Code Atlas User Manual

Yaobin Ouyang

Contents

Introduction 1

Installation 3

Quick Start 6

Feature List 15

- Show Functions/Variables/Classes 15
- Find Function Calls 15
- Find Class Members 16
- Find Class Hierarchy 16
- Find Variable's References 17
- Jump to Other Function/Variable/ Class 17
- Delete Nodes or Edges 17
- Find Call Path 18
- Add Functions to Black List 18
- Selection 19
- Save Call Graph 21
- Add comments to Nodes 23
- Add Edge Manually 24

Introduction

Many programmers, especially who are new to programming, find it difficult to look at and understand the code. That's because it's difficult to find out the call relationships between functions. Sometimes one has to remember call relationships between many functions in order to understand how a module works. I used to draw call graph on paper, so I thought it may be good to use a program to do this automatically. That's the origin of this project.

At the beginning, I thought I can use a map to visualize the whole program, this map can help users to navigate the code, understand the structure of program. I read some paper and make several demos based on Qt Creator or Understand. I got in touch with some theories and algorithms in data mining, visualization and software engineering. However, the goal I made is too big. I had to deal with many trivial details. So the process was slow and the demos were far from useful.

Then I graduated and entered a game company. There was always a lot of work to do and I had to give up my big goal. I decided to solve the code reading problem in my work first and then refine the system gradually. My company uses python for game logic, and my colleagues like Sublime Text very much, because of its convenient navigation and coding features. So my new goal is to write a plug-in for Sublime Text, which can help me to navigate in python call graph. I called my plug-in Code Atlas because I hope it can tell users where to go, just like a map for us when we go to a new place. After I went to another company, my main coding language turned to C++, so I made Code Atlas support C++ and add some useful features in it.

What can Code Atlas do? Mainly, helping programmer to navigate the code. When a programmer begin with an existing project, he knows nothing about it and has to spend a lot of time reading code, in order to find out call relationships between functions, important call patterns and code for important features or algorithms. Using Code Atlas, a user can navigate in the call graph conveniently (just by using Alt + Up/Down/Left/Right Arrow). Also, he can save the call graph he read before and write comments on important functions and show, which helps him to remember code read before quickly and avoid inefficient repeated work.

Now the Code Atlas support C/C++ and python. Theoretically, Java and C# are supported, but I

haven't tested them yet.

I wrote a simple tutorial on my blog, here is the link:

http://www.cnblogs.com/dydx/p/5393802.html

Here is a tutorial on youtube:

https://www.youtube.com/playlist?list=PLN16zMWJLkHLgHhTJUIkwp5chgnFz9_NH

All advices or comments are welcome, my e-mail is 549088764@qq.com, and my github homepage is https://github.com/league1991.

Sept 16, 2016

Installation

Following tools are needed:

Name	Version	Usage	Webpage
Python	3.4	Code Atlas uses python	https://www.python.org/ftp/python/3.4.0/python-3.4.0.msi
PyQt	Correspond Qt 4.8 and Python3.4	UI library	https://sourceforge.net/projects/pyqt/files/PyQt4/PyQt-4.11.4/PyQt4-4.11.4-gpl-Py3.4-Qt4.8.7-x32.exe/download
Sublime Text	3.x	Code editor	https://download.sublimetext.com/Sublime%20Text%20Build%203114%20Setup.exe
Understand	3.x	Code analysis tool	https://scitools.com/ (official site, registration is needed before downloading)
CodeAtlas	None	Code Atlas itself	https://github.com/league1991/CodeAtlasSublime (Click "Clone or download" and download the zip package)

Note that all the packages above should be 32-bit. One should ensure the corresponding python version of PyQt should be 3.4. That's because Understand's python API is for python 3.4.

Here is the installation steps:

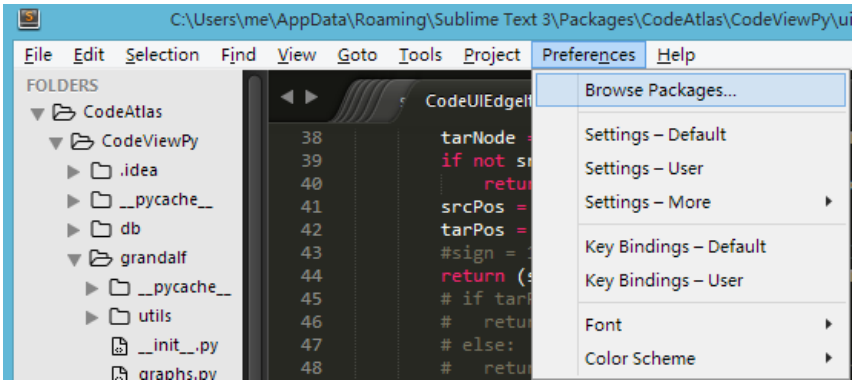
（1）Install Understand, just double-click the installation package and follow the wizard.

（2）Install Sublime Text, just double-click the installation package and follow the wizard.

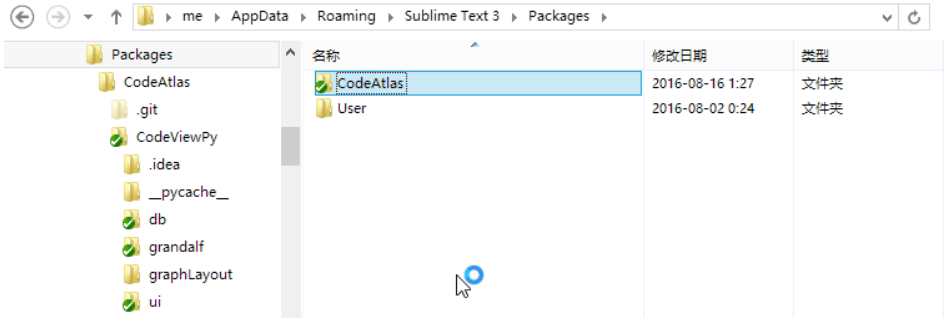
（3）Install Python, just double-click the installation package and follow the wizard.

（4）Install PyQt, just double-click the installation package and follow the wizard.

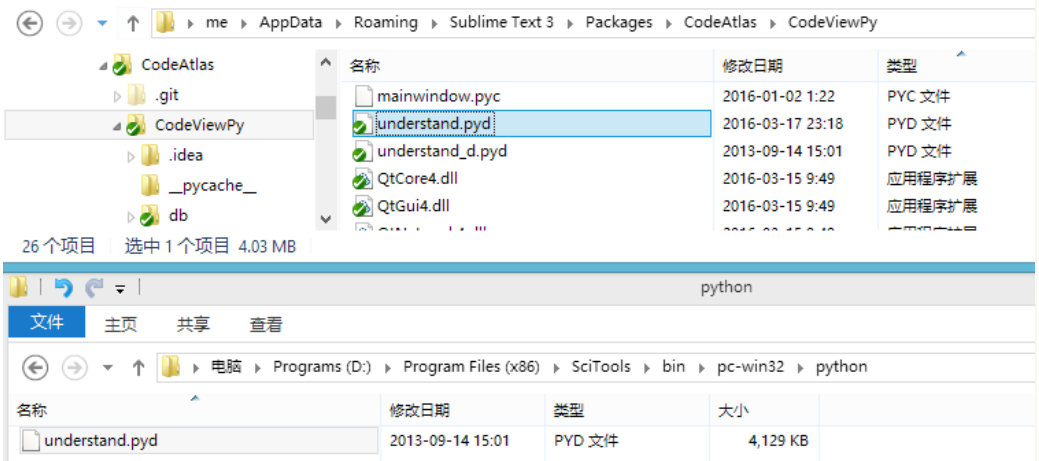
（5）Open Sublime Text，Click Preferences -> Browse Packages and open the Packages folder.



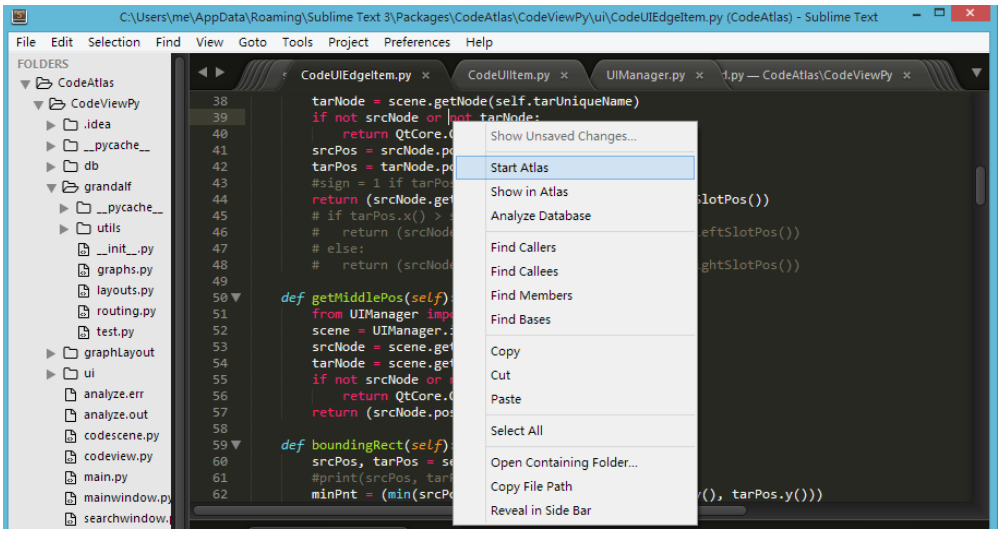
（6）Extract the zip package of Code Atlas to that folder.



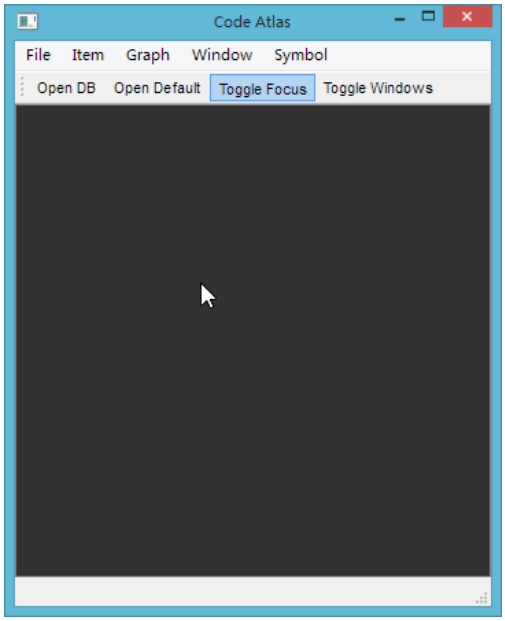
（7）Replace Packages/CodeAtlas/CodeViewPy/understand.pyd with the file under Understand's installation folder(Usually in SciTools\bin\pc-win32\python). The pyd file is Understand's API library. Replacing it can prove we can use the newer version of Understand API.



（8）Now Code Atlas is installed. In order to check it, you can open Sublime Text, press right mouse button in the coding area and choose "Start Atlas".



If the Code Atlas is installed successfully, you can see this window.



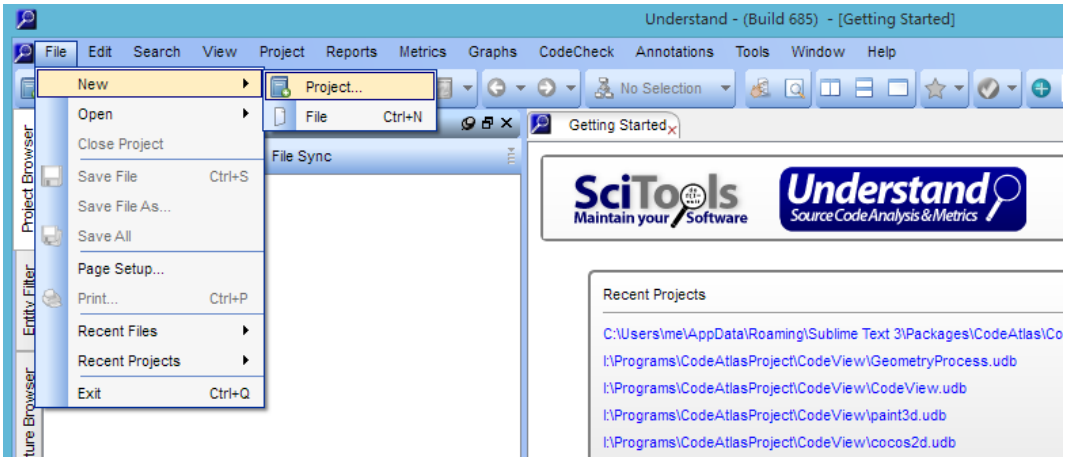
Now we can use the Code Atlas!

Quick Start

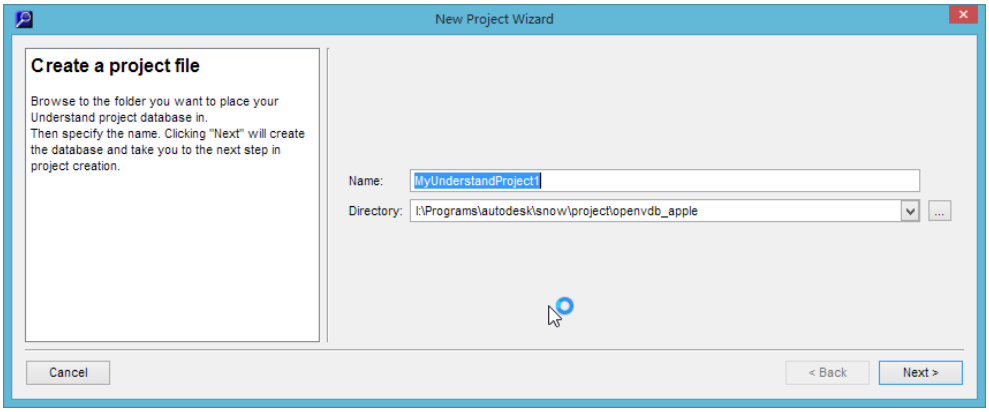
When navigating the code, Code Atlas query Understand database and show the result using node-link graph. So a user should build the Understand database first, open Code Atlas in Sublime Text and open the Understand database in Code Atlas.

Suppose we have a project with C++ source code. There may be a Visual Studio solution in it, or may not. Here are the steps of setting up Understand database and using Code Atlas to navigate the code:

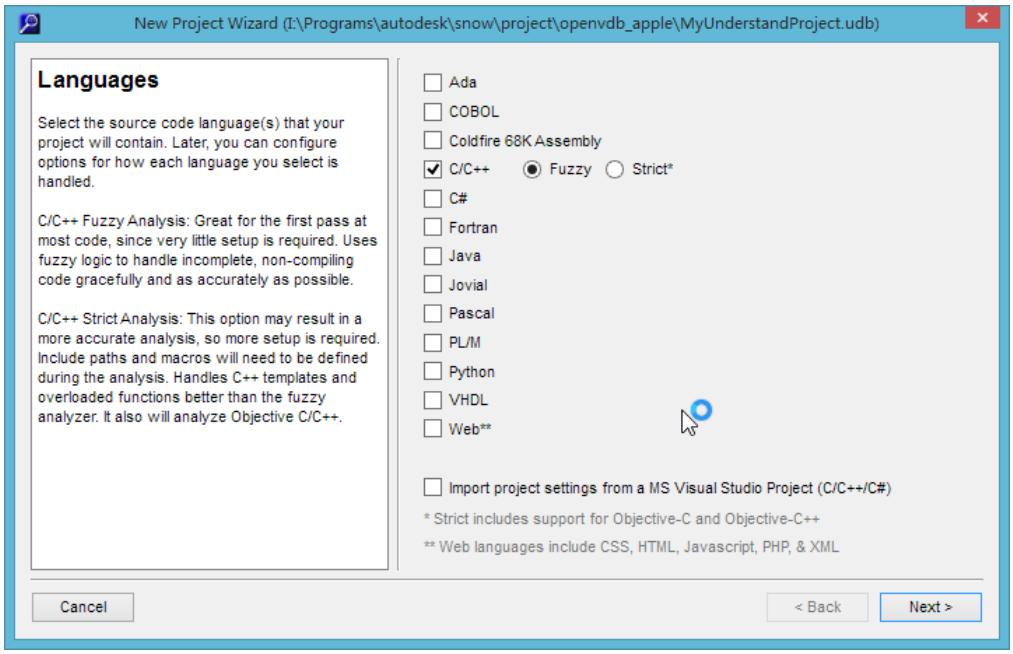
（1）Open Understand, and click File -> New -> Project.



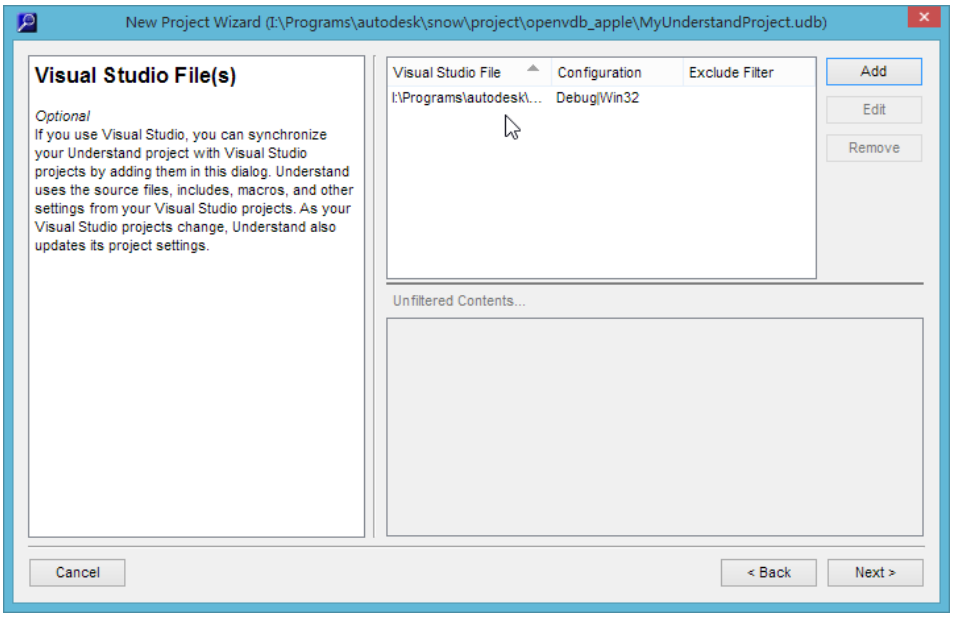
（2）Now we can see a dialog. Please enter the name of the database in the "Name" box and choose the code folder in the "Directory" box.



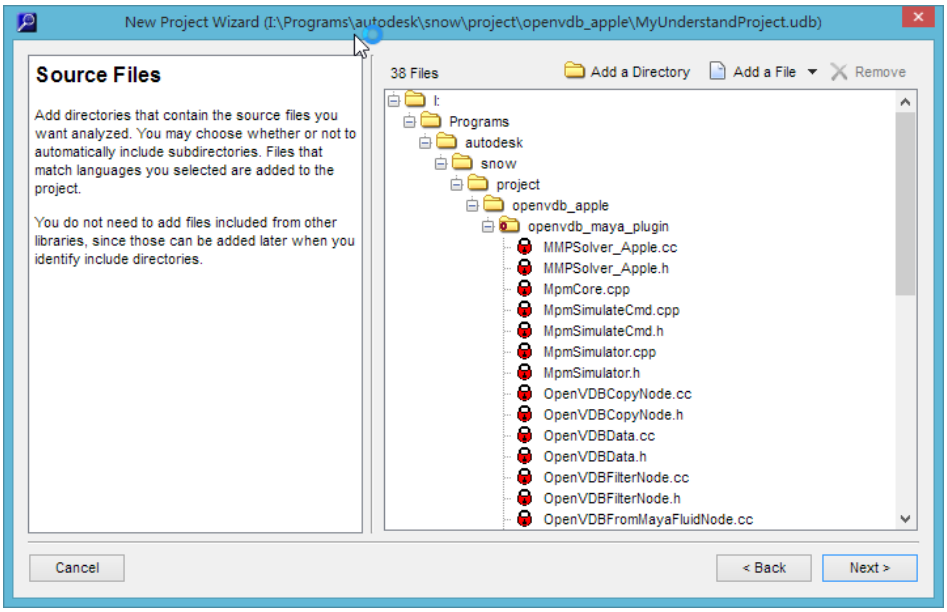
（3）Choose C/C++ language. Note that there is a check box at the bottom, asking you to import Visual Studio project. If you have such project, check it and the analysis result will be more accurate.



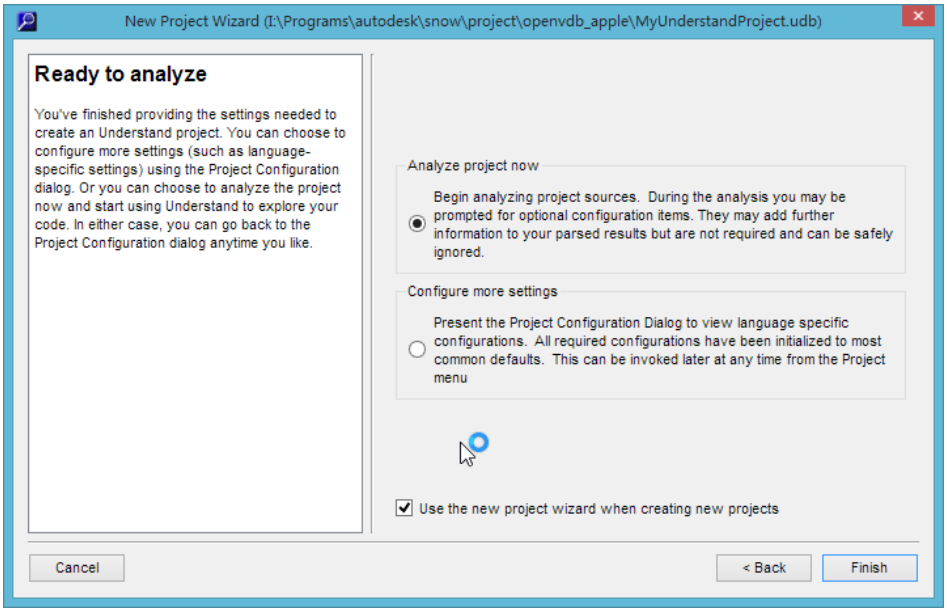
（4）If you choose to import Visual Studio project last step, please specify the solution(.sln) file of the project.



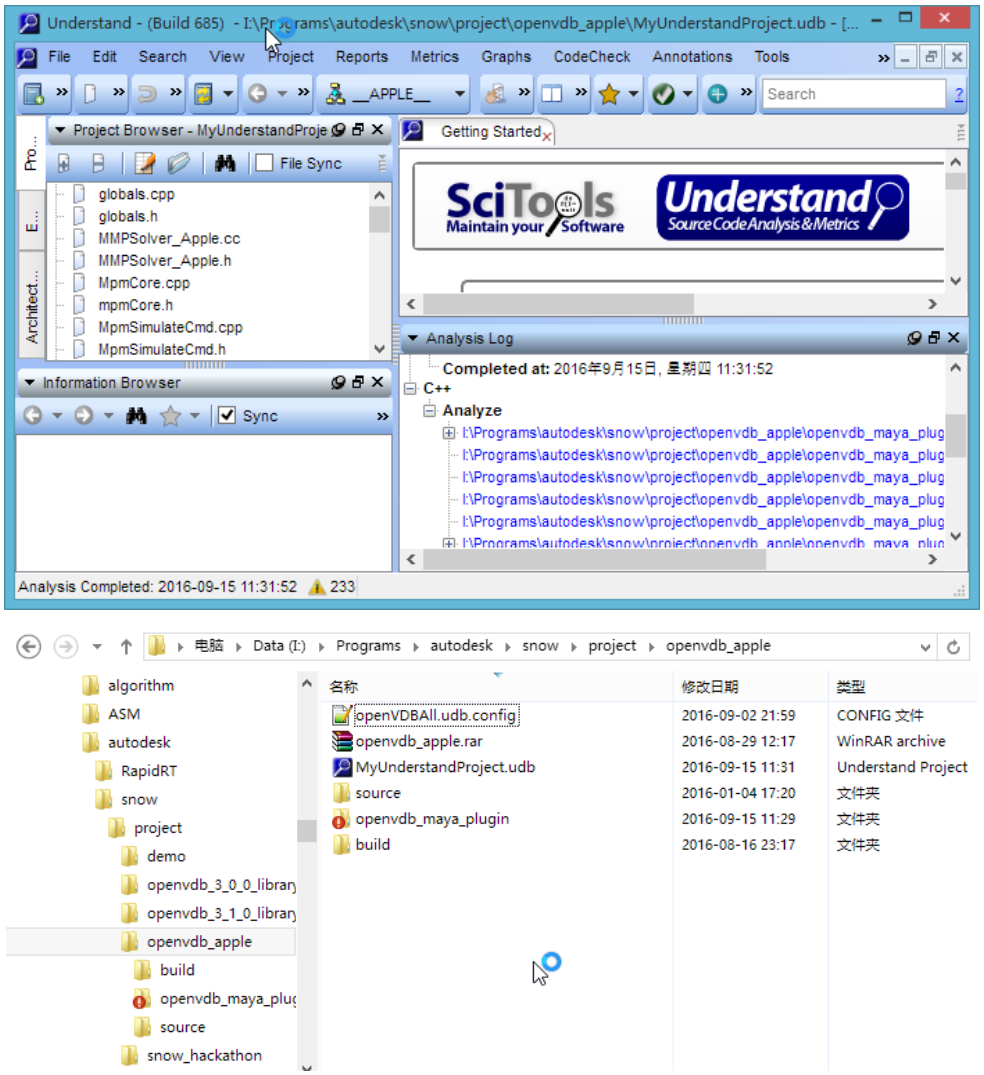
（5）Specify the project folder.



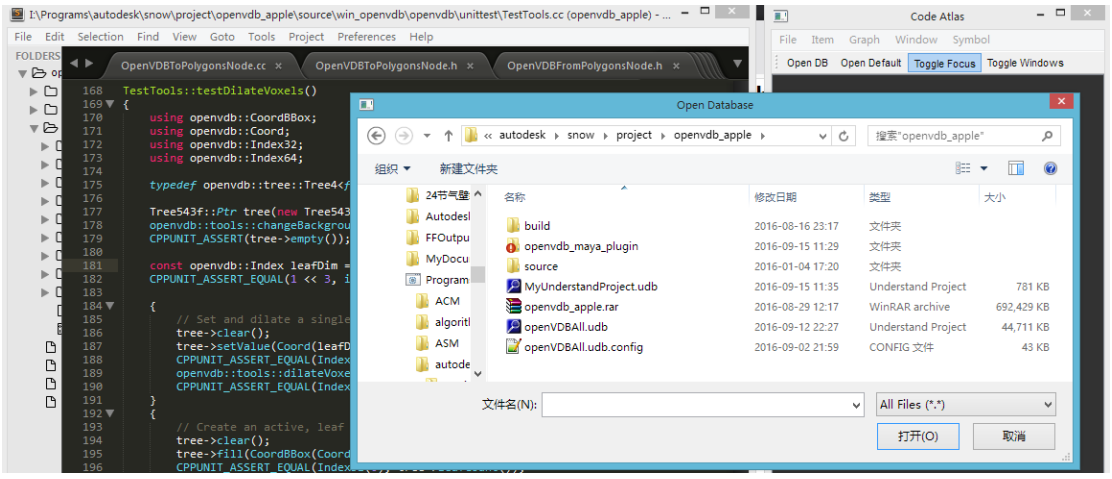
（6）Finish and begin code analysis.



（7）After code analysis, you can see "Analysis Complete" at the left bottom of the Understand window. Also, you'll find a .udb file in the folder specified in the first step, that's the database we want. Close Understand.

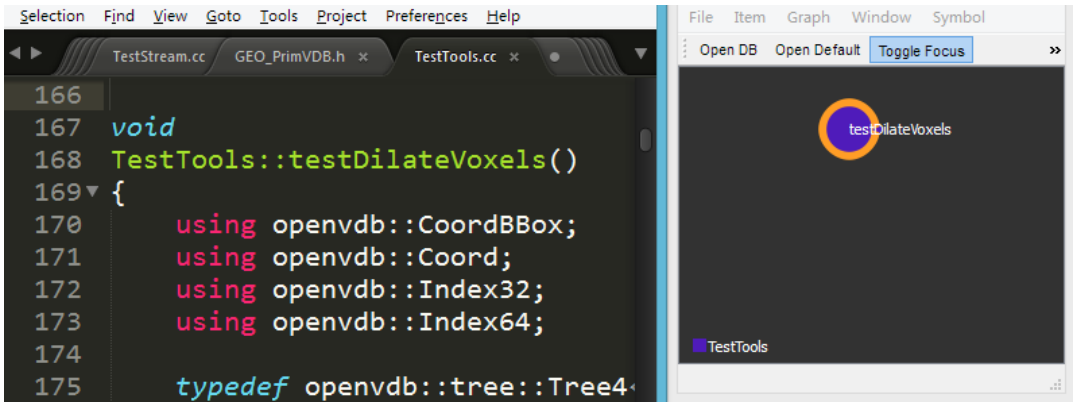


（8）Now the database is generated. Open Sublime Text, press "Start Atlas" in the code editing area. Then in Code Atlas's window, press "Open DB" button and choose the .udb file we generated before.



（9）The database is opened. In order to look into the callees of a function, one can move the

cursor in Sublime Text to the function we're interested in. Then press Alt + G and a purple disk will be shown on the plugin's viewport. This disk represents our function, and the orange edge around it represents it is selected. At the bottom left corner, there is a purple square labeled as "TestTools", because the function belongs to a class called "TestTools".

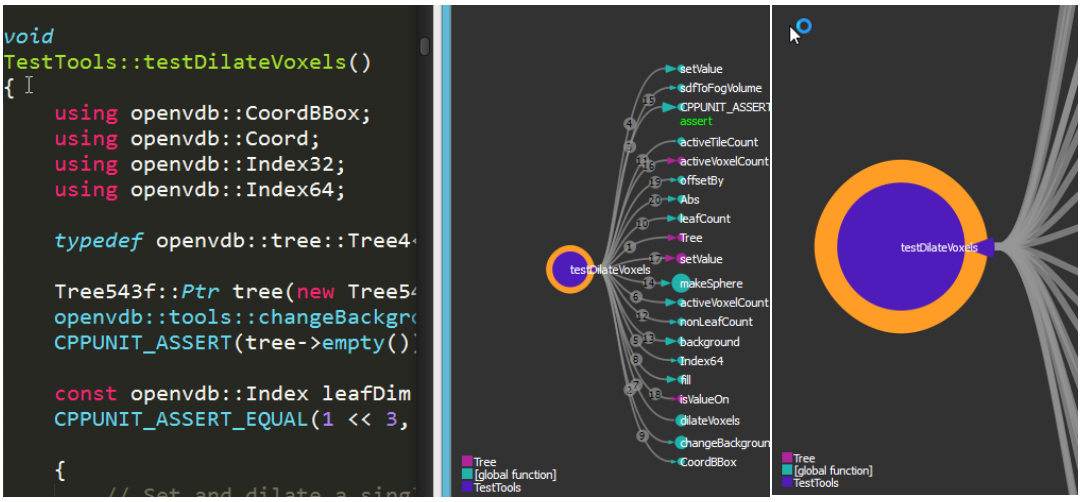


（10）If you want to see callees of the function, you can press Alt+V in Sublime Text and the callees will be shown in the viewport.

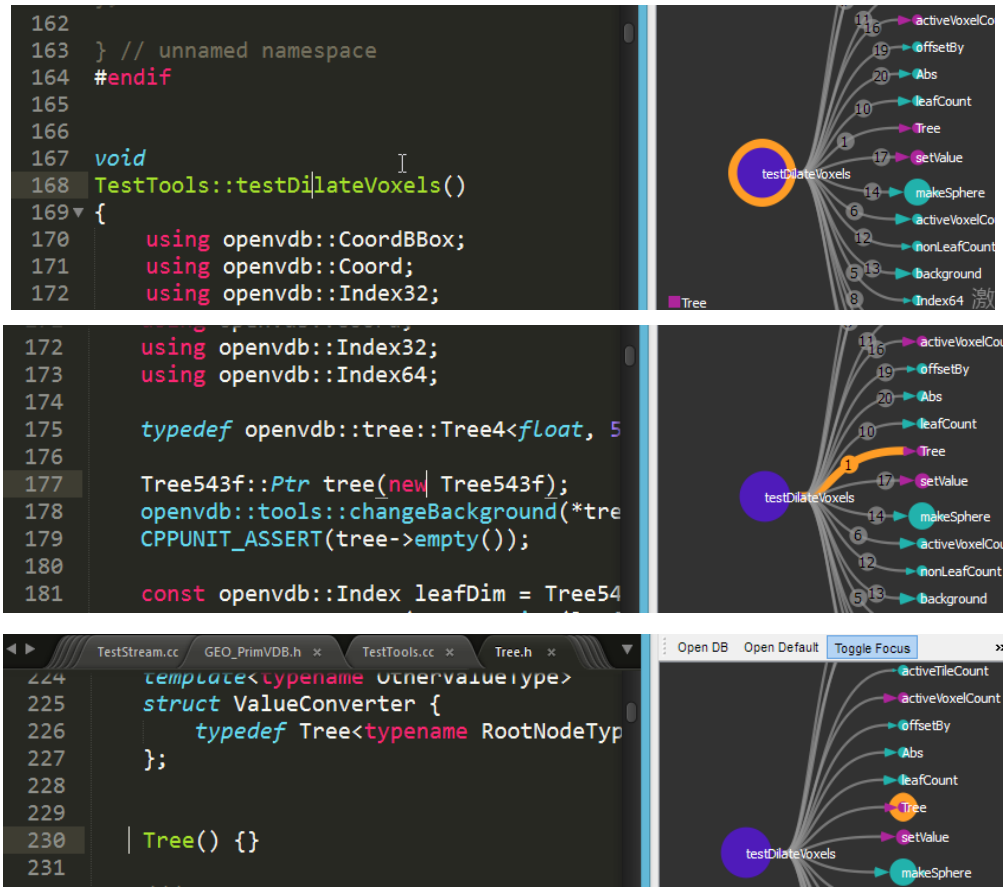
Note that the disks representing functions have different sizes. The bigger size a disk has, the more lines of code a functions has. For example, the function "makeSphere()" has more lines of code than "Abs()". Usually, a function with more lines of code contains some important logics or algorithms, which deserves further check.

For every function, there is a triangle on the left of the disk. The size of triangle implies the number of callers of that function. For some utility classes, such as std::vector, or Matrix class in some maths library, will be used in the whole program and result in they have a lot of callers. We can identify these functions by the size of triangles and can ignore them in most case, because they're usually trivial and boring.

Similar to those on the left, every function has a triangle on the right of the disk. The size of that triangle implies the number of callees of that functions. Usually, the more callees, the more possible a function has important logics or algorithms, which deserves further check.

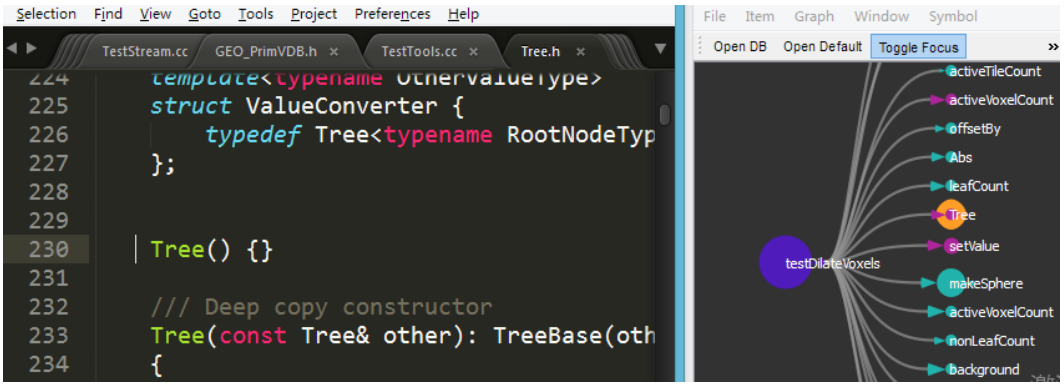


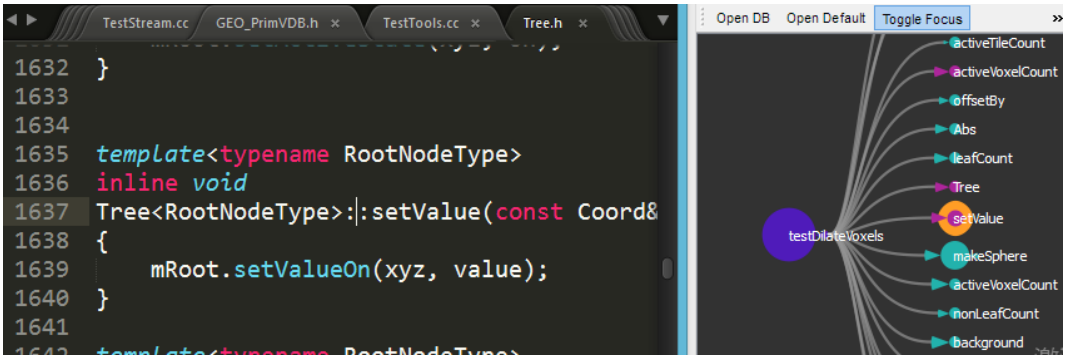
（11）How to check the code of the callees? Just press Alt + Right Arrow in Sublime Text, then the edge will be selected in viewport and Sublime Text will show the code of corresponding function call. Press Alt + Right again, the callee will be selected and the its code will be shown in Sublime Text. All these steps are shown in the figures below.



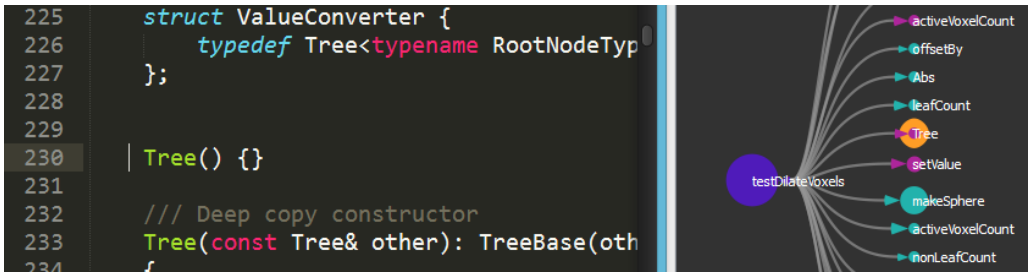
（12）Now we have jumped to the first callee, what about other callees? Just press Alt + Down Arrow in Sublime Text, then we can jump to the one below current selected function in viewport. Alt + Up, Alt + Left and Alt + Right are similar. In a word, we can use Alt+Up/Down/Left/Right

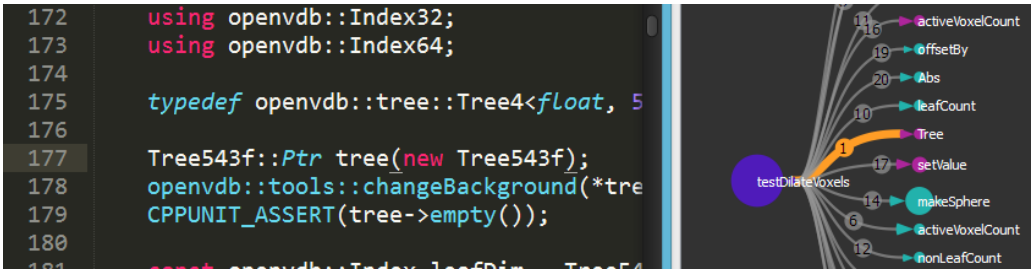
Arrow to jump to nearby function or function call edge.

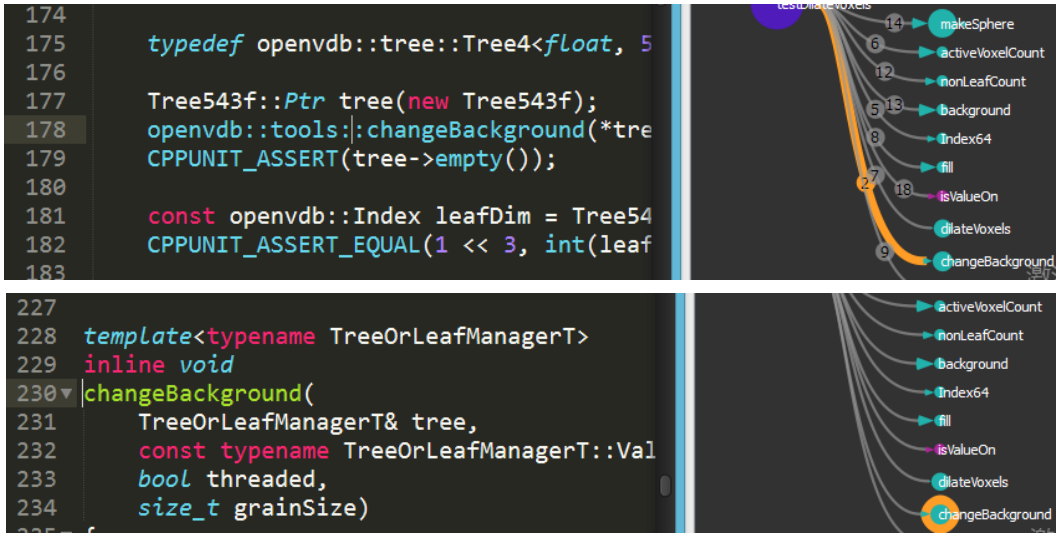




（13）But if I want to see the callees according to the calling order, what should I do? That's simple, too. Just select a call edge by pressing Alt + Up/Down/Left/Right Arrow, then by pressing Alt + Up or Down Arrow, the last or next function call edge (instead of the nearby edge)will be selected. Figures below show that. Note that the numbers on edges indicate call order of functions.

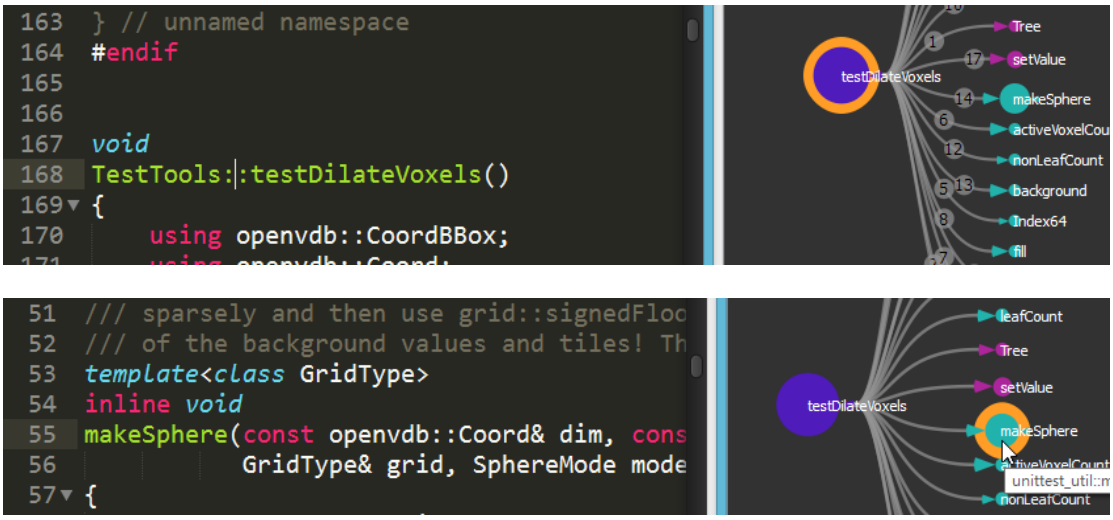






（14）All operations above are completed by using keyboard short-cut. Compared with using mouse, using keyboard short-cut is recommended for Code Atlas. Why? Because when you are reading code or writing program, it will cost much more time by switching your hand between keyboard and mouse frequently. Some code editors in Linux, and some new editors in these years, such as Sublime Text, allow user to do things just using keyboard.

However, if you are a newbie and no familiar with those key short-cuts, you may prefer using mouse. Code Atlas also allows you to do that. Double-clicking any function in viewport, then Sublime Text will show the corresponding code (See the figures below). Double-clicking any edge in viewport, Sublime Text will jump to the line of function call. Also, if the mouse cursor is not in viewport, the selected function or edge will be centered automatically.



（15）If you close Code Atlas and call graphs you dig out are lost, that is very depressing. So when you close Code Atlas, the state of viewport will be saved to a file in the same folder as the

udb file automatically. The file has the same name as the udb file, appending a .config extension. Above are the basic operations of Code Atlas. Besides these operations, users can also see class hierarchy, search invocation path between a function to another, save call graph, write comments on functions, etc. All of these will be introduced in next chapter.

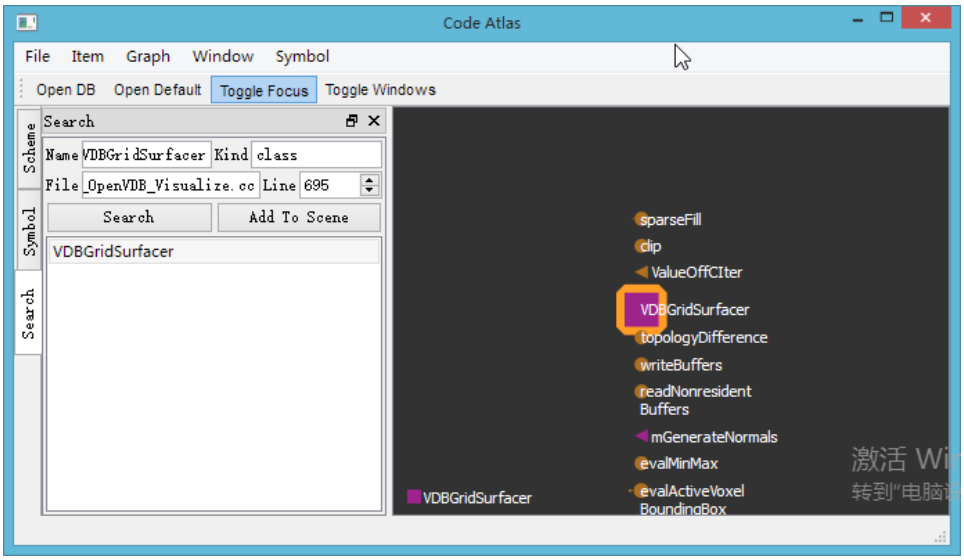
Feature List

In this chapter we will introduce all features of Code Atlas. Before reading this chapter, it is recommended to finish last chapter first and understand the basic operations of the Code Atlas.

Show Functions/Variables/Classes

> *In Sublime Text, place cursor on the function/variable/class you want to see, then press Alt + G*

Besides pressing key short-cut in Sublime Text, you can also search directly in the Code Atlas's UI. Just Press "Toggle Windows" and choose "Search" tab, then you will see the search window. Input name, kind(including class/variable/function), file path(optional) and line number(optional) and press "Search", then the results will be shown in the list below. Choose one and press "Add To Scene", then the class/variable/function will be added to viewport. When a function has many overloaded versions, or is a virtual function, the analysis result of Understand may be inaccurate, and the system can't find the right function when you press Alt + G. In these cases, you can use Search window to find the correct function.



Find Function Calls

> *1. In Sublime Text or viewport, press Alt+C, callers of current selected function will*

be shown

2. Press Alt + V, callees will be shown

In order to avoid more and more nodes shown in viewport, making it become slow and user get lost, the viewport will delete the oldest node automatically. However, if a function has too many callees or callers, press Alt + C/V will make the original function deleted. So a user should judge the number of callees/callers first, by checking the size of triangles in a function's icon.

Find Class Members

1. In Sublime Text, place cursor on the name of a class, press Alt + G, and the class will be shown in viewport

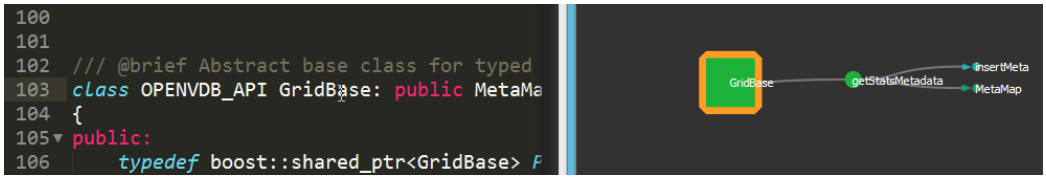
2. Press Alt + M, members of the class will be shown

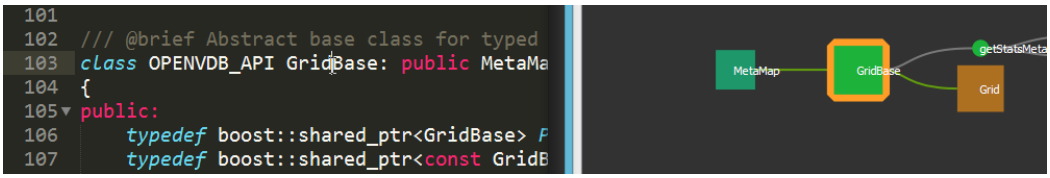
In the figures above, a class is represented as a square, and a variable is represented as a triangle. Similar to functions, the size of a square represents the number of a class's member. The size of variables' triangles are all the same.

Find Class Hierarchy

Select a class in viewport, press Alt + B, its base class(es) and derived class(es) will be shown

Just like the figures below. The base class is on the left and the derived class is on the right.

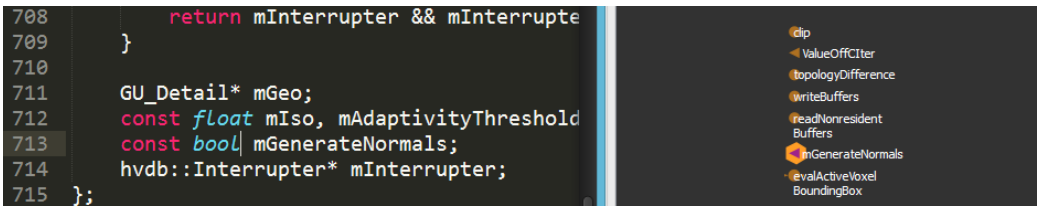


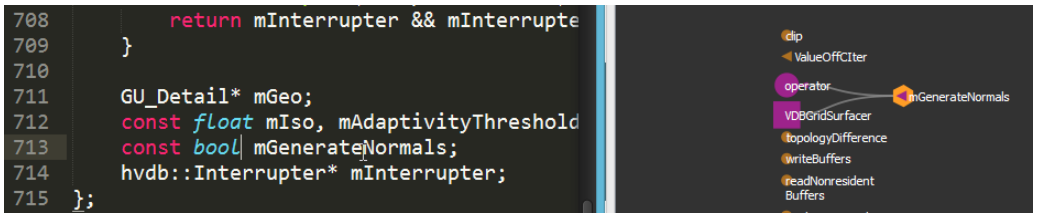


Find Variable's References

> *Select a variable, press Alt + U, the class defining it and the functions using it will be shown.*

Just like the figures below.





Jump to Other Function/Variable/ Class

> *In Sublime Text, press Alt + Up/Down/Left/Right Arrow*

Readers can refer "Quick Start" for more information.

Delete Nodes or Edges

Sometimes you are only interested in a part of classes, functions and variables, and you can delete others.

> *1. Select the node or edge you want to delete*

2. Press Alt + Delete in Sublime Text, or press Delete in viewport, it will be deleted

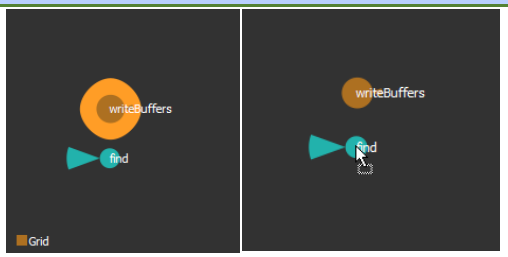
After we delete a node or edge, a nearby node or edge will be chosen automatically, so you can continue to use keyboard short-cuts to navigate.

Find Call Path

Sometimes you will want to find if there exists call path between one function to another.

1. Show function A and B in viewport

2. Drag A to B, and call path(s) from A to B will be shown



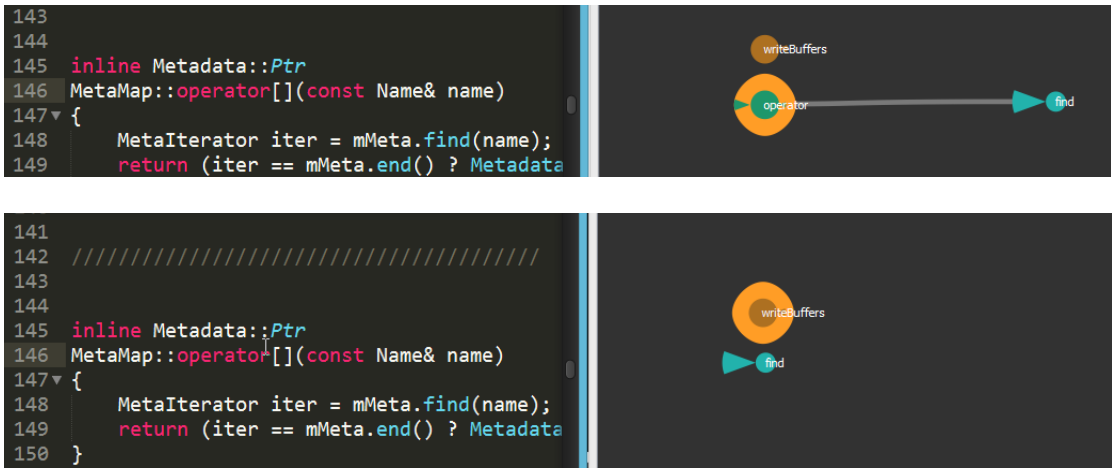
Using this feature, one can find main call hierarchy of a module quickly. Just find two functions, usually functions doing specific task first, and check if there exists call paths. If call paths exist, then check every node on the path and find more important functions.

Add Functions to Black List

Some functions, like std::vector::size(), are invoked everywhere in the whole program. Since everyone knows what it does, there is no need to show it in call graph. For these functions, you

can add them to black list and it will no longer be shown.

> *1. Select a function*
>
> *2. Press Alt + I, the function will be added to black list*

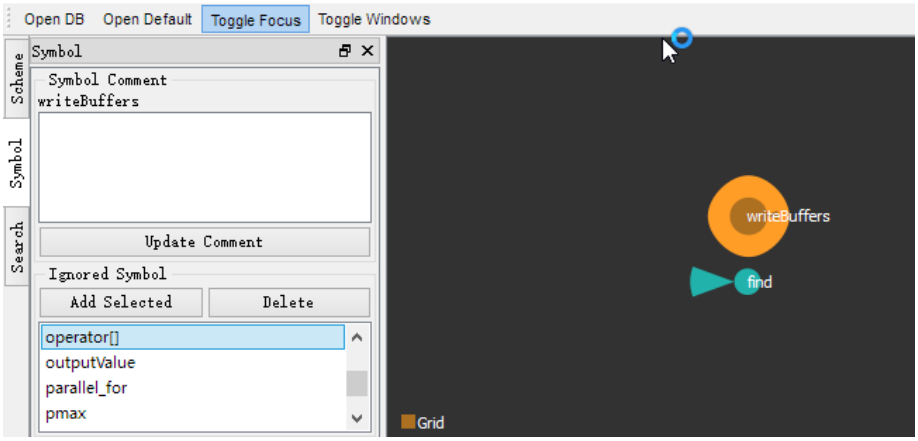


The function added to black list will be deleted immediately. Next time when we press Alt + V on its caller, or Alt + C on its callee, it will never be shown.

So how to remove a function from black list? Just do the following:

> *1. Press Toggle Windows and click "Symbol" tab, the black list is shown in "Ignored Symbol" box.*
>
> *2. Select a function in the black list and press Delete, it will be removed and can be shown in viewport again.*

You can see the operator[]() function we added to black list in the figure below.

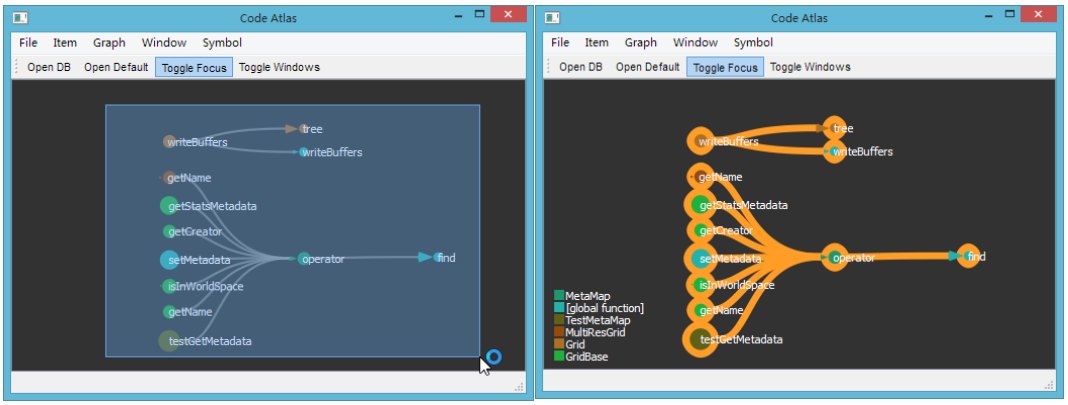


Selection

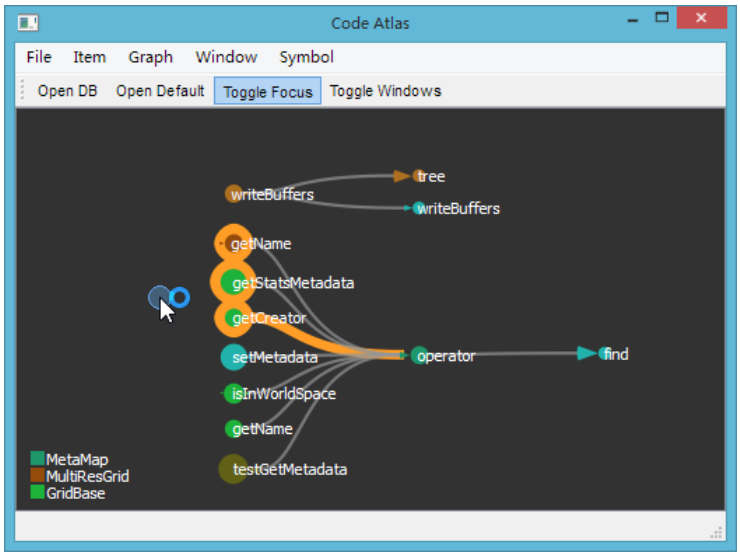
There are two ways to select nodes and edges using mouse: frame selection and brush

selection.

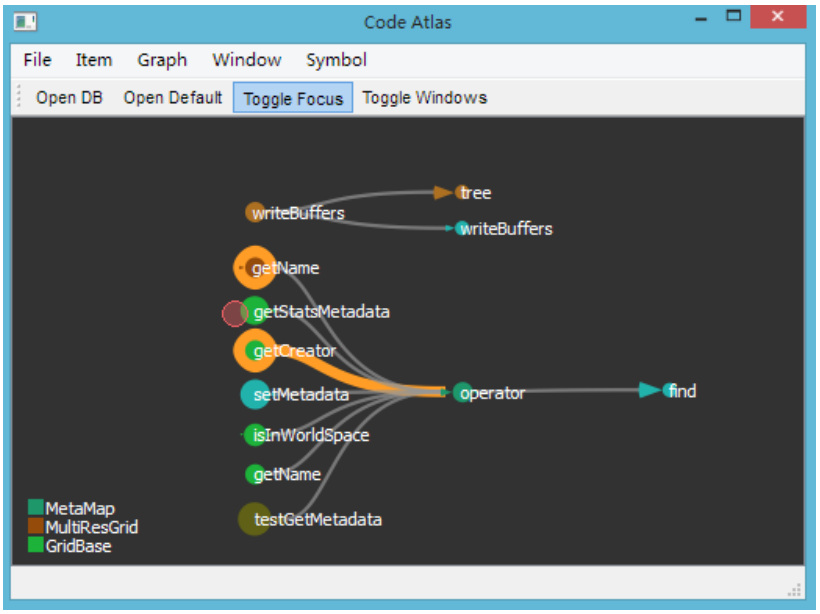
Frame selection is simple. Press blank area of viewport will clear selection.



Sometimes using frame selection may select something we don't want. In this case we can use brush selection. Press Ctrl key, then the mouse cursor in viewport will become a blue circle. Then press left mouse button and drag the mouse over nodes or edges. The nodes or edges under cursor will be selected.



But if I want to deselect something? Just press Shift key and drag the mouse over nodes or edges, and they will be deselected.

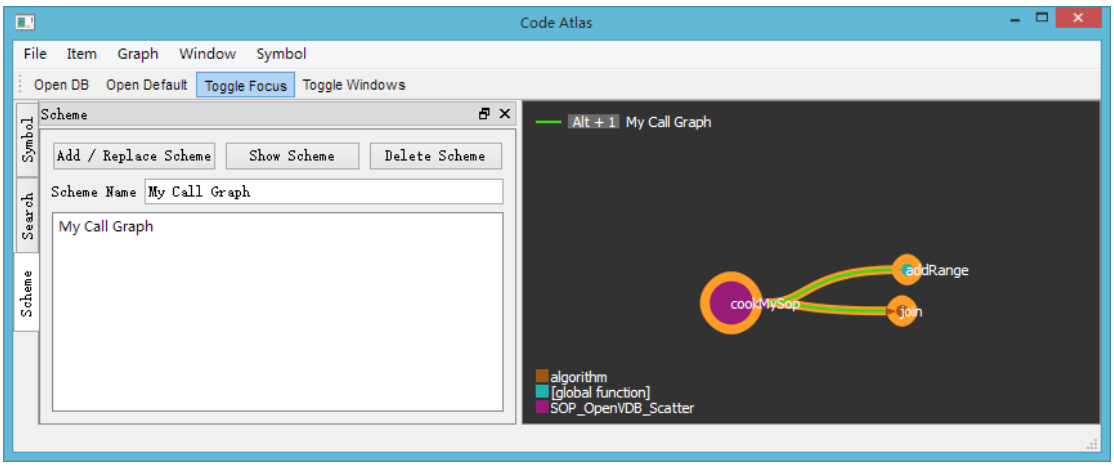


Save Call Graph

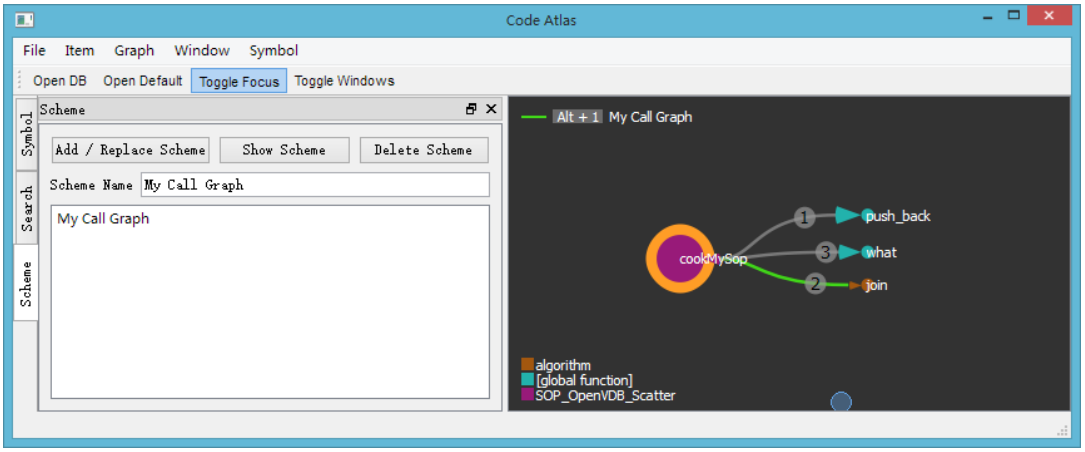
Sometimes we want to save the call graph and review it next time. Here are the steps:

> *1. Select nodes and edges to save*
>
> *2. Press Toggle Windows and go to the "Scheme" tab*
>
> *3. Input the name of call graph in "Scheme Name" box, press Add/Replace Scheme*

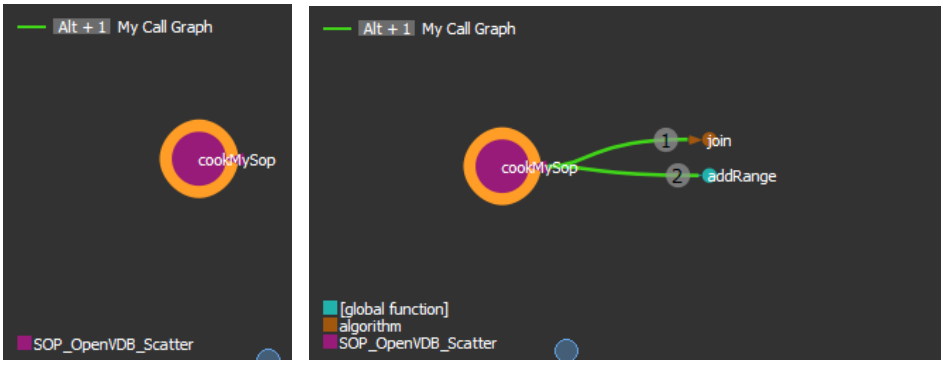
After that, we can see the call graph name added to the list below, just like the figure below.



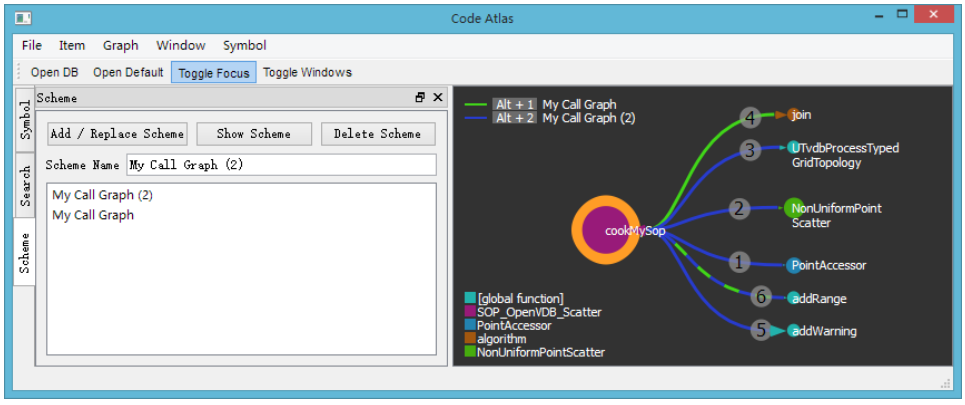
Another changes is, when we select node or edge of the previous call graph, all edges belong to that call graph will become colorful.



Also, a legend will appear at the top left corner of viewport. In the figure above, it reads "Alt + 1 My Call Graph". That means if we press "Alt + 1" in Sublime Text, the whole call graph will be shown. Just like the figures below:



If a function belongs to multiple call graphs, there will be several legends. If an edge belongs to multiple call graphs, it will be shown using dash line with different colors, with each color corresponding a call graph. Press Alt + number key according to the legend, the corresponding call graph will be shown.



There is "Show Scheme" and "Delete Scheme" button on the "Scheme" tab. You can use them to show or delete the call graph selected in the list.

Making good use of this feature will improve the efficiency of reading code. For example, we can

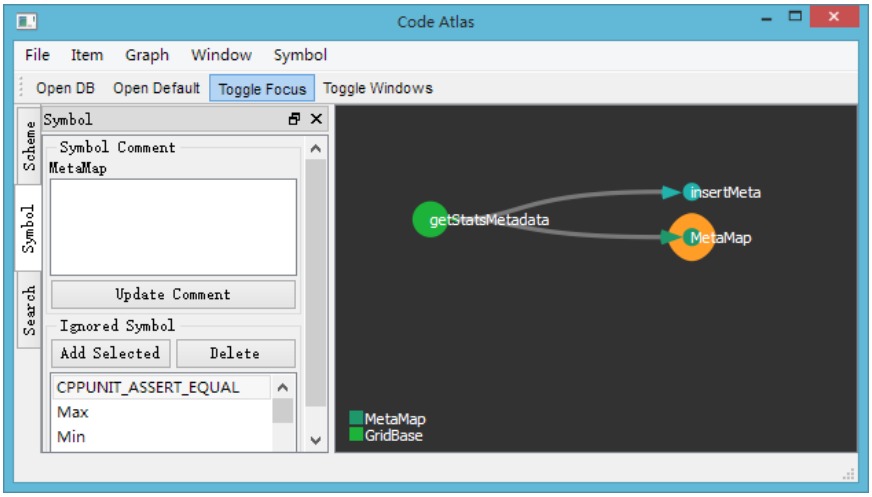
save the call graph of some important functions. Next time when we continue to work on these functions, we can use Alt + number key to get back the call graph, and use it to navigate.

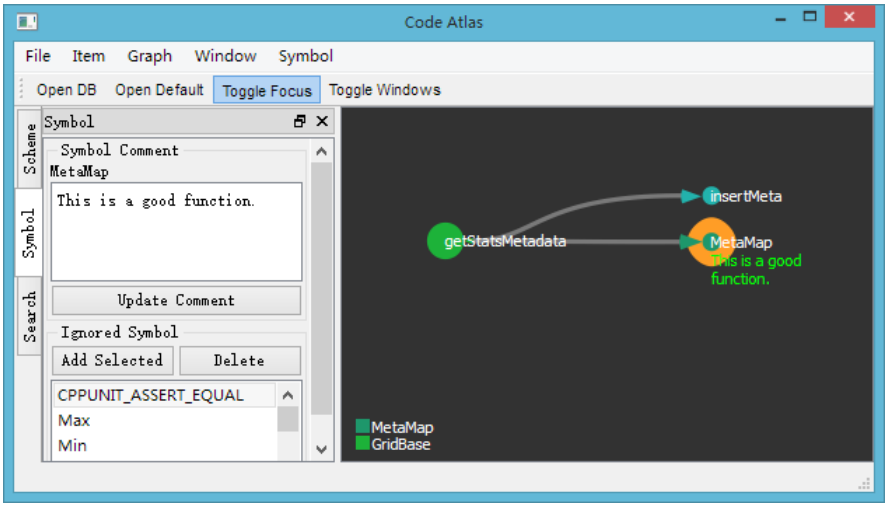
Add comments to Nodes

You can write down notes for important functons/variables/classes by adding comments to them.

> *1. Select the function/Variable/Class you want to comment*
>
> *2. Press "Toggle Windows" and go to "Symbol" Tab*
>
> *3. Input your comments in the "Symbol Comment" box and press "Update Comment"*

You can find several lines of green text appearing below the name of the function/variable/class. That's the comment you input.



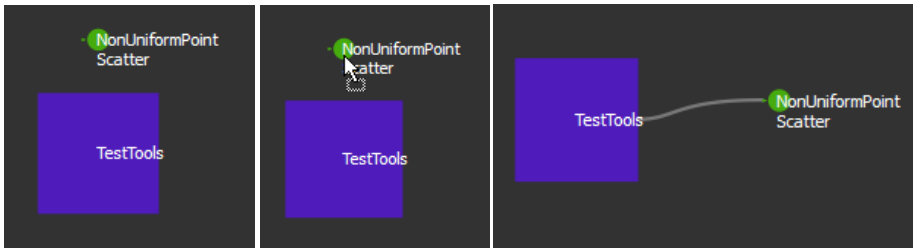


Add Edge Manually

Sometimes Understand's code analysis result will be inaccurate. For example, a virtual function may have many overloaded version. In that case, you can specify call edge manually. Actually, the meaning the manually added edges are not restricted, and you can use it to specify some custom relationship.

> *1. Show function/variable/class A and B in viewport*
>
> *2. Press right mouse button and drag A to B, then an edge from A to B is added*

Just like the figures below.



Now all features are introduced.