Instructions for server setup

1. Install django. Visit official installation guide for detailed instructions .

2. Go to **Extractor** directory in **source** folder .

3. - Open the project named **assembly** in Mono. Develop and run the project. Output will be generated in the file in same directory.

   OR

   - Use **Program.cs** file to get function list in json format in csharp. The output file will be generated in the directory of your project.

   OR

   - Use final_csharp.json . It is the json file for csharp function signatures.

   > ***NOTE:***
   > *It is advisible to use Mono - develop for easy and reliable use of the program.*
   > *Make sure you use proper class names as per your project name in csharp IDE you are using. The program uses **assembly** as project name*
   > *Make sure to add external dll file Newtonsoft.Json.dll in your project.*
   > *For mono-develop use the following instructions :*
   > *- In your project there should be a folder called References. Right click it and choose edit references.*
   > *- Then check the box next to the package you want to add.*

4. - Use **Reflect.java** program to get function list in json format for java.

   > ***NOTE:***
   > *Make sure you use proper class names and package names in java IDE you are using. The program is using project name **Refletion** and package name **Reflection** and class name **reflect**.*
   > *Configure the output path in your respective IDE you are using for java .*
   > *Add the two jar files **json-simple-1.1.1.jar** and **bcel-5.2.jar** to your project.*
   > *It is advisable to use eclipse for easy use.*

Following are instructions for eclipse IDE :

1. create a java project by name **reflection**
2. rename the present package as **reflection**
3. add a class by right clicking on your project name and name it **reflect**
4. set your output path by using the following path in project
   run -> run configurations -> Common
5. add external jars use : project -> properties -> Java Build Path -> Libraries -> Add external JARs
6. use program **Reflect.java** to get the desired output .

OR

- use output.txt file in Extractor directory in source . It contains the json data for java as we require

5. For generating data for C, use **extract.py** in **Extractor** directory. Simply run `python extract.py` , output will be generated in `Json_data/c_cpp_libraries` in json format.

6. For seeding database with previously generated function signatures, go to django project named **final_database** and run command `python manage.py seedatabase <language_name>` . For C, C#, Java use `c, csharp, java` respectively.

Detailed instructions

1. This django custom command i.e *seeddatabase* , takes json files from directory `Extractor/Json_data` .
2. The command reads all Json files one by one and add them to database.
3. For C, json files must be present in directory `Extractor/Json_data/c_cpp_libraries` and the file name must be same as the library they belong. Also add newly added library name to file `lib_name` which present in the directory `Extractor` .
4. For C# and Java, any new function signature must be added to corresponding json file present in directory `Extractor/Json_data` . `csharp_data` is json file for csharp functions and `java_data` is json file for java functions.
5. **Json object format for function**:

- C:

```
{
    "method_name" : "function name",
    "return_value" : "return type of function",
    "arguments" : "arguments separated by commas"
}
```

- C# :

```
{"method_name":"function name","return_type":"return type","parameters":["arguments seperated by commans"],"class_name":"parent class of function"}
```

- JAVA:

```
{ "Return type":"return type","Arguments_types":["arguements of functions, seperated by commas"],"Method name":"function name","Class Name":"parent class of function"}
```