Subject: Re: Very first impressions and.... [FEATURE REQUESTS]
Posted by mdelfede on Wed, 12 Sep 2007 22:15:52 GMT
View Forum Message <> Reply to Message

luzr wrote on Wed, 12 September 2007 21:52
How do you want to keep it in sync?

Note that rescanning after #define is typed is not enough.

Any change in file changes potential preprocessor output. This is not even related to macros, even if there are no macros, any change in file changes preprocessor output.

I don't understand your point, I think. You're saying that you scan the file on startup (or get from last saved cache), then only on demand. Let's start with an example :

```
#include <stdio.h>

int funca(int z)
{
  return 2*z;
}

int main(int argc, char *argv[])
{
  int x = 5;
  int y;

  y = funca(x);

}
```

the file here is saved.
When you reload, you add:

```
#include <stdio.h>

int funca(int z)
{
  return 2*z;
}

int funcb(int k)
{
  return k*k;
}

int main(int argc, char *argv[])
{
  int x = 5;
  int y;

  y = funca(x);
  x = funcb(  <-- here you have parameter suggestion

}
```

In this case, I don't know if you're parsing all the file again or only the added code... But let imagine that your program is very smart and chooses the hard and faster way. So you find the code difference from the previous parse, and add funcb() definition. Doing so makes things hard when you delete a function or rename it, but can be made, I think...
Let's change things, and add a #define and a #include ;

```
#include <stdio.h>

int funca(int z)
{
  return 2*z;
}

#define macroc(x, y) (x) + (y)

#include "mystuff.h"

int funcb(int k)
{
  return k*k;
}

int main(int argc, char *argv[])
{
  int x = 5;
  int y;

  y = funca(x);
  x = funcb(  <-- here you have parameter suggestion

}
```

Here you have 2 ways :
get a partial preprocessor and feed it with the #define and #include, but.... it's not enough, you miss the #include at the beginning. So, you MUST feed preprocessor with this also.
But it's not eno''ugh yet, if for example your "mystuff.h" test for 'funca' definition it fails, because you didn't feed the preprocessor with 'funca' definition.

To make it simple, you MUST feed the preprocessor with ALL code from the beginning of file, to be sure to catch all cases. No choice. So, it makes no sense to have a partial preprocessor; you need only a preprocessor that accept a truncated file as input. CPP does the job. Whathever you do, you must process the file from the beginning when you add # stuffs.

Quote:
All project files are scanned at ide startup(*). Then modified files are scanned before they are leaved or if the information is needed.

(*) in fact, all information is cached, so in reality only changed files are scanned at startup, unchanged file information is just loaded.

so, let's say, before scanning and caching the file, you add the preprocess pass.
On the former example (before macro stuff), you have the funca defined and main code. You add the funcb, then start type in main() the x = funcb(...) and here you must call the scanner, I think, or you'll miss funcb() parameter suggestion.
Simple way : ignore it, up to the next file reload.
Slow way, rescan all file from the beginning on the fly.
Smart way, scan only the diff.

Wathever you're doing, let's add the preprocess stuff now :

Simple way : ignore ecc ecc
Slow way, repreprocess and rescan from the beginning
Smart way : too complicated
Smartest way : Take simple way and do the slow way in a background thread. When the latter is done, swap the parsed threes!
The point is : the user don't notice a delay, he can continue typing AND have the completion with THE OLD stuff; when the new scan is done, the system switch to new stuff. As normally a programmer types (and think) much more slowly than a computer, he will feel like you made a Slow scan but with the speed of simple way.

Quote:
Any change in file is related to preprocessor.

That is the problem you seem to miss (or maybe I do not understand your method).

Maybe we're speaking about the same thing

Quote:
Note:

What you suggest might partially work, if preprocessor would be capable of exporting all defined macros together with preprocessor output. In that case, you could split the processing to processing of headers, which would yield two files (preprossed files and set of macros) and the main file, then you could e.g. include only macros for processing of main file, which would be much faster. But hey, this is what I call "partial preprocessing".

Not that there is another problem: You are not only interested in information contained in the .cpp file, but also in things defined in headers...

No, here you're right. If you want a full code completion, you MUST preprocess all stuffs, and that from begnning. It's of no use feed the preprocessor only with macros; some includes depends on previous macros, and viceversa.
The only way I see, wherever you use cpp or your preprocessor, is to smart preprocess/scan all stuff in background and keep a shadow copy of previous job while the new one is done.

I really don't see it too hard, the most is done in current parser... but maybe I'm wrong somewhere...

Ciao

Max