|  |
| --- |
| Personal Details |
| Name |
| Email |
| University |
| Course Pursued |
| Country of Residence |
| Phone |
| Date of Graduation |

|  |
| --- |
| **PROJECT PROPOSAL** |
| **Title :** Documentation Generator for ECL Code  **Deliverables :**   1. Ecldoc utility developed in Java and packages as executable JAR file. 2. A plugin to run ecldoc utility from ECL-IDE 3. Ecldoc utility will contain 3 major parts :    1. File Parser – It parses individual .ecl files into relevant intermediate format like XML.    2. Linker – If ecldoc is run on complete project, it generate link information (IMPORT, Inheritance in Modules, etc) between multiple ecl files.    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 output of Documentation Linker.    4. Output Generator (2.0) – Write a plugin to automatically convert XML output to required output using open source documentation generators like Sphinx/Doxygen   **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 version control systems so that ecldoc utility only processes files that are changed. |
| **DETAILS OF ECLDOC UTILITY :**  (In addition to those in specifications README.rst file provide with project description)  Example: Consider following project structures.  A E  |--- B |--- F.ecl  | ---- B1.ecl |--- G  | ---- B2.ecl |--- G1.ecl  |--- C.ecl  Where A and E are 2 different projects located in 2 different locations (eg - /home/usr/eclp/A & /mnt/eclp2/E).  To run ecldoc on A,   1. We specify docs output folder (eg - /home/usr/ecldocs) 2. Run : ecldoc [-o /home/usr/ecldocs] [-f html] ***<A>***   This generates a directory /home/usr/ecldocs/A-Doc with structure.  A-Doc  |-------- pkg.xml – contains information about directory structure of this project.  |-------- link.xml – contains information about internal and external links used in this project.  |-------- XML-A  |---XML- B  | ---- B1.xml  | ---- B2.xml  |--- C.xml  |-------- index.html – point of entry for Docs of project A.  |-------- PKG-A – contains all general files for A project.  |--- pkgtree.html (Table of Contents)  |--- Bundle.html (If bundle file present in original)  |-------- A  |---B  |---- B1.html  |---- B2.html  |--- C.html  A similar structure can be generated for E in, for example, /mnt/ecl2docs/E-Doc.  Note : pkg.xml contains similar info as pkgTree.html i.e. Table of Contents.  Format : ecldoc [options] ***<Source-Path>***  [Options] :   1. –f : output format. Potential values : html, txt, rest 2. –lp : link-paths. Provides paths to external directory (assuming it exists). See Example below. Use pkg.xml file in link-paths to make appropriate links. 3. –o : Output Folder where Docs directory (named ***<Source-Path>***-Doc) is created 4. –a : Provide all the above options as an argument file (See argfiles in javadoc). 5. Other options as required.   ***<Source-Path>*** : Specifies paths to all the files/directory whose documentation is needed. Any regex can be present in source path using <regex> format.  Eg - ***</home/usr/ecl/A/<\*>>***  Note : We do not have a –r option for recursion.  If we need to recur, we can specify the wildcard path like ***<A/<\*>>***  If we do not need recursion, we can specify wildcard path like ***<A/<\*.ecl>>***  Examples :   1. Consider we want to generate documentation for all files in A recursively. No external dependencies. (ie no stmts like IMPORT E; or IMPORT E.G; etc)   **Run :** ecldoc [–r] [-f html] [-o /home/usr/ecldocs] ***</home/usr/eclp/A/<\*>>***  **Output :** Generate /home/usr/ecldocs/A-Doc   1. Consider we want to generate documentation for all files in E recursively. No external dependencies. (ie no stmts like IMPORT A; or IMPORT A.B; etc)   **Run :** ecldoc [-f html] [–r] [-o /mnt/ecl2docs/] ***</mnt/eclp2/E/<\*>>***  **Output :** Generate /mnt/ecl2docs/E-Doc   1. Consider we want to generate documentation for all files in A recursively. External dependencies. (eg stmts like IMPORT E; or IMPORT E.G; in B.B1.ecl etc)   **Run :** ecldoc [-f html] [–r] [–lp /mnt/ecl2docs/E-Doc] [-o /home/usr/ecldocs] ***</home/usr/eclp/A/<\*>>***  **Output :**   * 1. Generate /home/usr/ecldocs/A-Doc   2. A-Doc/link.xml contain appropriate external links to /mnt/eclp2docs/E-Doc   3. HTML files in A-Doc/A are appropriately linked to files in /mnt/ecl2docs/E-Doc/E  1. Consider we want to generate documentation for all files in B only. No external dependencies. (ie no stmts like IMPORT E; or IMPORT E.G; etc)   **Run :** ecldoc [–f html] [-o /home/usr/ecldocs] ***</home/usr/eclp/A/B>***  **Output :** Generate /home/usr/ecldocs/B-Docs . Note in this case, it is not /home/usr/ecldocs/A/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 [–r] ***</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 [–r] [-o /ecldocs] ***</ecl-ml/Pbblas>*** 2. ecldoc [–r] [–lp /ecldocs/PBblas-Doc] ***</ML>*** |

|  |  |
| --- | --- |
| Timeline Details |  |
| **Processing of Single File:**   1. Generation of appropriate combination of Lexer and Parser rules for ECL file processing. 2. Generation of Lexer and Parser rules for Docstring processing. 3. Generation of XML Output for Intermediate Representation. |  |
| **Generation of Linkage Information:**   1. Placeholders to provide appropriate linkage info for IMPORTs in XML files or @see tags. 2. Placeholders for appropriate linkage info for internal links .   eg:  For ecl code :  EXPORT Abc := RECORD  …  END;  EXPORT a\_func(DATASET(Abc) some\_data) := FUNCTION  …  END;  We might want to link Abc in DATASET to its definition in Abc RECORD. |  |
| **Implementation of Internal Linkage Information:**   1. Determine which links are internal to current file and which are external. 2. Replace Internal links using <LINK href=”#name-of-link-tag”>…</LINK> as well as what we are linking to <tag name=”name-of-link-tag”>. For example :   <ARGS type=”RECORD” name=”a1”>Abc</ARGS>  …  <ARGS type=”FUNCTION”>  DATASET(<LINK href=”#a1”>Abc</LINK>) some\_data  </ARGS> |  |
| **Directory Processing:**  Until now, we have only dealt with single file processing. Now we move onto complete project processing to generate XML representation.   1. Java 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 wildcard path, get the largest prefix with no regex. That will form the \*-Doc potion of output directory.  Eg – for **/home/usr/ecl/D**/D***<\d+>***/DD***<\d+>***.ecl, the output directory will be (assuming –o /home/usr/docs),  /home/usr/ecl/D-Doc   1. Generate backbone of output directory structure i.e.   --- pkg.xml  --- link.xml,  --- overview.xml,  --- XML-***<Source-Path> Directory***,  --- PKG-***<Source-Path> Directory***, etc. |  |
| **Walking the Code Directory:**   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 corresponding directory within output directory and recur inside. |  |
|  |  |
|  |  |