|  |  |
| --- | --- |
| Personal Details |  |
| Name | SARTHAK JAIN |
| Email | successar@gmail.com |
| University | DELHI TECHNOLOGICAL UNIVERSITY |
| Course Pursued | BTECH (COMPUTER ENGINEERING) |
| Country of Residence | INDIA |
| Phone | +91 7838691965 |
| Date of Graduation | 01/07/2017 |
| Github Link | https://github.com/successar |

|  |
| --- |
| **PROJECT PROPOSAL** |
| **Title :** Documentation Generator for ECL Code  **Deliverables :**   1. Modified Eclcc for outputting meta information for documentation. 2. Ecldoc utility for generating actual documentation for the project 3. A plugin to generate docs directly throught ECLIDE 4. Ecldoc utility will contain 3 major parts:    1. Glob Pattern Parser – It takes in a glob pattern to specify which files we need docs for and generate relevant Meta information in XML format using eclcc.    2. Linker – It generates link information (IMPORT, Inheritance in Modules, etc) between multiple doc files using the meta information in each.    3. Output Generator – It convert XML format into output format (Text/HTML/reST/others) with appropriate directory structure. Link information is encoded (for example using hyperlinks in HTML format) using the output of Documentation Linker.   **Wishlist :**   1. ECLDoclet API similar to Doclet API in Javadoc for custom output format. 2. ECLTaglet API similar to Taglet API in Javadoc for custom tags. 3. Links to a watch utility so that ecldoc utility only processes files or parts of file that are changed. |
| **DETAILS OF ECLDOC UTILITY:**  (In addition to those in specifications README.rst file provide with project description)  Note: I am processing glob patterns in new ecldoc rather than eclcc because the current version of eclcc focuses on compilation of single ecl file and its dependencies as required. Glob patterns are needed for documentation and not compilation and therefore, I do not believe it will be reasonable design decision to club 2 different functionalities in single eclcc utility. We will modify eclcc to egenrate appropriate dependency information for example using –Md option, but the processing of dependencies will take place in ecldoc. This might increase the overall runtime for documentation generator, but it will reduce the complexity of eclcc which is needed for fast compilation.  Example: Consider following project structures.  A E  |--- B |--- F.ecl  | ---- B1.ecl |--- G  | ---- B2.ecl |--- G1.ecl  |--- C.ecl  Path to A : /ecl/A  Path to E : /mnt/E  To run ecldoc on A,   1. We specify docs output folder using –o option. [-o /ecldocs/A-Doc] 2. Run : ecldoc [-o /ecldocs/A-Doc] ***/ecl/A/<\*>***   This generates a directory /ecldocs/A-Doc with structure.  A-Doc  |-------- pkg.xml – contains information about the directory structure of this project.  |-------- A  |--- A.index.xml – each\*.index.xml file will contain info about that directory  |--- B  |--- B.index.xml  |--- B1.xml  |--- B2.xml  |--- C.xml  **pkg.xml** :  <DIR name=”A”>  <DIR name=”B”>  <FILE name=”B1.xml”> A.B.B1 </FILE>  <FILE name=”B2.xml”> A.B.B2 </FILE>  </DIR>  <FILE name=”C.xml”> A.C </FILE>  </DIR>  A similar structure can be generated for E --- ecldoc [-o /ecldocs/E-Doc] /mnt/E/<\*>  E-Doc  |-------- pkg.xml  |-------- E  |--- E.index.xml  |--- F.xml  |--- G  |--- G.index.xml  |--- G1.xml    Format : ecldoc [options] ***Source-Path***  [Options]:  -f : output format. Potential values: HTML, txt, reST  -l : link-paths. Provides paths to external docs directory that you want to link.  -o : Output Folder where doc will reside  -a : Provide all the above options as an argument file (See argfiles in Javadoc).  ***Source-Path***: Specifies paths to all the files/directory whose documentation is needed. Any regex can be present in source path using <regex> format.  E.g. - ***/ecl/A/<\*>***  If we need to recur, we can specify the wildcard path like ***/ecl/A/<\*>***  If we do not need recursion, we can specify wildcard path like ***/ecl/A/<\*>.ecl***  Examples :   1. Consider we want to generate documentation for all files in A recursively.   No external dependencies ( no IMPORT E; or IMPORT E.G; )  **Run :** ecldoc [-f html] [-o /ecldocs/A-Doc] ***/ecl/A/<\*>***  **Output :** Generate /ecldocs/A-Doc   1. Consider we want to generate documentation for all files in E recursively.   No external dependencies (no IMPORT A; or IMPORT A.B; )  **Run :** ecldoc [-f html] [-o /ecldocs/E-Doc] ***/mnt/E/<\*>***  **Output :** Generate /ecldocs/E-Doc   1. Consider we want to generate documentation for all files in A recursively.   External dependencies ( IMPORT E.G; in A/B/B1.ecl )  **Run :** ecldoc [-f html] [–lp /ecldocs/E-Doc] [-o /ecldocs/A-Doc] ***/ecl/A/<\*>***  **Output :**   * 1. Generate /ecldocs/A-Doc   2. IMPORT E.G; => XML file in A-Doc/A/B/B1.xml is appropriately linked to file in /ecldocs/E-Doc/E/G/G.index.xml   3. IMPORT E.G.G1; => link will be to /ecldocs/E-Doc/E/G/G1.xml  1. Consider we want to generate documentation for all files in B only.   No external dependencies  **Run :** ecldoc [–f html] [-o /ecldocs/B-Doc] ***/home/usr/eclp/A/B/<\*>.ecl***  **Output :** Generate /ecldocs/B-Docs |
| Real Example:  Consider ecl-ml library which contains ML and PBblas.  Many files in ML refer to PBblas as IMPORT PBlas;  So if we run ecldoc ***/ecl-ml*** – All links between ML and PBblas are done automatically.  However if we want Docs for ML and PBblas separately, we **run :**   1. ecldoc [-o /ecldocs/PBblas-Doc] ***/ecl-ml/Pbblas*** 2. ecldoc [–lp /ecldocs/PBblas-Doc] ***/ML*** |

|  |  |
| --- | --- |
| Timeline Details | Week No |
| **Processing of Single File:**   1. Modifying Meta generation system for generating documentation of single file using eclcc. 2. Generation of Parser rules for Docstring processing. 3. Convert Meta output into appropriate xml representation for documentation. 4. Processing of dependencies from Meta output if they match glob pattern for documented files. | 1-2 |
| **Generation of Linkage Information:**   1. Placeholders to provide appropriate linkage info for IMPORTs in XML files or @see tags.   E.g.  **ECL :** IMPORT E.G.G1;  **XML :**  <IMPORT>  <A href=”%DOCPATH%/E/G/G1.xml”> E.G.G1 </A>  </IMPORT> | 2 |
| **Directory Processing:**  Until now, we have only dealt with single file processing. Now we move onto complete project processing to generate XML representation.   1. Python Code – parse [options] given in command line. 2. Check to see if all –lp paths are valid and contain pkg.xml file. 3. Process any wildcards in ***<source-path>*** - filename portion.   To process starting point of wildcard path, get the largest prefix with no regex.  Generate backbone of output directory structure , similar to A-Doc | 3 |
| **Walking the Code Directory (1): Generate pkg.xml**   1. Walk the Directory, starting from largest prefix w/o regex. 2. Match the regex in source path with file/directory names. 3. If file name match, append <FILE name> </FILE> to pkg.xml. 4. If directory name match, create corresponding subdirectory using <DIR name> … </DIR> and recur within … . | 3 |
| **Walking the Code Directory (1): Generate actual doc files**   1. Walk the Directory, starting from largest prefix w/o regex. 2. Match the regex in source path with file/directory names. 3. If file name match, generate a corresponding .xml file in output directory 4. If directory name match, create the corresponding subdirectory within output directory and recur within the subdirectory. | 4 |
| **Integrating External Link Information:**  **External Links are 2 types:**  **Run : ecldoc [options] </A/\*>**   1. **External to file but internal to current run**   **E.g. In A/B/B1.ecl**  **IMPORT A.C;**   1. **External to file and external to current run**   **E.g. In A/B/B1.ecl**  **IMPORT E.F;**   1. Process pkg.xml files from link-paths to get info about Type (b). 2. Process pkg.xml from current run to get info about Type(a) 3. While generating XML files from ECL, check if corresponding link placeholder matches some link in pkg.xml files (See Example below.) 4. If yes, create logical link URL to the linked file. 5. If no, keep the placeholder for future link additions.   E.g. Consider ECL Code for A.B.B1.ecl file in A project as before.  **ECL Code:**  IMPORT E.G.G1;  RUN: ecldoc [-f][-o /home/usr/docs][-lp /mnt/ecl2docs/E-Doc] ***</home/usr/ecl/A/\*>***  **pkg.xml file in /mnt/ecl2docs/E-Doc/pkg.xml :**  <DIR name=”E”>  <DIR name=”G”>  <FILE name=”G1.xml”> E.G.G1 </FILE>  </DIR>  <FILE name=”F.xml”> E.F </FILE>  </DIR>  **XML Code in /home/usr/docs/A-Doc/XML-A/XML-B/B1.xml :**  <IMPORT>  <A href=”/ecldocs/E-Doc/E/G/G1.xml”>  E.G.G1  </A>  </IMPORT> | 4-5 |
| **HTML Output Generator :**  HTML Document will be generated according to following rules :   1. Imports in the file 2. Export Tree – A TOC for the file with nested structure (using nested lists, etc) with <docstring-type> : <Attribute-signature> format. 3. Exports Information – Here we do not use nesting but linear format.   Each Export will be of format :  -----------------------------------------------------------  <docstring-type> : <Attribute-Signature>  <Content>  -----------------------------------------------------------  This type of formatting will allow nesting structure to be represented independent of the content of the comment (which may be large), yielding a cleaner interface.  **See Attached file (example.html) to this proposal for a sample.**  Generator :   1. We use XSLT to convert XML to HTML Directly. 2. Or, we could run a second utility on top of ecldoc to generate physical HTML Documentation.   **Text Output Generator:**  Convert XML to Text directly with appropriate indentation.  This will be a trivial conversion since XML is already in a nested format.  **RestructuredText Output Generator :**  Convert XML to RestructuredText Format with appropriate link information using :ref tags in reST. | 6, 7 |
| **ECL-IDE plugin:**   1. Generate ECL-IDE plugin to run ecldoc directly. | 8 |

|  |
| --- |
| **Testing Information:** |
| During all tasks in above timeline, unit tests will be generated for each new Bold heading.  Test cases will of 3 types:   1. Tests are written specifically for the current task. 2. Running the ecldoc on ECL-Samples Folder. 3. Running the ecldoc on ECL-ML folder.   2 Types of tests evaluation will be done:   1. Tests on the correctness of single file output. 2. Tests on the correctness of directory output. |

|  |
| --- |
| **Knowledge and Skills:** |
| Languages: C++, C, Java, Python, ECL, SQL, R |
| Libraries : Regex (in C++, Java, Python), ANTLR, lex/flex/Jflex, yacc/bison/cup |
| Web Dev : HTML/CSS, Javascript, XML, XSL , XQuery, AJAX, JSON, DJango, Ruby-on-Rails, MySQL |
| Relevant Projects Done (On Github):   1. Implementation of 3 working compilers:    1. Slang: a procedural language with C-like syntax    2. COOL: an OOP Language with inheritance    3. YScript: a scripting language like Lua with concurrency, garbage collection, and VM Interpreter 2. Implementation of working CLI-OS 3. Implementation of complete website using HTML/CSS, Javascript, Django, AJAX with XML and JSON for data exchange 4. Implementation of website in Ruby-on-Rails |