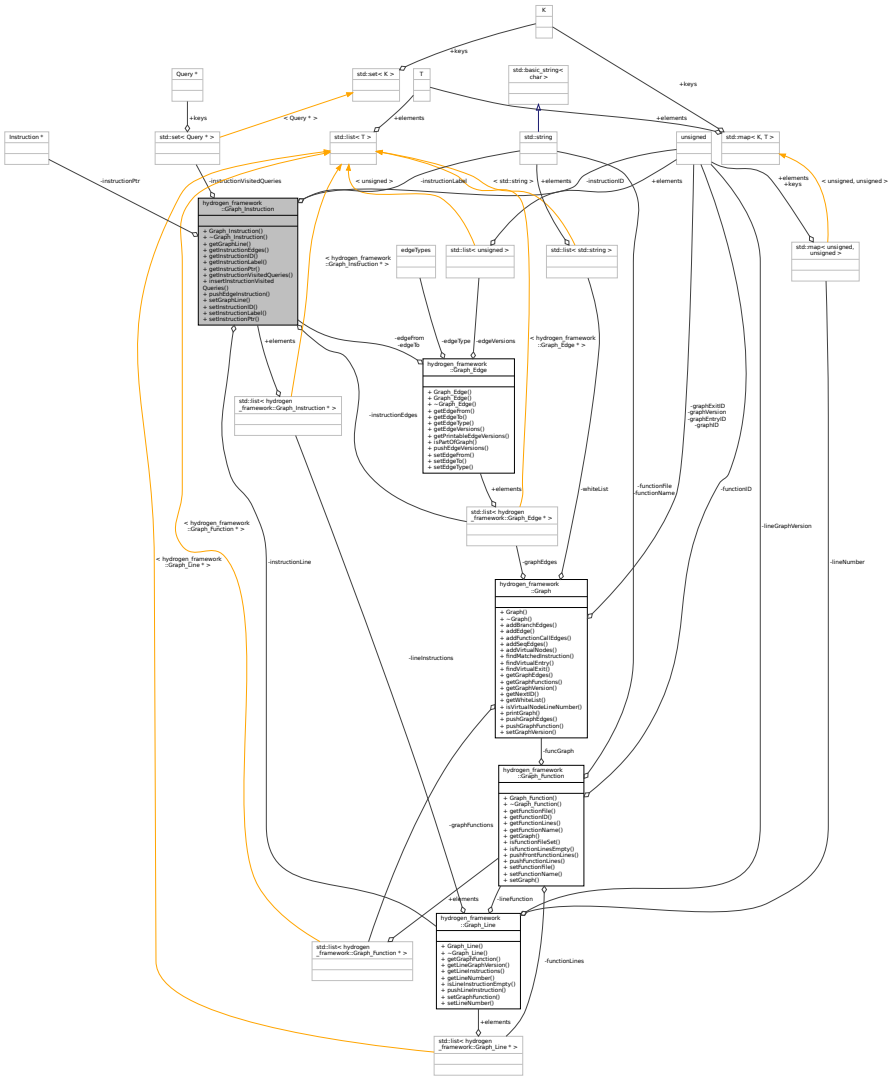
The documentation for this class was generated from the following files:

- Graph_Function.hpp
- Graph_Function.cpp

5.12 hydrogen_framework::Graph_Instruction Class Reference

```
#include <Graph_Instruction.hpp>
```

Collaboration diagram for hydrogen_framework::Graph_Instruction:



Public Member Functions

- Graph_Instruction ()
- ~Graph_Instruction ()
- Graph_Line * getGraphLine ()
- std::list< Graph_Edge * > getInstructionEdges ()
- unsigned getInstructionID ()
- std::string getInstructionLabel ()
- llvm::Instruction * getInstructionPtr ()
- std::set< Query * > getInstructionVisitedQueries ()
- void insertInstructionVisitedQueries (Query *q)
- void pushEdgeInstruction (Graph_Edge *edge)
- void setGraphLine (Graph_Line *line)
- void setInstructionID (unsigned ID)
- void setInstructionLabel (std::string label)
- void setInstructionPtr (llvm::Instruction *I)

Private Attributes

- std::list< Graph_Edge * > instructionEdges
- unsigned instructionID
- std::string instructionLabel
- Graph_Line * instructionLine
- llvm::Instruction * instructionPtr
- std::set< Query * > instructionVisitedQueries

5.12.1 Detailed Description

Graph_Instruction Class: To store individual LLVM instructions

Definition at line 21 of file Graph_Instruction.hpp.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Graph_Instruction()

```
hydrogen_framework::Graph_Instruction::Graph_Instruction ( )  [inline]
```

Constructor

Definition at line 26 of file Graph_Instruction.hpp.

5.12.2.2 ~Graph_Instruction()

```
hydrogen_framework::Graph_Instruction::~Graph_Instruction ( )  [inline]
```

Destructor

Definition at line 31 of file Graph_Instruction.hpp.

5.12.3 Member Function Documentation

5.12.3.1 getGraphLine()

```
Graph_Line* hydrogen_framework::Graph_Instruction::getGraphLine ( )  [inline]
```

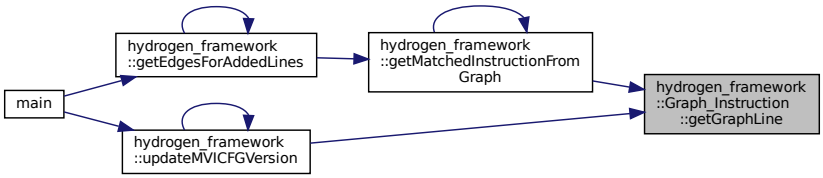
Return pointer to encompassing Graph_Line

Definition at line 82 of file Graph_Instruction.hpp.

References instructionLine.

Referenced by hydrogen_framework::getMatchedInstructionFromGraph(), and hydrogen_framework::updateMVICFGVersion().

Here is the caller graph for this function:



5.12.3.2 getInstructionEdges()

```
std::list<Graph_Edge *> hydrogen_framework::Graph_Instruction::getInstructionEdges ( )  [inline]
```

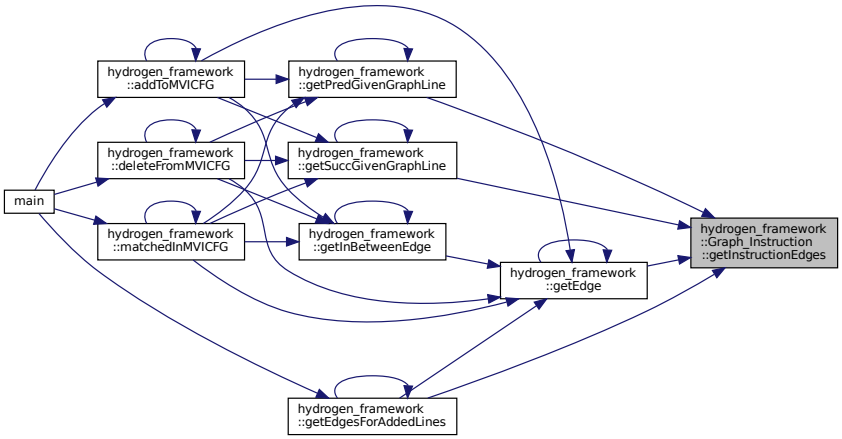
Return instructionEdges

Definition at line 72 of file Graph_Instruction.hpp.

References instructionEdges.

Referenced by hydrogen_framework::getEdge(), hydrogen_framework::getEdgesForAddedLines(), hydrogen_framework::getPredGive and hydrogen_framework::getSuccGivenGraphLine().

Here is the caller graph for this function:



5.12.3.3 getInstructionID()

```
unsigned hydrogen_framework::Graph_Instruction::getInstructionID ( )  [inline]
```

Return instructionID

Definition at line 61 of file Graph_Instruction.hpp.

References instructionID.

5.12.3.4 getInstructionLabel()

```
std::string hydrogen_framework::Graph_Instruction::getInstructionLabel ( )  [inline]
```

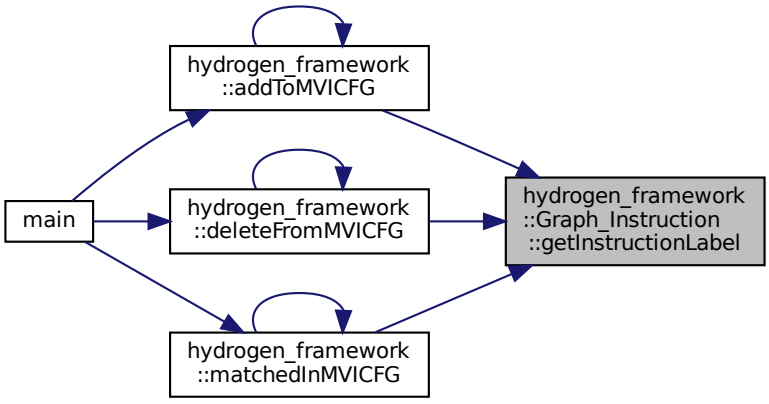
Return instructionLabel

Definition at line 56 of file Graph_Instruction.hpp.

References instructionLabel.

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), and hydrogen_framework::matchedInMVICFG().

Here is the caller graph for this function:



5.12.3.5 getInstructionPtr()

```
llvm::Instruction* hydrogen_framework::Graph_Instruction::getInstructionPtr ( )  [inline]
```

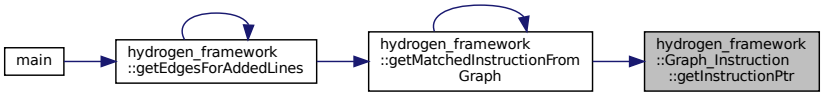
Get instructionPtr Can return NULL

Definition at line 67 of file Graph_Instruction.hpp.

References instructionPtr.

Referenced by hydrogen_framework::getMatchedInstructionFromGraph().

Here is the caller graph for this function:



5.12.3.6 getInstructionVisitedQueries()

```
std::set<Query *> hydrogen_framework::Graph_Instruction::getInstructionVisitedQueries ( )
[inline]
```

Return instructionVisitedQueries

Definition at line 87 of file Graph_Instruction.hpp.

References instructionVisitedQueries.

5.12.3.7 insertInstructionVisitedQueries()

```
void hydrogen_framework::Graph_Instruction::insertInstructionVisitedQueries (
            Query * q )  [inline]
```

Insert query as pointer into instructionVisitedQueries

Definition at line 92 of file Graph_Instruction.hpp.

References instructionVisitedQueries.

5.12.3.8 pushEdgeInstruction()

```
void hydrogen_framework::Graph_Instruction::pushEdgeInstruction (
            Graph_Edge * edge )  [inline]
```

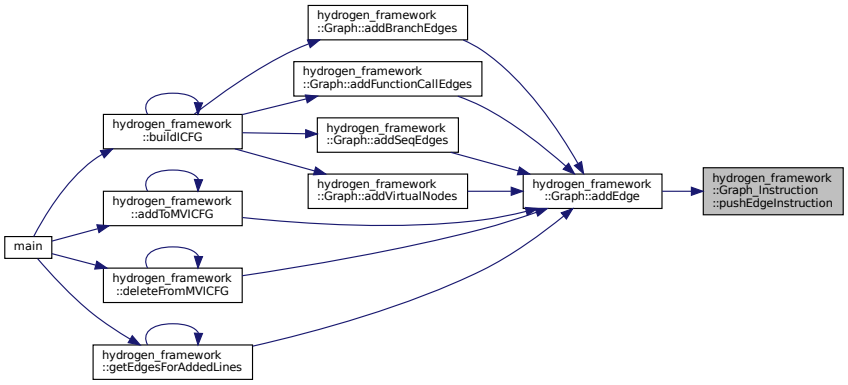
Push Graph_Edge into instructionEdges list

Definition at line 51 of file Graph_Instruction.hpp.

References instructionEdges.

Referenced by hydrogen_framework::Graph::addEdge().

Here is the caller graph for this function:



5.12.3.9 setGraphLine()

```
void hydrogen_framework::Graph_Instruction::setGraphLine (
            Graph_Line * line )  [inline]
```

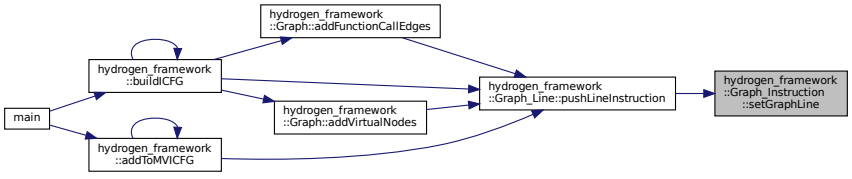
Set pointer to encompassing Graph_Line

Definition at line 77 of file Graph_Instruction.hpp.

References instructionLine.

Referenced by hydrogen_framework::Graph_Line::pushLineInstruction().

Here is the caller graph for this function:



5.12.3.10 setInstructionID()

```
void hydrogen_framework::Graph_Instruction::setInstructionID (
            unsigned ID )  [inline]
```

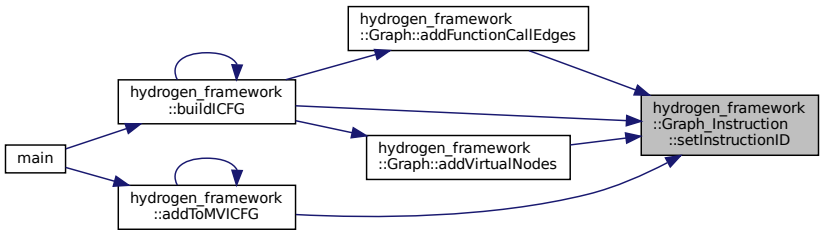
Set instructionID

Definition at line 36 of file Graph_Instruction.hpp.

References instructionID.

Referenced by hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::addToMVICFG(), hydrogen_framework::Graph::addVirtualNodes(), and hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.12.3.11 setInstructionLabel()

```
void hydrogen_framework::Graph_Instruction::setInstructionLabel (
            std::string label )  [inline]
```

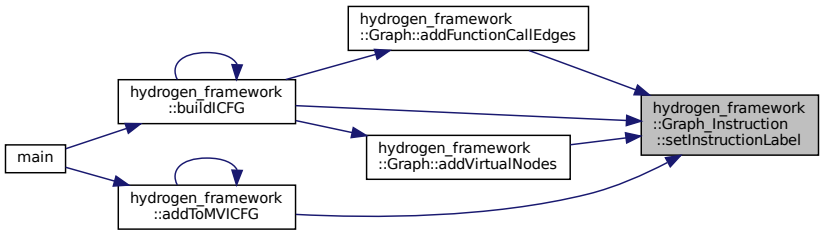
Set instructionLabel

Definition at line 41 of file Graph_Instruction.hpp.

References instructionLabel.

Referenced by hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::addToMVICFG(), hydrogen_framework::Graph::addVirtualNodes(), and hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.12.3.12 setInstructionPtr()

```
void hydrogen_framework::Graph_Instruction::setInstructionPtr (
            llvm::Instruction * I )  [inline]
```

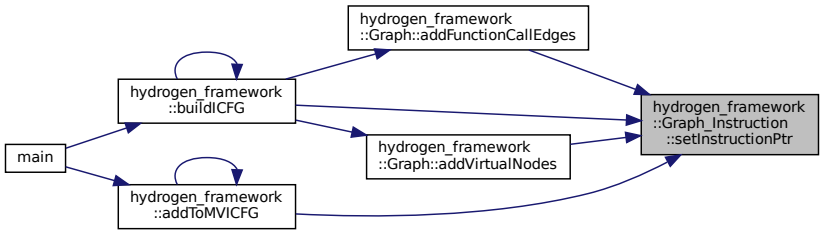
Set instructionPtr

Definition at line 46 of file Graph_Instruction.hpp.

References instructionPtr.

Referenced by hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::addToMVICFG(), hydrogen_framework::Graph::addVirtualNodes(), and hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.12.4 Field Documentation

5.12.4.1 instructionEdges

```
std::list<Graph_Edge *> hydrogen_framework::Graph_Instruction::instructionEdges  [private]
```

Container for edges in the instruction

Definition at line 98 of file Graph_Instruction.hpp.

Referenced by getInstructionEdges(), and pushEdgeInstruction().

5.12.4.2 instructionID

```
unsigned hydrogen_framework::Graph_Instruction::instructionID  [private]
```

Instruction ID

Definition at line 95 of file Graph_Instruction.hpp.

Referenced by getInstructionID(), and setInstructionID().

5.12.4.3 instructionLabel

```
std::string hydrogen_framework::Graph_Instruction::instructionLabel  [private]
```

Instruction label or text

Definition at line 96 of file Graph_Instruction.hpp.

Referenced by getInstructionLabel(), and setInstructionLabel().

5.12.4.4 instructionLine

```
Graph_Line* hydrogen_framework::Graph_Instruction::instructionLine  [private]
```

Points to the Graph_Line that encompasses this

Definition at line 99 of file Graph_Instruction.hpp.

Referenced by getGraphLine(), and setGraphLine().

5.12.4.5 instructionPtr

```
llvm::Instruction* hydrogen_framework::Graph_Instruction::instructionPtr  [private]
```

Instruction LLVM Pointer

Definition at line 97 of file Graph_Instruction.hpp.

Referenced by getInstructionPtr(), and setInstructionPtr().

5.12.4.6 instructionVisitedQueries

```
std::set<Query *> hydrogen_framework::Graph_Instruction::instructionVisitedQueries  [private]
```

Container for Queries that have visited this

Definition at line 100 of file Graph_Instruction.hpp.

Referenced by getInstructionVisitedQueries(), and insertInstructionVisitedQueries().

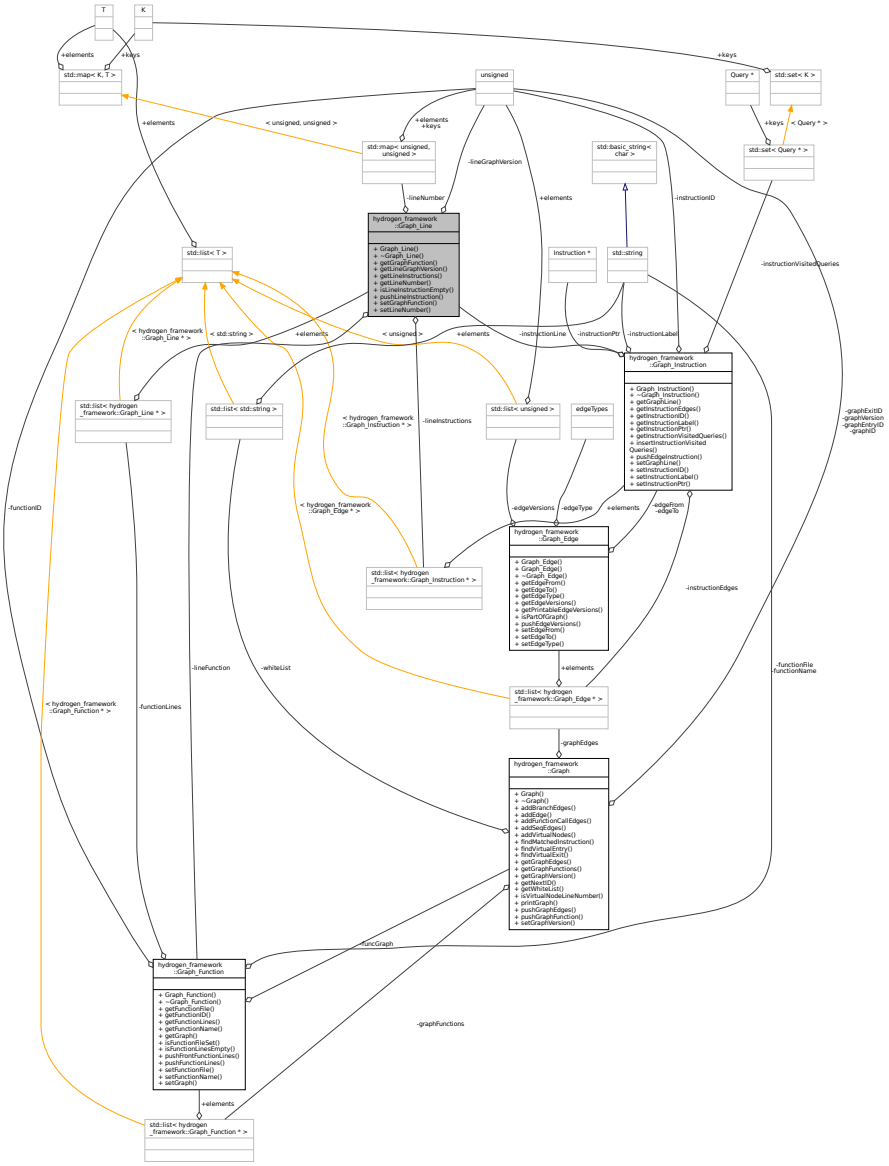
The documentation for this class was generated from the following file:

- Graph_Instruction.hpp

5.13 hydrogen_framework::Graph_Line Class Reference

`#include <Graph_Line.hpp>`

Collaboration diagram for hydrogen_framework::Graph_Line:



Public Member Functions

- Graph_Line (unsigned Version)
- ∼Graph_Line ()
- Graph_Function ∗ getGraphFunction ()
- unsigned getLineGraphVersion ()
- std::list< Graph_Instruction ∗ > getLineInstructions ()
- unsigned getLineNumber (unsigned Version)
- bool isLineInstructionEmpty ()
- void pushLineInstruction (Graph_Instruction ∗inst)
- void setGraphFunction (Graph_Function ∗func)
- void setLineNumber (unsigned Version, unsigned line)

Private Attributes

- Graph_Function * lineFunction
- unsigned lineGraphVersion
- std::list< Graph_Instruction * > lineInstructions
- std::map< unsigned, unsigned > lineNumber

5.13.1 Detailed Description

Graph_Line Class: Container for storing LLVM instruction belonging to a line

Definition at line 19 of file Graph_Line.hpp.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Graph_Line()

```
hydrogen_framework::Graph_Line::Graph_Line (
            unsigned Version )  [inline]
```

Constructor

Definition at line 24 of file Graph_Line.hpp.

5.13.2.2 ~Graph_Line()

```
hydrogen_framework::Graph_Line::~Graph_Line ( )  [inline]
```

Destructor

Definition at line 29 of file Graph_Line.hpp.

References lineInstructions.

5.13.3 Member Function Documentation

5.13.3.1 getGraphFunction()

```
Graph_Function* hydrogen_framework::Graph_Line::getGraphFunction ( )  [inline]
```

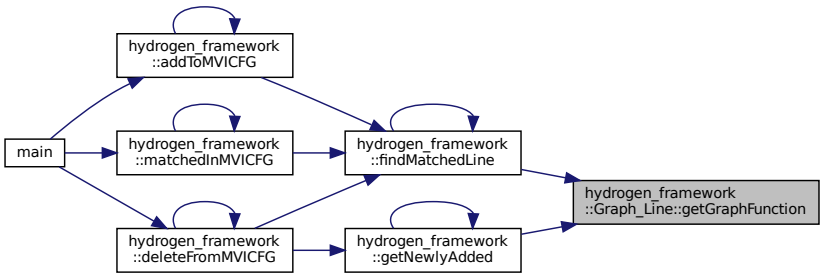
Return pointer to encompassing Graph_Line

Definition at line 65 of file Graph_Line.hpp.

References lineFunction.

Referenced by hydrogen_framework::findMatchedLine(), and hydrogen_framework::getNewlyAdded().

Here is the caller graph for this function:



5.13.3.2 getLineGraphVersion()

```
unsigned hydrogen_framework::Graph_Line::getLineGraphVersion ( )  [inline]
```

Return lineGraphVersion

Definition at line 70 of file Graph_Line.hpp.

References lineGraphVersion.

5.13.3.3 getLineInstructions()

```
std::list<Graph_Instruction *> hydrogen_framework::Graph_Line::getLineInstructions ( )  [inline]
```

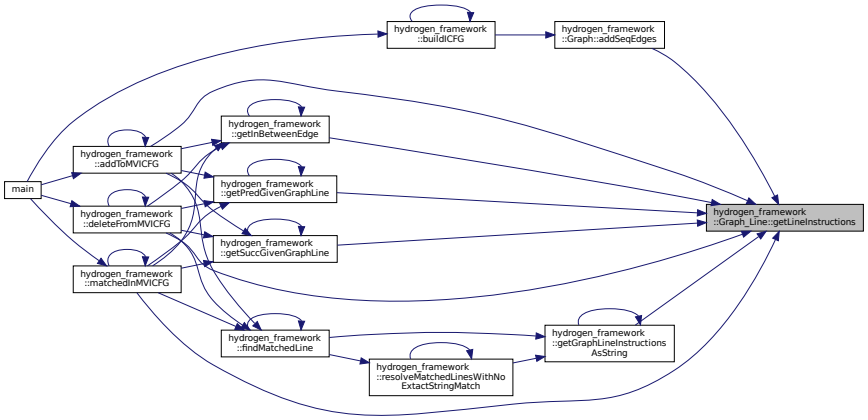
Return lineInstructions

Definition at line 55 of file Graph_Line.hpp.

References lineInstructions.

Referenced by hydrogen_framework::Graph::addSeqEdges(), hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFr
hydrogen_framework::getGraphLineInstructionsAsString(), hydrogen_framework::getInBetweenEdge(), hydrogen_framework::getPre
hydrogen_framework::getSuccGivenGraphLine(), and hydrogen_framework::matchedInMVICFG().

Here is the caller graph for this function:



5.13.3.4 getLineNumber()

```
unsigned hydrogen_framework::Graph_Line::getLineNumber (
            unsigned Version )
```

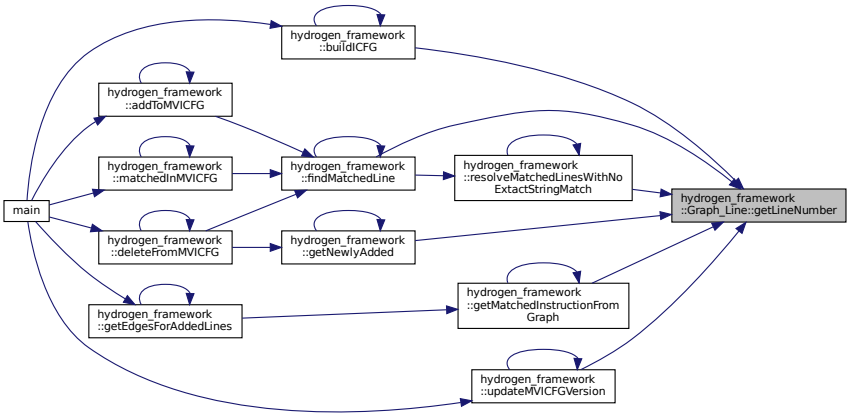
Get lineNumber given a version Returns zero if no mapping found

Definition at line 20 of file Graph_Line.cpp.

References lineNumber.

Referenced by hydrogen_framework::buildICFG(), hydrogen_framework::findMatchedLine(), hydrogen_framework::getMatchedInstruc hydrogen_framework::getNewlyAdded(), hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch(), and hydrogen_framework::updateMVICFGVersion().

Here is the caller graph for this function:



5.13.3.5 isLineInstructionEmpty()

```
bool hydrogen_framework::Graph_Line::isLineInstructionEmpty ( )  [inline]
```

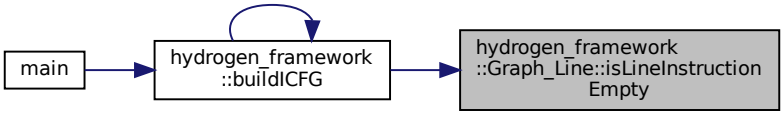
Return true if lineInstructions is empty

Definition at line 45 of file Graph_Line.hpp.

References lineInstructions.

Referenced by hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.13.3.6 pushLineInstruction()

```
void hydrogen_framework::Graph_Line::pushLineInstruction (
            Graph_Instruction * inst )
```

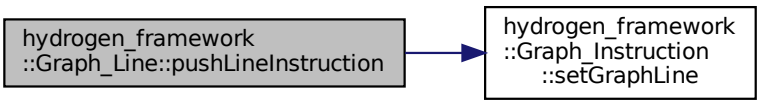
Push the Graph_Instruction at the back of the list

Definition at line 15 of file Graph_Line.cpp.

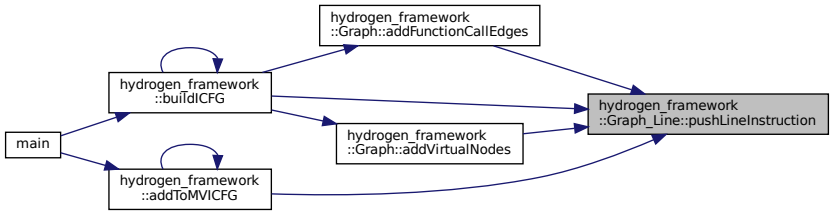
References lineInstructions, and hydrogen_framework::Graph_Instruction::setGraphLine().

Referenced by hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::addToMVICFG(), hydrogen_framework::Graph::addVirtualNodes(), and hydrogen_framework::buildICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.3.7 setGraphFunction()

```
void hydrogen_framework::Graph_Line::setGraphFunction (
            Graph_Function * func )  [inline]
```

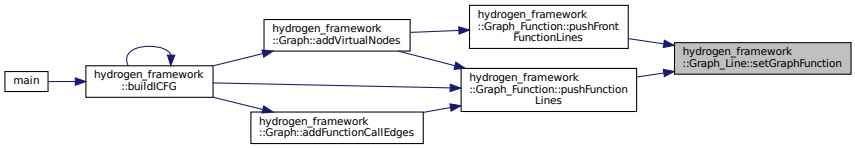
Set pointer to encompassing Graph_Function

Definition at line 60 of file Graph_Line.hpp.

References lineFunction.

Referenced by hydrogen_framework::Graph_Function::pushFrontFunctionLines(), and hydrogen_framework::Graph_Function::pushF

Here is the caller graph for this function:



5.13.3.8 setLineNumber()

```
void hydrogen_framework::Graph_Line::setLineNumber (
            unsigned Version,
            unsigned line )
```

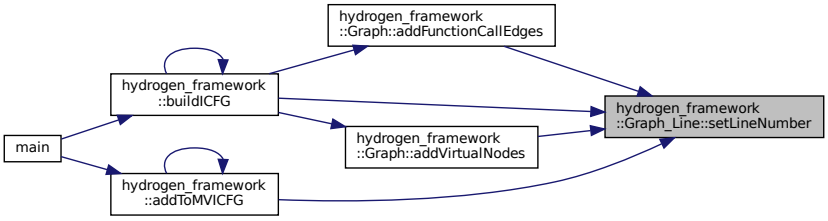
Set lineNumber

Definition at line 11 of file Graph_Line.cpp.

References lineNumber.

Referenced by hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::addToMVICFG(), hydrogen_framework::Graph::addVirtualNodes(), and hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.13.4 Field Documentation

5.13.4.1 lineFunction

```
Graph_Function* hydrogen_framework::Graph_Line::lineFunction  [private]
```

Points to the Graph_Function that encompasses this

Definition at line 75 of file Graph_Line.hpp.

Referenced by getGraphFunction(), and setGraphFunction().

5.13.4.2 lineGraphVersion

```
unsigned hydrogen_framework::Graph_Line::lineGraphVersion  [private]
```

The graph version in which this line was introduced

Definition at line 76 of file Graph_Line.hpp.

Referenced by getLineGraphVersion().

5.13.4.3 lineInstructions

```
std::list<Graph_Instruction *> hydrogen_framework::Graph_Line::lineInstructions  [private]
```

Container for instruction in the line

Definition at line 74 of file Graph_Line.hpp.

Referenced by getLineInstructions(), isLineInstructionEmpty(), pushLineInstruction(), and ~Graph_Line().

5.13.4.4 lineNumber

```
std::map<unsigned, unsigned> hydrogen_framework::Graph_Line::lineNumber  [private]
```

Map between graphVersion and line Number

Definition at line 73 of file Graph_Line.hpp.

Referenced by getLineNumber(), and setLineNumber().

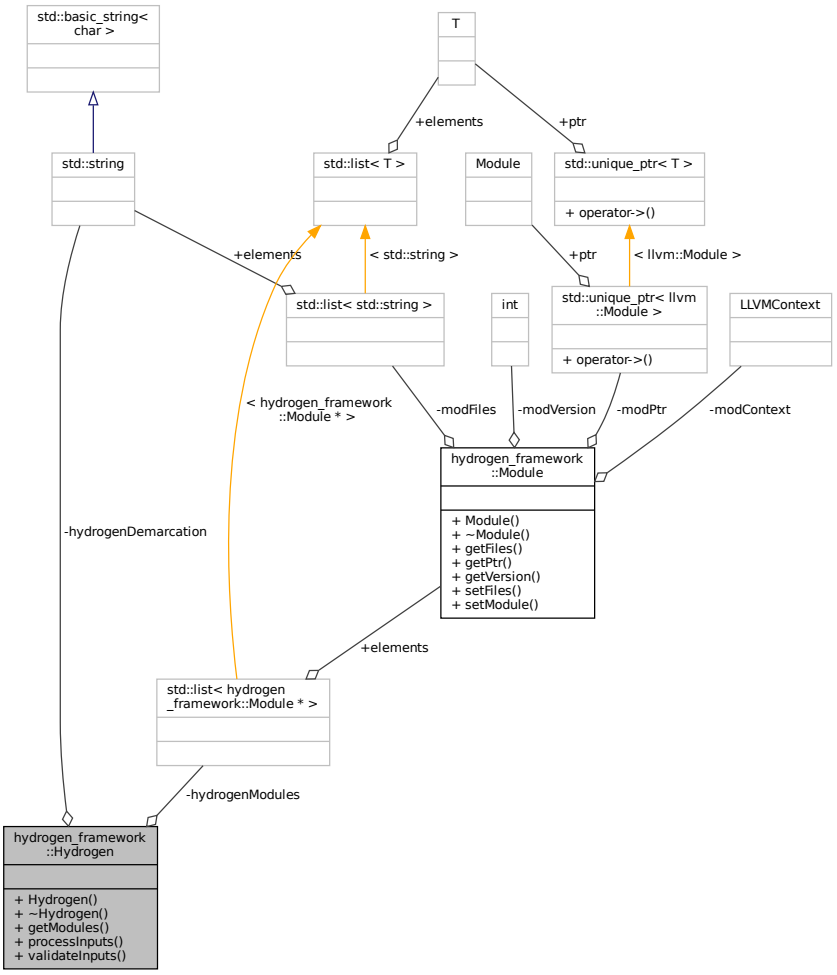
The documentation for this class was generated from the following files:

- Graph_Line.hpp
- Graph_Line.cpp

5.14 hydrogen_framework::Hydrogen Class Reference

```
#include <Get_Input.hpp>
```

Collaboration diagram for hydrogen_framework::Hydrogen:



Public Member Functions

- Hydrogen ()
- ~Hydrogen ()
- std::list< Module * > getModules ()
- bool processInputs (int c, char *files[])
- bool validateInputs (int c, char *files[])

Private Attributes

- std::string hydrogenDemarcation
- std::list< Module * > hydrogenModules

5.14.1 Detailed Description

Hydrogen Class: Hydrogen Framework data structures and functions

Definition at line 21 of file Get_Input.hpp.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 Hydrogen()

```
hydrogen_framework::Hydrogen::Hydrogen ( )  [inline]
```

Constructor for hydrogen class Sets the demarcation variable

Definition at line 27 of file Get_Input.hpp.

References hydrogenDemarcation.

5.14.2.2 ~Hydrogen()

```
hydrogen_framework::Hydrogen::~Hydrogen ( )  [inline]
```

Destructor

Definition at line 32 of file Get_Input.hpp.

References hydrogenModules.

5.14.3 Member Function Documentation

5.14.3.1 getModules()

```
std::list<Module *> hydrogen_framework::Hydrogen::getModules ( )  [inline]
```

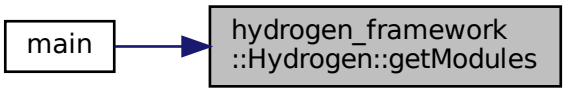
Return hydrogenModules

Definition at line 49 of file Get_Input.hpp.

References hydrogenModules.

Referenced by main().

Here is the caller graph for this function:



5.14.3.2 processInputs()

```
bool hydrogen_framework::Hydrogen::processInputs (
            int c,
            char * files[] )
```

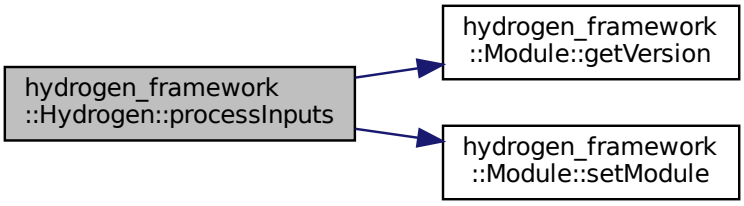
Process provided inputs. Returns FALSE if any of the Module cannot be parsed properly.

Definition at line 27 of file Get_Input.cpp.

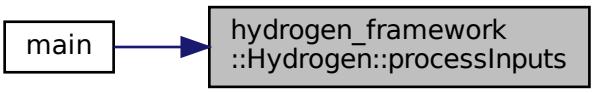
References hydrogen_framework::Module::getVersion(), hydrogenDemarcation, hydrogenModules, and hydrogen_framework::Module

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.3.3 validateInputs()

```
bool hydrogen_framework::Hydrogen::validateInputs (
            int c,
            char * files[] )
```

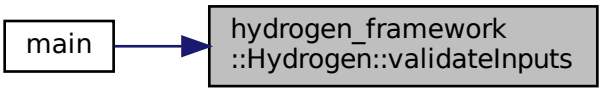
Validate provided inputs. Returns FALSE if any of the provided input is not present.

Definition at line 10 of file Get_Input.cpp.

References hydrogenDemarcation.

Referenced by main().

Here is the caller graph for this function:



5.14.4 Field Documentation

5.14.4.1 hydrogenDemarcation

```
std::string hydrogen_framework::Hydrogen::hydrogenDemarcation  [private]
```

Setting demarcation string for inputs

Definition at line 52 of file Get_Input.hpp.

Referenced by Hydrogen(), processInputs(), and validateInputs().

5.14.4.2 hydrogenModules

```
std::list<Module *> hydrogen_framework::Hydrogen::hydrogenModules  [private]
```

Container for storing LLVM Modules

Definition at line 53 of file Get_Input.hpp.

Referenced by getModules(), processInputs(), and ~Hydrogen().

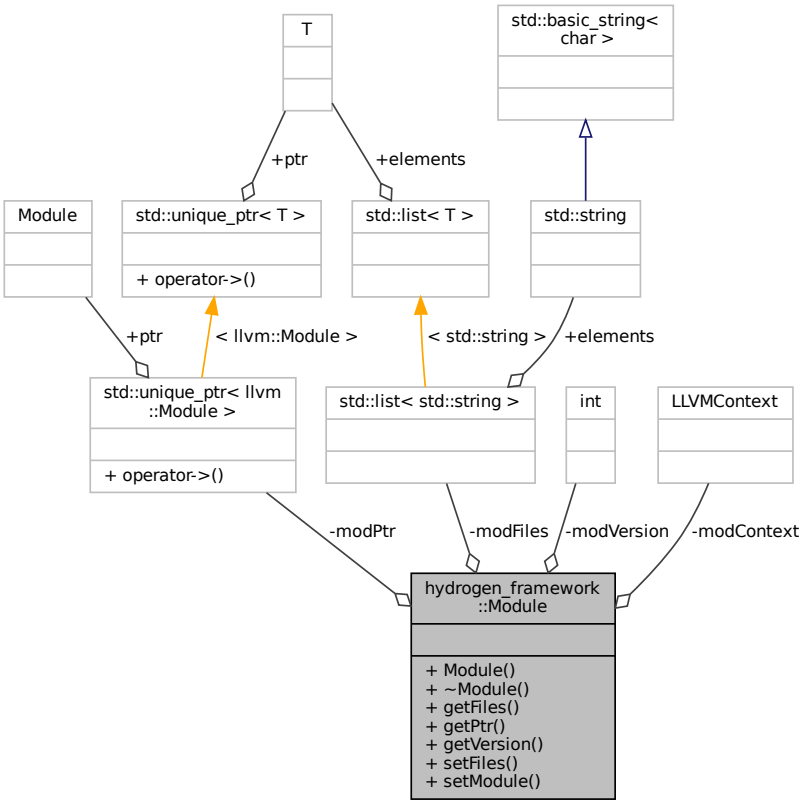
The documentation for this class was generated from the following files:

- Get_Input.hpp
- Get_Input.cpp

5.15 hydrogen_framework::Module Class Reference

```
#include <Module.hpp>
```

Collaboration diagram for hydrogen_framework::Module:



Public Member Functions

- Module ()
- ~Module ()
- std::list< std::string > getFiles ()
- std::unique_ptr< llvm::Module > & getPtr ()
- int getVersion ()
- void setFiles (std::list< std::string > file)
- bool setModule (int ver, std::string file)

Private Attributes

- llvm::LLVMContext modContext
- std::list< std::string > modFiles
- std::unique_ptr< llvm::Module > modPtr
- int modVersion

5.15.1 Detailed Description

LLVM Module class: Hold the LLVM modules and associated files

Definition at line 22 of file Module.hpp.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Module()

```
hydrogen_framework::Module::Module ( )  [inline]
```

Constructor for module class Set version to zero

Definition at line 28 of file Module.hpp.

References modVersion.

5.15.2.2 ~Module()

```
hydrogen_framework::Module::~Module ( )  [inline]
```

Destructor

Definition at line 33 of file Module.hpp.

References modFiles.

5.15.3 Member Function Documentation

5.15.3.1 getFiles()

```
std::list<std::string> hydrogen_framework::Module::getFiles ( )  [inline]
```

Return modFiles;

Definition at line 59 of file Module.hpp.

References modFiles.

5.15.3.2 getPtr()

```
std::unique_ptr<llvm::Module>& hydrogen_framework::Module::getPtr ( )  [inline]
```

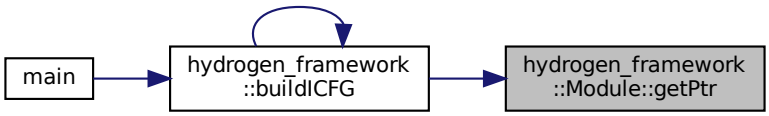
Return modPtr

Definition at line 54 of file Module.hpp.

References modPtr.

Referenced by hydrogen_framework::buildICFG().

Here is the caller graph for this function:



5.15.3.3 getVersion()

```
int hydrogen_framework::Module::getVersion ( )  [inline]
```

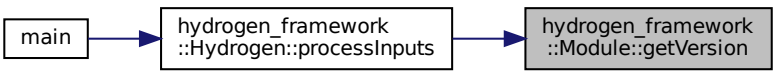
Return modVersion

Definition at line 49 of file Module.hpp.

References modVersion.

Referenced by hydrogen_framework::Hydrogen::processInputs().

Here is the caller graph for this function:



5.15.3.4 setFiles()

```
void hydrogen_framework::Module::setFiles (
            std::list< std::string > file )  [inline]
```

Set modFiles by swapping out with the incoming list of files

Definition at line 44 of file Module.hpp.

References modFiles.

5.15.3.5 setModule()

```
bool hydrogen_framework::Module::setModule (
            int ver,
            std::string file )
```

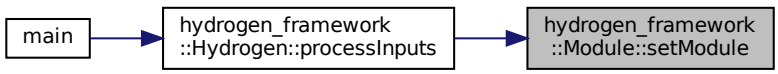
Set module by initializing all the values except modFiles Returns FALSE if LLVM IR parsing error is found

Definition at line 8 of file Module.cpp.

References modContext, modPtr, and modVersion.

Referenced by hydrogen_framework::Hydrogen::processInputs().

Here is the caller graph for this function:



5.15.4 Field Documentation

5.15.4.1 modContext

```
llvm::LLVMContext hydrogen_framework::Module::modContext  [private]
```

LLVM Module Context

Definition at line 63 of file Module.hpp.

Referenced by setModule().

5.15.4.2 modFiles

```
std::list<std::string> hydrogen_framework::Module::modFiles  [private]
```

Source files for the LLVM Module

Definition at line 65 of file Module.hpp.

Referenced by getFiles(), setFiles(), and ~Module().

5.15.4.3 modPtr

```
std::unique_ptr<llvm::Module> hydrogen_framework::Module::modPtr  [private]
```

LLVM Module Pointer

Definition at line 64 of file Module.hpp.

Referenced by getPtr(), and setModule().

5.15.4.4 modVersion

```
int hydrogen_framework::Module::modVersion  [private]
```

Module Version

Definition at line 62 of file Module.hpp.

Referenced by getVersion(), Module(), and setModule().

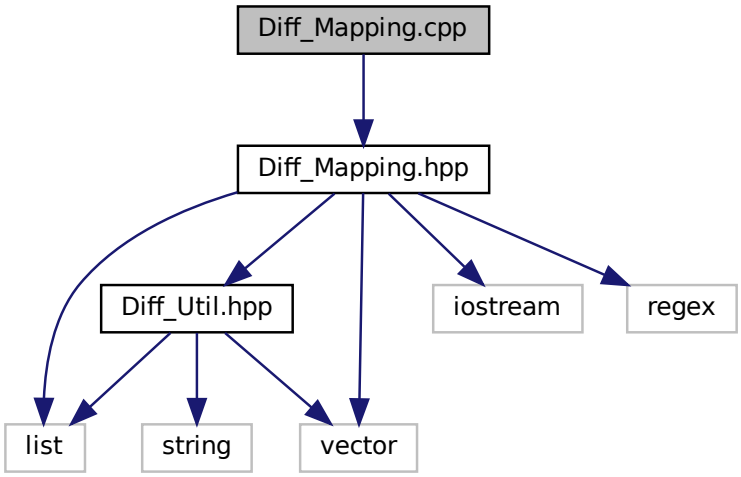
The documentation for this class was generated from the following files:

- Module.hpp
- Module.cpp

6 File Documentation

6.1 Diff_Mapping.cpp File Reference

`#include "Diff_Mapping.hpp"`
Include dependency graph for Diff_Mapping.cpp:



6.1.1 Detailed Description

Author

Ashwin K J

Implementing Diff_Mapping.hpp

Definition in file Diff_Mapping.cpp.

6.2 Diff_Mapping.cpp

```
00001
00006 #include "Diff_Mapping.hpp"
00007 namespace hydrogen_framework {
00008 void Diff_Mapping::putMapping(std::vector<sesElem> seqVector) {
00009   for (auto iter : seqVector) {
00010     elemInfo info;
00011     switch (iter.second.type) {
00012     case SES_ADD:
00013       info.beforeIdx = iter.second.beforeIdx;
00014       info.afterIdx = iter.second.afterIdx;
00015       info.type = SES_ADD;
00016       addedLines.push_back(iter.second.afterIdx);
00017       break;
00018     case SES_DELETE:
00019       info.beforeIdx = iter.second.beforeIdx;
00020       info.afterIdx = iter.second.afterIdx;
00021       info.type = SES_DELETE;
00022       deletedLines.push_back(iter.second.beforeIdx);
```

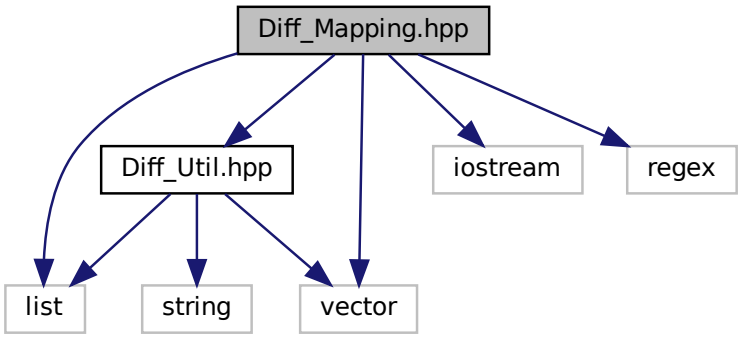
```
00023       break;
00024     case SES_COMMON:
00025       info.beforeIdx = iter.second.beforeIdx;
00026       info.afterIdx = iter.second.afterIdx;
00027       info.type = SES_COMMON;
00028       matchedLines.insert(std::pair<long long, long long>(iter.second.beforeIdx,
     iter.second.afterIdx));
00029       break;
00030     }
00031     lineMap.push_back(info);
00032   } // End loop for seqVector
00033 } // End putMapping
00034
00035 void Diff_Mapping::printMapping() {
00036   std::cout « "File name : " « fileName « "\n";
00037   for (auto iter : lineMap) {
00038     std::string type;
00039     switch (iter.type) {
00040     case SES_ADD:
00041       type = "+";
00042       break;
00043     case SES_DELETE:
00044       type = "-";
00045       break;
00046     case SES_COMMON:
00047       type = " ";
00048     } // End switch for iter.type
00049     std::cout « iter.beforeIdx « ":" « iter.afterIdx « "\t" « type « "\n";
00050   } // End loop for lineMap
00051 } // End printMapping
00052
00053 void Diff_Mapping::printAddedLines() {
00054   for (auto iter : addedLines) {
00055     std::cout « SES_MARK_ADD « " " « iter « "\n";
00056   } // End loop for addedLines
00057 } // End printAddedLines
00058
00059 void Diff_Mapping::printDeletedLines() {
00060   for (auto iter : deletedLines) {
00061     std::cout « SES_MARK_DELETE « " " « iter « "\n";
00062   } // End loop for deletedLines
00063 } // End printDeletedLines
00064
00065 void Diff_Mapping::printMatchedLines() {
00066   for (auto iter : matchedLines) {
00067     std::cout « iter.first « ":" « iter.second « "\n";
00068   } // End loop for matchedLines
00069 } // End printMatchedLines
00070
00071 void Diff_Mapping::printFileInfo() {
00072   std::cout « "File : " « getFileName() « "\n";
00073   printAddedLines();
00074   printDeletedLines();
00075   printMatchedLines();
00076   std::cout « "---\n";
00077 } // End printFileInfo
00078
00079 long long Diff_Mapping::getAfterLineNumber(long long currLine) {
00080   for (auto iter : lineMap) {
00081     if (iter.beforeIdx == currLine) {
00082       return iter.afterIdx;
00083     } // End check for currLine
00084   }   // End loop for lineMap
00085   return std::numeric_limits<unsigned>::max();
00086 } // End getNewLineNumber
00087
00088 long long Diff_Mapping::getBeforeLineNumber(long long currLine) {
00089   for (auto iter : lineMap) {
00090     if (iter.afterIdx == currLine) {
00091       return iter.beforeIdx;
00092     } // End check for currLine
00093   }   // End loop for lineMap
00094   return std::numeric_limits<unsigned>::max();
00095 } // End getOldLineNumber
00096 } // namespace hydrogen_framework
```

6.3 Diff_Mapping.hpp File Reference

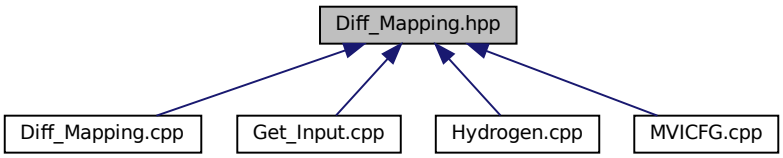
```
#include "Diff_Util.hpp"
#include <iostream>
#include <list>
```

```
#include <regex>
#include <vector>
```

Include dependency graph for Diff_Mapping.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Diff_Mapping

6.3.1 Detailed Description

Author

Ashwin K J

Diff_Mapping Class: Generating line mapping information per diff file

Definition in file Diff_Mapping.hpp.

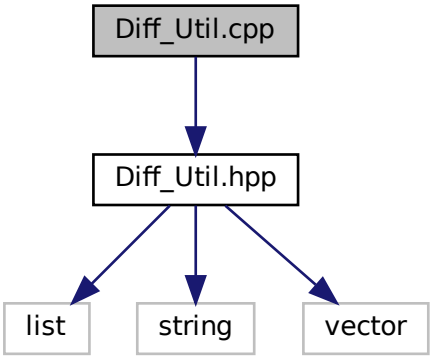
6.4 Diff_Mapping.hpp

```
00001
00006 #ifndef DIFF_MAPPING_H
00007 #define DIFF_MAPPING_H
00008
00009 #include "Diff_Util.hpp"
00010 #include <iostream>
00011 #include <list>
00012 #include <regex>
00013 #include <vector>
00014 namespace hydrogen_framework {
00018 class Diff_Mapping : public Diff_Vars {
00019 public:
00023   Diff_Mapping(std::string name) : fileName(name) {}
00024
00028    Diff_Mapping() { lineMap.clear(); }
00029
00033   void putMapping(std::vector<sesElem> seqVector);
00034
00038   std::list<elemInfo> getMapping() { return lineMap; }
00039
00043   std::string getFileName() { return fileName; }
00044
00048   std::list<long long> getAddedLines() { return addedLines; }
00049
00053   std::list<long long> getDeletedLines() { return deletedLines; }
00054
00058   std::map<long long, long long> getMatchedLines() { return matchedLines; }
00059
00063   void printMapping();
00064
00068   void printAddedLines();
00069
00073   void printDeletedLines();
00074
00078   void printMatchedLines();
00079
00083   void printFileInfo();
00084
00089   long long getAfterLineNumber(long long currLine);
00090
00095   long long getBeforeLineNumber(long long currLine);
00096
00097 private:
00098   std::string fileName;
00099   std::list<elemInfo> lineMap;
00100   std::list<long long> addedLines;
00101   std::list<long long> deletedLines;
00102   std::map<long long, long long>
00103       matchedLines;
00104 };                  // End Diff_Mapping Class
00105 } // namespace hydrogen_framework
00106 #endif
```

6.5 Diff_Util.cpp File Reference

```
#include "Diff_Util.hpp"
```

Include dependency graph for Diff_Util.cpp:



6.5.1 Detailed Description

Author

Ashwin K J

Implementing Diff_Util.hpp

Definition in file Diff_Util.cpp.

6.6 Diff_Util.cpp

```
00001
00006 #include "Diff_Util.hpp"
00007 namespace hydrogen_framework {
00008 void Diff_Ses::addSequence(elem e, long long beforeIdx, long long afterIdx, const int type) {
00009   elemInfo info;
00010   info.beforeIdx = beforeIdx;
00011   info.afterIdx = afterIdx;
00012   info.type = type;
00013   sesElem pe(e, info);
00014   if (!deletesFirst) {
00015     sequenceDS.push_back(pe);
00016   } // End check for deletesFirst
00017   switch (type) {
00018   case SES_DELETE:
00019     onlyCopy = false;
00020     onlyAdd = false;
00021     if (deletesFirst) {
00022       sequenceDS.insert(sequenceDS.begin() + nextDeleteIdx, pe);
00023       nextDeleteIdx++;
00024     } // End check for deletesFirst
00025     break;
00026   case SES_COMMON:
00027     onlyAdd = false;
00028     onlyDelete = false;
00029     if (deletesFirst) {
00030       sequenceDS.push_back(pe);
00031       nextDeleteIdx = sequenceDS.size();
00032     } // End check for deletesFirst
00033     break;
00034   case SES_ADD:
00035     onlyDelete = false;
00036     onlyCopy = false;
00037     if (deletesFirst) {
```

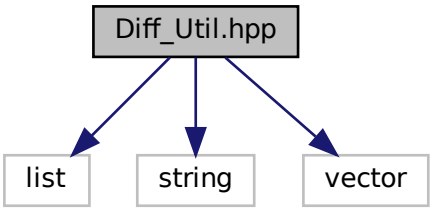
```
00038       sequenceDS.push_back(pe);
00039     } // End check for deletesFirst
00040     break;
00041   } // End switch for SES_DELETE
00042 } // End addSequence
00043
00044 void Diff_Util::compose() {
00045   pathCordinates.reserve(MAX_CORDINATES_SIZE);
00046   long long p = -1;
00047   fp = new long long[M + N + 3];
00048   std::fill(&fp[0], &fp[M + N + 3], -1);
00049   path = editPath(M + N + 3);
00050   fill(path.begin(), path.end(), -1);
00051 ONP:
00052   do {
00053     ++p;
00054     for (long long k = -p; k <= static_cast<long long>(delta) - 1; ++k) {
00055       fp[k + offset] = snake(k, fp[k - 1 + offset] + 1, fp[k + 1 + offset]);
00056     } // End loop for delta + 1
00057     for (long long k = static_cast<long long>(delta) + p; k >= static_cast<long long>(delta) + 1; -k)
      {
00058       fp[k + offset] = snake(k, fp[k - 1 + offset] + 1, fp[k + 1 + offset]);
00059     } // End loop for delta - 1
00060     fp[delta + offset] = snake(static_cast<long long>(delta), fp[delta - 1 + offset] + 1, fp[delta + 1
      + offset]);
00061   } while (fp[delta + offset] != static_cast<long long>(N) && pathCordinates.size() <
      MAX_CORDINATES_SIZE);
00062
00063   long long r = path[delta + offset];
00064   P cordinate;
00065   editPathCordinates epc(0);
00066
00067   while (r != -1) {
00068     cordinate.x = pathCordinates[(size_t)r].x;
00069     cordinate.y = pathCordinates[(size_t)r].y;
00070     epc.push_back(cordinate);
00071     r = pathCordinates[(size_t)r].k;
00072   } // End loop for r!= -1
00073
00074   // Record Longest Common Subsequence & Shortest Edit Script
00075   if (!recordSequence(epc)) {
00076     pathCordinates.resize(0);
00077     epc.resize(0);
00078     p = -1;
00079     goto ONP;
00080   } // End check for recordSequence
00081   delete[] this->fp;
00082 } // End compose
00083
00084 void Diff_Util::init() {
00085   M = distance(A.begin(), A.end());
00086   N = distance(B.begin(), B.end());
00087   if (M < N) {
00088     swapped = false;
00089   } else {
00090     std::swap(A, B);
00091     std::swap(M, N);
00092     swapped = true;
00093   } // End check for M < N
00094   delta = N - M;
00095   offset = M + 1;
00096   fp = NULL;
00097 } // End init
00098
00099 long long Diff_Util::snake(const long long &k, const long long &above, const long long &below) {
00100   long long r = above > below ? path[(size_t)k - 1 + offset] : path[(size_t)k + 1 + offset];
00101   long long y = std::max(above, below);
00102   long long x = y - k;
00103   while ((size_t)x < M && (size_t)y < N &&
00104          (swapped ? cmp.impl(B[(size_t)y], A[(size_t)x]) : cmp.impl(A[(size_t)x], B[(size_t)y]))) {
00105     ++x;
00106     ++y;
00107   } // End loop for swapped
00108
00109   path[(size_t)k + offset] = static_cast<long long>(pathCordinates.size());
00110   P p;
00111   p.x = x;
00112   p.y = y;
00113   p.k = r;
00114   pathCordinates.push_back(p);
00115   return y;
00116 } // End snake
00117
00118 bool Diff_Util::recordSequence(const editPathCordinates &v) {
00119   sequence_const_iter x(A.begin());
00120   sequence_const_iter y(B.begin());
00121   long long x_idx, y_idx;   // line number for Unified Format
```

```
00122   long long px_idx, py_idx; // cordinates
00123   bool complete = false;
00124   x_idx = y_idx = 1;
00125   px_idx = py_idx = 0;
00126   for (size_t i = v.size() - 1; !complete; --i) {
00127     while (px_idx < v[i].x || py_idx < v[i].y) {
00128       if (v[i].y - v[i].x > py_idx - px_idx) {
00129         if (!wasSwapped()) {
00130           ses.addSequence(*y, 0, y_idx, SES_ADD);
00131         } else {
00132           ses.addSequence(*y, y_idx, 0, SES_DELETE);
00133         } // End check for wasSwapped
00134         ++y;
00135         ++y_idx;
00136         ++py_idx;
00137       } else if (v[i].y - v[i].x < py_idx - px_idx) {
00138         if (!wasSwapped()) {
00139           ses.addSequence(*x, x_idx, 0, SES_DELETE);
00140         } else {
00141           ses.addSequence(*x, 0, x_idx, SES_ADD);
00142         } // End check for wasSwapped
00143         ++x;
00144         ++x_idx;
00145         ++px_idx;
00146       } else {
00147         if (!wasSwapped()) {
00148           ses.addSequence(*x, x_idx, y_idx, SES_COMMON);
00149         } else {
00150           ses.addSequence(*y, y_idx, x_idx, SES_COMMON);
00151         } // End check for wasSwapped
00152         ++x;
00153         ++y;
00154         ++x_idx;
00155         ++y_idx;
00156         ++px_idx;
00157         ++py_idx;
00158       } // End check for v.y -v.x
00159     }   // End loop for px_idx & py_idx
00160     if (i == 0)
00161       complete = true;
00162   } // End loop for complete
00163
00164   if (x_idx > static_cast<long long>(M) && y_idx > static_cast<long long>(N)) {
00165     // all recording succeeded
00166   } else {
00167     sequence A_(A.begin() + (size_t)x_idx - 1, A.end());
00168     sequence B_(B.begin() + (size_t)y_idx - 1, B.end());
00169     A = A_;
00170     B = B_;
00171     M = distance(A.begin(), A.end());
00172     N = distance(B.begin(), B.end());
00173     delta = N - M;
00174     offset = M + 1;
00175     delete[] fp;
00176     fp = new long long[M + N + 3];
00177     std::fill(&fp[0], &fp[M + N + 3], -1);
00178     std::fill(path.begin(), path.end(), -1);
00179     return false;
00180   } // End check for x_idx
00181   return true;
00182 } // End snake
00183 } // namespace hydrogen_framework
```

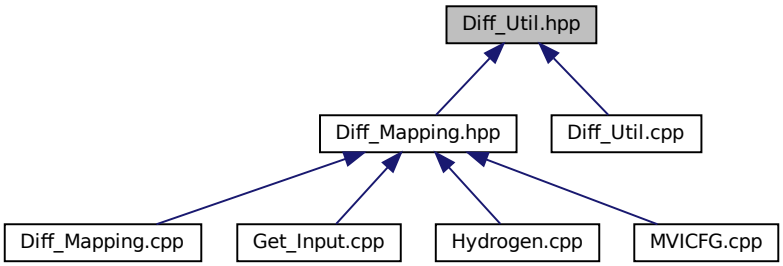
6.7 Diff_Util.hpp File Reference

```
#include <list>
#include <string>
#include <vector>
```

Include dependency graph for Diff_Util.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Diff_Compare
- class hydrogen_framework::Diff_Sequence
- class hydrogen_framework::Diff_Ses
- class hydrogen_framework::Diff_Util
- class hydrogen_framework::Diff_Vars
- struct hydrogen_framework::Diff_Vars::eleminfo
- struct hydrogen_framework::Diff_Vars::Point

6.7.1 Detailed Description

Author

Ashwin K J

Diff Util Class: Generating the diff between the files

Definition in file Diff_Util.hpp.

6.8 Diff_Util.hpp

```
00001
00006 #ifndef DIFF_UTIL_H
00007 #define DIFF_UTIL_H
00008
00009 #include <list>
00010 #include <string>
00011 #include <vector>
00012 namespace hydrogen_framework {
00016 class Diff_Compare {
00017 public:
00021   Diff_Compare() {}
00025   virtual  Diff_Compare() {}
00030   virtual inline bool impl(const std::string &e1, const std::string &e2) const { return e1 == e2; }
00031 };
00032
00036 class Diff_Vars {
00037 public:
00041   Diff_Vars() {}
00042
00046   virtual  Diff_Vars() {}
00047
00051   enum SES_TYPE { SES_DELETE = -1, SES_COMMON = 0, SES_ADD = 1 };
00052
00053   std::string SES_MARK_DELETE = "-";
00054   std::string SES_MARK_COMMON = " ";
00055   std::string SES_MARK_ADD = "+";
00060   typedef struct eleminfo {
00061     long long beforeIdx;
00062     long long afterIdx;
00063     int type;
00067     bool operator==(const eleminfo &other) const {
00068       return (this->beforeIdx == other.beforeIdx && this->afterIdx == other.afterIdx && this->type ==
      other.type);
00069     }
00070   } elemInfo;
00071
00075   typedef struct Point {
00076     long long x;
00077     long long y;
00078     long long k;
00079   } P;
00080
00081   const unsigned long long MAX_CORDINATES_SIZE = 2000000;
00082   typedef std::vector<long long> editPath;
00083   typedef std::vector<P> editPathCordinates;
00084   typedef std::string elem;
00085   typedef std::vector<elem> sequence;
00086   typedef std::pair<elem, elemInfo> sesElem;
00087   typedef std::vector<sesElem> sesElemVec;
00088   typedef std::list<elem> elemList;
00089   typedef std::vector<elem> elemVec;
00090   typedef typename sesElemVec::iterator sesElemVec_iter;
00091   typedef typename elemList::iterator elemList_iter;
00092   typedef typename sequence::iterator sequence_iter;
00093   typedef typename sequence::const_iterator sequence_const_iter;
00094   typedef typename elemVec::iterator elemVec_iter;
00095 };
00096
00100 class Diff_Sequence : public Diff_Vars {
00101 public:
00105   Diff_Sequence() {}
00109   virtual  Diff_Sequence() {}
00110
00114   elemVec getSequence() const { return sequence; }
00115
00120   void addSequence(elem e) { sequence.push_back(e); }
00121
00122 protected:
00123   elemVec sequence;
00124 };
00125
00129 class Diff_Ses : public Diff_Sequence {
00130 public:
00134   Diff_Ses() : onlyAdd(true), onlyDelete(true), onlyCopy(true), deletesFirst(false) { nextDeleteIdx =
      0; }
00135
00139   Diff_Ses(bool moveDel) : onlyAdd(true), onlyDelete(true), onlyCopy(true), deletesFirst(moveDel) {
     nextDeleteIdx = 0; }
00140
00144    Diff_Ses() {}
00145
00149   bool isOnlyAdd() const { return onlyAdd; }
00150
00154   bool isOnlyDelete() const { return onlyDelete; }
00155
```

```
00159   bool isOnlyCopy() const { return onlyCopy; }
00160
00164   bool isOnlyOneOperation() const { return isOnlyAdd() || isOnlyDelete() || isOnlyCopy(); }
00165
00169   bool isChange() const { return !onlyCopy; }
00170
00174   void addSequence(elem e, long long beforeIdx, long long afterIdx, const int type);
00175
00179   sesElemVec getSequence() const { return sequenceDS; }
00180
00181 private:
00182   sesElemVec sequenceDS;
00183   bool onlyAdd;
00184   bool onlyDelete;
00185   bool onlyCopy;
00186   bool deletesFirst;
00187   size_t nextDeleteIdx;
00188 };
00189
00193 class Diff_Util : Diff_Vars {
00194 public:
00198   Diff_Util() {}
00199
00203   Diff_Util(const sequence &a, const sequence &b) : A(a), B(b), ses(false) { init(); }
00204
00208    Diff_Util() {}
00209
00213   Diff_Ses getSes() const { return ses; }
00214
00220   void compose();
00221
00222 private:
00223   sequence A;
00224   sequence B;
00225   size_t M;
00226   size_t N;
00227   size_t delta;
00228   size_t offset;
00229   long long *fp;
00230   Diff_Ses ses;
00231   editPath path;
00232   editPathCordinates pathCordinates;
00233   bool swapped;
00234   Diff_Compare cmp;
00238   void init();
00239
00243   long long snake(const long long &k, const long long &above, const long long &below);
00244
00248   bool recordSequence(const editPathCordinates &v);
00249
00253   bool inline wasSwapped() const { return swapped; }
00254 }; // End Diff Class
00255 } // namespace hydrogen_framework
00256 #endif
```

6.9 Get_Input.cpp File Reference

```
#include "Get_Input.hpp"
#include "Diff_Mapping.hpp"
#include "Module.hpp"
```

Include dependency graph for Get_Input.cpp:

6.9.1 Detailed Description

Author

Ashwin K J

Implementing Get_Input.hpp

Definition in file Get_Input.cpp.

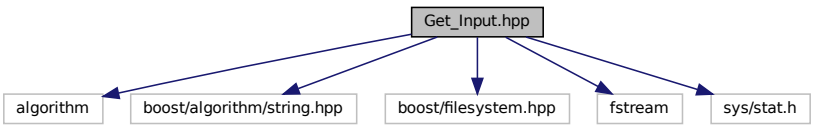
6.10 Get_Input.cpp

```
00001
00006 #include "Get_Input.hpp"
00007 #include "Diff_Mapping.hpp"
00008 #include "Module.hpp"
00009 namespace hydrogen_framework {
00010 bool Hydrogen::validateInputs(int c, char *files[]) {
00011   for (int index = 1; index < c; index++) {
00012     std::string file = files[index];
00013     struct stat buffer;
00014     int status = stat(file.c_str(), &buffer);
00015     if (status == -1) {
00016       if (file == hydrogenDemarcation) {
00017         continue;
00018       } // End check for hydrogenDemarcation
00019       std::cerr « file « " not accessible\n"
00020                 « "Please recheck the input\n";
00021       return false;
00022     } // End check for status
00023   }   // End loop for inputs
00024   return true;
00025 } // End validateInputs
00026
00027 bool Hydrogen::processInputs(int c, char *files[]) {
00028   int countModules = 0;
00029   /* Getting all the modules first */
00030   int index = 1;
00031   for (; index < c; ++index) {
00032     std::string file = files[index];
00033     if (file == hydrogenDemarcation) {
00034       break;
00035     } // End check for hydrogenDemarcation
00036     countModules++;
00037     Module *module = new Module();
00038     if (!module->setModule(countModules, file)) {
00039       return false;
00040     } // End check for module
00041     hydrogenModules.push_back(module);
00042   } // End module loop
00043   /* Getting the files associated with it */
00044   bool filesForAllVersions = false;
00045   for (int i = 1; i <= countModules; ++i) {
00046     std::list<std::string> versionFiles;
00047     for (++index; index < c; ++index) {
00048       std::string file = files[index];
00049       /* Checking for proper loop exit */
00050       if (file == hydrogenDemarcation) {
00051         if (i == (countModules - 1)) {
00052           filesForAllVersions = true;
00053         } // End check for countModules
00054         break;
00055       } // End check for hydrogenDemarcation
00056       versionFiles.push_back(file);
00057     } // End loop for versionFiles
00058     auto moduleIter = std::find_if(std::begin(hydrogenModules), std::end(hydrogenModules),
00059                                    [=](Module *mod) { return (mod->getVersion() == i); });
00060     if (moduleIter != hydrogenModules.end()) {
00061       (*moduleIter)->setFiles(versionFiles);
00062     } // End check for hydrogenDemarcation
00063   }   // End file loop
00064   if (!filesForAllVersions) {
00065     std::cerr « "Insufficient no of file versions provided\n"
00066               « "Please recheck your input\n";
00067     return false;
00068   } // End check for filesForAllVersions
00069   return true;
00070 } // End processInputs
00071 } // namespace hydrogen_framework
```

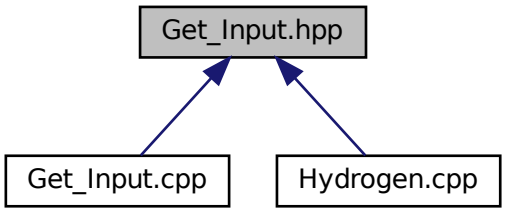
6.11 Get_Input.hpp File Reference

```
#include <algorithm>
#include <boost/algorithm/string.hpp>
#include <boost/filesystem.hpp>
#include <fstream>
#include <sys/stat.h>
```

Include dependency graph for Get_Input.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Hydrogen

6.11.1 Detailed Description

Author

Ashwin K J

Get_Input Class: Getting the input from the user

Definition in file Get_Input.hpp.

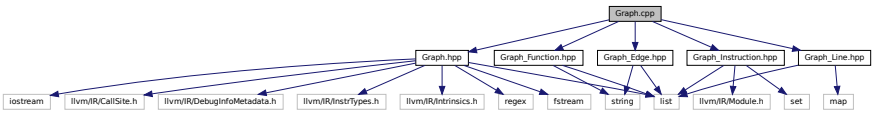
6.12 Get_Input.hpp

```
00001
00006 #ifndef GET_INPUT_H
00007 #define GET_INPUT_H
00008
00009 #include <algorithm>
00010 #include <boost/algorithm/string.hpp>
00011 #include <boost/filesystem.hpp>
00012 #include <fstream>
00013 #include <sys/stat.h>
00014 namespace hydrogen_framework {
00015 /* Forward declaration */
00016 class Module;
00017
00021 class Hydrogen {
00022 public:
00027   Hydrogen() { hydrogenDemarcation = "::"; }
00028
00032    Hydrogen() { hydrogenModules.clear(); }
00033
00038   bool validateInputs(int c, char *files[]);
00039
00044   bool processInputs(int c, char *files[]);
00045
00049   std::list<Module *> getModules() { return hydrogenModules; }
00050
00051 private:
00052   std::string hydrogenDemarcation;
00053   std::list<Module *> hydrogenModules;
00054 };                                    // End hydrogen class
00055 } // namespace hydrogen_framework
00056 #endif
```

6.13 Graph.cpp File Reference

```
#include "Graph.hpp"
#include "Graph_Edge.hpp"
#include "Graph_Function.hpp"
#include "Graph_Instruction.hpp"
#include "Graph_Line.hpp"
```

Include dependency graph for Graph.cpp:



Functions

- void hydrogen_framework::getLocationInfo (llvm::Instruction &I, unsigned int &DILocLine, std::string &DIFile)

6.13.1 Detailed Description

Author

Ashwin K J

Implementing Graph.hpp

Definition in file Graph.cpp.

6.13.2 Function Documentation

6.13.2.1 getLocationInfo()

```
void hydrogen_framework::getLocationInfo (
             llvm::Instruction & I,
             unsigned int & DILocLine,
             std::string & DIFile )
```

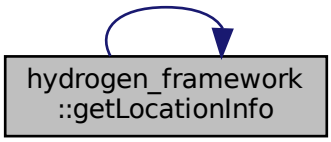
Find the line number and file name of the given LLVM instruction Will return 0 if no information found

Definition at line 296 of file Graph.cpp.

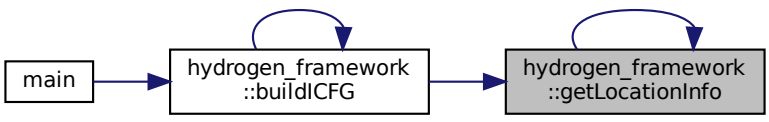
References hydrogen_framework::getLocationInfo().

Referenced by hydrogen_framework::buildICFG(), and hydrogen_framework::getLocationInfo().

Here is the call graph for this function:



Here is the caller graph for this function:



6.14 Graph.cpp

```
00001
00006 #include "Graph.hpp"
00007 #include "Graph_Edge.hpp"
00008 #include "Graph_Function.hpp"
00009 #include "Graph_Instruction.hpp"
00010 #include "Graph_Line.hpp"
00011 namespace hydrogen_framework {
00012 void Graph::pushGraphFunction(Graph_Function *func) {
00013   func->setGraph(this);
00014   graphFunctions.push_back(func);
00015 } // End pushGraphFunction
00016
00017 void Graph::addEdge(Graph_Instruction *from, Graph_Instruction *to, Graph_Edge *edge) {
00018   from->pushEdgeInstruction(edge);
00019   to->pushEdgeInstruction(edge);
00020   pushGraphEdges(edge);
00021 } // End addEdge
00022
00023 void Graph::addSeqEdges(Graph_Line *line) {
00024   std::list<Graph_Instruction *> instructions = line->getLineInstructions();
00025   for (auto inst = instructions.begin(), instEnd = instructions.end(); inst != instEnd; ++inst) {
00026     /* Double check to make sure it is not Br */
00027     llvm::Instruction *llvmInst = (*inst)->getInstructionPtr();
00028     if (llvmInst) {
00029       if (llvmInst->getOpcode() == llvm::Instruction::Br) {
00030         continue;
00031       } // End check for Br
00032     }   // End check for llvmInst
00033     auto nextInst = std::next(inst);
00034     if (nextInst != instEnd) {
00035       Graph_Edge *seqEdge = new Graph_Edge(*inst, *nextInst, Graph_Edge::SEQUENTIAL, graphVersion);
00036       addEdge(*inst, *nextInst, seqEdge);
00037     } // End check for instEnd
00038   }   // End loop for inst
00039 } // End addSeqEdges
00040
00041 Graph_Instruction *Graph::findMatchedInstruction(llvm::Instruction *matchInst) {
00042   for (auto func : graphFunctions) {
00043     for (auto line : func->getFunctionLines()) {
00044       for (auto inst : line->getLineInstructions()) {
00045         if (inst->getInstructionPtr() == matchInst) {
00046           return inst;
00047         } // End check for matchInst
00048       }   // End loop for inst
00049     }     // End loop for line
00050   }       // End loop for func
00051   return NULL;
00052 } // End findMatchedInstruction
00053
00054 Graph_Instruction *Graph::findVirtualEntry(std::string funcName) {
00055   for (auto func : graphFunctions) {
00056     if (func->getFunctionName() == funcName) {
00057       for (auto line : func->getFunctionLines()) {
00058         for (auto inst : line->getLineInstructions()) {
00059           if (inst->getInstructionLabel().find("Entry::") != std::string::npos) {
00060             return inst;
00061           } // End check for matchInst
00062         }   // End loop for inst
00063       }     // End loop for line
00064     }       // End check for function name
00065   }         // End loop for func
00066   return NULL;
00067 } // End findVirtualEntry
00068
00069 Graph_Instruction *Graph::findVirtualExit(std::string funcName) {
00070   for (auto func : graphFunctions) {
00071     if (func->getFunctionName() == funcName) {
00072       for (auto line : func->getFunctionLines()) {
00073         for (auto inst : line->getLineInstructions()) {
00074           if (inst->getInstructionLabel().find("Exit::") != std::string::npos) {
00075             return inst;
00076           } // End check for Exit
00077         }   // End loop for inst
00078       }     // End loop for line
00079     }       // End check for function name
00080   }         // End loop for func
00081   return NULL;
00082 } // End findVirtualExit
00083
00084 void Graph::addBranchEdges() {
00085   for (auto func : graphFunctions) {
00086     std::list<Graph_Line *> lines = func->getFunctionLines();
00087     for (auto line = lines.begin(); line != lines.end(); ++line) {
00088       std::list<Graph_Instruction *> instructions = (*line)->getLineInstructions();
00089       for (auto inst = instructions.begin(); inst != instructions.end(); ++inst) {
```

```
00090         llvm::Instruction *I = (*inst)->getInstructionPtr();
00091         if (I) {
00092           /* Adding edges for BB with multiple successors */
00093           if (I->isTerminator()) {
00094             unsigned int noSucc = I->getNumSuccessors();
00095             for (unsigned int iterSucc = 0; iterSucc < noSucc; ++iterSucc) {
00096               llvm::Instruction *iSucc =
      llvm::dyn_cast<llvm::Instruction>(I->getSuccessor(iterSucc)->begin());
00097               Graph_Instruction *iSuccInst = findMatchedInstruction(iSucc);
00098               if (iSucc) {
00099                 Graph_Edge *branchEdge = new Graph_Edge(*inst, iSuccInst, Graph_Edge::BRANCH,
      graphVersion);
00100                 addEdge(*inst, iSuccInst, branchEdge);
00101               } else {
00102                 std::cerr « "No matching Graph_Instruction found for edge from " «
      (*inst)->getInstructionLabel()
00103                           « "\n";
00104               } // End check for iSucc
00105             }   // End loop for iterSucc
00106           } else if ((*inst)->getInstructionID() == instructions.back()->getInstructionID()) {
00107             /* Adding Unique successors */
00108             auto nextLine = std::next(line);
00109             if (nextLine != lines.end()) {
00110               std::list<Graph_Instruction *> nextInstructions = (*nextLine)->getLineInstructions();
00111               auto nextI = nextInstructions.begin();
00112               if (nextI != nextInstructions.end()) {
00113                 Graph_Edge *seqEdge = new Graph_Edge(*inst, *nextI, Graph_Edge::SEQUENTIAL,
      graphVersion);
00114                 addEdge(*inst, *nextI, seqEdge);
00115               } // End check for nextI
00116             }   // End check for nextLine
00117           }     // End check for TerminatorInst
00118         }       // End check for I
00119       }         // End loop for inst
00120     }           // End llop for line
00121   }             // End loop for func
00122 } // End addBranchEdges
00123
00124 void Graph::addFunctionCallEdges() {
00125   /* External Node */
00126   Graph_Function *virtualNodeFunc = new Graph_Function(getNextID());
00127   virtualNodeFunc->setFunctionFile("External_Node_File");
00128   virtualNodeFunc->setFunctionName("External_Node_Func");
00129   Graph_Line *virtualNodeLine = new Graph_Line(graphVersion);
00130   virtualNodeLine->setLineNumber(graphVersion, graphEntryID);
00131   Graph_Instruction *externalNode = new Graph_Instruction();
00132   externalNode->setInstructionID(getNextID());
00133   externalNode->setInstructionLabel("External_Node");
00134   externalNode->setInstructionPtr(NULL);
00135   virtualNodeLine->pushLineInstruction(externalNode);
00136   virtualNodeFunc->pushFunctionLines(virtualNodeLine);
00137   pushGraphFunction(virtualNodeFunc);
00138   std::list<std::string> funcNotFoud;
00139   /* Get the whitelisted function names and merge with funcNotFoud */
00140   std::list<std::string> whiteListedFunc = Graph::getWhiteList();
00141   funcNotFoud.insert(funcNotFoud.end(), whiteListedFunc.begin(), whiteListedFunc.end());
00142   for (auto func : graphFunctions) {
00143     for (auto line : func->getFunctionLines()) {
00144       for (auto inst : line->getLineInstructions()) {
00145         llvm::Instruction *I = inst->getInstructionPtr();
00146         if (I) {
00147           if (auto callSite = llvm::CallSite(I)) {
00148             const llvm::Function *Callee = callSite.getCalledFunction();
00149             if (!Callee || !llvm::Intrinsic::isLeaf(Callee->getIntrinsicID())) {
00150               /* Call Extern */
00151               Graph_Edge *callEdge = new Graph_Edge(inst, externalNode, Graph_Edge::EXTERNAL_CALL,
      graphVersion);
00152               addEdge(inst, externalNode, callEdge);
00153             } else if (!Callee->isIntrinsic()) {
00154               /* Add Edge based on function name */
00155               bool noEntry = false;
00156               bool noExit = false;
00157               if (!Callee->getName().empty()) {
00158                 std::string funcName = Callee->getName();
00159                 /* Call site to Entry */
00160                 Graph_Instruction *virtualEntry = findVirtualEntry(funcName);
00161                 if (virtualEntry) {
00162                   Graph_Edge *callEdge = new Graph_Edge(inst, virtualEntry, Graph_Edge::CALL,
      graphVersion);
00163                   addEdge(inst, virtualEntry, callEdge);
00164                 } else {
00165                   noEntry = true;
00166                 } // End check for virtualEntry
00167                 /* Exit to Call site */
00168                 Graph_Instruction *virtualExit = findVirtualExit(funcName);
00169                 if (virtualExit) {
00170                   Graph_Edge *callEdge = new Graph_Edge(virtualExit, inst, Graph_Edge::CALL,
```

```
      graphVersion);
00171                   addEdge(virtualExit, inst, callEdge);
00172                 } else {
00173                   noExit = true;
00174                 } // End check for virtualExit
00175                 auto findFunc = std::find_if(std::begin(funcNotFoud), std::end(funcNotFoud),
00176                                              [=](std::string name) { return name == funcName; });
00177                 if (findFunc == funcNotFoud.end()) {
00178                   if (noEntry && noExit) {
00179                     funcNotFoud.push_back(funcName);
00180                     std::cerr << "Call edges not formed for " << funcName << "\n";
00181                   } else if (noEntry) {
00182                     std::cerr << "No Virtual Entry found for " << funcName << "\n";
00183                   } else if (noExit) {
00184                     std::cerr << "No Virtual Exit found for " << funcName << "\n";
00185                   } // End check for noEntry & noExit combinations
00186                 }   // End check for findFunc
00187               } else {
00188                 std::cerr << "Unknown function call from Instruction " << inst->getInstructionLabel() <<
      "\n";
00189               } // End check for Callee name
00190             }   // End check for Callee Intrinsic
00191           }     // End check for callSite
00192         }       // End check for I
00193       }         // End loop for inst
00194     }           // End loop for line
00195   }             // End loop for func
00196 } // End addFunctionCallEdges
00197
00198 void Graph::addVirtualNodes(Graph_Function *func) {
00199   std::string funcName = func->getFunctionName();
00200   Graph_Line *virtualLine = new Graph_Line(graphVersion);
00201   /* Entry Node */
00202   virtualLine->setLineNumber(graphVersion, graphEntryID);
00203   Graph_Instruction *virtualNode = new Graph_Instruction();
00204   virtualNode->setInstructionID(getNextID());
00205   virtualNode->setInstructionLabel("Entry::" + funcName);
00206   virtualNode->setInstructionPtr(NULL);
00207   virtualLine->pushLineInstruction(virtualNode);
00208   auto *to = func->getFunctionLines().front()->getLineInstructions().front();
00209   func->pushFrontFunctionLines(virtualLine);
00210   Graph_Edge *virtualEdgeEntry = new Graph_Edge(virtualNode, to, Graph_Edge::VIRTUAL, graphVersion);
00211   addEdge(virtualNode, to, virtualEdgeEntry);
00212   /* Exit Node */
00213   virtualLine = new Graph_Line(graphVersion);
00214   virtualLine->setLineNumber(graphVersion, graphExitID);
00215   virtualNode = new Graph_Instruction();
00216   virtualNode->setInstructionID(getNextID());
00217   virtualNode->setInstructionLabel("Exit::" + funcName);
00218   virtualNode->setInstructionPtr(NULL);
00219   virtualLine->pushLineInstruction(virtualNode);
00220   auto *from = func->getFunctionLines().back()->getLineInstructions().back();
00221   func->pushFunctionLines(virtualLine);
00222   Graph_Edge *virtualEdgeExit = new Graph_Edge(from, virtualNode, Graph_Edge::VIRTUAL, graphVersion);
00223   addEdge(from, virtualNode, virtualEdgeExit);
00224 } // End addVirtualNodes
00225
00226 void Graph::printGraph(std::string graphName) {
00227   std::ofstream gFile(graphName + ".dot", std::ios::trunc);
00228   if (!gFile.is_open()) {
00229     std::cerr << "Unable to open file for printing the output\n";
00230     return;
00231   } // End check for gFile
00232   /* Initialize graph */
00233   gFile << "digraph \"MVICFG\" {\n";
00234   gFile << "\tlabel=\"" << graphName << "\";\n";
00235   /* Generating Nodes */
00236   gFile << "/* Generating Nodes */\n";
00237   for (auto func : graphFunctions) {
00238     gFile << "\tsubgraph cluster_" << func->getFunctionID() << " {\n";
00239     gFile << "\t\tlabel=\"" << func->getFunctionName() << "\";\n";
00240     for (auto line : func->getFunctionLines()) {
00241       for (auto inst : line->getLineInstructions()) {
00242         std::string outputString = std::regex_replace(inst->getInstructionLabel(), std::regex("\""),
      "\\\"");
00243         gFile << "\t\t\"" << inst->getInstructionID() << "\" [label=\"" <<
      line->getLineNumber(graphVersion)
00244               << "::" << outputString << "\"];\n";
00245       } // End loop for inst
00246     }   // End loop for line
00247     gFile << "\t}\n";
00248   } // End loop for func
00249   /* Generating Edges*/
00250   gFile << "\n/* Generating Edges */\n";
00251   for (auto edge : graphEdges) {
00252     std::string outputString = "\t\t\"" + std::to_string(edge->getEdgeFrom()->getInstructionID()) +
      "\" -> \"" +
```

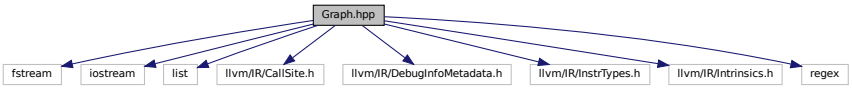
```
00253                               std::to_string(edge->getEdgeTo()->getInstructionID());
00254     switch (edge->getEdgeType()) {
00255     case Graph_Edge::SEQUENTIAL:
00256       outputString += "\" [arrowhead = normal, penwidth = 1.0, color = black, label=\"" +
00257                       edge->getPrintableEdgeVersions() + "\"];\n";
00258       break;
00259     case Graph_Edge::BRANCH:
00260       outputString += "\" [arrowhead = dot, penwidth = 1.0, color = black, label=\"" +
00261                       edge->getPrintableEdgeVersions() + "::Branch\"];\n";
00262       break;
00263     case Graph_Edge::VIRTUAL:
00264       outputString += "\" [arrowhead = normal, penwidth = 1.0, color = pink, label=\"" +
00265                       edge->getPrintableEdgeVersions() + "::Virtual\"];\n";
00266       break;
00267     case Graph_Edge::CALL:
00268       outputString += "\" [arrowhead = odot, penwidth = 1.0, color = blue, label=\"" +
00269                       edge->getPrintableEdgeVersions() + "::Call\"];\n";
00270       break;
00271     case Graph_Edge::EXTERNAL_CALL:
00272       outputString += "\" [arrowhead = odot, penwidth = 1.0, color = yellow, label=\"" +
00273                       edge->getPrintableEdgeVersions() + "::External_Call\"];\n";
00274       break;
00275     case Graph_Edge::MVICFG_ADD:
00276       outputString += "\" [arrowhead = normal, penwidth = 1.0, color = green, label=\"" +
00277                       edge->getPrintableEdgeVersions() + "::Add\"];\n";
00278       break;
00279     case Graph_Edge::MVICFG_DEL:
00280       outputString += "\" [arrowhead = normal, penwidth = 1.0, color = red, label=\"" +
00281                       edge->getPrintableEdgeVersions() + "::Del\"];\n";
00282       break;
00283     case Graph_Edge::ANY:
00284       std::cerr « "Should not have ANY as edgeType\n";
00285       outputString += "\" [arrowhead = normal, penwidth = 2.0, color = red, label=\"" +
00286                       edge->getPrintableEdgeVersions() + "::ANY\"];\n";
00287       break;
00288     } // End switch for edge
00289     gFile « outputString;
00290   } // End loop for edge
00291   /* Finalizing graph */
00292   gFile « "}\n";
00293   gFile.close();
00294 } // End printGraph
00295
00296 void getLocationInfo(llvm::Instruction &I, unsigned int &DILocLine, std::string &DIFile) {
00297   if (llvm::DILocation *DILoc = I.getDebugLoc()) {
00298     DILocLine = DILoc->getLine();
00299     DIFile = DILoc->getFilename();
00300     return;
00301   } else {
00302     bool resolvedLine = false;
00303     /* Search backward */
00304     llvm::Instruction *I_iter = &I;
00305     do {
00306       auto prev = I_iter->getPrevNode();
00307       if (prev) {
00308         if (prev->getDebugLoc()) {
00309           DILocLine = prev->getDebugLoc()->getLine();
00310           DIFile = prev->getDebugLoc()->getFilename();
00311           resolvedLine = true;
00312           return;
00313         } // End check for getDebugLoc
00314         I_iter = I_iter->getPrevNode();
00315       } else {
00316         break;
00317       } // End check for prev
00318     } while (!resolvedLine);
00319     /* Search forward */
00320     if (!resolvedLine) {
00321       llvm::Instruction *I_iter = &I;
00322       do {
00323         auto next = I_iter->getNextNode();
00324         if (next) {
00325           if (next->getDebugLoc()) {
00326             DILocLine = next->getDebugLoc()->getLine();
00327             DIFile = next->getDebugLoc()->getFilename();
00328             resolvedLine = true;
00329             return;
00330           } // End check for getDebugLoc
00331           I_iter = I_iter->getNextNode();
00332         } else {
00333           break;
00334         } // End check for next
00335       } while (!resolvedLine);
00336     } // End check for resolvedLine
00337   }   // End check for getDebugLoc
00338 } // End getLocationInfo
00339
```

```
00340 bool Graph::isVirtualNodeLineNumber(unsigned lineNumber) {
00341   if (lineNumber == graphEntryID || lineNumber == graphExitID) {
00342     return true;
00343   } // End check for Exit and Entry
00344   return false;
00345 } // End isVirtualNode
00346 } // namespace hydrogen_framework
```

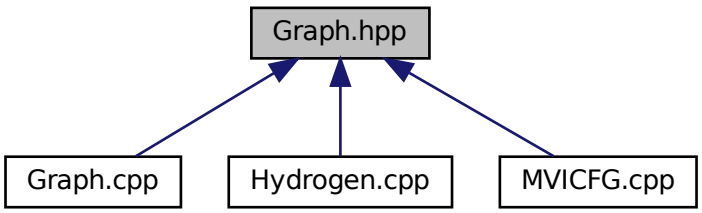
6.15 Graph.hpp File Reference

```
#include <fstream>
#include <iostream>
#include <list>
#include <llvm/IR/CallSite.h>
#include <llvm/IR/DebugInfoMetadata.h>
#include <llvm/IR/InstrTypes.h>
#include <llvm/IR/Intrinsics.h>
#include <regex>
```

Include dependency graph for Graph.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Graph

Functions

- void hydrogen_framework::getLocationInfo (llvm::Instruction &I, unsigned int &DILocLine, std::string &DIFile)

6.15.1 Detailed Description

Author

Ashwin K J

Graph Class: Graph Data-structure

Definition in file Graph.hpp.

6.15.2 Function Documentation

6.15.2.1 getLocationInfo()

```
void hydrogen_framework::getLocationInfo (
             llvm::Instruction & I,
             unsigned int & DILocLine,
             std::string & DIFile )
```

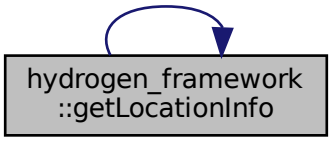
Find the line number and file name of the given LLVM instruction Will return 0 if no information found

Definition at line 296 of file Graph.cpp.

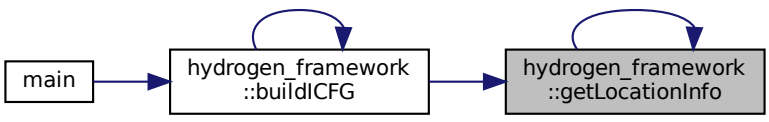
References hydrogen_framework::getLocationInfo().

Referenced by hydrogen_framework::buildICFG(), and hydrogen_framework::getLocationInfo().

Here is the call graph for this function:



Here is the caller graph for this function:



6.16 Graph.hpp

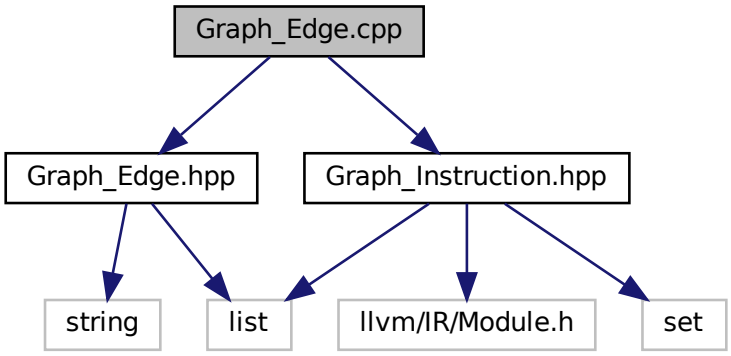
```
00001
00006 #ifndef GRAPH_H
00007 #define GRAPH_H
00008
00009 /* #include "Graph_Function.hpp" */
00010 #include <fstream>
00011 #include <iostream>
00012 #include <list>
00013 #include <llvm/IR/CallSite.h>
00014 #include <llvm/IR/DebugInfoMetadata.h>
00015 #include <llvm/IR/InstrTypes.h>
00016 #include <llvm/IR/Intrinsics.h>
00017 #include <regex>
00018 namespace hydrogen_framework {
00019 /* Forward declaration */
00020 class Graph_Edge;
00021 class Graph_Function;
00022 class Graph_Instruction;
00023 class Graph_Line;
00024
00028 class Graph {
00029 public:
00034   Graph(unsigned ver)
00035       : graphID(0), graphVersion(ver), graphEntryID(std::numeric_limits<unsigned int>::max() - 1),
00036         graphExitID(std::numeric_limits<unsigned int>::max() - 2) {
00037     whiteList.push_back("__isoc99_scanf");
00038     whiteList.push_back("printf");
00039     whiteList.push_back("malloc");
00040     whiteList.push_back("strlen");
00041     whiteList.push_back("strcpy");
00042     whiteList.push_back("strcmp");
00043     whiteList.push_back("free");
00044     whiteList.push_back("getpwnam");
00045     whiteList.push_back("__ctype_b_loc");
00046     whiteList.push_back("tolower");
00047     whiteList.push_back("setpwent");
00048     whiteList.push_back("getpwent");
00049     whiteList.push_back("strchr");
00050     whiteList.push_back("strcasecmp");
00051     whiteList.push_back("perror");
00052     whiteList.push_back("toupper");
00053     whiteList.push_back("malloc");
00054     whiteList.push_back("strlen");
00055     whiteList.push_back("strcpy");
00056     whiteList.push_back("strcmp");
00057     whiteList.push_back("free");
00058     whiteList.push_back("getpwnam");
00059     whiteList.push_back("__ctype_b_loc");
00060     whiteList.push_back("tolower");
00061     whiteList.push_back("setpwent");
00062     whiteList.push_back("getpwent");
00063     whiteList.push_back("strchr");
00064     whiteList.push_back("strcasecmp");
00065     whiteList.push_back("perror");
00066     whiteList.push_back("snprintf");
00067     whiteList.push_back("toupper");
00068   }
00069
00073    Graph() {
00074     graphEdges.clear();
00075     graphFunctions.clear();
00076   }
00077
00081   unsigned getNextID() { return ++graphID; }
00082
00086   unsigned getGraphVersion() { return graphVersion; }
00087
00091   void setGraphVersion(unsigned ver) { graphVersion = ver; }
00092
00096   void pushGraphEdges(Graph_Edge *edge) { graphEdges.push_back(edge); }
00097
00101   void pushGraphFunction(Graph_Function *func);
00102
00106   void addSeqEdges(Graph_Line *line);
00107
00111   void addBranchEdges();
00112
00116   void addVirtualNodes(Graph_Function *func);
00117
00122   void addFunctionCallEdges();
00123
00127   void addEdge(Graph_Instruction *from, Graph_Instruction *to, Graph_Edge *edge);
00128
00132   void printGraph(std::string graphName);
00133
```

```
00138   Graph_Instruction *findMatchedInstruction(llvm::Instruction *matchInst);
00139
00144   Graph_Instruction *findVirtualEntry(std::string funcName);
00145
00150   Graph_Instruction *findVirtualExit(std::string funcName);
00151
00155   std::list<Graph_Function *> getGraphFunctions() { return graphFunctions; }
00156
00160   bool isVirtualNodeLineNumber(unsigned lineNumber);
00161
00165   std::list<Graph_Edge *> getGraphEdges() { return graphEdges; }
00166
00170   std::list<std::string> getWhiteList() { return whiteList; }
00171
00172 private:
00173   unsigned graphID;
00174   unsigned graphVersion;
00175   unsigned graphEntryID;
00176   unsigned graphExitID;
00177   std::list<Graph_Edge *> graphEdges;
00178   std::list<Graph_Function *> graphFunctions;
00179   std::list<std::string> whiteList;
00180 };                                          // End Graph Class
00181
00186 void getLocationInfo(llvm::Instruction &I, unsigned int &DILocLine, std::string &DIFile);
00187 } // namespace hydrogen_framework
00188 #endif
```

6.17 Graph_Edge.cpp File Reference

```
#include "Graph_Edge.hpp"
#include "Graph_Instruction.hpp"
```

Include dependency graph for Graph_Edge.cpp:



6.17.1 Detailed Description

Author

Ashwin K J

Implementing Graph_Edge.hpp

Definition in file Graph_Edge.cpp.

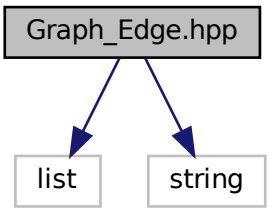
6.18 Graph_Edge.cpp

```
00001
00006 #include "Graph_Edge.hpp"
00007 #include "Graph_Instruction.hpp"
00008 namespace hydrogen_framework {
00009 std::string Graph_Edge::getPrintableEdgeVersions() {
00010   std::string ver;
00011   for (auto v : edgeVersions) {
00012     ver = ver + "V" + std::to_string(v) + ",";
00013   } // End loop for edgveVersions
00014   ver.pop_back();
00015   return ver;
00016 } // End getPrintableEdgeVersions
00017
00018 bool Graph_Edge::isPartOfGraph(unsigned graphVersion) {
00019   auto findVer = std::find_if(std::begin(edgeVersions), std::end(edgeVersions),
00020                               [=](unsigned ver) { return (ver == graphVersion); });
00021   if (findVer != edgeVersions.end()) {
00022     return true;
00023   } // End loop for edgeVersions
00024   return false;
00025 } // End isPartOfGraph
00026 } // namespace hydrogen_framework
```

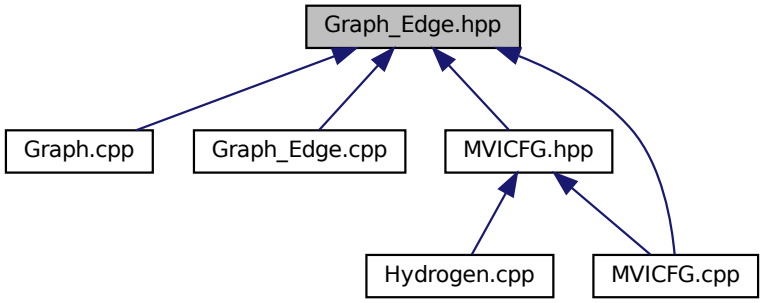
6.19 Graph_Edge.hpp File Reference

```
#include <list>
#include <string>
```

Include dependency graph for Graph_Edge.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Graph_Edge

6.19.1 Detailed Description

Author

Ashwin K J

Graph_Edge Class: Graph Data-structure Edges

Definition in file Graph_Edge.hpp.

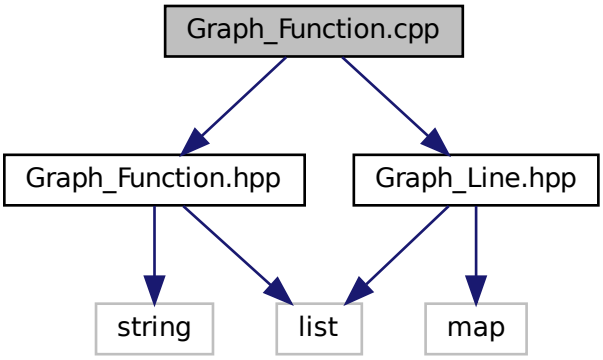
6.20 Graph_Edge.hpp

```
00001
00006 #ifndef GRAPH_EDGE_H
00007 #define GRAPH_EDGE_H
00008
00009 #include <list>
00010 #include <string>
00011 namespace hydrogen_framework {
00012 /* Forward Declaration */
00013 class Graph_Instruction;
00014
00018 class Graph_Edge {
00019 public:
00023   Graph_Edge() : edgeFrom(NULL), edgeTo(NULL), edgeType(edgeTypes::ANY) {}
00024
00028   enum edgeTypes { SEQUENTIAL, BRANCH, CALL, EXTERNAL_CALL, VIRTUAL, MVICFG_ADD, MVICFG_DEL, ANY };
00029
00033   Graph_Edge(Graph_Instruction *from, Graph_Instruction *to, edgeTypes type, unsigned ver)
00034       : edgeFrom(from), edgeTo(to), edgeType(type) {
00035     edgeVersions.push_back(ver);
00036   }
00037
00041    Graph_Edge() { edgeVersions.clear(); }
00042
00046   void setEdgeFrom(Graph_Instruction *I) { edgeFrom = I; }
00047
00051   void setEdgeTo(Graph_Instruction *I) { edgeTo = I; }
00052
00056   void setEdgeType(edgeTypes type) { edgeType = type; }
00057
00061   void pushEdgeVersions(int ver) { edgeVersions.push_back(ver); }
00062
00066   Graph_Instruction *getEdgeFrom() { return edgeFrom; }
00067
00071   Graph_Instruction *getEdgeTo() { return edgeTo; }
00072
00076   edgeTypes getEdgeType() { return edgeType; }
00077
00081   std::list<unsigned> getEdgeVersions() { return edgeVersions; }
00082
00086   std::string getPrintableEdgeVersions();
00087
00092   bool isPartOfGraph(unsigned graphVersion);
00093
00094 private:
00095   Graph_Instruction *edgeFrom;
00096   Graph_Instruction *edgeTo;
00097   edgeTypes edgeType;
00098   std::list<unsigned> edgeVersions;
00099 };                                  // End Graph_Edge Class
00100 } // namespace hydrogen_framework
00101 #endif
```

6.21 Graph_Function.cpp File Reference

```
#include "Graph_Function.hpp"
#include "Graph_Line.hpp"
```

Include dependency graph for Graph_Function.cpp:



6.21.1 Detailed Description

Author

Ashwin K J

Implementing Graph_Function.hpp

Definition in file Graph_Function.cpp.

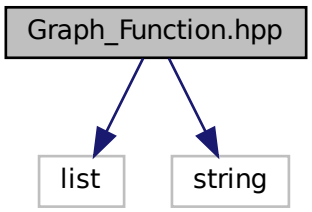
6.22 Graph_Function.cpp

```
00001
00006 #include "Graph_Function.hpp"
00007 #include "Graph_Line.hpp"
00008
00009 namespace hydrogen_framework {
00010 void Graph_Function::pushFunctionLines(Graph_Line *line) {
00011   line->setGraphFunction(this);
00012   functionLines.push_back(line);
00013 } // End pushFunctionLines
00014
00015 void Graph_Function::pushFrontFunctionLines(Graph_Line *line) {
00016   line->setGraphFunction(this);
00017   functionLines.push_front(line);
00018 } // End pushFrontFunctionLines
00019 } // namespace hydrogen_framework
```

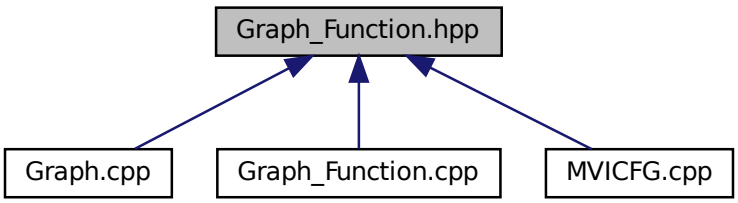
6.23 Graph_Function.hpp File Reference

```
#include <list>
#include <string>
```

Include dependency graph for Graph_Function.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Graph_Function

6.23.1 Detailed Description

Author

Ashwin K J

Graph_Function Class: Graph Data-structure for storing function source lines

Definition in file Graph_Function.hpp.

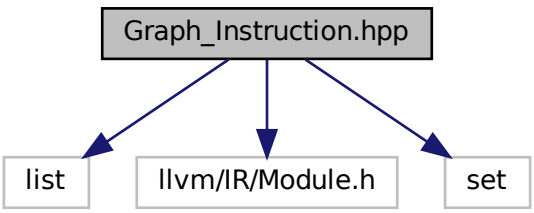
6.24 Graph_Function.hpp

```
00001
00006 #ifndef GRAPH_FUNCTION_H
00007 #define GRAPH_FUNCTION_H
00008
00009 #include <list>
00010 #include <string>
00011 namespace hydrogen_framework {
00012 /* Forward declaration */
00013 class Graph;
00014 class Graph_Line;
00015
00019 class Graph_Function {
00020 public:
00024   Graph_Function(unsigned id) : functionID(id), funcGraph(NULL) {}
00025
00029    Graph_Function() { functionLines.clear(); }
00030
00034   void setFunctionName(std::string name) { functionName = name; }
00035
00039   void setFunctionFile(std::string name) { functionFile = name; }
00040
00044   bool isFunctionFileSet() { return !functionFile.empty(); }
00045
00049   void pushFunctionLines(Graph_Line *line);
00050
00054   void pushFrontFunctionLines(Graph_Line *line);
00055
00059   bool isFunctionLinesEmpty() { return functionLines.empty(); }
00060
00064   std::list<Graph_Line *> getFunctionLines() { return functionLines; }
00065
00069   std::string getFunctionName() { return functionName; }
00070
00074   unsigned getFunctionID() { return functionID; }
00075
00079   std::string getFunctionFile() { return functionFile; }
00080
00084   void setGraph(Graph *graph) { funcGraph = graph; }
00085
00089   Graph *getGraph() { return funcGraph; }
00090
00091 private:
00092   unsigned functionID;
00093   std::string functionName;
00094   std::string functionFile;
00095   std::list<Graph_Line *> functionLines;
00096   Graph *funcGraph;
00097 };                                      // End Graph_Function Class
00098 } // namespace hydrogen_framework
00099 #endif
```

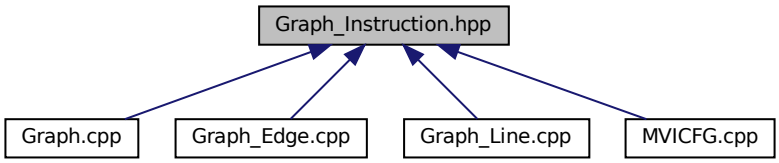
6.25 Graph_Instruction.hpp File Reference

```
#include <list>
#include <llvm/IR/Module.h>
#include <set>
```

Include dependency graph for Graph_Instruction.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Graph_Instruction

6.25.1 Detailed Description

Author

Ashwin K J

Graph_Instruction Class: Graph Data-structure for LLVM instructions

Definition in file Graph_Instruction.hpp.

6.26 Graph_Instruction.hpp

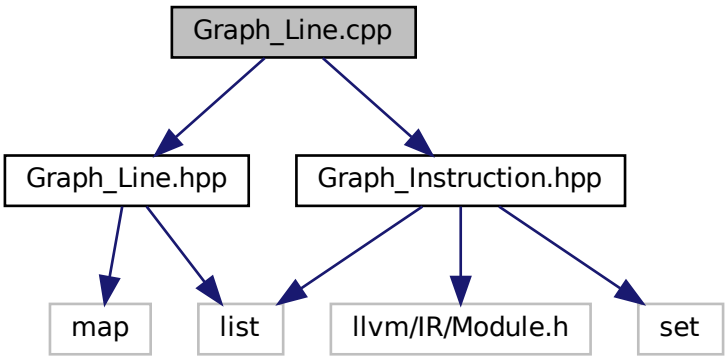
```
00001
00006 #ifndef GRAPH_INSTRUCTION_H
00007 #define GRAPH_INSTRUCTION_H
00008
00009 #include <list>
00010 #include <llvm/IR/Module.h>
00011 #include <set>
00012 namespace hydrogen_framework {
00013 /* Forward declaration */
00014 class Graph_Edge;
00015 class Graph_Line;
00016 class Query;
00017
00021 class Graph_Instruction {
00022 public:
00026   Graph_Instruction() : instructionID(0), instructionPtr(NULL), instructionLine(NULL) {}
00027
00031    Graph_Instruction() {}
00032
00036   void setInstructionID(unsigned ID) { instructionID = ID; }
00037
00041   void setInstructionLabel(std::string label) { instructionLabel = label; }
00042
00046   void setInstructionPtr(llvm::Instruction *I) { instructionPtr = I; }
00047
00051   void pushEdgeInstruction(Graph_Edge *edge) { instructionEdges.push_back(edge); }
00052
00056   std::string getInstructionLabel() { return instructionLabel; }
00057
00061   unsigned getInstructionID() { return instructionID; }
00062
00067   llvm::Instruction *getInstructionPtr() { return instructionPtr; }
00068
00072   std::list<Graph_Edge *> getInstructionEdges() { return instructionEdges; }
00073
00077   void setGraphLine(Graph_Line *line) { instructionLine = line; }
00078
```

```
00082   Graph_Line *getGraphLine() { return instructionLine; }
00083
00087   std::set<Query *> getInstructionVisitedQueries() { return instructionVisitedQueries; }
00088
00092   void insertInstructionVisitedQueries(Query *q) { instructionVisitedQueries.insert(q); }
00093
00094 private:
00095   unsigned instructionID;
00096   std::string instructionLabel;
00097   llvm::Instruction *instructionPtr;
00098   std::list<Graph_Edge *> instructionEdges;
00099   Graph_Line *instructionLine;
00100   std::set<Query *> instructionVisitedQueries;
00101 };                                          // End Graph_Instruction Class
00102 } // namespace hydrogen_framework
00103 #endif
```

6.27 Graph_Line.cpp File Reference

```
#include "Graph_Line.hpp"
#include "Graph_Instruction.hpp"
```

Include dependency graph for Graph_Line.cpp:



6.27.1 Detailed Description

Author

Ashwin K J

Implementing Graph_Line.hpp

Definition in file Graph_Line.cpp.

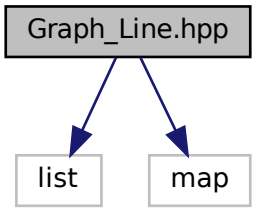
6.28 Graph_Line.cpp

```
00001
00006 #include "Graph_Line.hpp"
00007 #include "Graph_Instruction.hpp"
00008
00009 namespace hydrogen_framework {
00010
00011 void Graph_Line::setLineNumber(unsigned Version, unsigned line) {
00012   lineNumber.insert(std::pair<unsigned, unsigned>(Version, line));
00013 } // End setLineNumber
00014
00015 void Graph_Line::pushLineInstruction(Graph_Instruction *inst) {
00016   inst->setGraphLine(this);
00017   lineInstructions.push_back(inst);
00018 } // End pushLineInstruction;
00019
00020 unsigned Graph_Line::getLineNumber(unsigned Version) {
00021   auto searchLine = lineNumber.find(Version);
00022   if (searchLine != lineNumber.end()) {
00023     return searchLine->second;
00024   } // End check for searchLine
00025   return 0;
00026 } // End getLineNumber
00027 } // namespace hydrogen_framework
```

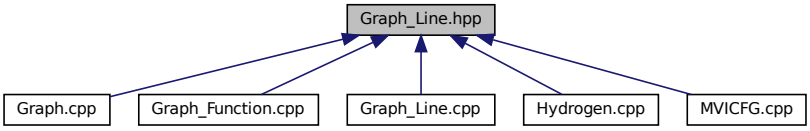
6.29 Graph_Line.hpp File Reference

```
#include <list>
#include <map>
```

Include dependency graph for Graph_Line.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Graph_Line

6.29.1 Detailed Description

Author

Ashwin K J

Graph_Line Class: Graph Data-structure for source line

Definition in file Graph_Line.hpp.

6.30 Graph_Line.hpp

```
00001
00006 #ifndef GRAPH_LINE_H
00007 #define GRAPH_LINE_H
00008
00009 #include <list>
00010 #include <map>
00011 namespace hydrogen_framework {
00012 /* Forward declaration */
00013 class Graph_Function;
00014 class Graph_Instruction;
00015
00019 class Graph_Line {
00020 public:
00024   Graph_Line(unsigned Version) : lineFunction(NULL), lineGraphVersion(Version) {}
00025
00029    Graph_Line() { lineInstructions.clear(); }
00030
00035   unsigned getLineNumber(unsigned Version);
00036
00040   void setLineNumber(unsigned Version, unsigned line);
00041
00045   bool isLineInstructionEmpty() { return lineInstructions.empty(); }
00046
00050   void pushLineInstruction(Graph_Instruction *inst);
00051
00055   std::list<Graph_Instruction *> getLineInstructions() { return lineInstructions; }
00056
00060   void setGraphFunction(Graph_Function *func) { lineFunction = func; }
00061
00065   Graph_Function *getGraphFunction() { return lineFunction; }
00066
00070   unsigned getLineGraphVersion() { return lineGraphVersion; }
00071
00072 private:
00073   std::map<unsigned, unsigned> lineNumber;
00074   std::list<Graph_Instruction *> lineInstructions;
00075   Graph_Function *lineFunction;
00076   unsigned lineGraphVersion;
00077 };                                          // End Graph_Line Class
00078 } // namespace hydrogen_framework
00079 #endif
```

6.31 Hydrogen.cpp File Reference

```
#include "Diff_Mapping.hpp"
#include "Get_Input.hpp"
#include "Graph.hpp"
#include "Graph_Line.hpp"
#include "MVICFG.hpp"
#include "Module.hpp"
#include <chrono>
```

Include dependency graph for Hydrogen.cpp:

Functions

- int main (int argc, char ∗argv[ ])

6.31.1 Detailed Description

Author

Ashwin K J

Definition in file Hydrogen.cpp.

6.31.2 Function Documentation

6.31.2.1 main()

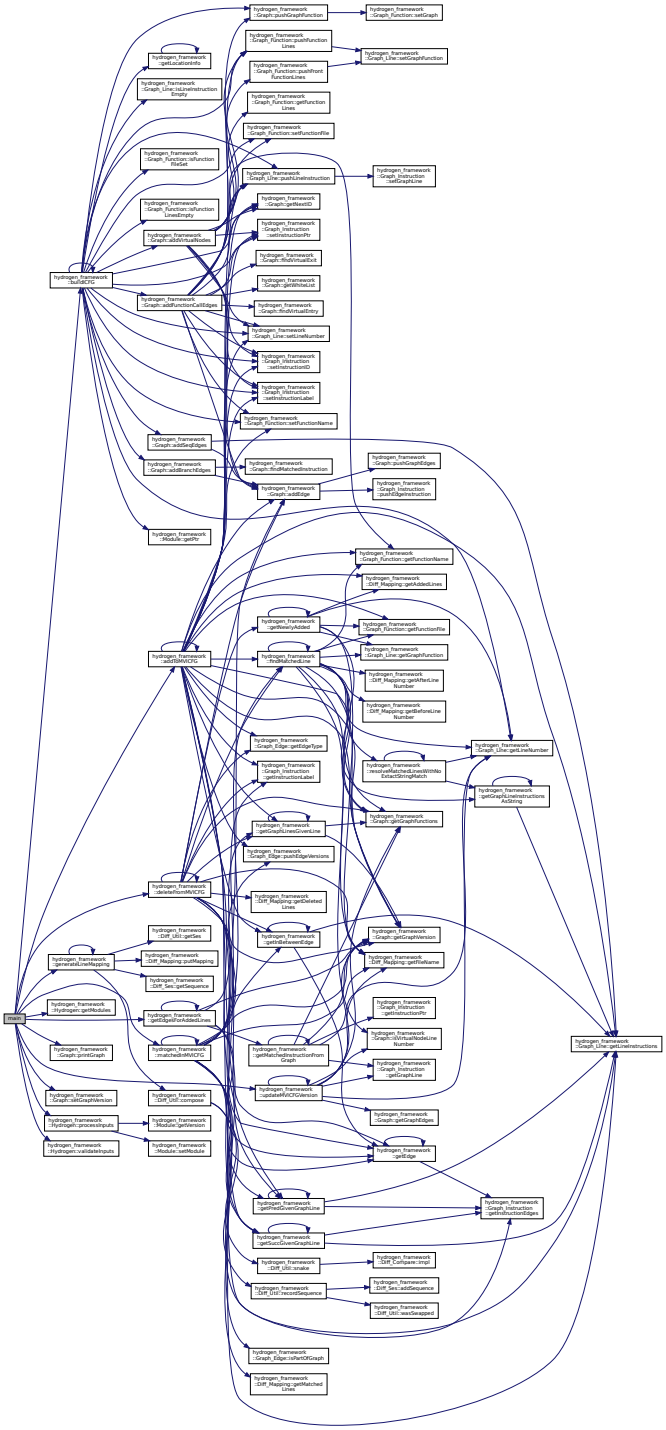
```
int main (
            int argc,
            char * argv[] )
```

Main function <Map From ICFG Graph_Line to MVICFG Graph_Line

Definition at line 18 of file Hydrogen.cpp.

References hydrogen_framework::addToMVICFG(), hydrogen_framework::buildICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::generateLineMapping(), hydrogen_framework::getEdgesForAddedLines(), hydrogen_framework::Hydrogen::ge hydrogen_framework::matchedInMVICFG(), hydrogen_framework::Graph::printGraph(), hydrogen_framework::Hydrogen::processInp hydrogen_framework::Graph::setGraphVersion(), hydrogen_framework::updateMVICFGVersion(), and hydrogen_framework::Hydroge

Here is the call graph for this function:



6.32 Hydrogen.cpp

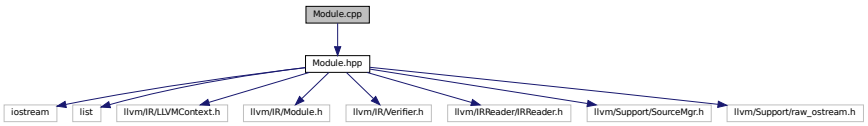
```
00001
00005 #include "Diff_Mapping.hpp"
00006 #include "Get_Input.hpp"
00007 #include "Graph.hpp"
00008 #include "Graph_Line.hpp"
00009 #include "MVICFG.hpp"
00010 #include "Module.hpp"
00011 #include <chrono>
```

```
00012
00013 using namespace hydrogen_framework;
00014
00018 int main(int argc, char *argv[]) {
00019   /* Getting the input */
00020   if (argc < 2) {
00021     std::cerr « "Insufficient arguments\n"
00022              « "The correct format is as follows:\n"
00023              « "<Path-to-Module1> <Path-to-Module2> .. <Path-to-ModuleN> :: "
00024              « "<Path-to-file1-for-Module1> .. <Path-to-fileN-for-Module1> :: "
00025              « "<Path-to-file2-for-Module2> .. <Path-to-fileN-for-Module2> ..\n"
00026              « "Note that '::' is the demarcation\n";
00027     return 1;
00028   } // End check for min argument
00029   Hydrogen framework;
00030   if (!framework.validateInputs(argc, argv)) {
00031     return 2;
00032   } // End check for valid Input
00033   if (!framework.processInputs(argc, argv)) {
00034     return 3;
00035   } // End check for processing Inputs
00036   std::list<Module *> mod = framework.getModules();
00037   /* Create ICFG */
00038   unsigned graphVersion = 1;
00039   Module *firstMod = mod.front();
00040   Graph *MVICFG = buildICFG(firstMod, graphVersion);
00041   /* Start timer */
00042   auto mvicfgStart = std::chrono::high_resolution_clock::now();
00043   /* Create MVICFG */
00044   for (auto iterModule = mod.begin(), iterModuleEnd = mod.end(); iterModule != iterModuleEnd;
      ++iterModule) {
00045     auto iterModuleNext = std::next(iterModule);
00046     /* Proceed as long as there is a next module */
00047     if (iterModuleNext != iterModuleEnd) {
00048       /* Container for added and deleted MVICFG lines */
00049       std::list<Graph_Line *> addedLines;
00050       std::list<Graph_Line *> deletedLines;
00051       std::map<Graph_Line *, Graph_Line *> matchedLines;
00052       std::list<Diff_Mapping> diffMap = generateLineMapping(*iterModule, *iterModuleNext);
00053       Graph *ICFG = buildICFG(*iterModuleNext, ++graphVersion);
00054       for (auto iter : diffMap) {
00055         /* iter.printFileInfo(); */
00056         std::list<Graph_Line *> iterAdd = addToMVICFG(MVICFG, ICFG, iter, graphVersion);
00057         std::list<Graph_Line *> iterDel = deleteFromMVICFG(MVICFG, ICFG, iter, graphVersion);
00058         std::map<Graph_Line *, Graph_Line *> iterMatch = matchedInMVICFG(MVICFG, ICFG, iter,
     graphVersion);
00059         addedLines.insert(addedLines.end(), iterAdd.begin(), iterAdd.end());
00060         deletedLines.insert(deletedLines.end(), iterDel.begin(), iterDel.end());
00061         matchedLines.insert(iterMatch.begin(), iterMatch.end());
00062       } // End loop for diffMap
00063       /* Update Map Edges */
00064       getEdgesForAddedLines(MVICFG, ICFG, addedLines, diffMap, graphVersion);
00065       /* Update the matched lines to get new temporary variable mapping for old lines */
00066       updateMVICFGVersion(MVICFG, addedLines, deletedLines, diffMap, graphVersion);
00067       /* Update Map Version */
00068       MVICFG->setGraphVersion(graphVersion);
00069     } // End check for iterModuleEnd
00070   } // End loop for Module
00071   /* Stop timer */
00072   auto mvicfgStop = std::chrono::high_resolution_clock::now();
00073   auto mvicfgBuildTime = std::chrono::duration_cast<std::chrono::milliseconds>(mvicfgStop -
     mvicfgStart);
00074   MVICFG->printGraph("MVICFG");
00075   std::cout « "Finished Building MVICFG in " « mvicfgBuildTime.count() « "ms\n";
00076   /* Write output to file */
00077   std::ofstream rFile("Result.txt", std::ios::trunc);
00078   if (!rFile.is_open()) {
00079     std::cerr « "Unable to open file for printing the output\n";
00080     return 5;
00081   } // End check for Result file
00082   rFile « "Input Args:\n";
00083   for (auto i = 0; i < argc; ++ i) {
00084     rFile « argv[i] « "  ";
00085   } // End loop for writing arguments
00086   rFile « "\n";
00087   rFile « "Finished Building MVICFG in " « mvicfgBuildTime.count() « "ms\n";
00088   rFile.close();
00089   return 0;
00090 } // End main
```

6.33 Module.cpp File Reference

```
#include "Module.hpp"
```

Include dependency graph for Module.cpp:



6.33.1 Detailed Description

Author

Ashwin K J

Implementing Module.hpp

Definition in file Module.cpp.

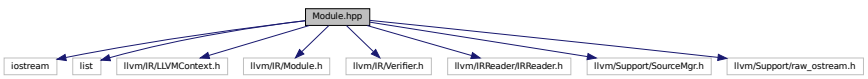
6.34 Module.cpp

```
00001
00006 #include "Module.hpp"
00007 namespace hydrogen_framework {
00008 bool Module::setModule(int ver, std::string file) {
00009   modVersion = ver;
00010   llvm::StringRef modulePath(file);
00011   llvm::SMDiagnostic error;
00012   modPtr = llvm::parseIRFile(modulePath, error, modContext);
00013   /* Parsing Error handling */
00014   if (!modPtr) {
00015     std::string errorMessage;
00016     llvm::raw_string_ostream output(errorMessage);
00017     /* error.print("Error in parsing the file ", output); */
00018     std::cerr « "Error in parsing the " « file « "\n";
00019     return false;
00020   } // End check for modPtr
00021   /* Verifying Module */
00022   if (llvm::verifyModule(*modPtr, &llvm::errs()) != 0) {
00023     std::cerr « "Error in verifying the Module : " « file « "\n";
00024     return false;
00025   } // End check for verifyModule
00026   return true;
00027 } // End setModule
00028 } // namespace hydrogen_framework
```

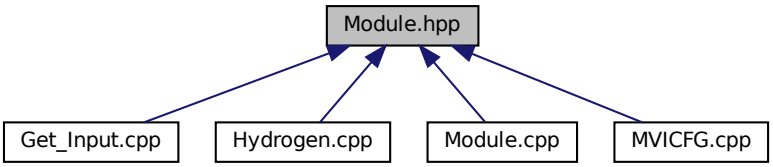
6.35 Module.hpp File Reference

```
#include <iostream>
#include <list>
#include <llvm/IR/LLVMContext.h>
#include <llvm/IR/Module.h>
#include <llvm/IR/Verifier.h>
#include <llvm/IRReader/IRReader.h>
#include <llvm/Support/SourceMgr.h>
#include <llvm/Support/raw_ostream.h>
```

Include dependency graph for Module.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class hydrogen_framework::Module

6.35.1 Detailed Description

Author

Ashwin K J

Module class: Managing LLVM Module

Definition in file Module.hpp.

6.36 Module.hpp

```
00001
00006 #ifndef MODULE_H
00007 #define MODULE_H
00008
00009 #include <iostream>
00010 #include <list>
00011 #include <llvm/IR/LLVMContext.h>
00012 #include <llvm/IR/Module.h>
00013 #include <llvm/IR/Verifier.h>
00014 #include <llvm/IRReader/IRReader.h>
00015 #include <llvm/Support/SourceMgr.h>
00016 #include <llvm/Support/raw_ostream.h>
00017
00018 namespace hydrogen_framework {
00022 class Module {
00023 public:
00028   Module() { modVersion = 0; }
00029
00033    Module() { modFiles.clear(); }
00034
00039   bool setModule(int ver, std::string file);
00040
00044   void setFiles(std::list<std::string> file) { modFiles.swap(file); }
00045
00049   int getVersion() { return modVersion; }
00050
00054   std::unique_ptr<llvm::Module> &getPtr() { return modPtr; }
00055
00059   std::list<std::string> getFiles() { return modFiles; }
00060
00061 private:
00062   int modVersion;
00063   llvm::LLVMContext modContext;
00064   std::unique_ptr<llvm::Module> modPtr;
00065   std::list<std::string> modFiles;
00066 };                                      // End module class
00067 } // namespace hydrogen_framework
00068 #endif
```

6.37 MVICFG.cpp File Reference

```
#include "MVICFG.hpp"
#include "Diff_Mapping.hpp"
#include "Graph.hpp"
#include "Graph_Edge.hpp"
#include "Graph_Function.hpp"
#include "Graph_Instruction.hpp"
#include "Graph_Line.hpp"
#include "Module.hpp"
```

Include dependency graph for MVICFG.cpp:

Functions

- std::list< Graph_Line * > hydrogen_framework::addToMVICFG (Graph *MVICFG, Graph *ICFG, Diff_↩ Mapping diff, unsigned Version)
- Graph * hydrogen_framework::buildICFG (Module *mod, unsigned graphVersion)
- std::list< Graph_Line * > hydrogen_framework::deleteFromMVICFG (Graph *MVICFG, Graph *ICFG, Diff↩ _Mapping diff, unsigned Version)
- Graph_Line * hydrogen_framework::findMatchedLine (Graph_Line *t, Graph *matchTo, Graph *matchFrom, Diff_Mapping diff)
- std::list< Diff_Mapping > hydrogen_framework::generateLineMapping (Module *firstMod, Module *secondMod)
- Graph_Edge * hydrogen_framework::getEdge (Graph_Instruction *fromNode, Graph_Instruction *toNode, Graph_Edge::edgeTypes type)
- void hydrogen_framework::getEdgesForAddedLines (Graph *MVICFG, Graph *ICFG, std::list< Graph_Line * > addedLines, std::list< Diff_Mapping > diffMap, unsigned Version)
- std::string hydrogen_framework::getGraphLineInstructionsAsString (Graph_Line *line)
- std::list< Graph_Line * > hydrogen_framework::getGraphLinesGivenLine (Graph *graph, long long lineNo, std::string fileName)
- Graph_Edge * hydrogen_framework::getInBetweenEdge (Graph_Line *fromLine, Graph_Line *toLine)
- Graph_Instruction * hydrogen_framework::getMatchedInstructionFromGraph (Graph *graphToMatch, Graph_Instruction *instToMatch)
- Graph_Line * hydrogen_framework::getNewlyAdded (Graph *MVICFG, Graph *ICFG, Graph_Line *new↩ Line, Diff_Mapping diff)
- std::list< Graph_Line * > hydrogen_framework::getPredGivenGraphLine (Graph_Line *line)
- std::list< Graph_Line * > hydrogen_framework::getSuccGivenGraphLine (Graph_Line *line)
- std::map< Graph_Line *, Graph_Line * > hydrogen_framework::matchedInMVICFG (Graph *MVICFG, Graph *ICFG, Diff_Mapping diff, unsigned Version)
- Graph_Line * hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch (std::list< Graph_Line * > matchedLines, std::string lineFromString, unsigned int graphVersion)
- void hydrogen_framework::updateMVICFGVersion (Graph *MVICFG, std::list< Graph_Line * > addedLines, std::list< Graph_Line * > deletedLines, std::list< Diff_Mapping > diffMap, unsigned Version)

6.37.1 Detailed Description

Author

Ashwin K J

Implementing MVICFG.hpp

Definition in file MVICFG.cpp.

6.37.2 Function Documentation

6.37.2.1 addToMVICFG()

```
std::list< Graph_Line * > hydrogen_framework::addToMVICFG (
            Graph * MVICFG,
            Graph * ICFG,
            Diff_Mapping diff,
            unsigned Version )
```

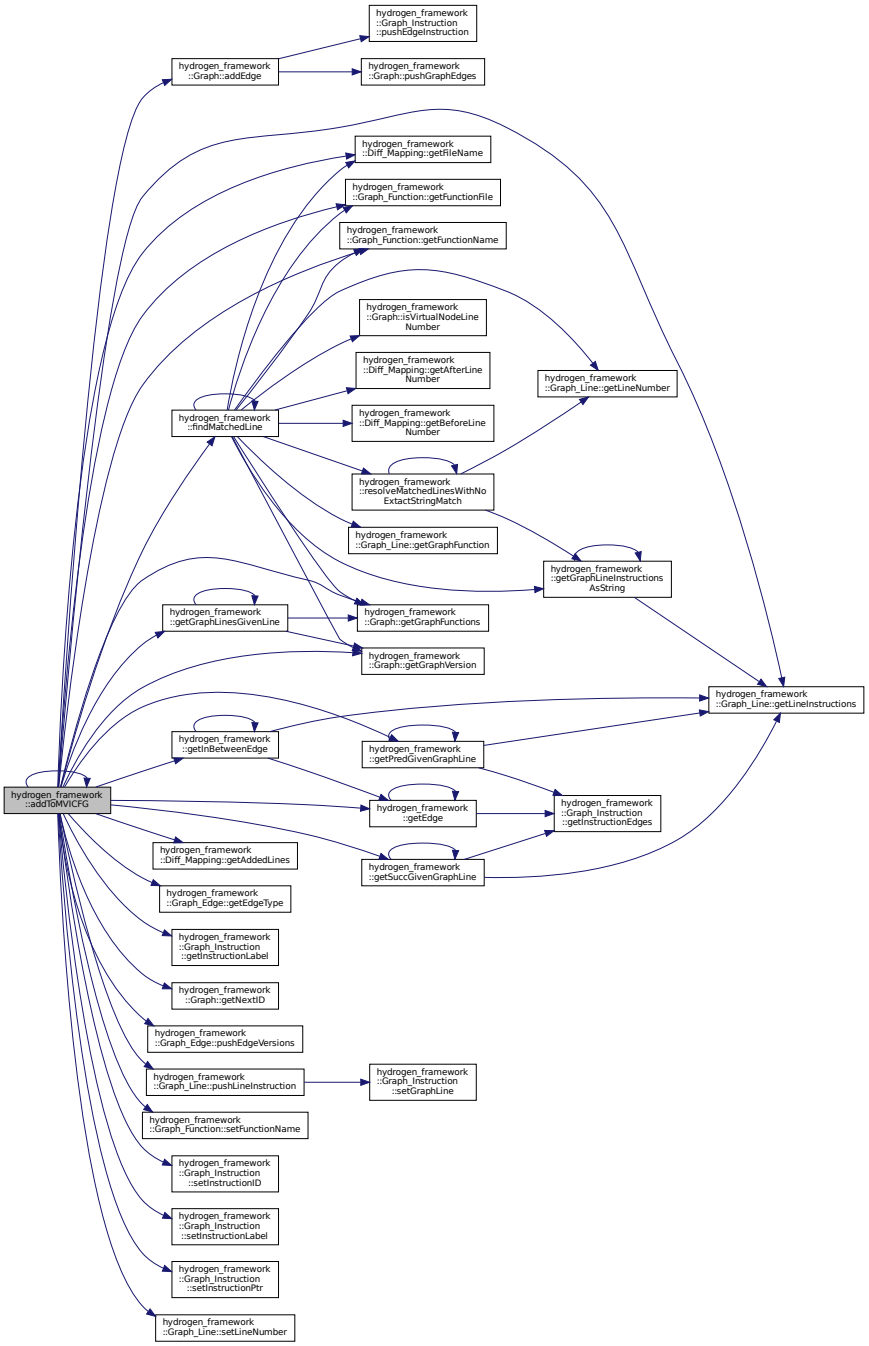
Add nodes to MVICFG and returns the added MVICFG lines

Definition at line 349 of file MVICFG.cpp.

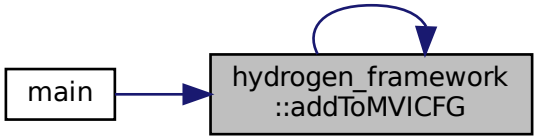
References hydrogen_framework::Graph::addEdge(), hydrogen_framework::addToMVICFG(), hydrogen_framework::findMatchedLine
hydrogen_framework::Diff_Mapping::getAddedLines(), hydrogen_framework::getEdge(), hydrogen_framework::Graph_Edge::getEdge
hydrogen_framework::Diff_Mapping::getFileName(), hydrogen_framework::Graph_Function::getFunctionFile(),
hydrogen_framework::Graph_Function::getFunctionName(), hydrogen_framework::Graph::getGraphFunctions(),
hydrogen_framework::getGraphLinesGivenLine(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::getInBetwe
hydrogen_framework::Graph_Instruction::getInstructionLabel(), hydrogen_framework::Graph_Line::getLineInstructions(),
hydrogen_framework::Graph::getNextID(), hydrogen_framework::getPredGivenGraphLine(), hydrogen_framework::getSuccGivenGra
hydrogen_framework::Graph_Edge::pushEdgeVersions(), hydrogen_framework::Graph_Line::pushLineInstruction(),
hydrogen_framework::Graph_Function::setFunctionName(), hydrogen_framework::Graph_Instruction::setInstructionID(),
hydrogen_framework::Graph_Instruction::setInstructionLabel(), hydrogen_framework::Graph_Instruction::setInstructionPtr(),
and hydrogen_framework::Graph_Line::setLineNumber().

Referenced by hydrogen_framework::addToMVICFG(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.2 buildICFG()

```
Graph * hydrogen_framework::buildICFG (
            Module * mod,
            unsigned graphVersion )
```

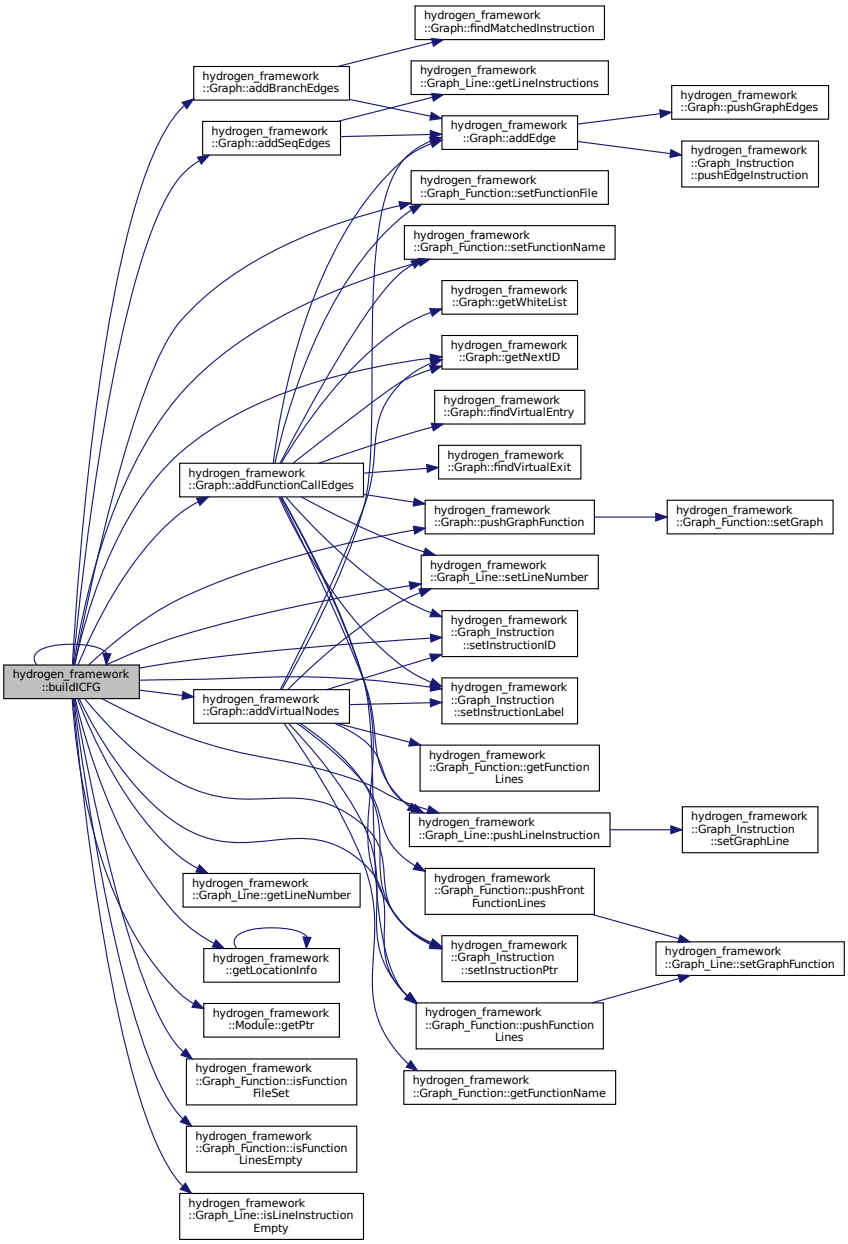
Build ICFG for the given module

Definition at line 15 of file MVICFG.cpp.

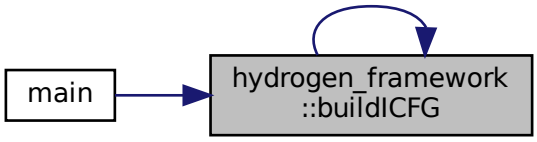
References hydrogen_framework::Graph::addBranchEdges(), hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::Graph::addSeqEdges(), hydrogen_framework::Graph::addVirtualNodes(), hydrogen_framework::buildICFG(), hydrogen_framework::Graph_Line::getLineNumber(), hydrogen_framework::getLocationInfo(), hydrogen_framework::Graph::getNextI hydrogen_framework::Module::getPtr(), hydrogen_framework::Graph_Function::isFunctionFileSet(), hydrogen_framework::Graph_Fu hydrogen_framework::Graph_Line::isLineInstructionEmpty(), hydrogen_framework::Graph_Function::pushFunctionLines(), hydrogen_framework::Graph::pushGraphFunction(), hydrogen_framework::Graph_Line::pushLineInstruction(), hydrogen_framework::Graph_Function::setFunctionFile(), hydrogen_framework::Graph_Function::setFunctionName(), hydrogen_framework::Graph_Instruction::setInstructionID(), hydrogen_framework::Graph_Instruction::setInstructionLabel(), hydrogen_framework::Graph_Instruction::setInstructionPtr(), and hydrogen_framework::Graph_Line::setLineNumber().

Referenced by hydrogen_framework::buildICFG(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.3 deleteFromMVICFG()

```
std::list< Graph_Line * > hydrogen_framework::deleteFromMVICFG (
              Graph * MVICFG,
              Graph * ICFG,
              Diff_Mapping diff,
              unsigned Version )
```

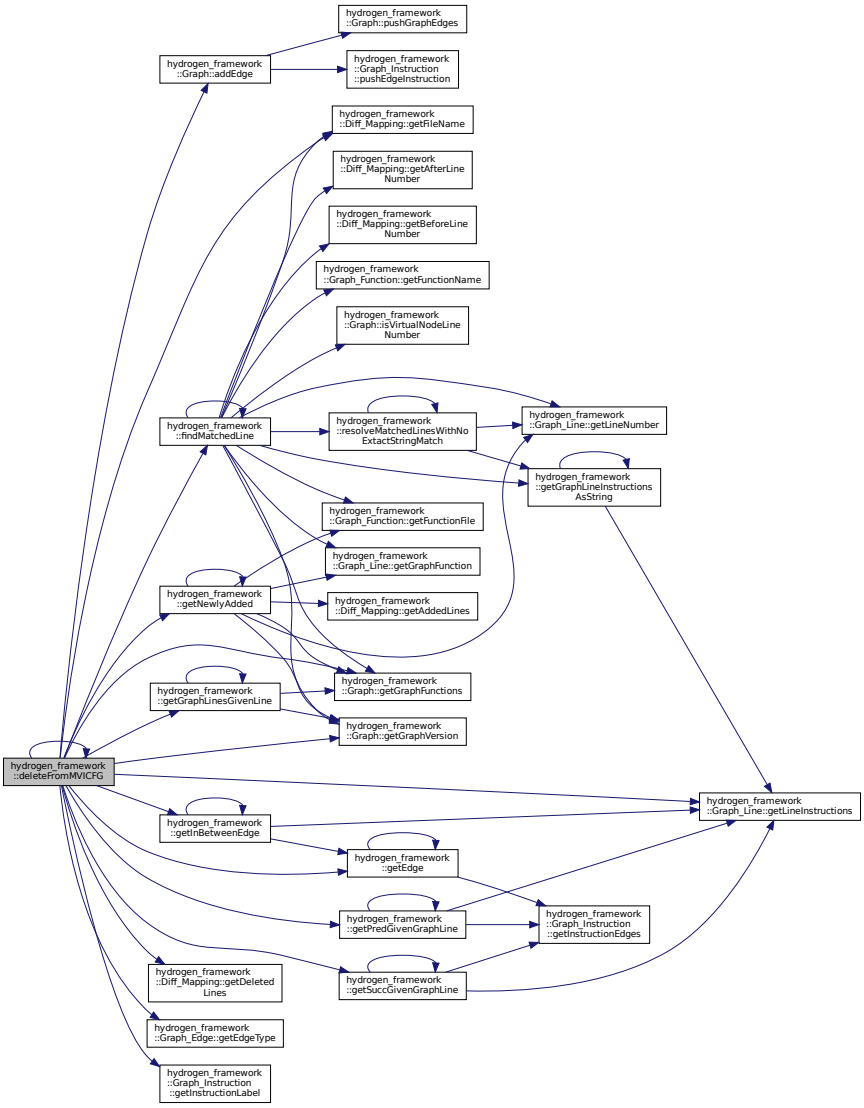
Mark deleted nodes in MVICFG and returns the deleted MVICFG lines

Definition at line 556 of file MVICFG.cpp.

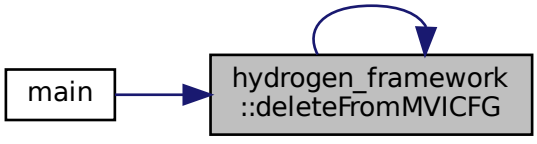
References hydrogen_framework::Graph::addEdge(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::findMatche
hydrogen_framework::Diff_Mapping::getDeletedLines(), hydrogen_framework::getEdge(), hydrogen_framework::Graph_Edge::getEdg
hydrogen_framework::Diff_Mapping::getFileName(), hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::getGr
hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::getInBetweenEdge(), hydrogen_framework::Graph_Instructio
hydrogen_framework::Graph_Line::getLineInstructions(), hydrogen_framework::getNewlyAdded(), hydrogen_framework::getPredGive
and hydrogen_framework::getSuccGivenGraphLine().

Referenced by hydrogen_framework::deleteFromMVICFG(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.4 findMatchedLine()

```
Graph_Line * hydrogen_framework::findMatchedLine (
             Graph_Line * t,
             Graph * matchTo,
             Graph * matchFrom,
             Diff_Mapping diff )
```

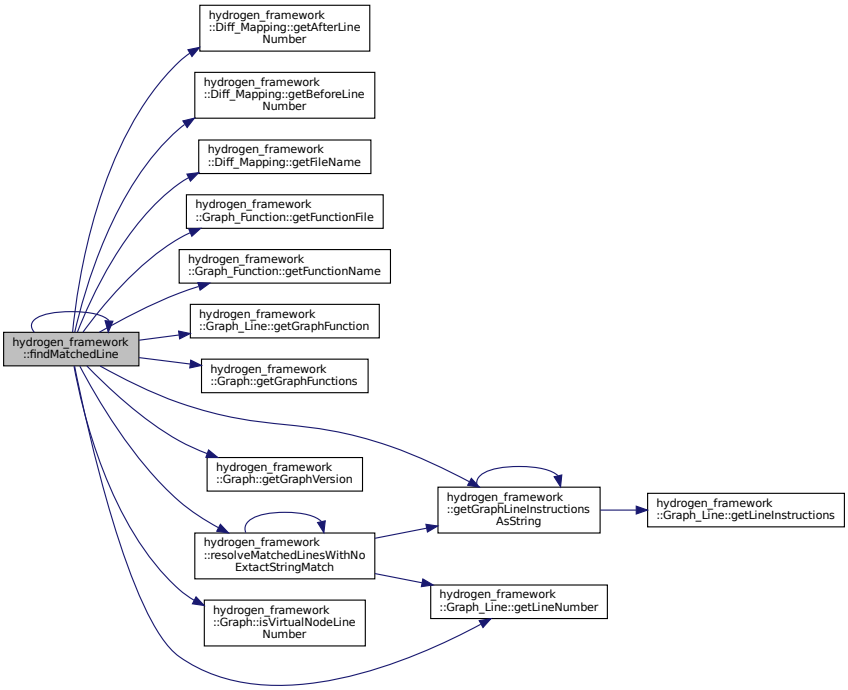
Find matched Node Returns NULL if no match found Always make sure to check that the Graph_Line is from the diff being used

Definition at line 233 of file MVICFG.cpp.

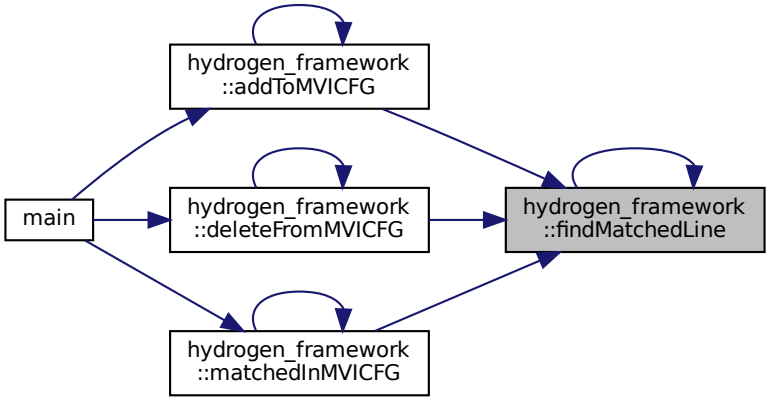
References hydrogen_framework::findMatchedLine(), hydrogen_framework::Diff_Mapping::getAfterLineNumber(), hydrogen_framework::Diff_Mapping::getBeforeLineNumber(), hydrogen_framework::Diff_Mapping::getFileName(), hydrogen_framework::Graph_Function::getFunctionFile(), hydrogen_framework::Graph_Function::getFunctionName(), hydrogen_framework::Graph_Line::getGraphFunction(), hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::getGraphLineInstructionsAsString(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Line::getLineNumber(), hydrogen_framework::Graph::isVirtualNodeLineNumber(), and hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::findMatch and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.5 generateLineMapping()

```
std::list< Diff_Mapping > hydrogen_framework::generateLineMapping (
            Module * firstMod,
            Module * secondMod )
```

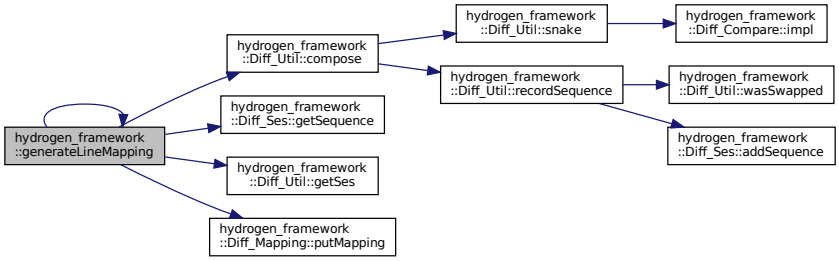
Generate Line Mappings between two modules

Definition at line 75 of file MVICFG.cpp.

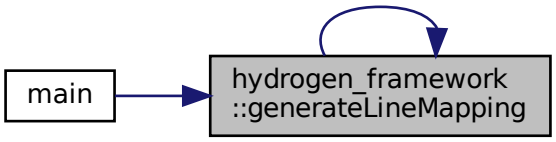
References hydrogen_framework::Diff_Util::compose(), hydrogen_framework::generateLineMapping(), hydrogen_framework::Diff_Ses
hydrogen_framework::Diff_Util::getSes(), and hydrogen_framework::Diff_Mapping::putMapping().

Referenced by hydrogen_framework::generateLineMapping(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.6 getEdge()

```
Graph_Edge * hydrogen_framework::getEdge (
             Graph_Instruction * fromNode,
             Graph_Instruction * toNode,
             Graph_Edge::edgeTypes type )
```

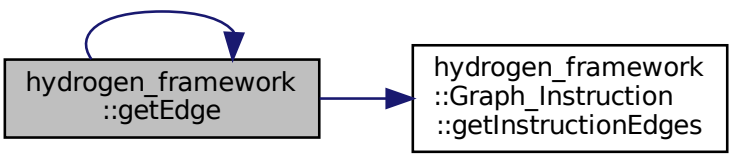
Get the edge between two given nodes Returns NULL if no match found

Definition at line 300 of file MVICFG.cpp.

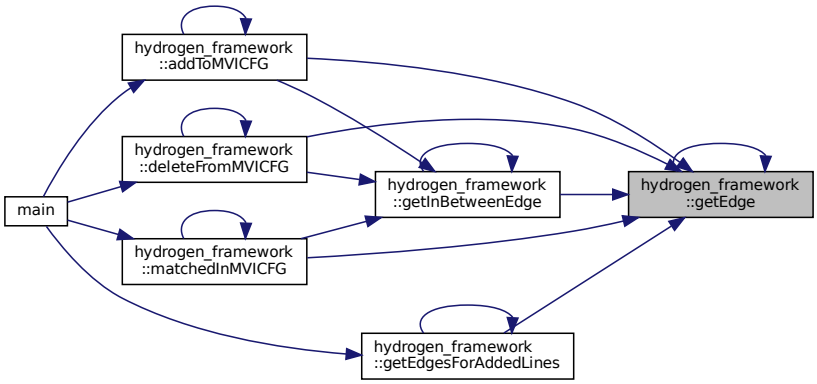
References hydrogen_framework::getEdge(), and hydrogen_framework::Graph_Instruction::getInstructionEdges().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getEdge(), hydrogen_framework::getEdgesForAddedLines(), hydrogen_framework::getInBetweenEdge(), and hydrogen_framework::matchedInM

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.7 getEdgesForAddedLines()

```
void hydrogen_framework::getEdgesForAddedLines (
            Graph * MVICFG,
            Graph * ICFG,
            std::list< Graph_Line * > addedLines,
            std::list< Diff_Mapping > diffMap,
            unsigned Version )
```

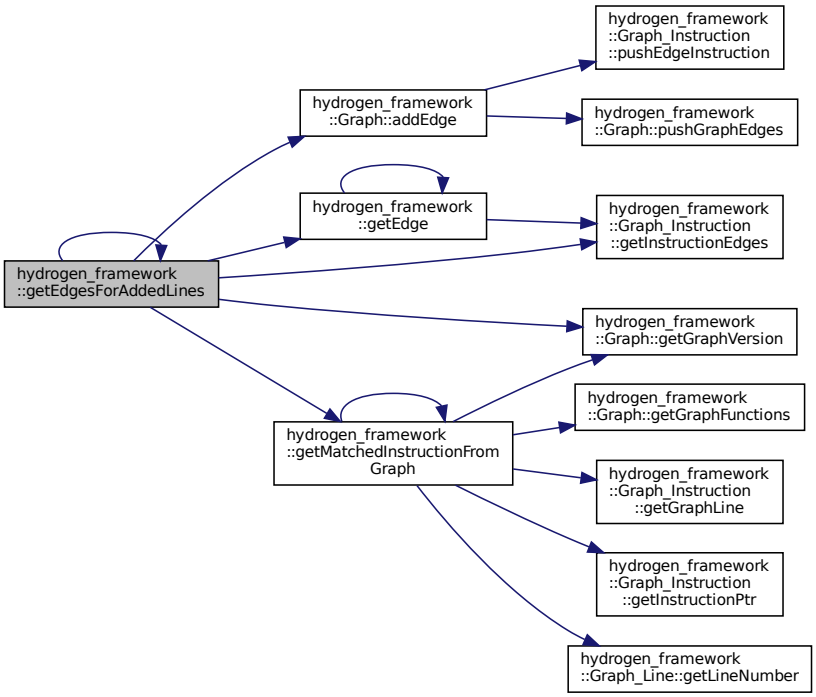
Import edges from ICFG instruction for added Graph_Line

Definition at line 523 of file MVICFG.cpp.

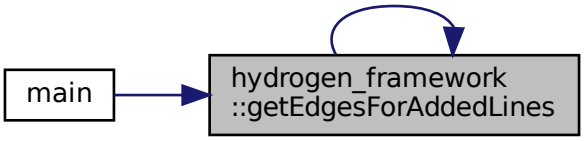
References hydrogen_framework::Graph::addEdge(), hydrogen_framework::getEdge(), hydrogen_framework::getEdgesForAddedLine
hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Instruction::getInstructionEdges(),
and hydrogen_framework::getMatchedInstructionFromGraph().

Referenced by hydrogen_framework::getEdgesForAddedLines(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.8 getGraphLineInstructionsAsString()

```
std::string hydrogen_framework::getGraphLineInstructionsAsString (
            Graph_Line * line )
```

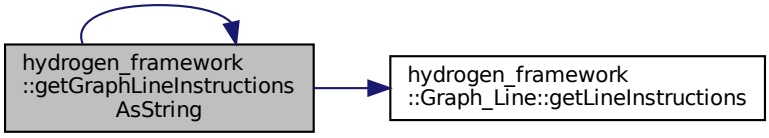
Get the OpCode of the Instructions in a Graph_Line as String in the order in which they appear Returns empty string if none of the Graph_Instruction had Instruction_Ptr

Definition at line 185 of file MVICFG.cpp.

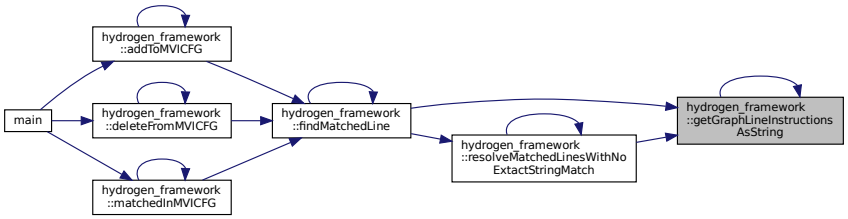
References hydrogen_framework::getGraphLineInstructionsAsString(), and hydrogen_framework::Graph_Line::getLineInstructions().

Referenced by hydrogen_framework::findMatchedLine(), hydrogen_framework::getGraphLineInstructionsAsString(), and hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.9 getGraphLinesGivenLine()

```
std::list< Graph_Line * > hydrogen_framework::getGraphLinesGivenLine (
            Graph * graph,
            long long lineNo,
            std::string fileName )
```

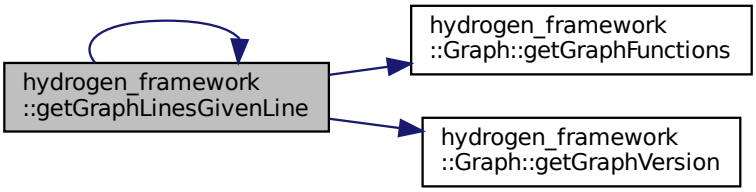
Get Graph_Line(s) from given source line Returns empty list if no Graph_Line is not found

Definition at line 139 of file MVICFG.cpp.

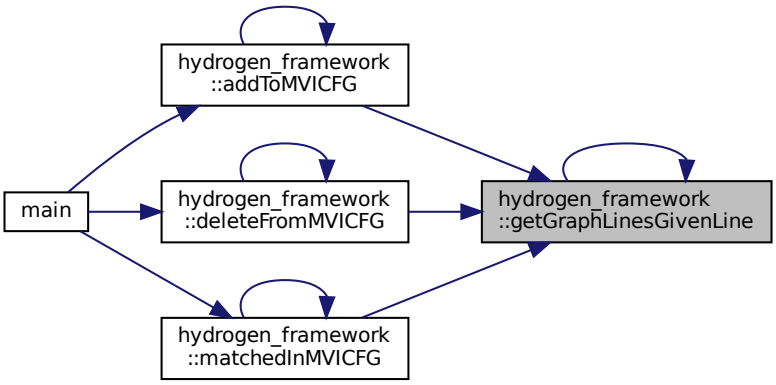
References hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::getGraphLinesGivenLine(), and hydrogen_framework::Graph::getGraphVersion().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getGraphl and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.10 getInBetweenEdge()

```
Graph_Edge * hydrogen_framework::getInBetweenEdge (
             Graph_Line * fromLine,
             Graph_Line * toLine )
```

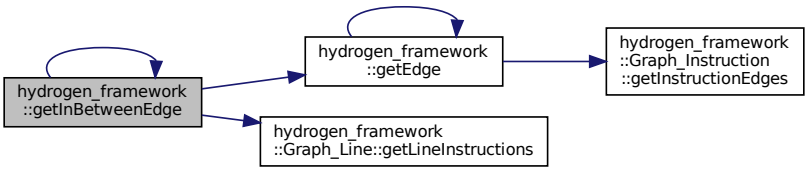
Get the first edge between two Graph_Line It does reverse propagation for Instructions of fromLine and forward propagation for toLine Instructions Used only when getEdge fails to find an edge where one is expected Returns NULL if no match is found

Definition at line 315 of file MVICFG.cpp.

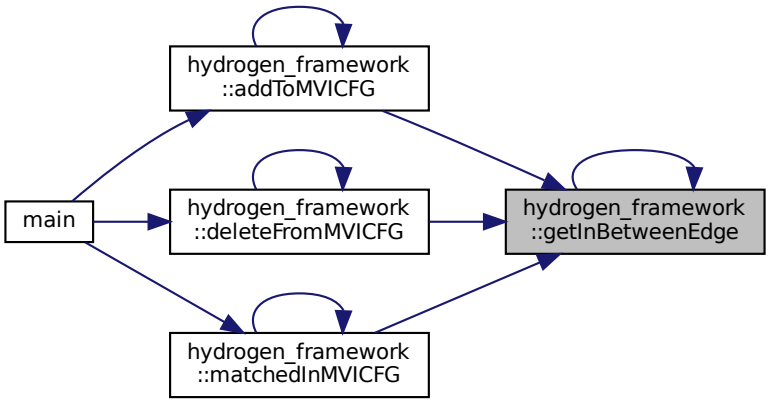
References hydrogen_framework::getEdge(), hydrogen_framework::getInBetweenEdge(), and hydrogen_framework::Graph_Line::get

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getInBetw and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.11 getMatchedInstructionFromGraph()

```
Graph_Instruction * hydrogen_framework::getMatchedInstructionFromGraph (
            Graph * graphToMatch,
            Graph_Instruction * instToMatch )
```

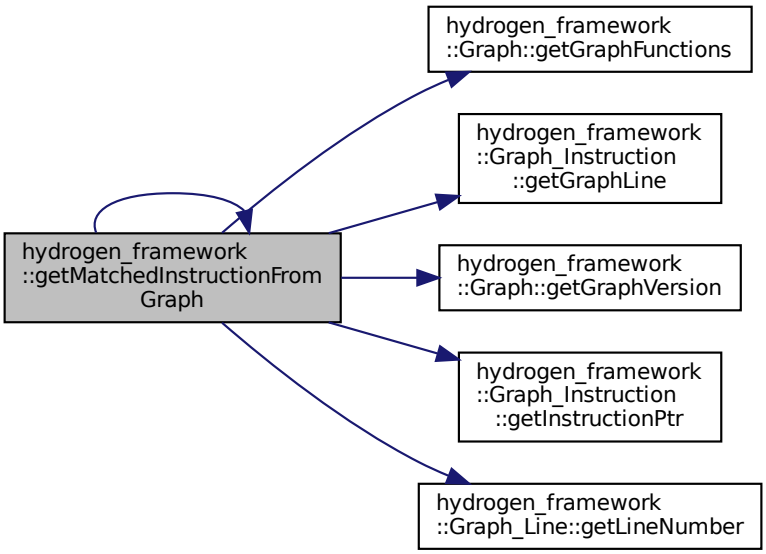
Get matching Graph_Instruction from given Graph given a Graph_Instruction using LLVM PTR Return NULL if no match is found

Definition at line 499 of file MVICFG.cpp.

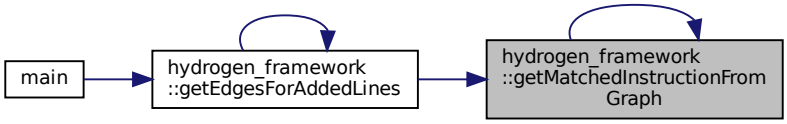
References hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::Graph_Instruction::getGraphLine(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Instruction::getInstructionPtr(), hydrogen_framework::Graph_Line::getLineNumber(), and hydrogen_framework::getMatchedInstructionFromGraph().

Referenced by hydrogen_framework::getEdgesForAddedLines(), and hydrogen_framework::getMatchedInstructionFromGraph().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.12 getNewlyAdded()

```
Graph_Line * hydrogen_framework::getNewlyAdded (
            Graph * MVICFG,
            Graph * ICFG,
            Graph_Line * newLine,
            Diff_Mapping diff )
```

Get the newly added MVICFG Graph_Line corresponding to the given ICFG Graph_Line Used only when find↩ MatchedLine fails to retrieve the same Returns NULL if no such line is found

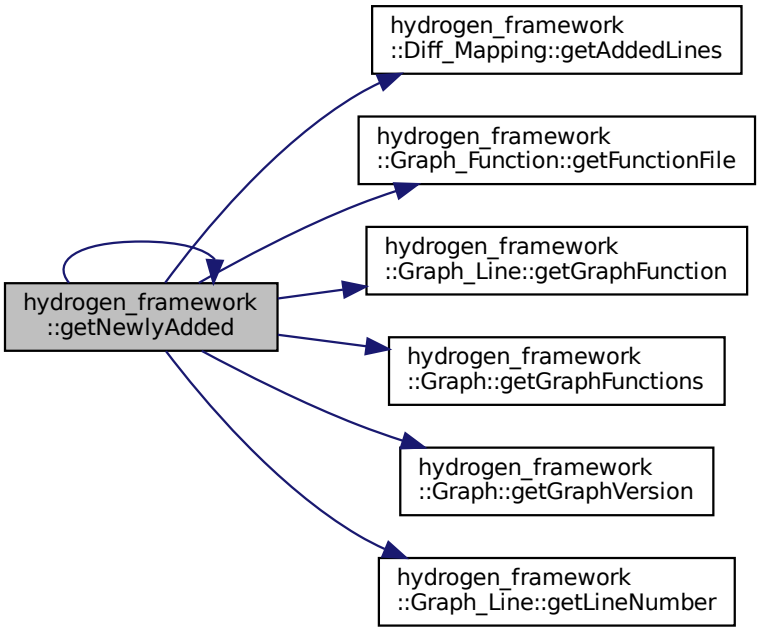
Definition at line 330 of file MVICFG.cpp.

References hydrogen_framework::Diff_Mapping::getAddedLines(), hydrogen_framework::Graph_Function::getFunctionFile(), hydrogen_framework::Graph_Line::getGraphFunction(), hydrogen_framework::Graph::getGraphFunctions(),

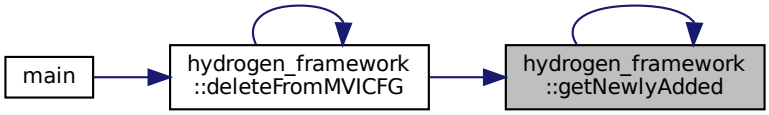
hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Line::getLineNumber(), and hydrogen_framework::getNewlyAdded().

Referenced by hydrogen_framework::deleteFromMVICFG(), and hydrogen_framework::getNewlyAdded().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.13 getPredGivenGraphLine()

```
std::list< Graph_Line * > hydrogen_framework::getPredGivenGraphLine (
            Graph_Line * line )
```

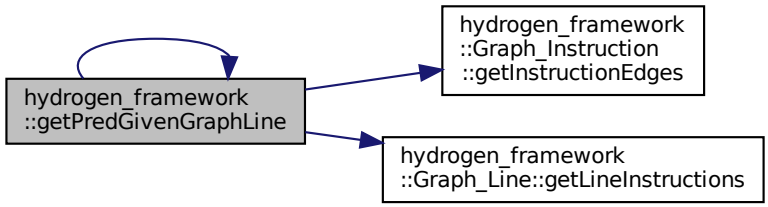
Get predecessor of a given Graph_Line

Definition at line 161 of file MVICFG.cpp.

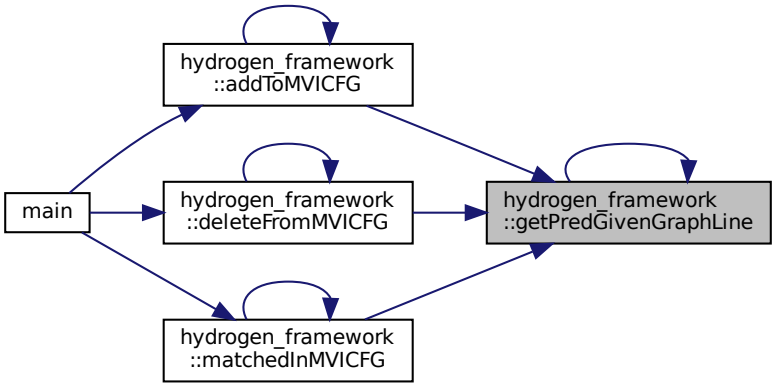
References hydrogen_framework::Graph_Instruction::getInstructionEdges(), hydrogen_framework::Graph_Line::getLineInstructions() and hydrogen_framework::getPredGivenGraphLine().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getPredGi and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.14 getSuccGivenGraphLine()

```
std::list< Graph_Line * > hydrogen_framework::getSuccGivenGraphLine (
            Graph_Line * line )
```

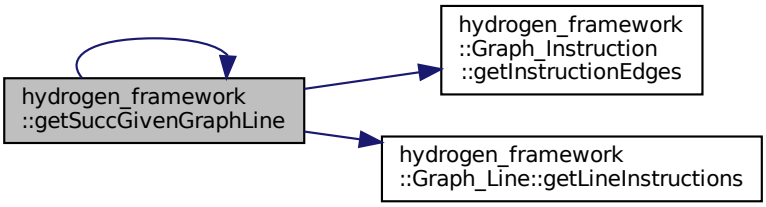
Get successor of a given Graph_Line

Definition at line 173 of file MVICFG.cpp.

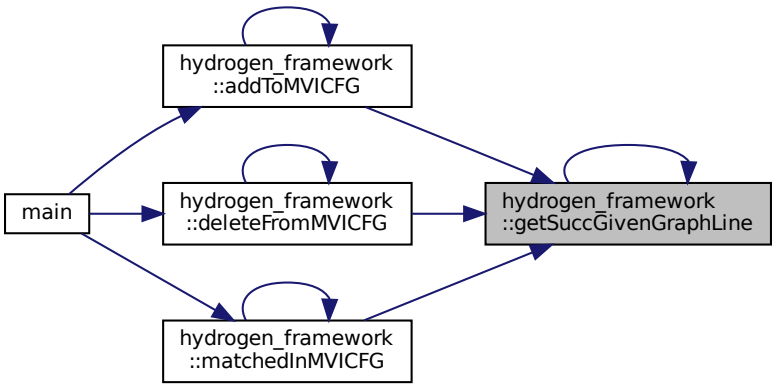
References hydrogen_framework::Graph_Instruction::getInstructionEdges(), hydrogen_framework::Graph_Line::getLineInstructions() and hydrogen_framework::getSuccGivenGraphLine().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getSuccG and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.15 matchedInMVICFG()

```
std::map< Graph_Line *, Graph_Line * > hydrogen_framework::matchedInMVICFG (
            Graph * MVICFG,
            Graph * ICFG,
            Diff_Mapping diff,
            unsigned Version )
```

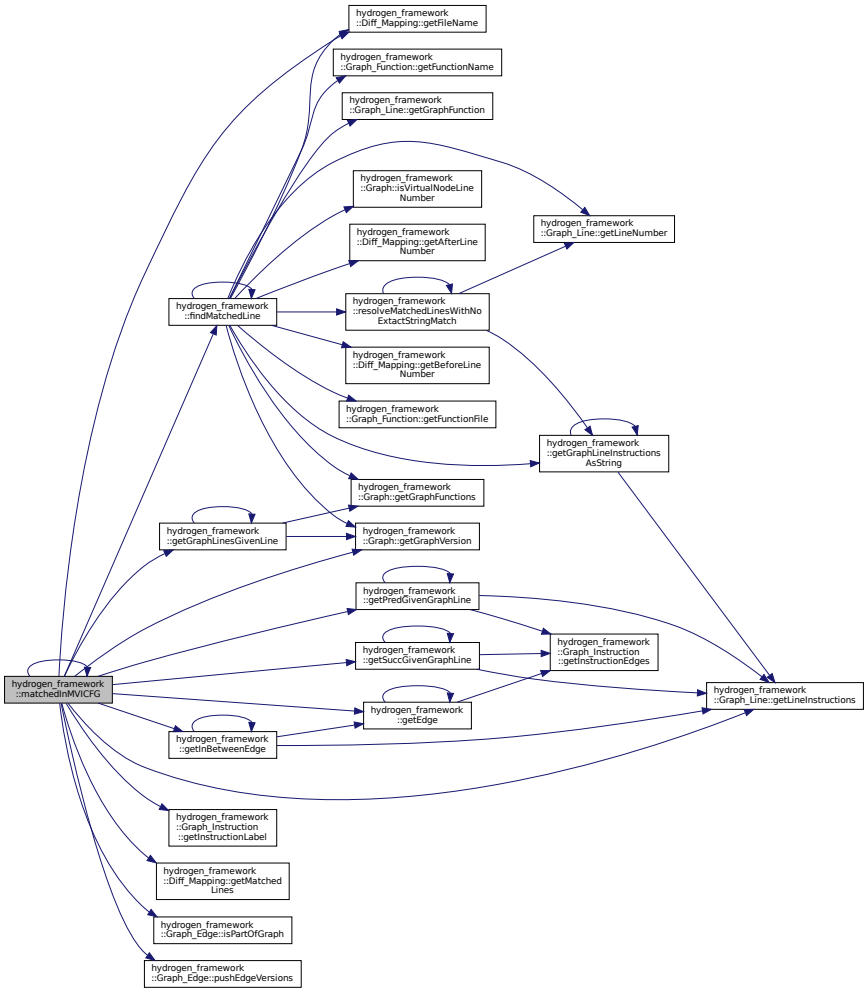
Returns the corresponding matched Graph_Line in MVICFG from ICFG

Definition at line 703 of file MVICFG.cpp.

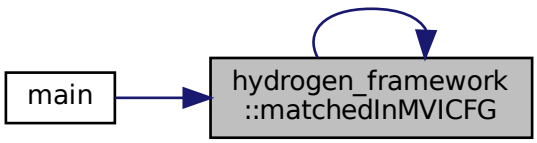
References hydrogen_framework::findMatchedLine(), hydrogen_framework::getEdge(), hydrogen_framework::Diff_Mapping::getFileN
hydrogen_framework::getGraphLinesGivenLine(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::getInBetwe
hydrogen_framework::Graph_Instruction::getInstructionLabel(), hydrogen_framework::Graph_Line::getLineInstructions(),
hydrogen_framework::Diff_Mapping::getMatchedLines(), hydrogen_framework::getPredGivenGraphLine(), hydrogen_framework::getS
hydrogen_framework::Graph_Edge::isPartOfGraph(), hydrogen_framework::matchedInMVICFG(), and hydrogen_framework::Graph_

Referenced by main(), and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.16 resolveMatchedLinesWithNoExtactStringMatch()

```
Graph_Line * hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch (
             std::list< Graph_Line * > matchedLines,
             std::string lineFromString,
             unsigned int graphVersion )
```

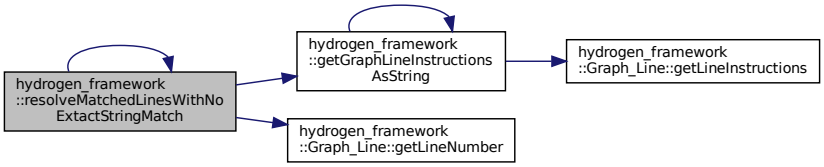
Heuristically try to find the closest Graph_Line match from a list of potential Graph_Line matches when no exact match is found using getGraphLineInstructionsAsString Currently will throw an warning if heuristic skips more than 2 OpCode to match the lines Returns NULL if no heuristic match is found

Definition at line 200 of file MVICFG.cpp.

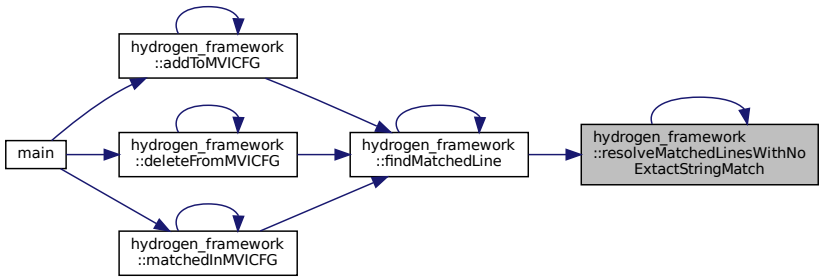
References hydrogen_framework::getGraphLineInstructionsAsString(), hydrogen_framework::Graph_Line::getLineNumber(), and hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch().

Referenced by hydrogen_framework::findMatchedLine(), and hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch().

Here is the call graph for this function:



Here is the caller graph for this function:



6.37.2.17 updateMVICFGVersion()

```
void hydrogen_framework::updateMVICFGVersion (
            Graph * MVICFG,
            std::list< Graph_Line * > addedLines,
            std::list< Graph_Line * > deletedLines,
            std::list< Diff_Mapping > diffMap,
            unsigned Version )
```

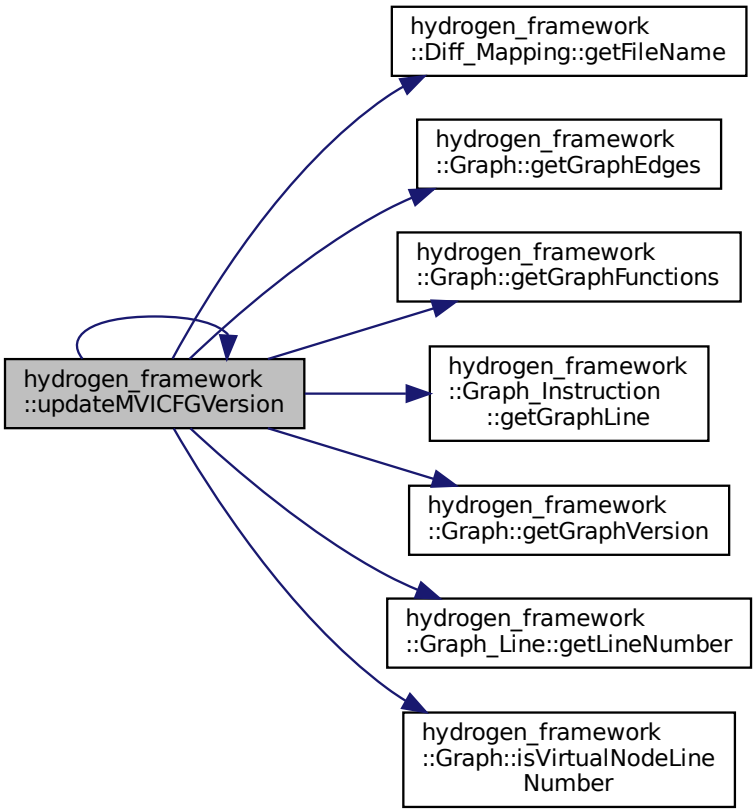
Update the Edge and Node information for MVICFG

Definition at line 814 of file MVICFG.cpp.

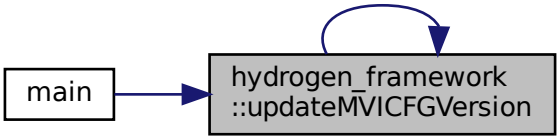
References hydrogen_framework::Diff_Mapping::getFileName(), hydrogen_framework::Graph::getGraphEdges(), hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::Graph_Instruction::getGraphLine(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Line::getLineNumber(), hydrogen_framework::Graph: and hydrogen_framework::updateMVICFGVersion().

Referenced by main(), and hydrogen_framework::updateMVICFGVersion().

Here is the call graph for this function:



Here is the caller graph for this function:



6.38 MVICFG.cpp

```
00001
00006 #include "MVICFG.hpp"
00007 #include "Diff_Mapping.hpp"
00008 #include "Graph.hpp"
00009 #include "Graph_Edge.hpp"
00010 #include "Graph_Function.hpp"
00011 #include "Graph_Instruction.hpp"
00012 #include "Graph_Line.hpp"
00013 #include "Module.hpp"
00014 namespace hydrogen_framework {
00015 Graph *buildICFG(Module *mod, unsigned graphVersion) {
00016   std::unique_ptr<llvm::Module> &modPtr = mod->getPtr();
00017   Graph *ICFG = new Graph(graphVersion);
00018   for (llvm::Function &F : (*modPtr)) {
00019     std::string funcName;
00020     Graph_Function *funcGraph = new Graph_Function(ICFG->getNextID());
00021     if (F.hasName()) {
00022       funcName = F.getName();
00023     } else {
00024       funcName = "Unknown_Function";
00025     } // End check for function name
00026     funcGraph->setFunctionName(funcName);
00027     Graph_Line *currentLineGraph = new Graph_Line(graphVersion);
00028     for (llvm::BasicBlock &BB : F) {
00029       for (llvm::Instruction &I : BB) {
00030         unsigned int DILocLine = 0;
00031         std::string DIFile = "Unknown_File";
00032         getLocationInfo(I, DILocLine, DIFile);
00033         /* Attach the line to current line if no debug information is found */
00034         if (DILocLine == 0) {
00035           DILocLine = currentLineGraph->getLineNumber(graphVersion);
00036         } // End check for DILocLine
00037         /* Create new Graph_Line container whenever new DILocLine is encountered */
00038         if (DILocLine != currentLineGraph->getLineNumber(graphVersion)) {
00039           if (!currentLineGraph->isLineInstructionEmpty()) {
00040             funcGraph->pushFunctionLines(currentLineGraph);
00041             ICFG->addSeqEdges(currentLineGraph);
00042           } // End check for isLineInstructionEmpty
00043           currentLineGraph = new Graph_Line(graphVersion);
00044         } // End check for continuation for current line
00045         if (!funcGraph->isFunctionFileSet()) {
00046           funcGraph->setFunctionFile(DIFile);
00047         } // End check for isFunctionFileSet
00048         currentLineGraph->setLineNumber(graphVersion, DILocLine);
00049         std::string instLabel;
00050         llvm::raw_string_ostream rInstLabel(instLabel);
00051         I.print(rInstLabel);
00052         Graph_Instruction *currentInstGraph = new Graph_Instruction();
00053         currentInstGraph->setInstructionLabel(instLabel);
00054         currentInstGraph->setInstructionID(ICFG->getNextID());
00055         llvm::Instruction *iTmp = &I;
00056         currentInstGraph->setInstructionPtr(iTmp);
00057         currentLineGraph->pushLineInstruction(currentInstGraph);
00058       } // End loop for BasicBlock
00059     }   // End loop for Function
00060     if (!currentLineGraph->isLineInstructionEmpty()) {
00061       funcGraph->pushFunctionLines(currentLineGraph);
00062       ICFG->addSeqEdges(currentLineGraph);
00063     } // End check for isLineInstructionEmpty
00064     if (!funcGraph->isFunctionLinesEmpty()) {
00065       ICFG->pushGraphFunction(funcGraph);
```

```
00066       ICFG->addVirtualNodes(funcGraph);
00067     } // End check for isFunctionLinesEmpty
00068   }   // End loop for Module
00069   ICFG->addBranchEdges();
00070   ICFG->addFunctionCallEdges();
00071   /* ICFG->printGraph("Graph_" + std::to_string(graphVersion)); */
00072   return ICFG;
00073 } // End buildICFG
00074
00075 std::list<Diff_Mapping> generateLineMapping(Module *firstMod, Module *secondMod) {
00076   std::list<Diff_Mapping> diffMap;
00077   std::list<std::string> processedFiles;
00078   /* Process files from first module */
00079   for (auto iterFile : (firstMod)->getFiles()) {
00080     std::list<std::string> nextModuleFiles = (secondMod)->getFiles();
00081     auto fileMatch = std::find_if(std::begin(nextModuleFiles), std::end(nextModuleFiles),
      [=](std::string f) {
00082       return (boost::filesystem::path(f).filename() == boost::filesystem::path(iterFile).filename());
00083     });
00084     Diff_Mapping::sequence ALines, BLines;
00085     if (fileMatch != nextModuleFiles.end()) {
00086       /* Matching file exist */
00087       processedFiles.push_back(boost::filesystem::path(iterFile).filename().c_str());
00088       std::ifstream Aifs(iterFile.c_str());
00089       std::ifstream Bifs((*fileMatch).c_str());
00090       Diff_Mapping::elem buf;
00091       while (getline(Aifs, buf)) {
00092         ALines.push_back(buf);
00093       } // End loop for Aifs
00094       while (getline(Bifs, buf)) {
00095         BLines.push_back(buf);
00096       } // End loop for Bifs
00097     } else {
00098       /* File no longer exist */
00099       processedFiles.push_back(boost::filesystem::path(iterFile).filename().c_str());
00100       std::ifstream Aifs(iterFile.c_str());
00101       Diff_Mapping::elem buf;
00102       while (getline(Aifs, buf)) {
00103         ALines.push_back(buf);
00104       } // End loop for Aifs
00105     }   // End check for nextModuleFiles
00106     Diff_Util diff(ALines, BLines);
00107     diff.compose();
00108     Diff_Ses s = diff.getSes();
00109     Diff_Mapping file(boost::filesystem::path(iterFile).filename().c_str());
00110     file.putMapping(s.getSequence());
00111     /* file.printMapping(); */
00112     diffMap.push_back(file);
00113   } // End loop for first module file processing
00114   /* Check for new files in next module */
00115   for (auto iterFile : (secondMod)->getFiles()) {
00116     auto fileMatch = std::find_if(std::begin(processedFiles), std::end(processedFiles),
00117                                   [=](std::string f) { return (f ==
      boost::filesystem::path(iterFile).filename()); });
00118     if (fileMatch == processedFiles.end()) {
00119       /* New file exist */
00120       processedFiles.push_back(boost::filesystem::path(iterFile).filename().c_str());
00121       Diff_Mapping::sequence ALines, BLines;
00122       std::ifstream Bifs(iterFile.c_str());
00123       Diff_Mapping::elem buf;
00124       while (getline(Bifs, buf)) {
00125         BLines.push_back(buf);
00126       } // End loop for Bifs
00127       Diff_Util diff(ALines, BLines);
00128       diff.compose();
00129       Diff_Ses s = diff.getSes();
00130       Diff_Mapping file(boost::filesystem::path(iterFile).filename().c_str());
00131       file.putMapping(s.getSequence());
00132       /* file.printMapping(); */
00133       diffMap.push_back(file);
00134     } // End check for processedFiles
00135   }   // End loop for second module file processing
00136   return diffMap;
00137 } // End generateLineMapping
00138
00139 std::list<Graph_Line *> getGraphLinesGivenLine(Graph *graph, long long lineNo, std::string fileName) {
00140   std::list<Graph_Line *> graphLines;
00141   bool foundLine = false;
00142   for (auto func : graph->getGraphFunctions()) {
00143     /* Matching with correct diff File */
00144     if (func->getFunctionFile() == fileName) {
00145       for (auto line : func->getFunctionLines()) {
00146         /* Matching line of diff */
00147         if (line->getLineNumber(graph->getGraphVersion()) == lineNo) {
00148           graphLines.push_back(line);
00149           foundLine = true;
00150         } // End check for lineNo
```

```
00151       }   // End loop for line
00152       /* Same line cannot be spread across functions. Hence stop search if at least one line found */
00153       if (foundLine) {
00154         return graphLines;
00155       } // End check for foundLine
00156     }   // End check for fileName
00157   }     // End loop for Functions
00158   return graphLines;
00159 } // End getGraphLinesGivenLine
00160
00161 std::list<Graph_Line *> getPredGivenGraphLine(Graph_Line *line) {
00162   std::list<Graph_Line *> pred;
00163   Graph_Instruction *frontInst = line->getLineInstructions().front();
00164   std::list<Graph_Edge *> edges = frontInst->getInstructionEdges();
00165   for (auto iter : edges) {
00166     if (iter->getEdgeTo() == frontInst) {
00167       pred.push_back(iter->getEdgeFrom()->getGraphLine());
00168     } // End check for frontInst
00169   }   // End loop for edges
00170   return pred;
00171 } // End getPredGivenGraphLine
00172
00173 std::list<Graph_Line *> getSuccGivenGraphLine(Graph_Line *line) {
00174   std::list<Graph_Line *> succ;
00175   Graph_Instruction *backInst = line->getLineInstructions().back();
00176   std::list<Graph_Edge *> edges = backInst->getInstructionEdges();
00177   for (auto iter : edges) {
00178     if (iter->getEdgeFrom() == backInst) {
00179       succ.push_back(iter->getEdgeTo()->getGraphLine());
00180     } // End check for backInst
00181   }   // End loop for edges
00182   return succ;
00183 } // End getSuccGivenGraphLine
00184
00185 std::string getGraphLineInstructionsAsString(Graph_Line *line) {
00186   std::string lineString;
00187   /* Iterate through the Graph_Line and make a string representation of the Instruction OpCode */
00188   for (auto inst : line->getLineInstructions()) {
00189     /* If Ptr is not found, then it won't be present in the other version as well */
00190     if (inst->getInstructionPtr() != NULL) {
00191       lineString.append(inst->getInstructionPtr()->getOpcodeName()).append(" ");
00192     } // End check for Instruction Ptr
00193   }   // End loop for Graph_Line
00194   if (!lineString.empty()) {
00195     lineString.pop_back();
00196   } // End check for empty string before remove trailing space
00197   return lineString;
00198 } // End getGraphLineInstructionsAsString
00199
00200 Graph_Line *resolveMatchedLinesWithNoExtactStringMatch(std::list<Graph_Line *> matchedLines,
     std::string lineFromString,
00201                                                        unsigned int graphVersion) {
00202   int minDiff = std::numeric_limits<int>::max();
00203   Graph_Line *tmp = NULL;
00204   for (auto line : matchedLines) {
00205     /* First remove matched OpCodes */
00206     std::string lineToString = getGraphLineInstructionsAsString(line);
00207     std::size_t pos = lineToString.find(lineFromString);
00208     if (pos != std::string::npos) {
00209       lineToString.erase(pos, lineFromString.length());
00210       boost::trim(lineToString);
00211     } // End check for match in lineToString
00212     /* Get the OpCodes remaining */
00213     int countOpCode = 0;
00214     if (!lineToString.empty()) {
00215       countOpCode = 1;
00216       for (auto ch : lineToString) {
00217         if (ch == ' ') {
00218           ++countOpCode;
00219         } // End check for space
00220       }   // End loop for counting space
00221     }     // End check for empty lineToString after removing matches
00222     if (countOpCode < minDiff) {
00223       minDiff = countOpCode;
00224       tmp = line;
00225     } // End check to update minDiff
00226   }   // End loop for matchedLines
00227   if (minDiff > 2) {
00228     std::cerr << "The heuristically matched line for " << tmp->getLineNumber(graphVersion) << "might be
     incorrect\n";
00229   } // End check for minDiff
00230   return tmp;
00231 } // End resolveMatchedLinesWithNoExtactStringMatch
00232
00233 Graph_Line *findMatchedLine(Graph_Line *t, Graph *matchTo, Graph *matchFrom, Diff_Mapping diff) {
00234   /* Extra check to ensure correct diff File */
00235   if (diff.getFileName() != t->getGraphFunction()->getFunctionFile()) {
```

```
00236     std::cerr « "findMatchedLine is using wrong diff File\n";
00237     std::cerr « "Skipping match for " « t->getLineNumber(matchFrom->getGraphVersion()) « " from "
00238               « matchFrom->getGraphVersion() « " to " « matchTo->getGraphVersion() « "\n";
00239     return NULL;
00240   } // End check for diff File name
00241   unsigned lineFrom = t->getLineNumber(matchFrom->getGraphVersion());
00242   unsigned lineTo = 0;
00243   /* Check for virtual node */
00244   if (matchFrom->isVirtualNodeLineNumber(lineFrom)) {
00245     /* Virtual nodes have same unique line number across graphs*/
00246     lineTo = lineFrom;
00247   } else {
00248     if (matchTo->getGraphVersion() > matchFrom->getGraphVersion()) {
00249       /* Matching MVICFG to ICFG */
00250       if (lineFrom == 0) {
00251         /* New lines will have line numbers already in them */
00252         lineTo = t->getLineNumber(matchTo->getGraphVersion());
00253       } else {
00254         lineTo = diff.getAfterLineNumber(lineFrom);
00255       } // End check for lineFrom
00256     } else {
00257       /* Matching ICFG to MVICFG */
00258       lineTo = diff.getBeforeLineNumber(lineFrom);
00259       if (lineTo == 0) {
00260         /* Line was deleted, but match is requested. Set lineTo to max
00261          * and let the calling function deal with it */
00262         lineTo = std::numeric_limits<unsigned>::max();
00263       } // End check for lineTo
00264     }   // End check for matchTo > matchFrom
00265   }     // End check for isVirtualNodeLineNumber
00266   if (lineTo != std::numeric_limits<unsigned>::max()) {
00267     for (auto func : matchTo->getGraphFunctions()) {
00268       if (func->getFunctionFile() == t->getGraphFunction()->getFunctionFile()) {
00269         if (func->getFunctionName() == t->getGraphFunction()->getFunctionName()) {
00270           std::list<Graph_Line *> matchedLines;
00271           for (auto line : func->getFunctionLines()) {
00272             if (lineTo == line->getLineNumber(matchTo->getGraphVersion())) {
00273               std::string lineToString = getGraphLineInstructionsAsString(line);
00274               std::string lineFromString = getGraphLineInstructionsAsString(t);
00275               if (lineToString == lineFromString) {
00276                 return line;
00277               } else {
00278                 matchedLines.push_back(line);
00279               } // End check for lineTo and lineFrom string
00280             }   // End check for matching line number
00281           }     // End loop for lines
00282           /* If there is a match at this point heuristically match it rather than return NULL */
00283           if (!matchedLines.empty()) {
00284             /* If only one match is there, then we don't have to work much */
00285             if (matchedLines.size() == 1) {
00286               return matchedLines.front();
00287             } else {
00288               std::string lineFromString = getGraphLineInstructionsAsString(t);
00289               return resolveMatchedLinesWithNoExtactStringMatch(matchedLines, lineFromString,
00290                                                                 matchTo->getGraphVersion());
00291             } // End check for matchedLines size
00292           }   // End check for empty matchedLines
00293         }     // End check for Function name check
00294       }       // End check for File name
00295     }         // End loop for functions
00296   }           // End check for virtual node check
00297   return NULL;
00298 } // End findMatchedLine
00299
00300 Graph_Edge *getEdge(Graph_Instruction *fromNode, Graph_Instruction *toNode, Graph_Edge::edgeTypes
      type) {
00301   for (auto edge : fromNode->getInstructionEdges()) {
00302     if (edge->getEdgeFrom() == fromNode) {
00303       if (edge->getEdgeTo() == toNode) {
00304         if (type == edge->getEdgeType()) {
00305           return edge;
00306         } else if (type == Graph_Edge::ANY) {
00307           return edge;
00308         } // End check for ANY
00309       }   // End check for toNode
00310     }     // End check for fromNode
00311   }       // End loop for Instructions
00312   return NULL;
00313 } // End getEdge
00314
00315 Graph_Edge *getInBetweenEdge(Graph_Line *fromLine, Graph_Line *toLine) {
00316   std::list<Graph_Instruction *> fromLineInstructions = fromLine->getLineInstructions();
00317   for (auto fromLineInstIter = fromLineInstructions.rbegin(); fromLineInstIter !=
      fromLineInstructions.rend();
00318        ++fromLineInstIter) {
00319     Graph_Instruction *fromLineInst = *fromLineInstIter;
00320     for (auto toLineInstIter : toLine->getLineInstructions()) {
```

```
00321       Graph_Edge *checkEdge = getEdge(fromLineInst, toLineInstIter, Graph_Edge::ANY);
00322       if (checkEdge) {
00323         return checkEdge;
00324       } // End check for checkEdge
00325     }   // End loop for toLine
00326   }     // End loop for fromLine
00327   return NULL;
00328 } // End getInBetweenEdge
00329
00330 Graph_Line *getNewlyAdded(Graph *MVICFG, Graph *ICFG, Graph_Line *newLine, Diff_Mapping diff) {
00331   std::list<long long> addedLines = diff.getAddedLines();
00332   auto findAdd = std::find_if(std::begin(addedLines), std::end(addedLines),
00333                               [=](long long no) { return (no ==
      newLine->getLineNumber(ICFG->getGraphVersion())); });
00334   if (findAdd != addedLines.end()) {
00335     for (auto func : MVICFG->getGraphFunctions()) {
00336       /* Compare line number within same file */
00337       if (func->getFunctionFile() == newLine->getGraphFunction()->getFunctionFile()) {
00338         for (auto line : func->getFunctionLines()) {
00339           if (newLine->getLineNumber(ICFG->getGraphVersion()) ==
      line->getLineNumber(ICFG->getGraphVersion())) {
00340             return line;
00341           } // End check for line and newLine numbers
00342         }   // End loop for line
00343       }     // End check for function file
00344     }       // End loop for function
00345   }         // End check for addLines.end
00346   return NULL;
00347 } // End getNewlyAdded
00348
00349 std::list<Graph_Line *> addToMVICFG(Graph *MVICFG, Graph *ICFG, Diff_Mapping diff, unsigned Version) {
00350   std::list<long long> addedLines = diff.getAddedLines();
00351   std::string fileName = diff.getFileName();
00352   std::list<Graph_Line *> N;
00353   std::list<Graph_Line *> icfgN;
00354   /*Identify all added lines */
00355   for (auto line : addedLines) {
00356     std::list<Graph_Line *> addedGraphLines;
00357     addedGraphLines = getGraphLinesGivenLine(ICFG, line, fileName);
00358     if (addedGraphLines.empty()) {
00359       std::cerr << "Graph_Line for line " << line << ":" << fileName << " not found in ICFG Ver " << Version
      << "\n";
00360       std::cerr << "Skipping this line and continuing\n";
00361       continue;
00362     } // End check for addedGraphLines
00363     for (auto addedLine : addedGraphLines) {
00364       Graph_Function *func = addedLine->getGraphFunction();
00365       /* Get corresponding MVICFG Graph_Function */
00366       std::list<Graph_Function *> mvicfgFunctions = MVICFG->getGraphFunctions();
00367       auto findMvicfgFunc =
00368           std::find_if(std::begin(mvicfgFunctions), std::end(mvicfgFunctions), [=](Graph_Function
      *mvicfgfunc) {
00369             return mvicfgfunc->getFunctionName() == func->getFunctionName();
00370           });
00371       /* Create new one if it doesn't exist */
00372       Graph_Function *mvicfgFunc;
00373       if (findMvicfgFunc == mvicfgFunctions.end()) {
00374         mvicfgFunc = new Graph_Function(MVICFG->getNextID());
00375         Graph_Function *mvicfgFunc = new Graph_Function(MVICFG->getNextID());
00376         mvicfgFunc->setFunctionName(func->getFunctionName());
00377         mvicfgFunc->setFunctionFile(func->getFunctionFile());
00378       } else {
00379         mvicfgFunc = *findMvicfgFunc;
00380       } // End check for findMvicfgFunc
00381       /* Iterating through addedLine and adding instructions to MVICFG */
00382       Graph_Line *newLine = new Graph_Line(ICFG->getGraphVersion());
00383       newLine->setLineNumber(MVICFG->getGraphVersion(), 0);
00384       newLine->setLineNumber(ICFG->getGraphVersion(),
      addedLine->getLineNumber(ICFG->getGraphVersion()));
00385       for (auto inst : addedLine->getLineInstructions()) {
00386         Graph_Instruction *newInstruction = new Graph_Instruction();
00387         newInstruction->setInstructionLabel(inst->getInstructionLabel());
00388         newInstruction->setInstructionID(MVICFG->getNextID());
00389         newInstruction->setInstructionPtr(inst->getInstructionPtr());
00390         newLine->pushLineInstruction(newInstruction);
00391       } // End loop for adding instructions
00392       mvicfgFunc->pushFunctionLines(newLine);
00393       N.push_back(newLine);
00394       icfgN.push_back(addedLine);
00395     } // End loop for processing addedGraphLines
00396   }   // End loop for identifying added lines
00397   for (auto n : N) {
00398     /* Proceed only if the function is in diff File being processed */
00399     if (n->getGraphFunction()->getFunctionFile() == fileName) {
00400       Graph_Line *nDash = findMatchedLine(n, ICFG, MVICFG, diff);
00401       if (!nDash) {
00402         std::cerr << "ICFG line corresponding to the added MVICFG line " <<
```

```
      n->getLineNumber(ICFG->getGraphVersion())
00403                   « " not found\n";
00404         continue;
00405       } // End check for nDash
00406       std::list<Graph_Line *> pred = getPredGivenGraphLine(nDash);
00407       std::list<Graph_Line *> succ = getSuccGivenGraphLine(nDash);
00408       std::list<Graph_Line *> T;
00409       T.insert(T.end(), pred.begin(), pred.end());
00410       T.insert(T.end(), succ.begin(), succ.end());
00411       for (auto t : T) {
00412         auto findT = std::find_if(std::begin(icfgN), std::end(icfgN), [=](Graph_Line *N) { return (N
      == t); });
00413         if (findT == icfgN.end()) {
00414           /* t in T but not in N */
00415           /* Proceed only if the function is in diff File being processed */
00416           if (t->getGraphFunction()->getFunctionFile() == fileName) {
00417             Graph_Line *tDash = findMatchedLine(t, MVICFG, ICFG, diff);
00418             if (tDash) {
00419               auto findTPred = std::find_if(std::begin(pred), std::end(pred), [=](Graph_Line *N) {
      return (N == t); });
00420               auto findTSucc = std::find_if(std::begin(succ), std::end(succ), [=](Graph_Line *N) {
      return (N == t); });
00421               if (findTPred != pred.end()) {
00422                 Graph_Instruction *nInst = n->getLineInstructions().front();
00423                 Graph_Instruction *tDashInst = tDash->getLineInstructions().back();
00424                 /* Check before adding the edge and if edge exist only add the version */
00425                 Graph_Edge *checkEdge = getEdge(tDashInst, nInst, Graph_Edge::ANY);
00426                 if (!checkEdge) {
00427                   /* Get edge type from ICFG */
00428                   Graph_Instruction *nDashInst = nDash->getLineInstructions().front();
00429                   Graph_Instruction *tInst = t->getLineInstructions().back();
00430                   Graph_Edge *getEdgeType = getEdge(tInst, nDashInst, Graph_Edge::ANY);
00431                   Graph_Edge::edgeTypes edgeType;
00432                   if (getEdgeType) {
00433                     edgeType = getEdgeType->getEdgeType();
00434                   } else {
00435                     bool foundEdge = false;
00436                     Graph_Edge *checkBetweenEdge = getInBetweenEdge(t, nDash);
00437                     if (checkBetweenEdge) {
00438                       foundEdge = true;
00439                       edgeType = checkBetweenEdge->getEdgeType();
00440                     } // End check for checkBetweenEdge
00441                     if (!foundEdge) {
00442                       std::cerr « "ICFG edge between " « tInst->getInstructionLabel() « " and "
00443                                « nDashInst->getInstructionLabel() « " not found\n";
00444                       std::cerr « "Setting edge type to MVICFG_ADD\n";
00445                       edgeType = Graph_Edge::MVICFG_ADD;
00446                     } // End check for foundEdge
00447                   }   // End check for getEdgeType
00448                   Graph_Edge *newEdge = new Graph_Edge(tDashInst, nInst, edgeType, Version);
00449                   MVICFG->addEdge(tDashInst, nInst, newEdge);
00450                 } else {
00451                   checkEdge->pushEdgeVersions(Version);
00452                 } // End check for checkEdge
00453               } else if (findTSucc != succ.end()) {
00454                 Graph_Instruction *tDashInst = tDash->getLineInstructions().front();
00455                 Graph_Instruction *nInst = n->getLineInstructions().back();
00456                 /* Check before adding the edge and if edge exist only add the version */
00457                 Graph_Edge *checkEdge = getEdge(nInst, tDashInst, Graph_Edge::ANY);
00458                 if (!checkEdge) {
00459                   /* Get edge type from ICFG */
00460                   Graph_Instruction *tInst = t->getLineInstructions().front();
00461                   Graph_Instruction *nDashInst = nDash->getLineInstructions().back();
00462                   Graph_Edge *getEdgeType = getEdge(nDashInst, tInst, Graph_Edge::ANY);
00463                   Graph_Edge::edgeTypes edgeType;
00464                   if (getEdgeType) {
00465                     edgeType = getEdgeType->getEdgeType();
00466                   } else {
00467                     bool foundEdge = false;
00468                     Graph_Edge *checkBetweenEdge = getInBetweenEdge(nDash, t);
00469                     if (checkBetweenEdge) {
00470                       foundEdge = true;
00471                       edgeType = checkBetweenEdge->getEdgeType();
00472                     } // End check for checkBetweenEdge
00473                     if (!foundEdge) {
00474                       std::cerr « "ICFG edge between " « tInst->getInstructionLabel() « " and "
00475                                « nDashInst->getInstructionLabel() « " not found\n";
00476                       std::cerr « "Setting edge type to MVICFG_ADD\n";
00477                       edgeType = Graph_Edge::MVICFG_ADD;
00478                     } // End check for foundEdge
00479                   }   // End check for edgeType
00480                   Graph_Edge *newEdge = new Graph_Edge(nInst, tDashInst, edgeType, Version);
00481                   MVICFG->addEdge(nInst, tDashInst, newEdge);
00482                 } else {
00483                   checkEdge->pushEdgeVersions(Version);
00484                 } // End check for checkEdge
00485               }   // End check for Predecessor & Successors
```

```
00486             } else {
00487               std::cerr « "No matching line found for " « t->getLineNumber(ICFG->getGraphVersion())
00488                        « " in MVICFG(A)\n";
00489             } // End check for tDash
00490           }   // End check to see if the function is in the same diff file
00491         }     // End check for find T
00492       }       // End loop for T
00493     }         // End check to see if the function is in the same diff file
00494   }           // End loop for adding edges for added lines
00495   /* Return the added lines */
00496   return N;
00497 } // End addToMVICFG
00498
00499 Graph_Instruction *getMatchedInstructionFromGraph(Graph *graphToMatch, Graph_Instruction *instToMatch)
      {
00500   for (auto func : graphToMatch->getGraphFunctions()) {
00501     for (auto line : func->getFunctionLines()) {
00502       std::list<Graph_Instruction *> lineInstList = line->getLineInstructions();
00503       std::_List_iterator<Graph_Instruction *> findInst;
00504       if (instToMatch->getInstructionPtr() == NULL) {
00505         /* This is a virtual node and they always share their line numbers */
00506         unsigned instToLineNumber =
      instToMatch->getGraphLine()->getLineNumber(graphToMatch->getGraphVersion());
00507         findInst = std::find_if(std::begin(lineInstList), std::end(lineInstList),
      [=](Graph_Instruction *inst) {
00508           return (inst->getGraphLine()->getLineNumber(graphToMatch->getGraphVersion() ==
      instToLineNumber));
00509         });
00510       } else {
00511         findInst = std::find_if(std::begin(lineInstList), std::end(lineInstList),
      [=](Graph_Instruction *inst) {
00512           return (inst->getInstructionPtr() == instToMatch->getInstructionPtr());
00513         });
00514       } // End check for instToMatch
00515       if (findInst != lineInstList.end()) {
00516         return *findInst;
00517       } // End check for findInst
00518     }   // End loop for line
00519   }     // End loop for func
00520   return NULL;
00521 } // End getMatchedInstructionFromGraph
00522
00523 void getEdgesForAddedLines(Graph *MVICFG, Graph *ICFG, std::list<Graph_Line *> addedLines,
00524                            std::list<Diff_Mapping> diffMap, unsigned Version) {
00525   for (auto line : addedLines) {
00526     for (auto lineInst : line->getLineInstructions()) {
00527       Graph_Instruction *lineDashInst = getMatchedInstructionFromGraph(ICFG, lineInst);
00528       if (!lineDashInst) {
00529         std::cerr « "No match found in ICFG for instruction " « lineInst->getInstructionLabel() «
      "\n";
00530         std::cerr « "Skipping Instruction\n";
00531         continue;
00532       } // End check for lineDashInst
00533       for (auto edgeDash : lineDashInst->getInstructionEdges()) {
00534         Graph_Instruction *fromDash = edgeDash->getEdgeFrom();
00535         Graph_Instruction *from = getMatchedInstructionFromGraph(MVICFG, fromDash);
00536         if (!from) {
00537           /* This edge would have been added by addToMVICFG */
00538           continue;
00539         } // End check for from
00540         Graph_Instruction *toDash = edgeDash->getEdgeTo();
00541         Graph_Instruction *to = getMatchedInstructionFromGraph(MVICFG, toDash);
00542         if (!to) {
00543           /* This edge would have been added by addToMVICFG */
00544           continue;
00545         } // End check for to
00546         Graph_Edge *checkEdge = getEdge(from, to, edgeDash->getEdgeType());
00547         if (!checkEdge) {
00548           Graph_Edge *newEdge = new Graph_Edge(from, to, edgeDash->getEdgeType(),
      ICFG->getGraphVersion());
00549           MVICFG->addEdge(from, to, newEdge);
00550         } // End check for checkEdge
00551       }   // End loop for adding edges
00552     }     // End loop for lineInst
00553   }       // End loop for line
00554 } // End getEdgesForAddedLines
00555
00556 std::list<Graph_Line *> deleteFromMVICFG(Graph *MVICFG, Graph *ICFG, Diff_Mapping diff, unsigned
      Version) {
00557   std::list<long long> deletedLines = diff.getDeletedLines();
00558   std::string fileName = diff.getFileName();
00559   std::list<Graph_Line *> N;
00560   /* Identify all deleted lines */
00561   for (auto line : deletedLines) {
00562     std::list<Graph_Line *> deletedGraphLines;
00563     deletedGraphLines = getGraphLinesGivenLine(MVICFG, line, fileName);
00564     if (deletedGraphLines.empty()) {
```

```
00565       std::cerr « "Graph_Line for line " « line « " not found in MVICFG\n";
00566       std::cerr « "Skipping this line and continuing\n";
00567       continue;
00568     } // End check for deletedGraphLines
00569     for (auto deleteLine : deletedGraphLines) {
00570       /* Mark as deleted in ICFG version */
00571       deleteLine->setLineNumber(ICFG->getGraphVersion(), 0);
00572       /* Add deleted line to N */
00573       N.push_back(deleteLine);
00574     } // End loop for processing deletedGraphLines
00575   }   // End loop for identifying the deleted lines
00576   for (auto func : MVICFG->getGraphFunctions()) {
00577     /* Proceed only if the function is in diff File being processed */
00578     if (func->getFunctionFile() == fileName) {
00579       for (auto n : func->getFunctionLines()) {
00580         auto findLine = std::find_if(std::begin(N), std::end(N), [=](Graph_Line *N) { return (N == n);
      });
00581         if (findLine == N.end()) {
00582           /* n not in N but in MVICFG */
00583           std::list<Graph_Line *> pred = getPredGivenGraphLine(n);
00584           std::list<Graph_Line *> succ = getSuccGivenGraphLine(n);
00585           std::list<Graph_Line *> T;
00586           T.insert(T.end(), pred.begin(), pred.end());
00587           T.insert(T.end(), succ.begin(), succ.end());
00588           for (auto t : T) {
00589             auto findT = std::find_if(std::begin(N), std::end(N), [=](Graph_Line *N) { return (N ==
      t); });
00590             if (findT != N.end()) {
00591               /* n has a successor or predecessor in N */
00592               /* Proceed only if the function is in diff File being processed */
00593               if (n->getGraphFunction()->getFunctionFile() == fileName) {
00594                 Graph_Line *nDash = findMatchedLine(n, ICFG, MVICFG, diff);
00595                 if (!nDash) {
00596                   /* Check if 'n' exist in MVICFG currently, otherwise nDash won't exist obviously */
00597                   if (n->getLineNumber(MVICFG->getGraphVersion()) == 0) {
00598                     continue;
00599                   } // End check for n's line number
00600                 }   // End check for nDash
00601                 if (nDash) {
00602                   std::list<Graph_Line *> predDash = getPredGivenGraphLine(nDash);
00603                   std::list<Graph_Line *> succDash = getSuccGivenGraphLine(nDash);
00604                   std::list<Graph_Line *> MDash;
00605                   MDash.insert(MDash.end(), predDash.begin(), predDash.end());
00606                   MDash.insert(MDash.end(), succDash.begin(), succDash.end());
00607                   for (auto mDash : MDash) {
00608                     /* Proceed only if the function is in diff File being processed */
00609                     if (mDash->getGraphFunction()->getFunctionFile() == fileName) {
00610                       Graph_Line *m = findMatchedLine(mDash, MVICFG, ICFG, diff);
00611                       if (!m) {
00612                         /* Check if it was newly added ICFG line */
00613                         m = getNewlyAdded(MVICFG, ICFG, mDash, diff);
00614                       } // End check for m
00615                       if (m) {
00616                         auto findMPred = std::find_if(std::begin(predDash), std::end(predDash),
00617                                                       [=](Graph_Line *N) { return (N == mDash); });
00618                         auto findMSucc = std::find_if(std::begin(succDash), std::end(succDash),
00619                                                       [=](Graph_Line *N) { return (N == mDash); });
00620                         if (findMPred != predDash.end()) {
00621                           /* Check for edge between m and n */
00622                           Graph_Instruction *mInst = m->getLineInstructions().back();
00623                           Graph_Instruction *nInst = n->getLineInstructions().front();
00624                           Graph_Edge *checkEdge = getEdge(mInst, nInst, Graph_Edge::ANY);
00625                           if (!checkEdge) {
00626                             /* Get edge type from ICFG */
00627                             Graph_Instruction *nDashInst = nDash->getLineInstructions().back();
00628                             Graph_Instruction *mDashInst = mDash->getLineInstructions().front();
00629                             Graph_Edge *getEdgeType = getEdge(mDashInst, nDashInst, Graph_Edge::ANY);
00630                             Graph_Edge::edgeTypes edgeType;
00631                             if (getEdgeType) {
00632                               edgeType = getEdgeType->getEdgeType();
00633                             } else {
00634                               bool foundEdge = false;
00635                               Graph_Edge *checkBetweenEdge = getInBetweenEdge(mDash, nDash);
00636                               if (checkBetweenEdge) {
00637                                 foundEdge = true;
00638                                 edgeType = checkBetweenEdge->getEdgeType();
00639                               } // End check for checkBetweenEdge
00640                               if (!foundEdge) {
00641                                 std::cerr « "ICFG edge between " « mInst->getInstructionLabel() « "
      and "
00642                                          « nInst->getInstructionLabel() « " not found\n";
00643                                 std::cerr « "Setting edge type to MVICFG_DEL\n";
00644                                 edgeType = Graph_Edge::MVICFG_DEL;
00645                               } // End check for foundEdge
00646                             }   // End check for getEdgeType
00647                             Graph_Edge *newEdge = new Graph_Edge(mInst, nInst, edgeType, Version);
00648                             MVICFG->addEdge(mInst, nInst, newEdge);
```

```
00649                         } // End check for checkEdge
00650                       } else if (findMSucc != succDash.end()) {
00651                         /* Check for edge between n and m */
00652                         Graph_Instruction *nInst = n->getLineInstructions().back();
00653                         Graph_Instruction *mInst = m->getLineInstructions().front();
00654                         Graph_Edge *checkEdge = getEdge(nInst, mInst, Graph_Edge::ANY);
00655                         if (!checkEdge) {
00656                           /* Get edge type from ICFG */
00657                           Graph_Instruction *nDashInst = nDash->getLineInstructions().back();
00658                           Graph_Instruction *mDashInst = mDash->getLineInstructions().front();
00659                           Graph_Edge *getEdgeType = getEdge(nDashInst, mDashInst, Graph_Edge::ANY);
00660                           Graph_Edge::edgeTypes edgeType;
00661                           if (getEdgeType) {
00662                             edgeType = getEdgeType->getEdgeType();
00663                           } else {
00664                             bool foundEdge = true;
00665                             Graph_Edge *checkBetweenEdge = getInBetweenEdge(nDash, mDash);
00666                             if (checkBetweenEdge) {
00667                               foundEdge = true;
00668                               edgeType = checkBetweenEdge->getEdgeType();
00669                             } // End check for checkBetweenEdge
00670                             if (!foundEdge) {
00671                               std::cerr << "ICFG edge between " << nInst->getInstructionLabel() << "
      and "
00672                                         << mInst->getInstructionLabel() << " not found\n";
00673                               std::cerr << "Setting edge type to MVICFG_DEL\n";
00674                               edgeType = Graph_Edge::MVICFG_DEL;
00675                             } // End check for foundEdge
00676                           }   // End check for getEdgeType
00677                           Graph_Edge *newEdge = new Graph_Edge(nInst, mInst, edgeType, Version);
00678                           MVICFG->addEdge(nInst, mInst, newEdge);
00679                         } // End check for checkEdge
00680                       }   // End check for Predecessors and Successors
00681                     } else {
00682                       std::cerr << "No matching line found for " <<
      mDash->getLineNumber(ICFG->getGraphVersion())
00683                                 << " in MVICFG(D)\n";
00684                     } // End check for m
00685                   }   // End check to see if the function is in the same diff file
00686                 }     // End loop for MDash
00687               } else {
00688                 std::cerr << "No matching line found for " <<
      n->getLineNumber(MVICFG->getGraphVersion())
00689                           << " in ICFG(" << ICFG->getGraphVersion() << ")\n";
00690                 std::cerr << "file : " << fileName << "\n";
00691               } // End check for nDash
00692             }   // End check to see if the function is in the same diff file
00693           }     // End check for T in N
00694         }       // End loop for T
00695       }         // End check for findLine in N
00696     }           // End loop for n
00697   }             // End check to see if the function is in the same diff file
00698  }              // End loop for adding edges for deleted lines
00699  /* Return the deleted MVICFG lines */
00700  return N;
00701 } // End deleteFromMVICFG
00702
00703 std::map<Graph_Line *, Graph_Line *> matchedInMVICFG(Graph *MVICFG, Graph *ICFG, Diff_Mapping diff,
      unsigned Version) {
00704  std::map<long long, long long> matchedLines = diff.getMatchedLines();
00705  std::string fileName = diff.getFileName();
00706  std::map<Graph_Line *, Graph_Line *> matchedGraphLines;
00707  std::list<Graph_Line *> mvicfgM;
00708  /* Identify all the matched lines */
00709  for (auto line : matchedLines) {
00710    long long mvicfgLineNo = line.first;
00711    long long icfgLineNo = line.second;
00712    std::list<Graph_Line *> mvicfgGraphLines;
00713    mvicfgGraphLines = getGraphLinesGivenLine(MVICFG, mvicfgLineNo, fileName);
00714    /* No need to worry about matching lines that are not in MVICFG */
00715    if (!mvicfgGraphLines.empty()) {
00716      std::list<Graph_Line *> icfgGraphLines;
00717      icfgGraphLines = getGraphLinesGivenLine(ICFG, icfgLineNo, fileName);
00718      if (icfgGraphLines.empty()) {
00719        std::cerr << "Graph_Line for line " << icfgLineNo << " not found in ICFG\n";
00720        std::cerr << "Skipping this line and continuing\n";
00721        continue;
00722      } // End check for if line is present in ICFG
00723      if (mvicfgGraphLines.size() != icfgGraphLines.size()) {
00724        std::cerr << "Mismatch between the number of MVICFG and ICFG Graph_Lines for (" << line.first
00725                  << "::" << line.second << ")\n";
00726        std::cerr << "Skipping this line and continuing\n";
00727        continue;
00728      } // End check for mismatch in GraphLine size
00729      for (auto mvicfgLine = mvicfgGraphLines.begin(), icfgLine = icfgGraphLines.begin();
00730           mvicfgLine != mvicfgGraphLines.end() && icfgLine != icfgGraphLines.end(); ++mvicfgLine,
      ++icfgLine) {
```

```
00731         matchedGraphLines.insert(std::pair<Graph_Line *, Graph_Line *>(*icfgLine, *mvicfgLine));
00732         mvicfgM.push_back(*icfgLine);
00733       } // End loop for processing mvicfgGraphLines & icfgGraphLines
00734     }   // End check for if line is preset in MVICFG
00735   }     //  End loop for matchedLines
00736   for (auto line : matchedGraphLines) {
00737     Graph_Line *n = line.second;
00738     Graph_Line *nDash = line.first;
00739     std::list<Graph_Line *> pred = getPredGivenGraphLine(nDash);
00740     std::list<Graph_Line *> succ = getSuccGivenGraphLine(nDash);
00741     std::list<Graph_Line *> T;
00742     T.insert(T.end(), pred.begin(), pred.end());
00743     T.insert(T.end(), succ.begin(), succ.end());
00744     for (auto t : T) {
00745       auto findT = std::find_if(std::begin(mvicfgM), std::end(mvicfgM), [=](Graph_Line *N) { return (N
      == t); });
00746       if (findT != mvicfgM.end()) {
00747         /* t in T and in Matched */
00748         /* Proceed only if the function is in diff File being processed */
00749         if (t->getGraphFunction()->getFunctionFile() == fileName) {
00750           Graph_Line *tDash = findMatchedLine(t, MVICFG, ICFG, diff);
00751           if (tDash) {
00752             auto findTPred = std::find_if(std::begin(pred), std::end(pred), [=](Graph_Line *N) {
    return (N == t); });
00753             auto findTSucc = std::find_if(std::begin(succ), std::end(succ), [=](Graph_Line *N) {
    return (N == t); });
00754             if (findTPred != pred.end()) {
00755               Graph_Instruction *nInst = n->getLineInstructions().front();
00756               Graph_Instruction *tDashInst = tDash->getLineInstructions().back();
00757               /* Edge should exist in the MVICFG. Raise error otherwise */
00758               Graph_Edge *checkEdge = getEdge(tDashInst, nInst, Graph_Edge::ANY);
00759               bool foundEdge = false;
00760               if (!checkEdge) {
00761                 Graph_Edge *checkBetweenEdge = getInBetweenEdge(tDash, n);
00762                 if (checkBetweenEdge) {
00763                   foundEdge = true;
00764                   checkEdge = checkBetweenEdge;
00765                 } // End checkBetweenEdge
00766                 std::cerr « "MVICFG edge between " « tDashInst->getInstructionLabel() « " and "
00767                          « nInst->getInstructionLabel() « " not found\n";
00768                 std::cerr « "Skipping this predecessor edge\n";
00769               } else {
00770                 foundEdge = true;
00771               } // End check for checkEdge
00772               if (foundEdge) {
00773                 if (!checkEdge->isPartOfGraph(Version)) {
00774                   checkEdge->pushEdgeVersions(Version);
00775                 } // End check for isPartOfGraph
00776               }   // End check for foundEdge
00777             } else if (findTSucc != succ.end()) {
00778               Graph_Instruction *tDashInst = tDash->getLineInstructions().front();
00779               Graph_Instruction *nInst = n->getLineInstructions().back();
00780               /* Edge should exist in the MVICFG. Raise error otherwise */
00781               Graph_Edge *checkEdge = getEdge(nInst, tDashInst, Graph_Edge::ANY);
00782               bool foundEdge = false;
00783               if (!checkEdge) {
00784                 Graph_Edge *checkBetweenEdge = getInBetweenEdge(n, tDash);
00785                 if (checkBetweenEdge) {
00786                   foundEdge = true;
00787                   checkEdge = checkBetweenEdge;
00788                 } // End check for checkBetweenEdge
00789                 if (!foundEdge) {
00790                   std::cerr « "MVICFG edge between " « tDashInst->getInstructionLabel() « " and "
00791                            « nInst->getInstructionLabel() « " not found\n";
00792                   std::cerr « "Skipping this successor edge\n";
00793                 } // End check for foundEdge
00794               } else {
00795                 foundEdge = true;
00796               } // End check for checkEdge
00797               if (foundEdge) {
00798                 if (!checkEdge->isPartOfGraph(Version)) {
00799                   checkEdge->pushEdgeVersions(Version);
00800                 } // End check for isPartOfGraph
00801               }   // End check for foundEdge
00802             }     // End check for Successor
00803           } else {
00804             std::cerr « "No matching line found for " « t->getLineNumber(ICFG->getGraphVersion())
00805                      « " in MVICFG(M)\n";
00806           } // End check for tDash
00807         }   // End check to see if the function is in the same diff file
00808       }     // End check for find T
00809     }       // End loop for T
00810   }         // End loop for adding edges for matched lines
00811   return matchedGraphLines;
00812 } // End matchedInMVICFG
00813
00814 void updateMVICFGVersion(Graph *MVICFG, std::list<Graph_Line *> addedLines, std::list<Graph_Line *>
```

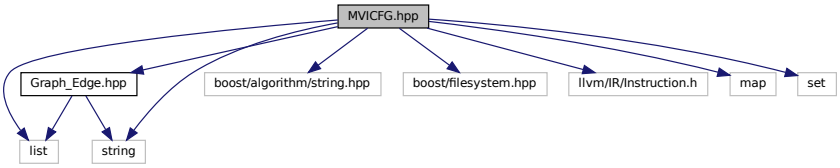
```
      deletedLines,
00815                           std::list<Diff_Mapping> diffMap, unsigned Version) {
00816   /* Update Graph_Line information */
00817   for (auto func : MVICFG->getGraphFunctions()) {
00818     auto findDiff = std::find_if(std::begin(diffMap), std::end(diffMap),
00819                                  [=](Diff_Mapping d) { return (d.getFileName() ==
      func->getFunctionFile()); });
00820     if (findDiff != diffMap.end()) {
00821       for (auto line : func->getFunctionLines()) {
00822         auto findInAdd = std::find_if(std::begin(addedLines), std::end(addedLines),
00823                                       [=](Graph_Line *addLine) { return (addLine == line); });
00824         auto findInDel = std::find_if(std::begin(deletedLines), std::end(deletedLines),
00825                                       [=](Graph_Line *delLine) { return (delLine == line); });
00826         if (findInAdd == addedLines.end() && findInDel == deletedLines.end()) {
00827           /* Line was neither added nor deleted */
00828           unsigned oldLineNumber = line->getLineNumber(MVICFG->getGraphVersion());
00829           if (!MVICFG->isVirtualNodeLineNumber(oldLineNumber)) {
00830             /* Substitute if the line is present in the MVICFG. Otherwise, it's mapping was done
      earlier */
00831             if (oldLineNumber != 0) {
00832               unsigned newLineNumber = findDiff->getAfterLineNumber(oldLineNumber);
00833               if (newLineNumber != std::numeric_limits<unsigned>::max()) {
00834                 line->setLineNumber(Version, newLineNumber);
00835               } else {
00836                 std::cerr << "Incorrect update line for " << oldLineNumber << "\n";
00837               } // End check for newLineNumber being max
00838             }   // End check for oldLineNumber
00839           } else {
00840             /* Assign the same number in this version also */
00841             unsigned newLineNumber = oldLineNumber;
00842             line->setLineNumber(Version, newLineNumber);
00843           } // End check for isVirtualNodeLineNumber
00844         }   // End check for deleted and added line
00845       }     // End loop for line
00846     } else {
00847       if (func->getFunctionFile() == "External_Node_File") {
00848         /* Assign the same number in this version also */
00849         for (auto line : func->getFunctionLines()) {
00850           unsigned oldLineNumber = line->getLineNumber(MVICFG->getGraphVersion());
00851           unsigned newLineNumber = oldLineNumber;
00852           line->setLineNumber(Version, newLineNumber);
00853         } // End loop for line
00854         continue;
00855       } // End check for External Node
00856       std::cerr << "No Line mapping found for " << func->getFunctionFile() << "\n";
00857     } // End check for diffMap.end
00858   }   // End loop for updating Graph_Line information
00859   /* Collect all nodes from which a new edge has originated for this version */
00860   std::list<Graph_Instruction *> mvicfgAddEdgesNodes;
00861   for (auto mvicfgEdge : MVICFG->getGraphEdges()) {
00862     Graph_Line *toLine = mvicfgEdge->getEdgeTo()->getGraphLine();
00863     auto findInAddToLine = std::find_if(std::begin(addedLines), std::end(addedLines),
00864                                         [=](Graph_Line *addLine) { return (addLine == toLine); });
00865     /* Collect all the Nodes from which an new edge for this version originates */
00866     if (findInAddToLine != addedLines.end()) {
00867       mvicfgAddEdgesNodes.push_back(mvicfgEdge->getEdgeFrom());
00868     } // End check for addedLines.end
00869   }   // End loop for collecting Graph_Instruction
00870   for (auto edge : MVICFG->getGraphEdges()) {
00871     Graph_Instruction *edgeFromInst = edge->getEdgeFrom();
00872     if (edgeFromInst->getGraphLine()->getLineNumber(Version) != 0) {
00873       /* The from Node is active for this version */
00874       Graph_Instruction *edgeToInst = edge->getEdgeTo();
00875       if (edgeToInst->getGraphLine()->getLineNumber(Version) != 0) {
00876         /* The to Node is active for this version */
00877         auto findInAddEgdeFrom =
00878             std::find_if(std::begin(mvicfgAddEdgesNodes), std::end(mvicfgAddEdgesNodes),
00879                          [=](Graph_Instruction *instComp) { return (edge->getEdgeFrom() == instComp);
      });
00880         if (findInAddEgdeFrom == mvicfgAddEdgesNodes.end()) {
00881           /* Neither the From node or To node were part of added edges */
00882           if (!edge->isPartOfGraph(Version)) {
00883             edge->pushEdgeVersions(Version);
00884           } // End check for isPartOfGraph
00885         }   // End if for findInAddEgdeFrom
00886       }     // End check for edgeToInst
00887     }       // End check for edgeFromInst
00888   }         // End loop for updating Graph_Edge information
00889 } // End updateMVICFGVersion
00890 } // namespace hydrogen_framework
```

6.39 MVICFG.hpp File Reference

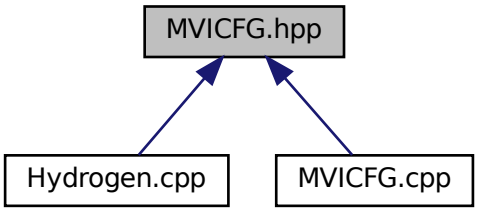
```
#include "Graph_Edge.hpp"
```

```
#include <boost/algorithm/string.hpp>
#include <boost/filesystem.hpp>
#include <list>
#include <llvm/IR/Instruction.h>
#include <map>
#include <set>
#include <string>
```

Include dependency graph for MVICFG.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- std::list< Graph_Line * > hydrogen_framework::addToMVICFG (Graph *MVICFG, Graph *ICFG, Diff_↩ Mapping diff, unsigned Version)
- Graph * hydrogen_framework::buildICFG (Module *mod, unsigned graphVersion)
- std::list< Graph_Line * > hydrogen_framework::deleteFromMVICFG (Graph *MVICFG, Graph *ICFG, Diff↩ _Mapping diff, unsigned Version)
- Graph_Line * hydrogen_framework::findMatchedLine (Graph_Line *t, Graph *matchTo, Graph *matchFrom, Diff_Mapping diff)
- std::list< Diff_Mapping > hydrogen_framework::generateLineMapping (Module *firstMod, Module *secondMod)
- Graph_Edge * hydrogen_framework::getEdge (Graph_Instruction *fromNode, Graph_Instruction *toNode, Graph_Edge::edgeTypes type)
- void hydrogen_framework::getEdgesForAddedLines (Graph *MVICFG, Graph *ICFG, std::list< Graph_Line * > addedLines, std::list< Diff_Mapping > diffMap, unsigned Version)
- std::string hydrogen_framework::getGraphLineInstructionsAsString (Graph_Line *line)
- std::list< Graph_Line * > hydrogen_framework::getGraphLinesGivenLine (Graph *graph, long long lineNo, std::string fileName)
- Graph_Edge * hydrogen_framework::getInBetweenEdge (Graph_Line *fromLine, Graph_Line *toLine)

- Graph_Instruction * hydrogen_framework::getMatchedInstructionFromGraph (Graph *graphToMatch, Graph_Instruction *instToMatch)
- Graph_Line * hydrogen_framework::getNewlyAdded (Graph *MVICFG, Graph *ICFG, Graph_Line *newLine, Diff_Mapping diff)
- std::list< Graph_Line * > hydrogen_framework::getPredGivenGraphLine (Graph_Line *line)
- std::list< Graph_Line * > hydrogen_framework::getSuccGivenGraphLine (Graph_Line *line)
- std::map< Graph_Line *, Graph_Line * > hydrogen_framework::matchedInMVICFG (Graph *MVICFG, Graph *ICFG, Diff_Mapping diff, unsigned Version)
- Graph_Line * hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch (std::list< Graph_Line * > matchedLines, std::string lineFromString, unsigned int graphVersion)
- void hydrogen_framework::updateMVICFGVersion (Graph *MVICFG, std::list< Graph_Line * > addedLines, std::list< Graph_Line * > deletedLines, std::list< Diff_Mapping > diffMap, unsigned Version)

6.39.1 Detailed Description

Author

Ashwin K J

MVICFG : Grouping together auxiliary function for MVICFG

Definition in file MVICFG.hpp.

6.39.2 Function Documentation

6.39.2.1 addToMVICFG()

```
std::list< Graph_Line * > hydrogen_framework::addToMVICFG (
              Graph * MVICFG,
              Graph * ICFG,
              Diff_Mapping diff,
              unsigned Version )
```

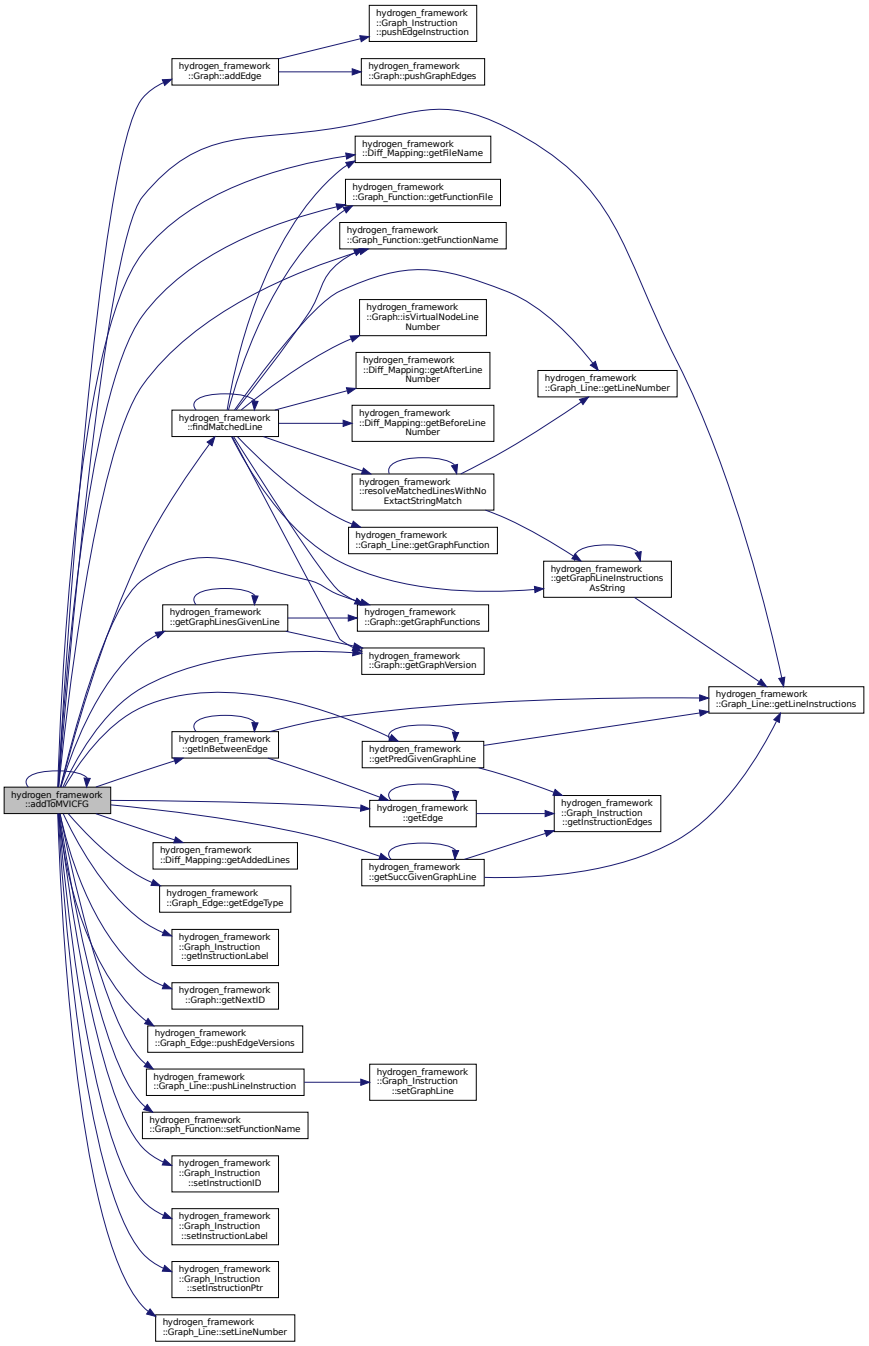
Add nodes to MVICFG and returns the added MVICFG lines

Definition at line 349 of file MVICFG.cpp.

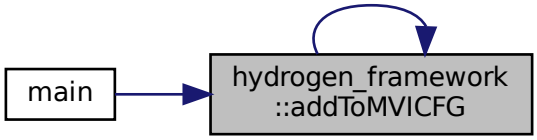
References hydrogen_framework::Graph::addEdge(), hydrogen_framework::addToMVICFG(), hydrogen_framework::findMatchedLine
hydrogen_framework::Diff_Mapping::getAddedLines(), hydrogen_framework::getEdge(), hydrogen_framework::Graph_Edge::getEdge
hydrogen_framework::Diff_Mapping::getFileName(), hydrogen_framework::Graph_Function::getFunctionFile(),
hydrogen_framework::Graph_Function::getFunctionName(), hydrogen_framework::Graph::getGraphFunctions(),
hydrogen_framework::getGraphLinesGivenLine(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::getInBetwe
hydrogen_framework::Graph_Instruction::getInstructionLabel(), hydrogen_framework::Graph_Line::getLineInstructions(),
hydrogen_framework::Graph::getNextID(), hydrogen_framework::getPredGivenGraphLine(), hydrogen_framework::getSuccGivenGra
hydrogen_framework::Graph_Edge::pushEdgeVersions(), hydrogen_framework::Graph_Line::pushLineInstruction(),
hydrogen_framework::Graph_Function::setFunctionName(), hydrogen_framework::Graph_Instruction::setInstructionID(),
hydrogen_framework::Graph_Instruction::setInstructionLabel(), hydrogen_framework::Graph_Instruction::setInstructionPtr(),
and hydrogen_framework::Graph_Line::setLineNumber().

Referenced by hydrogen_framework::addToMVICFG(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.2 buildICFG()

```
Graph * hydrogen_framework::buildICFG (
            Module * mod,
            unsigned graphVersion )
```

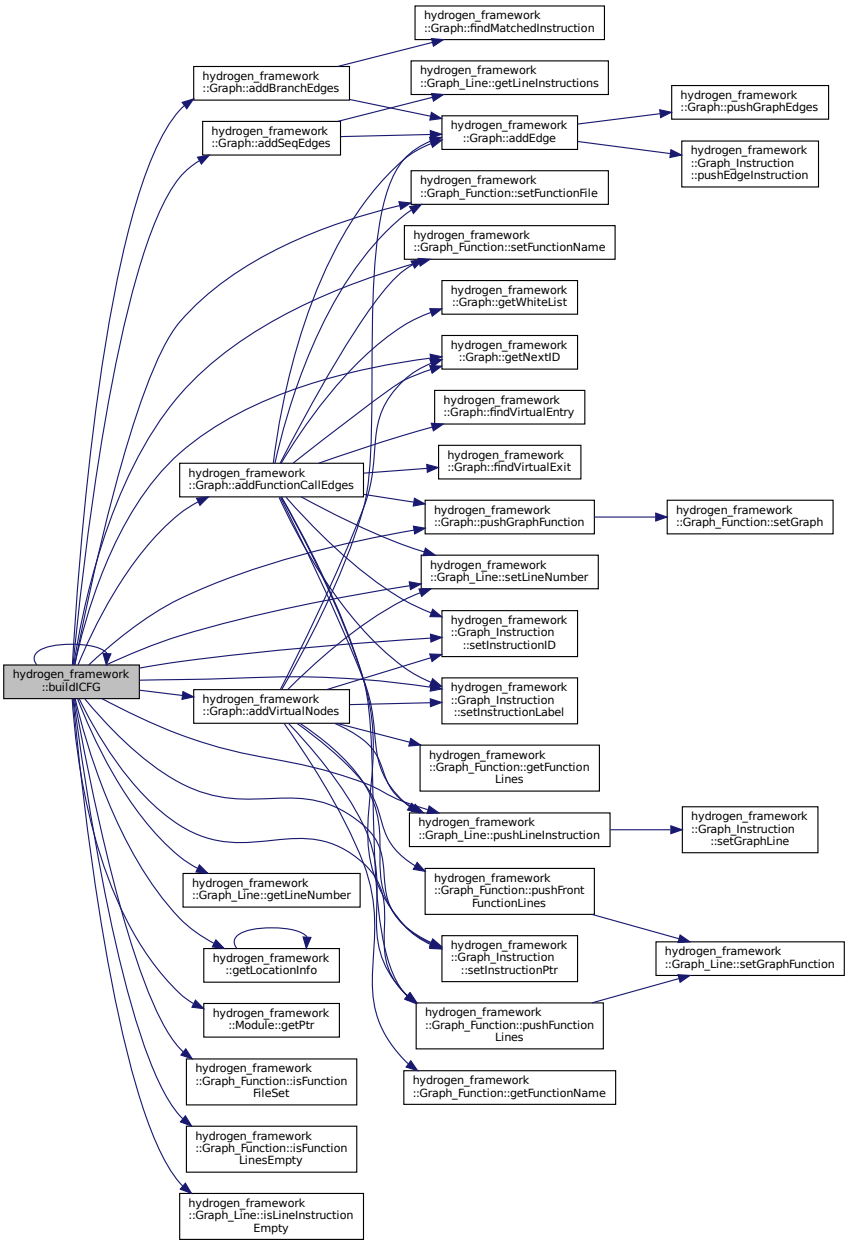
Build ICFG for the given module

Definition at line 15 of file MVICFG.cpp.

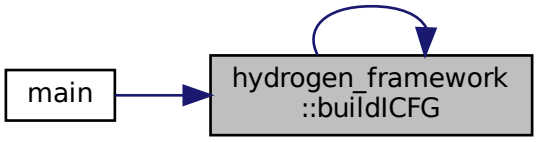
References hydrogen_framework::Graph::addBranchEdges(), hydrogen_framework::Graph::addFunctionCallEdges(), hydrogen_framework::Graph::addSeqEdges(), hydrogen_framework::Graph::addVirtualNodes(), hydrogen_framework::buildICFG(), hydrogen_framework::Graph_Line::getLineNumber(), hydrogen_framework::getLocationInfo(), hydrogen_framework::Graph::getNextI hydrogen_framework::Module::getPtr(), hydrogen_framework::Graph_Function::isFunctionFileSet(), hydrogen_framework::Graph_Fu hydrogen_framework::Graph_Line::isLineInstructionEmpty(), hydrogen_framework::Graph_Function::pushFunctionLines(), hydrogen_framework::Graph::pushGraphFunction(), hydrogen_framework::Graph_Line::pushLineInstruction(), hydrogen_framework::Graph_Function::setFunctionFile(), hydrogen_framework::Graph_Function::setFunctionName(), hydrogen_framework::Graph_Instruction::setInstructionID(), hydrogen_framework::Graph_Instruction::setInstructionLabel(), hydrogen_framework::Graph_Instruction::setInstructionPtr(), and hydrogen_framework::Graph_Line::setLineNumber().

Referenced by hydrogen_framework::buildICFG(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.3 deleteFromMVICFG()

```
std::list< Graph_Line * > hydrogen_framework::deleteFromMVICFG (
              Graph * MVICFG,
              Graph * ICFG,
              Diff_Mapping diff,
              unsigned Version )
```

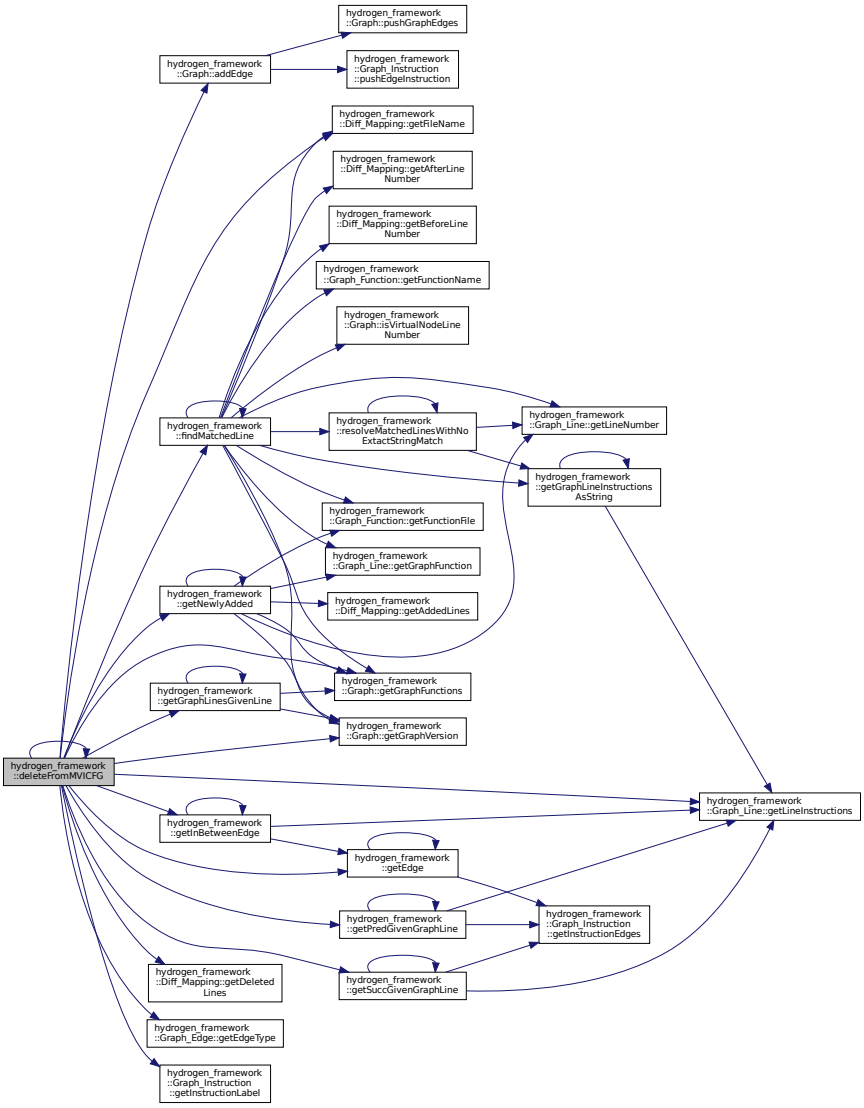
Mark deleted nodes in MVICFG and returns the deleted MVICFG lines

Definition at line 556 of file MVICFG.cpp.

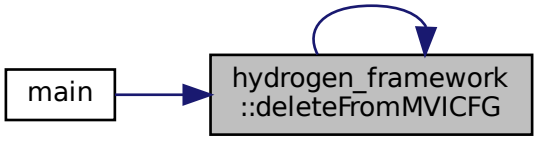
References hydrogen_framework::Graph::addEdge(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::findMatche
hydrogen_framework::Diff_Mapping::getDeletedLines(), hydrogen_framework::getEdge(), hydrogen_framework::Graph_Edge::getEdg
hydrogen_framework::Diff_Mapping::getFileName(), hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::getGr
hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::getInBetweenEdge(), hydrogen_framework::Graph_Instructio
hydrogen_framework::Graph_Line::getLineInstructions(), hydrogen_framework::getNewlyAdded(), hydrogen_framework::getPredGive
and hydrogen_framework::getSuccGivenGraphLine().

Referenced by hydrogen_framework::deleteFromMVICFG(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.4 findMatchedLine()

```
Graph_Line * hydrogen_framework::findMatchedLine (
             Graph_Line * t,
             Graph * matchTo,
             Graph * matchFrom,
             Diff_Mapping diff )
```

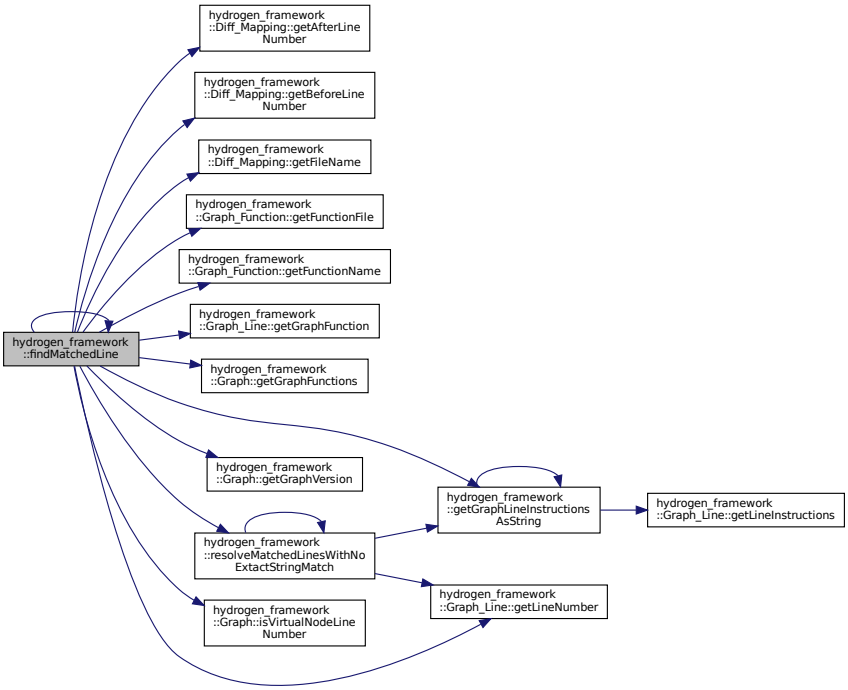
Find matched Node Returns NULL if no match found Always make sure to check that the Graph_Line is from the diff being used

Definition at line 233 of file MVICFG.cpp.

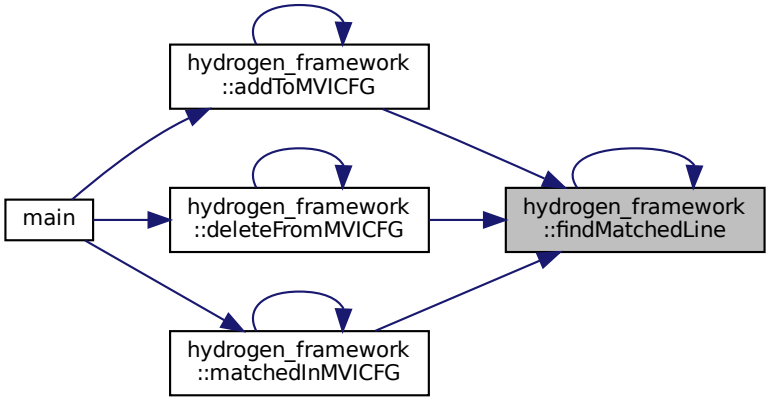
References hydrogen_framework::findMatchedLine(), hydrogen_framework::Diff_Mapping::getAfterLineNumber(), hydrogen_framework::Diff_Mapping::getBeforeLineNumber(), hydrogen_framework::Diff_Mapping::getFileName(), hydrogen_framework::Graph_Function::getFunctionFile(), hydrogen_framework::Graph_Function::getFunctionName(), hydrogen_framework::Graph_Line::getGraphFunction(), hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::getGraphLineInstructionsAsString(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Line::getLineNumber(), hydrogen_framework::Graph::isVirtualNodeLineNumber(), and hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::findMatch and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.5 generateLineMapping()

```
std::list< Diff_Mapping > hydrogen_framework::generateLineMapping (
            Module * firstMod,
            Module * secondMod )
```

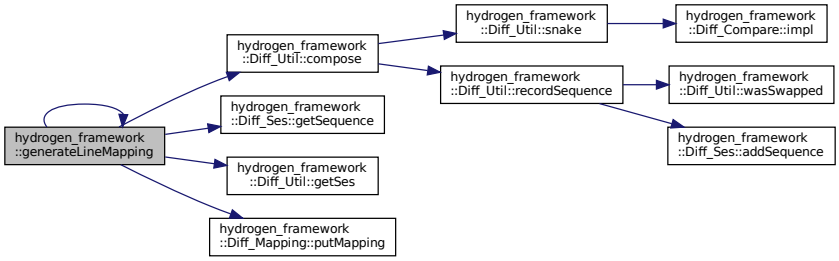
Generate Line Mappings between two modules

Definition at line 75 of file MVICFG.cpp.

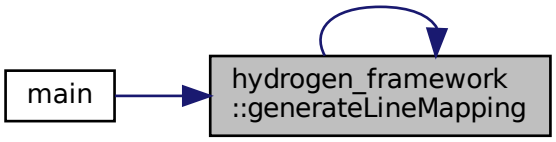
References hydrogen_framework::Diff_Util::compose(), hydrogen_framework::generateLineMapping(), hydrogen_framework::Diff_Ses
hydrogen_framework::Diff_Util::getSes(), and hydrogen_framework::Diff_Mapping::putMapping().

Referenced by hydrogen_framework::generateLineMapping(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.6 getEdge()

```
Graph_Edge * hydrogen_framework::getEdge (
             Graph_Instruction * fromNode,
             Graph_Instruction * toNode,
             Graph_Edge::edgeTypes type )
```

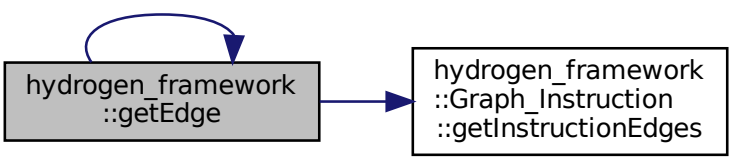
Get the edge between two given nodes Returns NULL if no match found

Definition at line 300 of file MVICFG.cpp.

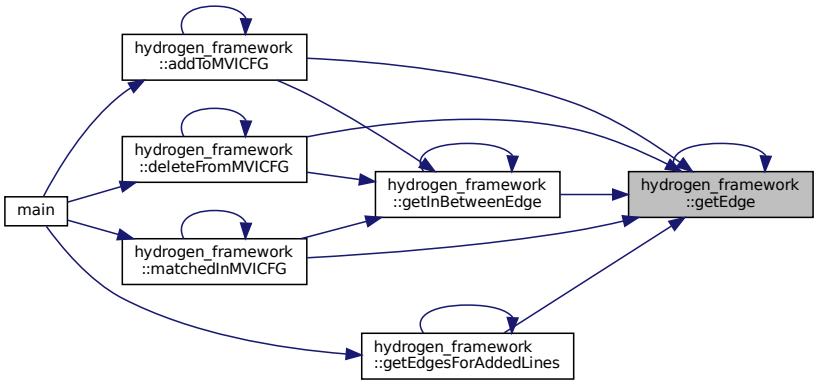
References hydrogen_framework::getEdge(), and hydrogen_framework::Graph_Instruction::getInstructionEdges().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getEdge(), hydrogen_framework::getEdgesForAddedLines(), hydrogen_framework::getInBetweenEdge(), and hydrogen_framework::matchedInM

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.7 getEdgesForAddedLines()

```
void hydrogen_framework::getEdgesForAddedLines (
            Graph * MVICFG,
            Graph * ICFG,
            std::list< Graph_Line * > addedLines,
            std::list< Diff_Mapping > diffMap,
            unsigned Version )
```

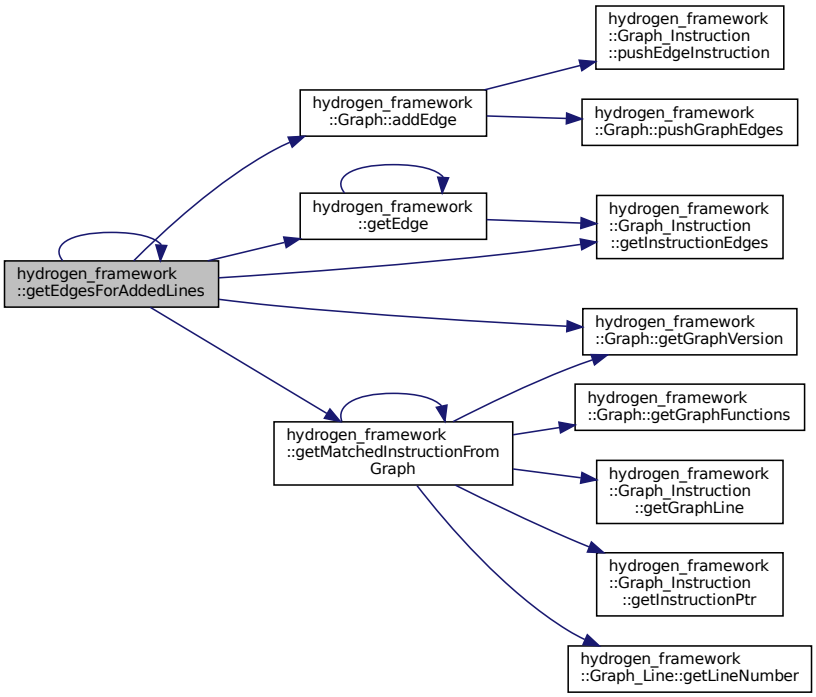
Import edges from ICFG instruction for added Graph_Line

Definition at line 523 of file MVICFG.cpp.

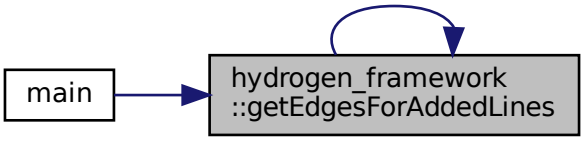
References hydrogen_framework::Graph::addEdge(), hydrogen_framework::getEdge(), hydrogen_framework::getEdgesForAddedLine
hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Instruction::getInstructionEdges(),
and hydrogen_framework::getMatchedInstructionFromGraph().

Referenced by hydrogen_framework::getEdgesForAddedLines(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.8 getGraphLineInstructionsAsString()

```
std::string hydrogen_framework::getGraphLineInstructionsAsString (
            Graph_Line * line )
```

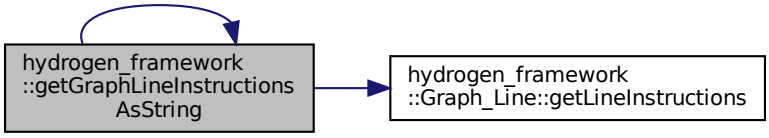
Get the OpCode of the Instructions in a Graph_Line as String in the order in which they appear Returns empty string if none of the Graph_Instruction had Instruction_Ptr

Definition at line 185 of file MVICFG.cpp.

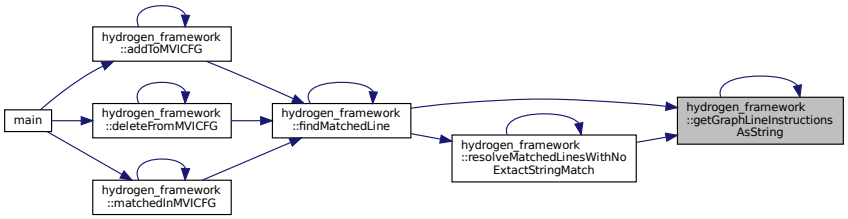
References hydrogen_framework::getGraphLineInstructionsAsString(), and hydrogen_framework::Graph_Line::getLineInstructions().

Referenced by hydrogen_framework::findMatchedLine(), hydrogen_framework::getGraphLineInstructionsAsString(), and hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.9 getGraphLinesGivenLine()

```
std::list< Graph_Line * > hydrogen_framework::getGraphLinesGivenLine (
            Graph * graph,
            long long lineNo,
            std::string fileName )
```

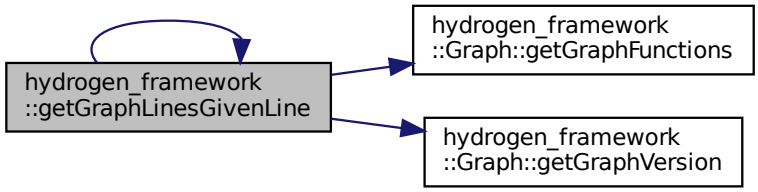
Get Graph_Line(s) from given source line Returns empty list if no Graph_Line is not found

Definition at line 139 of file MVICFG.cpp.

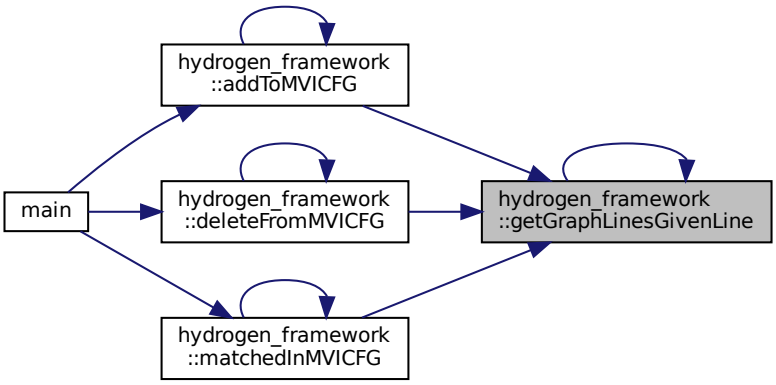
References hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::getGraphLinesGivenLine(), and hydrogen_framework::Graph::getGraphVersion().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getGraphl and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.10 getInBetweenEdge()

```
Graph_Edge * hydrogen_framework::getInBetweenEdge (
              Graph_Line * fromLine,
              Graph_Line * toLine )
```

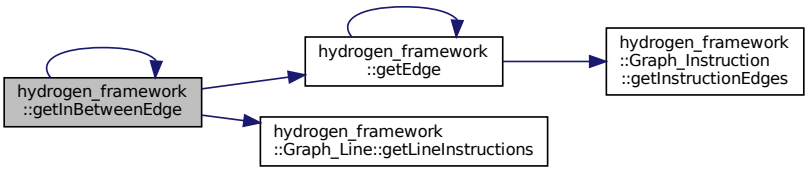
Get the first edge between two Graph_Line It does reverse propagation for Instructions of fromLine and forward propagation for toLine Instructions Used only when getEdge fails to find an edge where one is expected Returns NULL if no match is found

Definition at line 315 of file MVICFG.cpp.

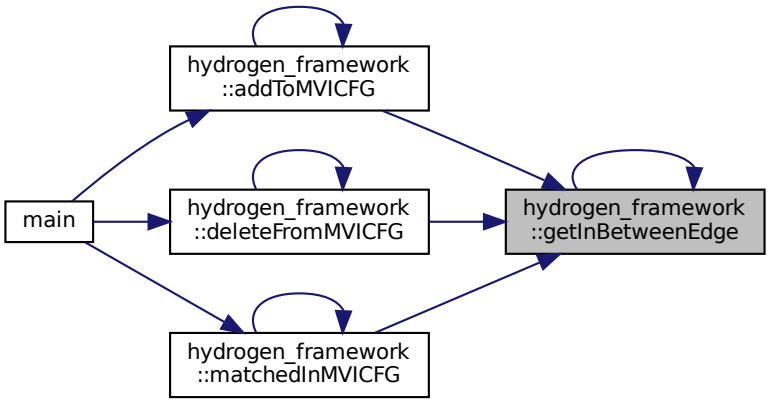
References hydrogen_framework::getEdge(), hydrogen_framework::getInBetweenEdge(), and hydrogen_framework::Graph_Line::get

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getInBetw and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.11 getMatchedInstructionFromGraph()

```
Graph_Instruction * hydrogen_framework::getMatchedInstructionFromGraph (
            Graph * graphToMatch,
            Graph_Instruction * instToMatch )
```

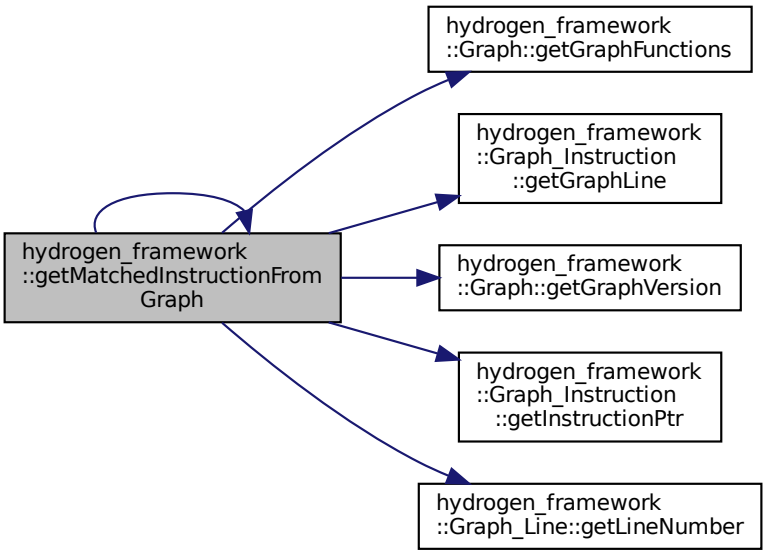
Get matching Graph_Instruction from given Graph given a Graph_Instruction using LLVM PTR Return NULL if no match is found

Definition at line 499 of file MVICFG.cpp.

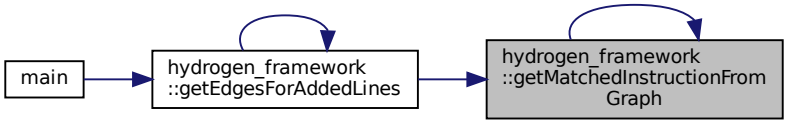
References hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::Graph_Instruction::getGraphLine(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Instruction::getInstructionPtr(), hydrogen_framework::Graph_Line::getLineNumber(), and hydrogen_framework::getMatchedInstructionFromGraph().

Referenced by hydrogen_framework::getEdgesForAddedLines(), and hydrogen_framework::getMatchedInstructionFromGraph().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.12 getNewlyAdded()

```
Graph_Line * hydrogen_framework::getNewlyAdded (
             Graph * MVICFG,
             Graph * ICFG,
             Graph_Line * newLine,
             Diff_Mapping diff )
```

Get the newly added MVICFG Graph_Line corresponding to the given ICFG Graph_Line Used only when find↩ MatchedLine fails to retrieve the same Returns NULL if no such line is found

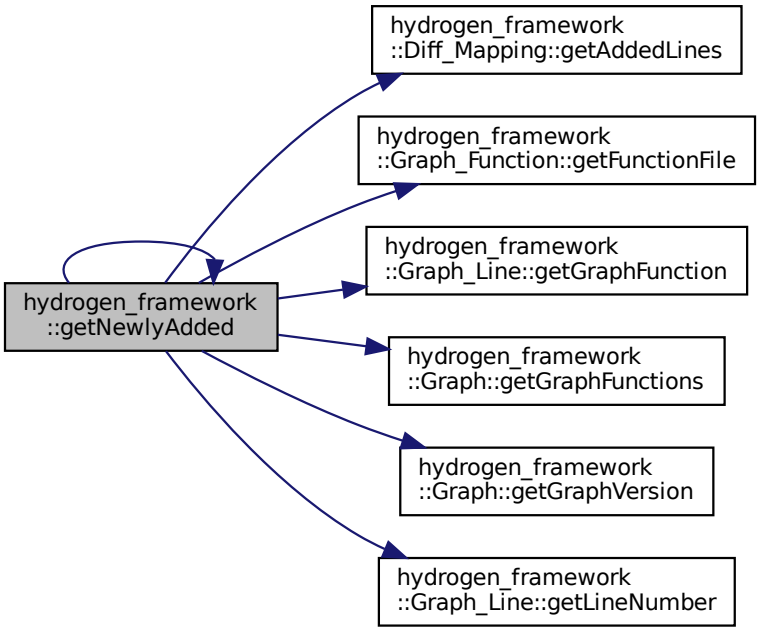
Definition at line 330 of file MVICFG.cpp.

References hydrogen_framework::Diff_Mapping::getAddedLines(), hydrogen_framework::Graph_Function::getFunctionFile(), hydrogen_framework::Graph_Line::getGraphFunction(), hydrogen_framework::Graph::getGraphFunctions(),

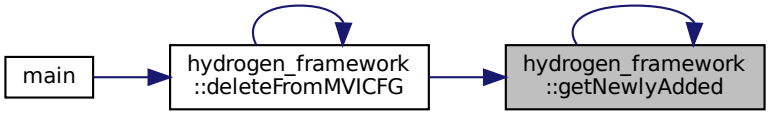
hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Line::getLineNumber(), and hydrogen_framework::getNewlyAdded().

Referenced by hydrogen_framework::deleteFromMVICFG(), and hydrogen_framework::getNewlyAdded().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.13 getPredGivenGraphLine()

```
std::list< Graph_Line * > hydrogen_framework::getPredGivenGraphLine (
            Graph_Line * line )
```

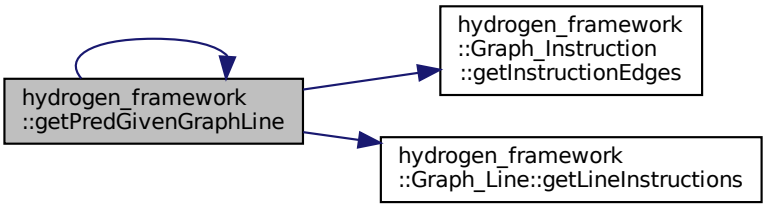
Get predecessor of a given Graph_Line

Definition at line 161 of file MVICFG.cpp.

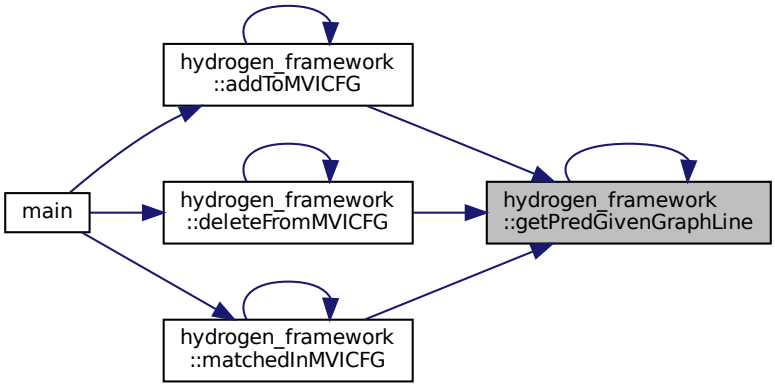
References hydrogen_framework::Graph_Instruction::getInstructionEdges(), hydrogen_framework::Graph_Line::getLineInstructions() and hydrogen_framework::getPredGivenGraphLine().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getPredGi and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.14 getSuccGivenGraphLine()

```
std::list< Graph_Line * > hydrogen_framework::getSuccGivenGraphLine (
            Graph_Line * line )
```

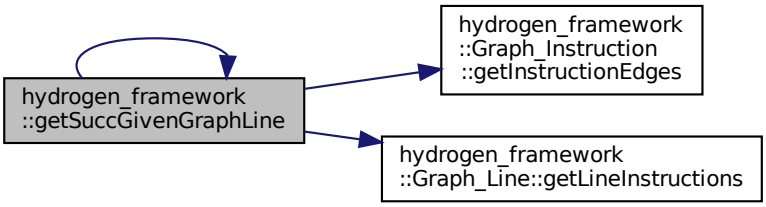
Get successor of a given Graph_Line

Definition at line 173 of file MVICFG.cpp.

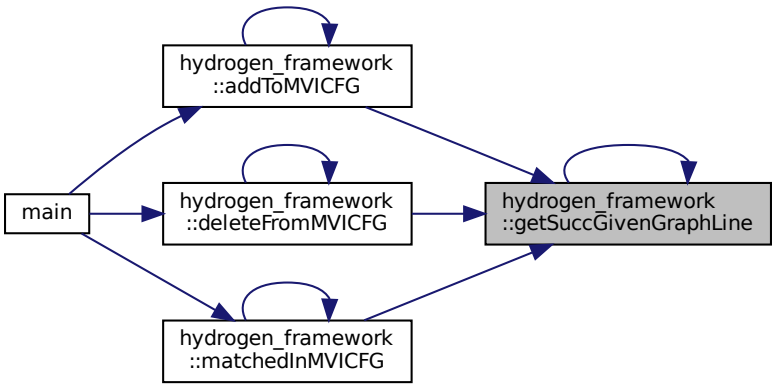
References hydrogen_framework::Graph_Instruction::getInstructionEdges(), hydrogen_framework::Graph_Line::getLineInstructions() and hydrogen_framework::getSuccGivenGraphLine().

Referenced by hydrogen_framework::addToMVICFG(), hydrogen_framework::deleteFromMVICFG(), hydrogen_framework::getSuccG and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.15 matchedInMVICFG()

```
std::map< Graph_Line *, Graph_Line * > hydrogen_framework::matchedInMVICFG (
            Graph * MVICFG,
            Graph * ICFG,
            Diff_Mapping diff,
            unsigned Version )
```

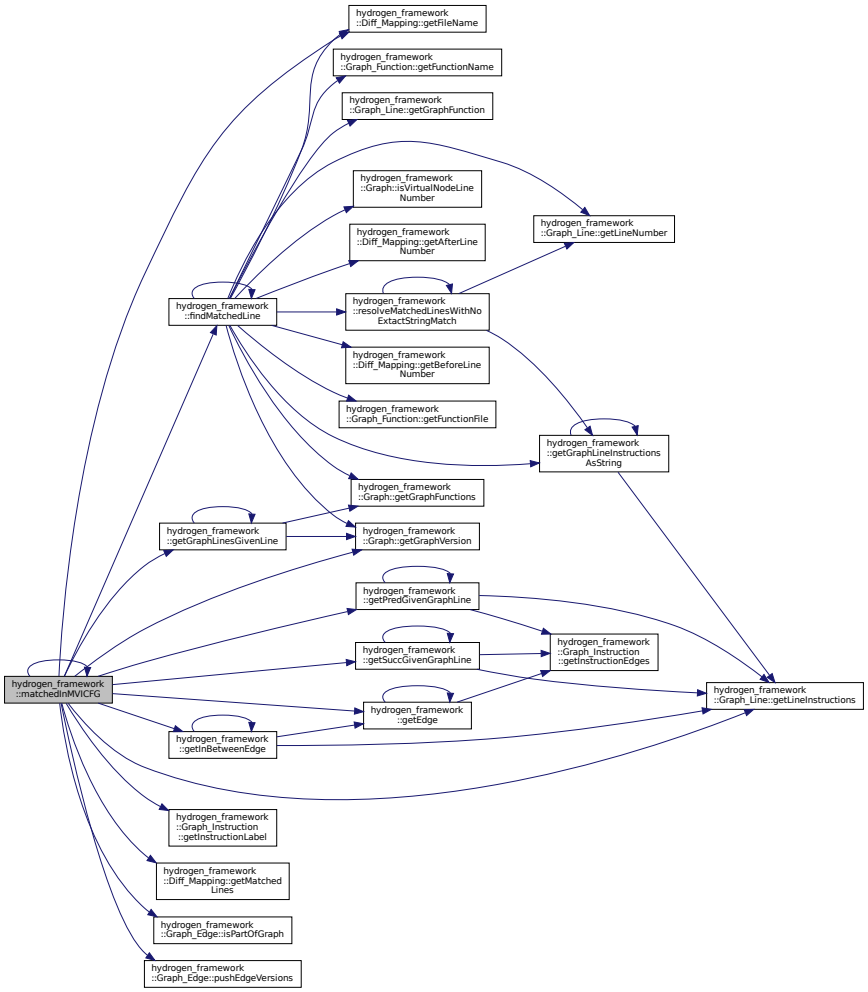
Returns the corresponding matched Graph_Line in MVICFG from ICFG

Definition at line 703 of file MVICFG.cpp.

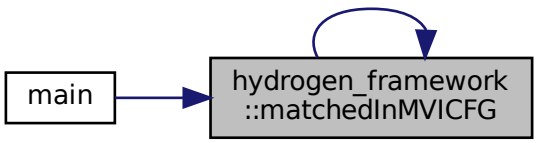
References hydrogen_framework::findMatchedLine(), hydrogen_framework::getEdge(), hydrogen_framework::Diff_Mapping::getFileN
hydrogen_framework::getGraphLinesGivenLine(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::getInBetwe
hydrogen_framework::Graph_Instruction::getInstructionLabel(), hydrogen_framework::Graph_Line::getLineInstructions(),
hydrogen_framework::Diff_Mapping::getMatchedLines(), hydrogen_framework::getPredGivenGraphLine(), hydrogen_framework::getS
hydrogen_framework::Graph_Edge::isPartOfGraph(), hydrogen_framework::matchedInMVICFG(), and hydrogen_framework::Graph_

Referenced by main(), and hydrogen_framework::matchedInMVICFG().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.16 resolveMatchedLinesWithNoExtactStringMatch()

```
Graph_Line * hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch (
             std::list< Graph_Line * > matchedLines,
             std::string lineFromString,
             unsigned int graphVersion )
```

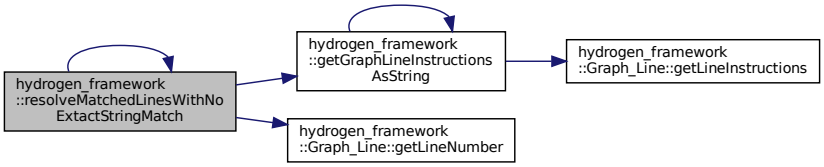
Heuristically try to find the closest Graph_Line match from a list of potential Graph_Line matches when no exact match is found using getGraphLineInstructionsAsString Currently will throw an warning if heuristic skips more than 2 OpCode to match the lines Returns NULL if no heuristic match is found

Definition at line 200 of file MVICFG.cpp.

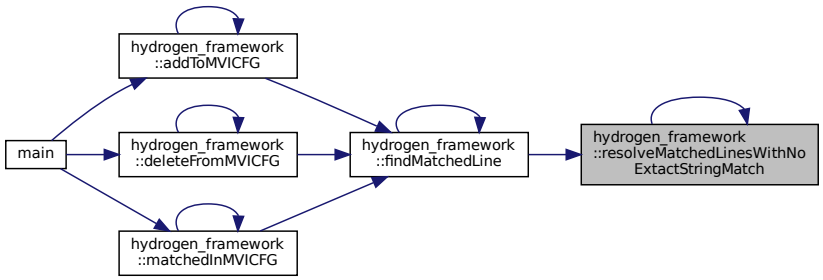
References hydrogen_framework::getGraphLineInstructionsAsString(), hydrogen_framework::Graph_Line::getLineNumber(), and hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch().

Referenced by hydrogen_framework::findMatchedLine(), and hydrogen_framework::resolveMatchedLinesWithNoExtactStringMatch().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.2.17 updateMVICFGVersion()

```
void hydrogen_framework::updateMVICFGVersion (
            Graph * MVICFG,
            std::list< Graph_Line * > addedLines,
            std::list< Graph_Line * > deletedLines,
            std::list< Diff_Mapping > diffMap,
            unsigned Version )
```

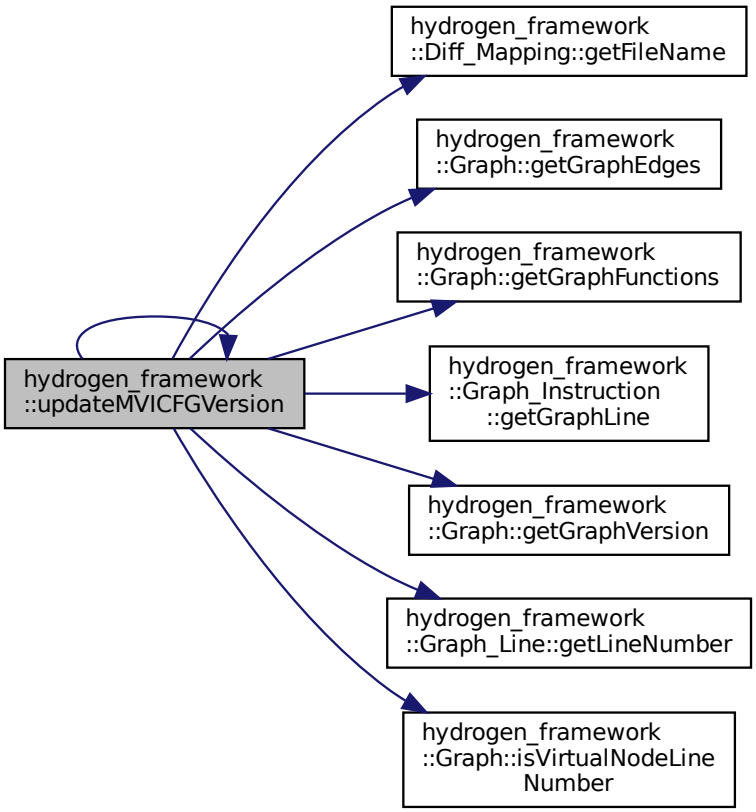
Update the Edge and Node information for MVICFG

Definition at line 814 of file MVICFG.cpp.

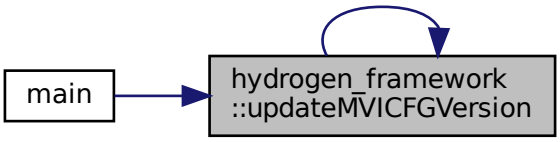
References hydrogen_framework::Diff_Mapping::getFileName(), hydrogen_framework::Graph::getGraphEdges(), hydrogen_framework::Graph::getGraphFunctions(), hydrogen_framework::Graph_Instruction::getGraphLine(), hydrogen_framework::Graph::getGraphVersion(), hydrogen_framework::Graph_Line::getLineNumber(), hydrogen_framework::Graph: and hydrogen_framework::updateMVICFGVersion().

Referenced by main(), and hydrogen_framework::updateMVICFGVersion().

Here is the call graph for this function:



Here is the caller graph for this function:



6.40 MVICFG.hpp

```
00001
00006 #ifndef MVICFG_H
00007 #define MVICFG_H
00008
00009 #include "Graph_Edge.hpp"
00010 #include <boost/algorithm/string.hpp>
00011 #include <boost/filesystem.hpp>
00012 #include <list>
00013 #include <llvm/IR/Instruction.h>
00014 #include <map>
00015 #include <set>
00016 #include <string>
00017 namespace hydrogen_framework {
00018 /* Forward declaration */
00019 class Diff_Mapping;
00020 class Graph;
00021 class Graph_Edge;
00022 class Graph_Instruction;
00023 class Graph_Line;
00024 class Module;
00025
00029 Graph *buildICFG(Module *mod, unsigned graphVersion);
00030
00034 std::list<Diff_Mapping> generateLineMapping(Module *firstMod, Module *secondMod);
00035
00040 std::list<Graph_Line *> getGraphLinesGivenLine(Graph *graph, long long lineNo, std::string fileName);
00041
00045 std::list<Graph_Line *> getPredGivenGraphLine(Graph_Line *line);
00046
00050 std::list<Graph_Line *> getSuccGivenGraphLine(Graph_Line *line);
00051
00056 std::string getGraphLineInstructionsAsString(Graph_Line *line);
00057
00064 Graph_Line *resolveMatchedLinesWithNoExtactStringMatch(std::list<Graph_Line *> matchedLines,
       std::string lineFromString,
00065                                                       unsigned int graphVersion);
00066
00072 Graph_Line *findMatchedLine(Graph_Line *t, Graph *matchTo, Graph *matchFrom, Diff_Mapping diff);
00073
00078 Graph_Edge *getEdge(Graph_Instruction *fromNode, Graph_Instruction *toNode, Graph_Edge::edgeTypes
       type);
00079
00086 Graph_Edge *getInBetweenEdge(Graph_Line *fromLine, Graph_Line *toLine);
00087
00093 Graph_Line *getNewlyAdded(Graph *MVICFG, Graph *ICFG, Graph_Line *newLine, Diff_Mapping diff);
00094
00098 std::list<Graph_Line *> addToMVICFG(Graph *MVICFG, Graph *ICFG, Diff_Mapping diff, unsigned Version);
00099
00104 Graph_Instruction *getMatchedInstructionFromGraph(Graph *graphToMatch, Graph_Instruction
       *instToMatch);
00105
00109 void getEdgesForAddedLines(Graph *MVICFG, Graph *ICFG, std::list<Graph_Line *> addedLines,
00110                            std::list<Diff_Mapping> diffMap, unsigned Version);
00111
00115 std::list<Graph_Line *> deleteFromMVICFG(Graph *MVICFG, Graph *ICFG, Diff_Mapping diff, unsigned
       Version);
00116
00120 std::map<Graph_Line *, Graph_Line *> matchedInMVICFG(Graph *MVICFG, Graph *ICFG, Diff_Mapping diff,
       unsigned Version);
00121
00125 void updateMVICFGVersion(Graph *MVICFG, std::list<Graph_Line *> addedLines, std::list<Graph_Line *>
       deletedLines,
```

```
00126                           std::list<Diff_Mapping> diffMap, unsigned Version);
00127 } // namespace hydrogen_framework
00128 #endif
```

Index