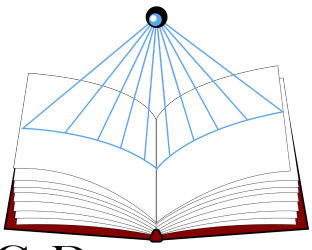
Critical Design Review Document

for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



R.C. DEVELOPMENT

Designed by Reading Club Development:
Matthew Brown, Gunnar Eastman, Jesiah Harris, Shea Keegan, Eli Story

Version 1.0
Dec 15, 2022

Executive Summary

The project set forth is to create a Python library of Macleod, using the Poetry package manager that can be installable via pip and be used to parse CLIF files, and to create a plugin for an existing IDE, Spyder, in which CLIF files can be easily opened, edited, and managed. This plugin is also to be called Macleod. Macleod stands for “Macleod - A Common Logic Environment for Ontology Development.” RCD is to accomplish these two tasks within the Fall 2022 and Spring 2023 semesters of the University of Maine’s academic year in correspondence with Dr. Torsten Hahmann and Jake Emerson. Thus far, RCD is prototyping the IDE through making mock-ups using framer and TKinter, though the third prototype used QT and met RCD’s clients’ approvals. The next steps are to learn about plugins, specifically how to write and manipulate them, and to begin work on increasing the accuracy of the currently out-of-date and slightly inaccurate parsing scripts. This Critical Design Review Document is to record the steps the RCD team has taken thus far, and to outline the next steps for the RCD team to complete in the near future.

Foreword

“Human knowledge is built incrementally and shared infrequently. In science we represent knowledge as models. These can be simple relationships between things, like the mechanics of a swinging pendulum, or highly complex, like the replication of DNA. Ongoing work in projects like COLORE, BFO, DOLCE, and many other scientific data models aim to unify how we communicate about data, information, and knowledge.

“An established specification for this kind of communication is called the Common Logic Interchange Format. However, there are obstacles to adoption. One significant obstacle in the path of this communication effort is the lack of a robust programming environment, such as an integrated development environment (IDE), that supports writing, debugging and otherwise working with complex logical statements in Common Logic. The project underway by R. C. Development is building the next step toward better scientific communication by updating the Common Logic text parser and IDE.”

- Jake Emerson, The Jackson Laboratory

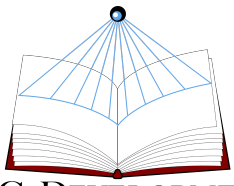
Preface

The goals of the project are to expand upon the existing CLIF parser, add a user interface for parsing and debugging, and make macleod available as a python library, which is to be done by the Reading Club Development (RCD) team during the Fall 2022 and Spring 2023 semesters of the University of Maine in partial fulfillment of the Computer Science BS degree for the University of Maine. There is thus limited development time, but the hope is

to produce a product which will satisfy the requirements set forth by the clients, Jake Emerson and Torsten Hahmann.

In preparation for this project, the RCD team has proven themselves to understand both the V-Shaped Development Process and Agile Development Process. The requirements of the system were set forth in the System Requirements Specification document (SRS), the design requirements enumerated in the System Design Document (SDD), and the user interface standards were set within the User Interface Design Document (UIDD). Each of these documents have been created under and with the express guidance of the clients via biweekly check-in meetings.

The planned updates to the Macleod software will allow for a more robust system of ontology writing, which will speed up the development and debugging of Common Logic ontologies. Specific use will primarily be in sciences, where reproducibility is difficult due to ambiguous data. An example given by Jake Emerson is that of the Mouse Genome Informatics team at Jackson Laboratory. This team currently uses Web Ontology Language (OWL) as their specification language. However, OWL is not as expressive as CLIF, so the team would use Macleod and the CLIF Parser to write ontologies to better represent the data they collect and the experiments they run.



R.C. DEVELOPMENT

Critical Design Review Document:

Table of Contents

List of Figures

List of Tables

Symbols (Nomenclature)

RCD: Reading Club Development
SPR: System Proposal Review
SRR: System Requirements Review
SRS: System Requirements Specification
SDR: System Definition Review
SDD: System Design Document
SSR: Software Specification Review
UIDD: User Interface Design Document
CLIF: Common Logic Interchange Format
OWL: Web Ontology Language
TPTP: Thousand Problem Theorem Prover
LADR: Library for Automated Deduction Research
FR: Functional Requirement
NFR: Non-Functional Requirement

Summary

Macleod was originally created by Dr. Torsten Hahmann, and later re-engineered by Robert Powell. This initial project set the groundwork for the general structural flow of the Macleod system: the user inputs a CLIF file to be parsed, which the system does; then the file can be translated to TPTP, OWL, or LADR formats; these new formats are then passed through a consistency checker, with the output eventually being whether the translations are consistent.

In the years since its inception, Macleod has had an ontology visualization interface, though this has since fallen into disuse and disrepair, and is currently nonfunctional. The parser itself is, in its current state, only covers about 90% of the CLIF language constructs, and the standards for CLIF have since been updated as well, meaning that the parser has even less complete coverage now. Thus lie the two chief development matters of RCD: create an IDE to be used for editing and debugging CLIF files by interacting with the existing Macleod backend, and expand the CLIF parser to cover more of the current CLIF standard. RCD will not interact with the conversions to the other file formats, nor the consistency checking.

The requirements were discussed via a System Requirement Review, which ultimately led to the production of the aforementioned SRS. These requirements that were covered were higher level in nature, discussing the functions of the system that is to be created, and lower-level requirements, such as what the system is to perform on, how the system is to be accessed, and the usual response times of the system.

Following the specifications of the system came the design requirements of the system. The System Definition Review was where the design requirements were gathered and discussed. These requirements allowed for the creation of the SRS, wherein the design elements of the project are set forth, and the full goals of the project are written.

Next came the user interface design, with the requirements elicited via a Software Specification Review with the clients. This review led to the production of the UIDD, where the user interface of the system was prototyped. In the UIDD, the requirements for the user interface were also produced, which the RCD team plans on following through to the end of the project.

The goals of this project have thus been laid out in the previously mentioned documents, with two ultimate goals. The first goal is to create a Python library via Poetry to allow for the easy and accurate parsing, conversion, and consistency checking of CLIF files. The

second goal is to create a plug-in for a pre-existing IDE (likely Spyder) which will allow for the convenient modification, parsing, and translation of CLIF files.

1. Introduction

The Common Logic Interchange Format (CLIF) Parser is a capstone project for Dr. Torsten Hahmann and Jake Emerson, in partial fulfillment of the Computer Science BS degree for the University of Maine, completed by the Reading Club Development (RCD) team. Dr. Hahmann is a professor at the University of Maine with affiliation to the Spatial Data Science Institute and research interests in knowledge representation, logic, and automated reasoning. Jake Emerson works for Jackson Laboratories in Bar Harbor, Maine, where he would implement this parser into the workflow of projects he is involved with. RCD consists of five seniors from the University of Maine: Matthew Brown, Gunnar Eastman, Jesiah Harris, Shea Keegan, and Eli Story.

1.1 Purpose of This Document

This CDRD is meant to illustrate the current state of the CLIF Parser and plugin that RCD has been tasked with developing. This SRS will also describe the work that RCD has completed on the project in the course of the first semester of the two-semester long Capstone class at the University of Maine. The intended readership of this document consists of the client and the RCD team so as to effectively document the progress on the CLIF Parser and IDE. The CLIF Parser is to be used by researchers so as to better document their methods for experiments so as to allow for more repeatability in their experiments.

2. Purpose

The field of biology is currently facing a “crisis of reproducibility” according to Jake Emerson, one of our clients and an engineer at Jackson Laboratory in Bar Harbor, Maine. The development of an ontological reader is, for him, paramount due to the congruity of semantics within ontological statements. The CLIF Parser is to be a stepping stone in the progress toward wider uptake of Common Logic, allowing for specifying more detailed semantics, and hopefully slightly mitigating this crisis of reproducibility.

On a smaller scale, the purpose of the CLIF Parser is primarily to parse CLIF files to ensure they are syntactically correct according to CLIF standards. The library should also translate from the CLIF syntax to TPTP, or other logic-based conventions. Another purpose of the product is to build an IDE for easier use of the previously developed Macleod, so Common Logic can have an IDE dedicated to supporting it.

3. Method

The RCD development team will use an agile approach, ensuring that the pre-existing product remains usable during the entire process. Efforts will be split between making

improvements to the functionality of the software itself, packaging the software as a python library, and designing an IDE based on the Spyder platform to provide some level of GUI for interacting with Macleod and make the product easier to use.

4. Design

This section will review the overall design of the product. It should look similar to how the product currently functions, this is to help ensure that RCD does not cause the product to stop working.

4.1 Design Parameters

For the project we are constrained to the limitations of Common Logic while building the parser, specifically the Common Logic Interchange Format (CLIF), as created by the International Organization for Standardization (ISO). This is to ensure that the files that are created align with the syntactic standards that have been set in place by the ISO. The purpose of this is to allow for ontologies to be consistent when translated into various other logical languages such as TPTP or LADR.

For the GUI we will be limited by the functionality of Spyder and its capabilities.

4.2 Design Approach

The basic design is a Pipe and Filter architecture model, wherein data is passed through a series of steps to an eventual output. In Fig. 1, the data is fed to the parser.py (the Parser), which is the beginning of the Macleod architecture. Specifically, using Macleod as a library allows for the calling of the Parser via the parse_clif function, and possibly the translating of the CLIF file into various formats. The parse_clif script generates an ontology object whether the translation is required or not, and the ontology object prints itself to the terminal. If conversion is necessary, then the specific language to be translated into is required, and the ontology object is translated into the relevant language and the converted file is output.

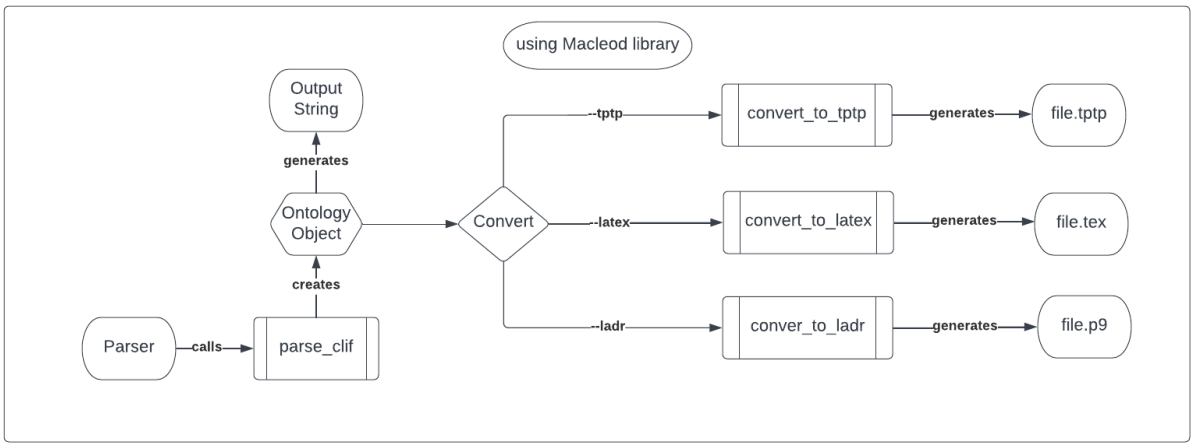


Fig. 1: Data Pipeline of Macleod Library: This diagram documents the flow of data through the Macleod system, from the parser to the eventual output.

Macleod is currently built on Python. All of the relevant scripts from Fig. 1 exist, though the parser is presently incomplete. The work of RCD is twofold on the parser: firstly is to package up Macleod into a Python library via Poetry that is then downloadable by Pip; secondly is to ensure full coverage of the CLIF standard.

Fig. 2 demonstrates how the Macleod plugin will interface with the Spyder IDE. The Macleod plugin will have the same general functionality as the Macleod library. The “parse” and “convert” options will be handled via buttons displayed on the IDE. One button will simply parse the file, and an array of buttons will be available to convert the file into different formats. However, if the parse fails at either button press, we will use Spyder’s error checking to display where the error is in the CLIF file that caused the parse to fail. The user will then be able to edit the file using Spyder’s normal editor.

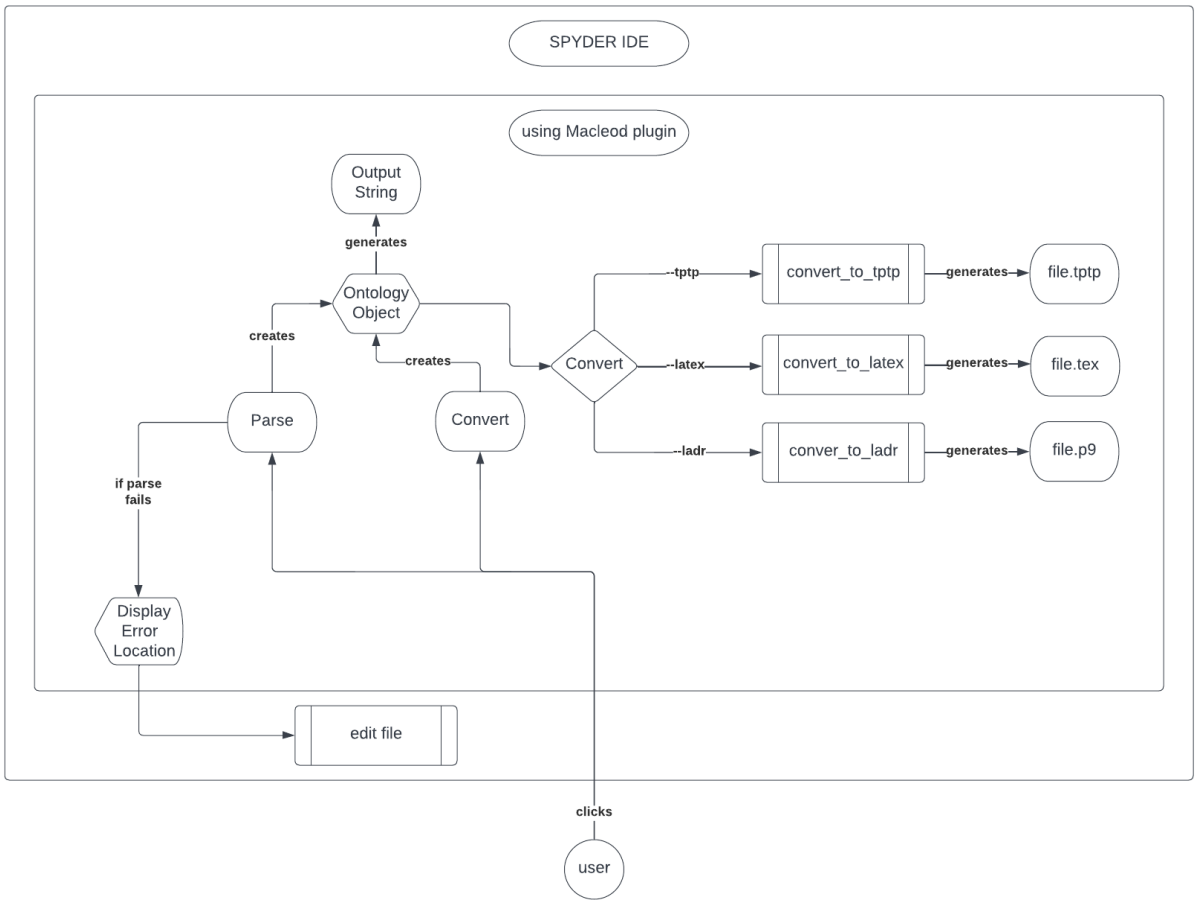


Fig. 2: IDE Architecture Diagram of Macleod: This diagram documents how the user input is processed by the system depending on which button the user clicks.

4.3 Design Problems

The largest issues facing RCD are whether we should make the IDE from scratch, how the library fits into the Macleod architecture, and how the parser truly operates.

The idea behind creating an IDE from scratch is that it would be entirely customizable to the needs of the clients. There would also only need to be an executable that is downloaded in order for the IDE to be able to be used seamlessly. However, the full construction of an IDE requires an extensive knowledge of how to use the tools provided through TKinter or Qt for Python. Essentially, it would have been quite an undertaking in order to fully construct a fully-featured IDE from scratch, more than we could likely accomplish, let alone accomplish in parallel with the parser expansion. Instead, we are choosing to create a Spyder plugin to accomplish similar purposes.

As for how the library fits into the Macleod architecture, we had a hard time distinguishing how to include the library into the system. The Python library will be installable via Pip and importable into a .py file, wherein the parse_clif script can be called with the file location specified. The library is to be utilized when parsing and/or translating is important to a python script, but the Macleod plugin will be utilized when a user is creating or curating CLIF files.

How the parser operates eluded the RCD team for a few months. The process of how to download and get the parser going required an extensive use of file organization and file manipulation. The user had to modify configuration files in two separate spots with quirks riddled throughout. Currently, only two members of the RCD team even have the ability to parse CLIF files by using the supplied Macleod parser. However, we have now written out steps that, though not universally applicable, are a good starting point on making Macleod more accessible.

4.4 Design Details

The fine details of the design have been discussed in UIDD §2. To summarize, the GUI will have five sections each with their own functionalities. The sections include; file navigation, file editor, console, error list, toolbar. Each section will interact with the parser by either calling functions in the parser or by displaying data that the parser generates.

For the Parser, we intend to follow the coding structure that already exists. Following this structure we will enhance the capabilities of the parser to accommodate parts of the common logic language that was not previously covered.

5. Equipment

No physical equipment is being used other than the personal computers of the RCD team. The team is running the current parser and plans on running the future library and plugin on Ubuntu Linux, Windows 10 and 11, and Mac Ventura.

6. Test

This section will contain our testing information. This will include the file types and formats that will be accepted, along with some tests that were run inorder to isolate syntax that isn't accepted, and their results.

6.1 Test Conditions

The one constant among tests is that inputs will be CLIF files. We will test translation into each of TPTP, LADR, and OWL formats.

6.2 Test Procedures

We were given three test files, all but one of them in the modern CLIF format, which does not currently work with the parser. Then, we ran them through the parser, and recorded our results. We are now working to isolate which conventions do not work, specifically starting with ensuring the preservation of comments. We are working to create new test files to isolate syntactic elements to ensure they are working, starting with a basic "text.clif" file.

Fig. 3: Test File text.clif: RCD's first original test file, isolating the syntactic elements of *forall*, *if*, *not*, *iff*, *compl*, and *ntpp*.

6.3 Test Results

We tested on four files, *rcc_basic_old.clif*, *rcc_basic.clif*, *rcc.clif*, and *text.clif*. We expected *rcc_basic.clif* and *rcc.clif* to fail due to them containing the updated comment syntax. Both *rcc_basic.clif* and *rcc.clif* failed due to the incorrect parsing of the comments because of the updated syntax, which is expected. The two files expected to parse successfully, *text.clif* and *rcc_basic_old.clif*, both did parse successfully. This means that we have isolated our first syntactic element to fix: comments.

7. Data

Unfortunately, there is not very much data to analyze currently. Of the four CLIF files that we have to test the parser on, two succeed and two fail. The two that fail, fail for the same reason, because the comment structure of CLIF was updated after the conception of

Macleod, and thus the parser has not been updated since the comment structure was updated. As for the two that succeed, *rcc_basic_old.clif* contains valuable insight into the conventions that do work, due to the fact that this file parses with several ontologies contained within it. As for *text.clif*, the file is too small to be of much use besides ensuring that the parser is set up correctly to parse.

File Name	Results	Output
rcc_basic_old.clif	Success	(base) matthewbrown@Matthews-MBP macleod % parse_clif -f /Users/matthewbrown/Documents/macleod/test_ontologies/rcc_basic_old.clif 2022-12-13 17:04:41,316 macleod.Filemgt INFO Config file read: /Users/matthewbrown/macleod/macleod_mac.conf 2022-12-13 17:04:41,316 macleod.Filemgt INFO Logging configuration file read: /Users/matthewbrown/macleod/logging.conf 2022-12-13 17:04:41,316 macleod.Filemgt DEBUG Started logging with MacleodConfigParser 2022-12-13 17:04:41,316 macleod.scripts.parser INFO Called script parse_clif ELIMINATING CONDITIONALS False 2022-12-13 17:04:41,322 macleod.scripts.parser INFO Starting to parse /Users/matthewbrown/Documents/macleod/test_ontologies/rcc_basic_old.clif
rcc_basic.clif	failure	Unexpected Token: 'http://colore.oor.net/mereotopology/rcc_basic.clif' :: "http://colore.oor.net/mereotopology/rcc_basic.clif " COMMENT LPAREN NONLOGICAL nonlogicals LPAREN NONLOGICAL NONLOGICAL ... TypeError: Error in function: bad nested term
rcc.clif	failure	Unexpected Token: 'http://colore.oor.net/mereotopology/rcc.clif' :: "http://colore.oor.net/mereotopology/rcc.clif " COMMENT LPAREN NONLOGICAL nonlogicals LPAREN NONLOGICAL NONLOGICAL

		... TypeError: Error in function: bad nested term
text.clif	success	(base) matthewbrown@Matthews-MBP macleod % parse_clif -f /Users/matthewbrown/Documents/macleod/test_ontologies/text.clif 2022-12-13 17:10:38,575 macleod.Filemgt INFO Config file read: /Users/matthewbrown/macleod/macleod_mac.conf 2022-12-13 17:10:38,575 macleod.Filemgt INFO Logging configuration file read: /Users/matthewbrown/macleod/logging.conf 2022-12-13 17:10:38,575 macleod.Filemgt DEBUG Started logging with MacleodConfigParser 2022-12-13 17:10:38,575 macleod.scripts.parser INFO Called script parse_clif ELIMINATING CONDITIONALS False 2022-12-13 17:10:38,581 macleod.scripts.parser INFO Starting to parse /Users/matthewbrown/Documents/macleod/test_ontologies/text.clif

Table 1: Results from Test Files: This table depicts that two of the test files succeeded and two of them failed, along with the output of the parse.

8. Conclusions

Throughout the semester, we have come to three conclusions. Firstly, according to the clients, the parser is approximately 90% correct already. Secondly, we have cemented the tools we have used in Spyder and Poetry. Thirdly, documentation is hard and time consuming.

As for the completeness of the parser, the parser being as complete as it is is enormously helpful because it means the RCD team has to do less work. Our work going forward is to isolate the syntactic elements that do not work, so that we can improve the parser. We have decided the first element to address is altering the parser to comments present in the documents it translates, ensuring that those comments are also present in the translation.

Isolating the tools to use to complete the client's proposal was more difficult than initially thought. In our SPR, there were a few packaging methods discussed, such as pyproject or

Poetry. We now know that we are to use Poetry to package Macleod. We also began by thinking we would be building the IDE off of Macleod's pre-existing native GUI. We were wrong, and any IDE version of Macleod needed to be developed by us. We initially believed that we would be writing our own IDE using TKinter or Qt, though now we know that we are to be building a plugin off the Spyder IDE that utilizes Macleod.

Thirdly, documentation is important, time-consuming, and difficult at times. Even through this difficulty, complying with client standards is paramount, and we need to develop in conjunction with writing documentation. The RCD team plans on developing more in the spring semester, but also plans on keeping the quality of documentation consistent.

9. Recommendations

On future projects, we will be more probing when it comes to requirements and overall system design to ensure that we are building the correct system from the outset. It felt as though most times we attempted to validate that we were understanding the project correctly, our understanding came up short and led us to numerous setbacks. We ultimately did not get as much as we expected to do this semester due to misunderstandings, mostly stemming from poor elicitation of requirements.

We need to be more communicative. Going forward, we would recommend involving the clients on the action items and open issues, thus giving them a line into what we plan on accomplishing on any given sprint. Our clients were expecting more code to be written and less documentation, due to a lack of communication by us, which could have been remedied easily.

References

Brown, M, et al., CLIF Parser System Requirement Specification, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_SRS.pdf.

Brown, M, et al., CLIF Parser System Design Document, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_SDD.pdf.

Brown, M, et al., CLIF Parser User Interface Design Document, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_UIDD.pdf.

Bibliography

Colore, Semantic Technologies Laboratory, http://stl.mie.utoronto.ca/colore/.

Hahmann, Torsten, *Macleod*, GitHub, 2022, https://github.com/thahmann/macleod.

Acknowledgement

We want to express our thanks to Jake Emerson. Every step of the way, Jake has been asking to aid us in development, in testing, even in writing documentation. Jake also wrote the foreword for this document, which was incredibly helpful, and for that we are very grateful.

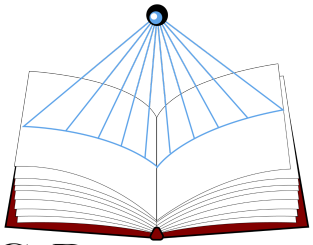
We also want to express our thanks to Dr. Torsten Hahmann for the opportunity to work on this project under his wisdom and tutelage. He is patient with us, and always insightful in his opinions.

We would also like to thank our Sister Team, Sled Dogs, for reviewing and performing quality assurance on our documentation.

Appendix A - SRS

System Requirements Specification Document

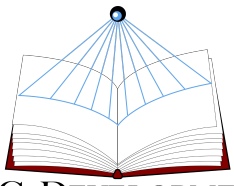
for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



R.C. DEVELOPMENT

Designed by Reading Club Development:
Matthew Brown, Gunnar Eastman, Jesiah Harris, Shea Keegan, Eli Story

Version 1.3
Dec. 13, 2022



R.C. DEVELOPMENT

System Requirements Specification:

Table of Contents

Date	Reason for Change	Version
10/18/2022	Initial Creation	1.0
10/27/2022	Specified when the Unit Testing Phase occurs.	1.1
12/01/2022	Updated and color-coded requirements	1.2
12/13/2022	Updated based on feedback	1.3

1. Introduction

The Common Logic Interchange Format (CLIF) Parser is a capstone project for Dr. Torsten Hahmann and Jake Emerson, in partial fulfillment of the Computer Science BS degree for the University of Maine, completed by the Reading Club Development (RCD) team. Dr. Hahmann is a professor at the University of Maine with affiliation to the Spatial Data Science Institute and research interests in knowledge representation, logic, and automated reasoning. Jake Emerson works for Jackson Laboratories in Bar Harbor, Maine, where he would implement this parser into the workflow of projects he is involved with. RCD consists of five seniors from the University of Maine: Matthew Brown, Gunnar Eastman, Jesiah Harris, Shea Keegan, and Eli Story.

This SRS will detail the functional and nonfunctional requirements set forth for this project, the deliverables necessary for completion, and document the consent of the team members and the client. The CLIF Parser is to be used by researchers so as to better document their methods for experiments so as to allow for more repeatability in their experiments.

1.1 Purpose of This Document

This SRS is meant to enumerate the functional and nonfunctional requirements set forth for the CLIF Parser that RCD has been tasked with developing. This SRS will also describe the work that RCD is to complete in the course of this project and the two-semester long Capstone class at the University of Maine. The intended readership of this document consists of the client and the RCD team so as to serve as an agreement between RCD and the clients on how to effectively develop the proposed CLIF Parser.

1.2 References

Macleod, Dr. Torsten Hahmann, GitHub, 2022, https://github.com/thahmann/macleod.

1.3 Purpose of the Product

The field of biology is currently facing a “crisis of reproducibility” according to Mr. Emerson. The development of an ontological reader is, for him, paramount due to the congruity of semantics within ontological statements. The CLIF Parser is to be a stepping stone in the progress toward unified Common Logic, allowing for more properly defined variables, and hopefully slightly mitigating this crisis of reproducibility.

On a smaller scale, the purpose of the CLIF Parser is primarily to parse CLIF files to ensure they are syntactically correct according to CLIF standards. The library should also translate from the CLIF syntax to TPTP, or other logic-based conventions. Another purpose of the product is to build upon the previously developed Macleod IDE, so Common Logic can have an IDE dedicated to supporting it.

1.4 Product Scope

The scope of the project is twofold. Firstly there is the parser. At the moment there is an existing parser that works, but is outdated and is missing some parsing capabilities. At the moment the installation of the parser is very complicated and time consuming. During the scope of this project, we hope to repair and further the development of the parser as well as turn it into a python library via Poetry. This will allow an easy installation for users.

The second part of this project is the IDE. Again there is an existing IDE, however it is slow and cumbersome. The goal to solve this problem is to create a Spyder plugin. This will allow the user to interact with the parser in an environment with many tools to their disposal.

Overall the project will include a parser for managing CLIF files and an IDE to help that management. The two systems will be separate. The parser can be used through the command line separate to the IDE. The IDE will rely on the parser to parse the files, therefore it can not be used as a standalone system.

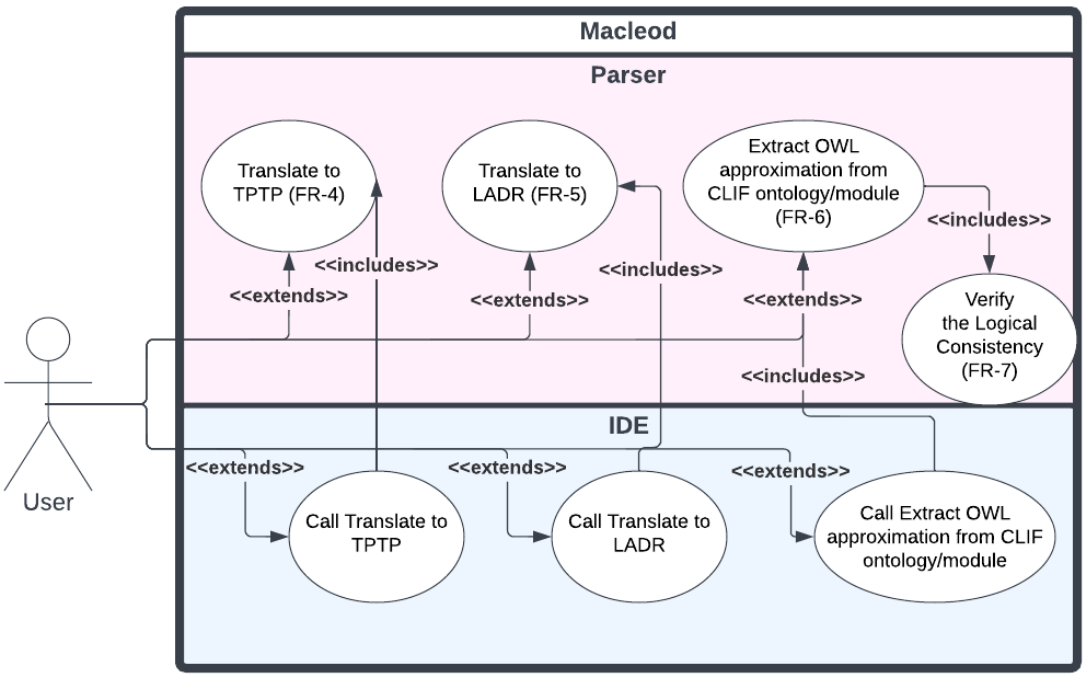


Fig. 1: Primary Use Case Diagram: This diagram depicts how Macleod is to be used as a parser and as an IDE.

This diagram briefly depicts the anticipated interactions between Macleod and the user. The scope of the system is intended to be that the system will translate CLIF files to other file formats and verify the syntactic and logical consistency of pre-existing CLIF files. The ability to edit the CLIF file will be granted if an error has been located.

2. Functional Requirements

In this section, we outline the requirements that detail the functionality of the system. Priority is measured on a scale from 1 to 5, with 5 being the most critical.

Blue = Already done, should work when project is finished
Pink = Needs to be completed
Green = New

1. The system shall identify Quantified Variables (and their scope and use).
2. The system shall identify Predicates.
3. The system shall identify Statements from file.
4. The system shall take CLIF and output TPTP.
5. The system shall take CLIF file and produce LADR output.
6. The system shall extract OWL approximation from CLIF ontology/module.
7. The system shall verify the logical consistency of a CLIF ontology or module.
8. The system shall prove theorems that encode intended consequences (e.g. properties of concepts and relations) of ontologies/modules.
9. The system shall preserve all comments.
10. The system shall bringing up an editor to fix syntax errors.
11. The system shall, identify Instructions from file.
12. The system shall provide a button to parse a CLIF file.
13. The system shall provide a button to parse and convert a file.

Number	FR-1
Name	The system shall identify Quantified Variables (and their scope and use).
Summary	The system shall identify variables in a provided logical statement
Priority	5
Preconditions	An input has been entered
Postconditions	Variables would have been identified
Primary Actor	

Secondary Actors	
Trigger	A logical statement is given as an input
Steps	The system shall examine each individual string of symbols separated by white space, and each string that is not defined by the file format shall be identified as a variable. (examples of non-variables include “for all” symbols, “for each”, punctuation, etc.)
Open Issues	
Tests	FR-1 will be tested during the Unit Testing phase (early 2023) by providing logical statements and asserting that the variables identified by the system are the same as those identified by the developers.

Number	FR-2
Name	The system shall identify Predicates.
Summary	The system will identify predicates and function symbols in a inputted logical statement
Priority	5
Preconditions	An input has been entered
Postconditions	Predicates would have been identified
Primary Actor	
Secondary Actors	
Trigger	A logical statement is given as an input
Steps	The system shall examine each symbol/word in the input, and each one

	that matches a list of predicate symbols that will be extracted from the file, shall be identified as a predicate.
Open Issues	
Tests	FR-2 will be tested during the Unit Testing phase by providing logical statements and asserting that the predicates identified by the system are the same as those identified by the developers.

Number	FR-3
Name	The system shall identify Statements from file.
Summary	The system can read a file and identify logical statements written inside
Priority	5
Preconditions	A file must be given
Postconditions	Statements are identified
Primary Actor	
Secondary Actors	
Trigger	A file location is given to read
Steps	
Open Issues	
Tests	FR-3 will be tested during the Unit Testing phase by providing a CLIF file and asserting that the logical statements identified by the system are the same as those identified by the developers.

Number	FR-4
Name	The system shall take CLIF and output TPTP.
Summary	The system should be able to take a CLIF file as an input and give a TPTP file as an output
Priority	5
Preconditions	The system is not currently working on another task.
Postconditions	A TPTP file is produced as output
Primary Actor	
Secondary Actors	
Trigger	A CLIF file is given as input
Steps	
Open Issues	
Tests	FR-4 will be tested during the Integration Testing phase, by providing the system with CLIF files of varying length and complexity and asserting that the system translates them to TPTP correctly.

Number	FR-5
Name	The system shall take CLIF file and produce LADR output.
Summary	The system should be able to take in a CLIF file and produce LADR output

Priority	4
Preconditions	CLIF file must be given and accepted by system
Postconditions	The system will produce LADR out from a given CLIF file
Primary Actor	
Secondary Actors	
Trigger	A CLIF file is given to the system
Steps	
Open Issues	
Tests	FR-5 will be tested during the Integration Testing phase, by providing the system with CLIF files of varying length and complexity and asserting that the system translates them to LADR correctly.

Number	FR-6
Name	The system shall extract OWL approximation from CLIF ontology/module.
Summary	The system should be able to extract an OWL (Web Ontology Language) approximation from a CLIF ontology or module
Priority	4
Preconditions	A CLIF ontology or module must be given to the system. The system must have the ability to translate CLIF ontologies to approximate OWL ontologies

Postconditions	An OWL approximation will be created/extracted by the system
Primary Actor	
Secondary Actors	
Trigger	A CLIF ontology/module is given to the system
Steps	
Open Issues	FR-6 will be tested during the Integration Testing phase, by providing the system with CLIF files of varying length and complexity and asserting that the system translates them to OWL correctly.

Number	FR-7
Name	The system shall verify the logical consistency of a CLIF ontology or module.
Summary	The system shall be able to verify the logical consistency of a CLIF ontology/module
Priority	5
Preconditions	The system must have access to theorem provers and an inputted CLIF file.
Postconditions	A logical CLIF ontology/module will be identified and ready for processing by the system
Primary Actor	
Secondary Actors	
Trigger	A CLIF ontology/module is given to the

	system
Steps	
Open Issues	
Tests	FR-7 will be tested during the Integration Testing phase, by providing the system with CLIF files of varying length and complexity, and correctness, and asserting that the system confirms correctly whether or not the files are logically consistent.

Number	FR-8
Name	The system shall prove theorems that encode intended consequences (e.g. properties of concepts and relations) of ontologies/modules.
Summary	The system should have theorem proving capabilities that can encode the intended consequences of ontologies/modules given to the system. Examples of intended consequences are properties of concepts and relations or competency questions.
Priority	3
Preconditions	The system must have a theorem prover or theorem proving capabilities that can process given ontologies/modules.
Postconditions	Intended consequences of an ontology or module will be proven and reported by the system
Primary Actor	

Secondary Actors	
Trigger	
Steps	
Open Issues	
Tests	FR-8 will be tested during the Integration Testing phase, by providing the system with CLIF files of varying length and complexity and asserting that the system translates them to TPTP correctly.

Number	FR-9
Name	The system shall preserve all comments
Summary	The system shall maintain any comments present in input CLIF files, and make them present in the associated output files.
Priority	3
Preconditions	A CLIF file with comments has been submitted as input.
Postconditions	The comments are still present in the translated output.
Primary Actor	User
Secondary Actors	
Trigger	
Steps	
Open Issues	
Tests	FR-9 will be tested during the Integration

	Phase by providing the system with CLIF files that have comments in various places and of various lengths, and ensuring those comments are intact in the output.

Number	FR-10
Name	The system shall bringing up an editor to fix syntax errors.
Summary	The system shall open CLIF files and point out errors.
Priority	3
Preconditions	The system is reading through a CLIF file
Postconditions	A CLIF editor has been opened to the location of the error so that it can be fixed.
Primary Actor	User
Secondary Actors	
Trigger	The system finds a syntax error
Steps	The system marks the location of the error, opens a CLIF editor, and scrolls to the location of the error.
Open Issues	
Tests	FR-10 shall be tested during the Integration Testing phase, by providing the system with CLIF files that have various syntactical errors in various places.

Number	FR-11
Name	The system shall, identify Instructions from

	file.
Summary	The system shall be able to identify instructions outside of the logic of the ontology, such as import statements and module declarations
Priority	5
Preconditions	A file must be given
Postconditions	Instructions are identified
Primary Actor	User
Secondary Actors	
Trigger	A file location is given to read
Steps	
Open Issues	
Tests	FR-11 will be tested during the Unit Testing phase by providing CLIF files with import statements, module declarations, and both, and assuring that those identified by the system match those identified by the developers.

Number	FR-12
Name	The system shall provide a button to parse a CLIF file.
Summary	The system shall provide one button that will parse a CLIF file and ensure that it is syntactically correct.
Priority	5

Preconditions	A CLIF file is open in spyder.
Postconditions	The file has been ensured to be syntactically correct or errors have been raised.
Primary Actor	User
Secondary Actors	
Trigger	The button is pressed
Steps	
Open Issues	
Tests	FR-12 will be tested during the Integration Testing phase by providing CLIF files with and without issues and testing them in spyder.

Number	FR-13
Name	The system shall provide a button to parse and convert a file.
Summary	The system shall provide one button that will parse a CLIF file and ensure that it is syntactically correct and translate it to an indicated other language.
Priority	5
Preconditions	A CLIF file is open in spyder.
Postconditions	The file has been translated or errors have been raised.
Primary Actor	User
Secondary Actors	

Trigger	The button is pressed
Steps	
Open Issues	
Tests	FR-13 will be tested during the Integration Testing phase by providing CLIF files with and without issues and testing them in spyder to see that they are translated correctly.

Table 1: Functional Requirements of Macleod: This table outlines the core functionalities that the system shall have upon completion.

3. Non-Functional Requirements

In this section, we will outline the requirements that do not directly involve the functionality of the system.

Blue = Already done, should work when project is finished
Pink = Needs to be completed
White = Should just work
Green = New

1. The system shall work on Linux, Mac, and Windows.
2. The system shall be installable via pip as a python library.
3. The GUI will respond to all inputs with at least a “loading” message within 2 seconds 90% of the time.
4. The system shall correctly translate at least 95% of input content.
5. The system shall translate an individual logical statement within 1 second 95% of the time, to ensure that the translation of a document does not take a prohibitive amount of time.
6. The system shall parse a CLIF file within 3 seconds 90% of the time.
7. The system shall be accompanied by a User Guide, which should allow users to use the system within 30 minutes.
8. The system itself shall not keep a record of any files it translates.
9. The system shall be installable as a spyder plugin.

Number	NFR-1

Description	The system shall work on Linux, Mac, and Windows.
Priority	5
Tests	All tests shall be performed on a Linux, Mac, and Windows system.

Number	NFR-2
Description	The system shall be installable via pip as a python library.
Priority	5
Tests	Test NFR-2: Once completed, we shall attempt to install the system using pip.

Number	NFR-3
Description	The GUI will respond to all inputs with at least a “loading” message within 2 seconds 90% of the time.
Priority	4
Tests	Test NFR-3: Measure the time it takes for the GUI to respond to various requests.

Number	NFR-4
Description	The system shall correctly translate at least 95% of input content.
Priority	4
Tests	Test NFR-4: Have the system translate

	small and large files and ensure that there are no errors within at least 95% of the resulting files.

Number	NFR-5
Description	The system shall translate an individual logical statement within 1 second 95% of the time, to ensure that the translation of a document does not take a prohibitive amount of time.
Priority	4
Tests	Test NFR-5: Have the system translate the same files as in Test NFR-4, and record how long it takes for the system to translate those files.

Number	NFR-6
Description	The system shall parse a CLIF file within 3 seconds 90% of the time.
Priority	4
Tests	Test NFR-6: Have the system parse files with similar variation as in Test NFR-4, and record how long it takes for the system to parse those files.

Number	NFR-7
Description	The system shall be accompanied by a User Guide, which should allow users to use the system within 30 minutes.

Priority	5
Tests	Test NFR-7: Our sister team in the Capstone class will be given access to the system and the User Guide, and we will determine whether they are able to make use of the system after 30 minutes.

Number	NFR-8
Description	The system itself shall not keep a record of any files it translates.
Priority	2
Tests	Test NFR-8

Number	NFR-9
Description	The system shall be installable as a spyder plugin
Priority	5
Tests	Test NFR-2: Once completed, we shall attempt to install the system as a spyder plugin

Table 2: Non-Functional Requirements of Macleod: This table outlines the requirements that do not involve the functionality of the system.

4. User Interface

This section briefly outlines any user interface technicalities that will need to be completed for this project.

See “User Interface Design Document for CLIF Parser.”

5. Deliverables

This section will enumerate the deliverables RCD is to produce throughout the Fall 2022 and Spring 2023 University of Maine semesters in accordance with the requirements set forth both by the Capstone class and the client's desired outcome.

The following deliverables shall be produced and given to the client:

Electronic files containing the following:

- Systems Requirement Specification
    - This will be shared via Google Docs.
    - We will send the SRS to the client digitally October 18, 2022.
- System Design Document
    - This will be shared via Google Docs.
    - We will send the SDD to the client digitally November 8, 2022.
- User Interface Design Document
    - This will be shared via Google Docs.
    - We will send the UIDD to the client digitally November 21, 2022.
- Code Inspection Report
    - This will be shared via Google Docs.
    - We will send the CIR to the client digitally in the spring, on approximately February 16, 2023
- User Manual
    - This will be shared via Google Docs.
    - We will send the UM to the client digitally in the spring, on approximately March 1, 2023
- Administrator Manual
    - This will be shared via Google Docs.
    - We will send the AM to the client digitally in the spring, on approximately March 14, 2023
- All source code
    - This will be stored on a GitHub repository which the client will have permanent access to.
- Macleod Verion 1.1
    - We will send the CIR to the client digitally in the spring, on approximately May 1, 2023
- Any other software required for installation and execution of the delivered program.
    - This will be stored on a GitHub repository which the client will have permanent access to.

6. Open Issues

This section will enumerate issues that have been raised, but are as of yet lacking a solution. These issues will be addressed later in development.

Open Issue	**Approximate Resolution Date**
1. Download and get Macleod up and running on our machines.	10/21/22
2. Isolate the first conventions of CLIF syntax we are to parse.	10/21/22
3. Make a Python library with Poetry release	4/1/23
4. Simplify configuration after install	5/1/23
5. Need to update CLIF BNF grammar	3/1/23
6. Cleanup GUI	5/1/23
7. Remove pyparsing dependency	2/1/23
8. Parser.py doesn't use prefix when importing	3/1/23
9. Parser doesn't like the /** <comment> **/ comments	2/1/23
10. Fix/Test setup.py	2/1/23

Appendix A-1

This appendix details the expectations that RCD shall uphold to the client upon completion of this document, and how future changes to this document shall be made.

RCD and the client, upon the signing of the document, are agreeing that this SRS contains a compilation of the nonfunctional and functional requirements necessary for the CLIF Parser. RCD and the client agree that these requirements are to be developed, tested, and integrated over the course of the Fall 2022 and Spring 2023 University of Maine semesters. The client, in signing this SRS, agrees that these requirements are sufficient for completion of this project. The team, RCD, agrees that these requirements are meant to be agile and flexible in nature, so if the need arises, the requirements may change in accordance with the client's wishes.

Any changes made to this document must be approved by all members of RCD and the client via signatures to an additional appendix wherein the changes are enumerated and detailed. Changes to this document include, but are not limited to, updating requirements, removing requirements, adding requirements, and changing structure as to be in accordance with the Capstone requirements for the University of Maine course this project is managed through. The signing of this appendix consents all members of RCD and the client that this structure of implementing changes is acceptable.

Name:	Signature:	Date:
Torsten Hahmann	_____	__/__/__
Jake Emerson	_____	__/__/__
Matthew Brown	_____	__/__/__
Gunnar Eastman	_____	__/__/__
Jesiah Harris	_____	__/__/__
Shea Keegan	_____	__/__/__
Eli Story	_____	__/__/__

Appendix A-2

This appendix will contain the agreement that all members of RCD have read and consent to the document in its entirety.

Through signing this appendix, we, as the members of RCD, agree that we have reviewed this document fully, we agree to the formatting of this document, and agree to the content that is located within this SRS. Each member of RCD may have minor disagreements with certain parts of this document, though by signing below, we agree that there are not any major points of contention within this SRS. We have all agreed to these terms and placed our signatures below.

Name:	Signature:	Date:
Matthew Brown Comments:	_____	__/__/__
Gunnar Eastman Comments:	_____	__/__/__
Jesiah Harris Comments:	_____	__/__/__
Shea Keegan Comments:	_____	__/__/__
Eli Story Comments:	_____	__/__/__

Appendix A-3

This appendix will outline the approximate contributions of each of the team members of RCD to the completion of this SRS.

Matthew Brown

- Formatted the document.
- Wrote all that was required from the template except the functional and nonfunctional requirements.
- Conducted review for RCD team on whole SRS.
- Attended client approval meeting.
- Edited the entire SRS.
- Worked on all sections.
- Contributed about 30% of the work.

Gunnar Eastman

- Created the document
- Co-authored the functional requirements.
- Authored the nonfunctional requirements.
- Conducted initial grammatical review.
- Edited the entire SRS.
- Worked on all sections.
- Contributed about 30% of the work.

Jesiah Harris

- Co-authored the functional requirements
- Attended the client approval meeting.
- Worked on §2
- Contributed about 15% of the work.

Shea Keegan

- Co-authored the functional requirements.
- Worked on §2
- Contributed about 15% of the work.

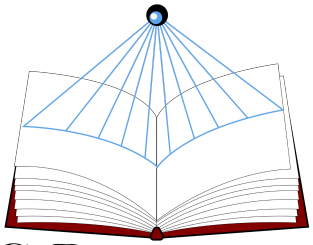
Eli Story

- Co-authored the functional and nonfunctional requirements.
- Constructed the context diagram.
- Worked on §1,2
- Contributed about 10% of the work.

Appendix B - SDD

System Design Document

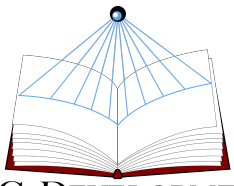
for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



R.C. DEVELOPMENT

Designed by Reading Club Development:
Matthew Brown, Gunnar Eastman, Jesiah Harris, Shea Keegan, Eli Story

Version 1.1
Dec 13, 2022



R.C. DEVELOPMENT

System Design Document:

Table of Contents

Date	Reason for Change	Version
11/9/2022	Initial Creation	1.0
12/13/2022	Updated based on feedback	1.1

1. Introduction

In this section of this System Design Document (SDD), the project will be introduced along with the purpose of the SDD, the references used and compiled by Reading Club Development (RCD), the purpose of the product, and the scope of the product.

This is a capstone project for Dr. Torsten Hahmann and Jake Emerson, in partial fulfillment of the Computer Science BS degree for the University of Maine. This SRS will detail the functional and nonfunctional requirements set forth for this project, the deliverables necessary for completion, and document the consent of the team members and the client.

The field of biology is currently facing a “crisis of reproducibility” according to Jake Emerson, one of our clients and an engineer at Jackson Laboratory in Bar Harbor, Maine. The development of an ontological reader is, for him, paramount due to the congruity of semantics within ontological statements. The CLIF Parser is to be a stepping stone in the progress toward unified Common Logic, allowing for more properly defined variables, and hopefully slightly mitigating this crisis of reproducibility.

On a smaller scale, the purpose of the CLIF Parser is primarily to parse CLIF files to ensure they are syntactically correct according to CLIF standards. The library should also translate from the CLIF syntax to TPTP, or other logic-based conventions. Another purpose of the product is to build upon the previously developed Macleod IDE, so Common Logic can have an IDE dedicated to supporting it.

1.1 Purpose of This Document

This SDD is meant to expand upon the design details for the Common Logic Interchange Format (CLIF) Parser that RCD has been tasked with developing. The intended readership of this document consists of the client and the RCD team so as to effectively develop the proposed CLIF Parser.

1.2 References

Macleod, Dr. Torsten Hahmann, GitHub, 2022, https://github.com/thahmann/macleod.

Brown, M, et al., CLIF Parser System Requirement Specification, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_SDD.pdf.

Colore, Semantic Technologies Laboratory, http://stl.mie.utoronto.ca/colore/.

2. System Architecture

Within this section are several design diagrams meant to illustrate in further detail the inner workings of the system. Many of the diagrams may outline the Macleod system as it currently exists, though this is to ensure that RCD does not deviate far from the current implementation of the system. Differences between the current Macleod implementation and the design diagrams highlight improvements that will be made by RCD, at the behest of the client, and in line with previously outlined functional requirements.

2.1 Architectural Design

The basic design is a Pipe and Filter architecture model, wherein data is passed through a series of steps to an eventual output. In Fig. 1, the data is fed to the Parser, which is the beginning of the Macleod architecture. Specifically, using Macleod as a library allows for the calling of the parser or parse_clif and possibly the translating of the CLIF file into various formats. The parse_clif script generates an ontology object whether the translation is required or not, and the ontology object prints itself to the terminal. If conversion is necessary, then the specific language to be translated into is required, and the ontology object is translated into the relevant language and the converted file is output.

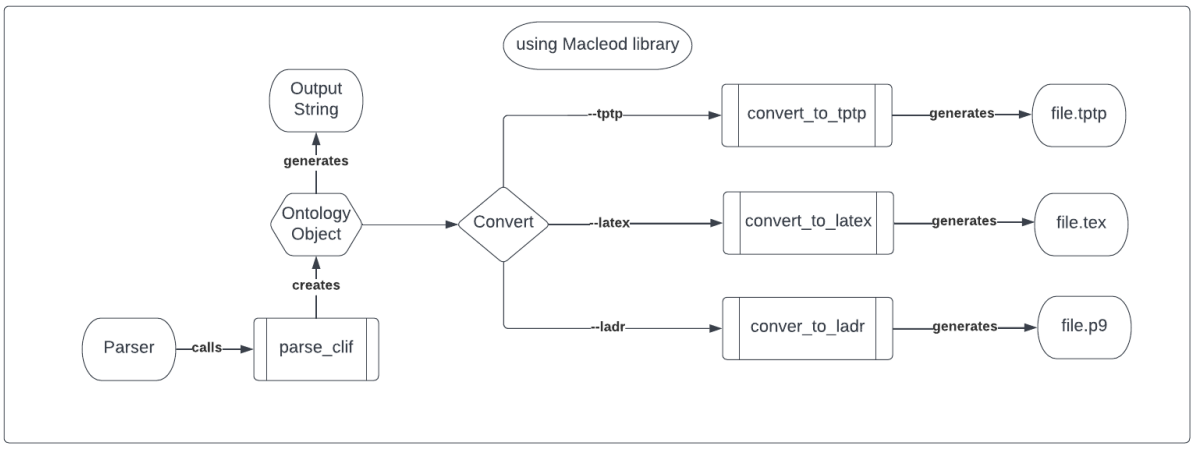


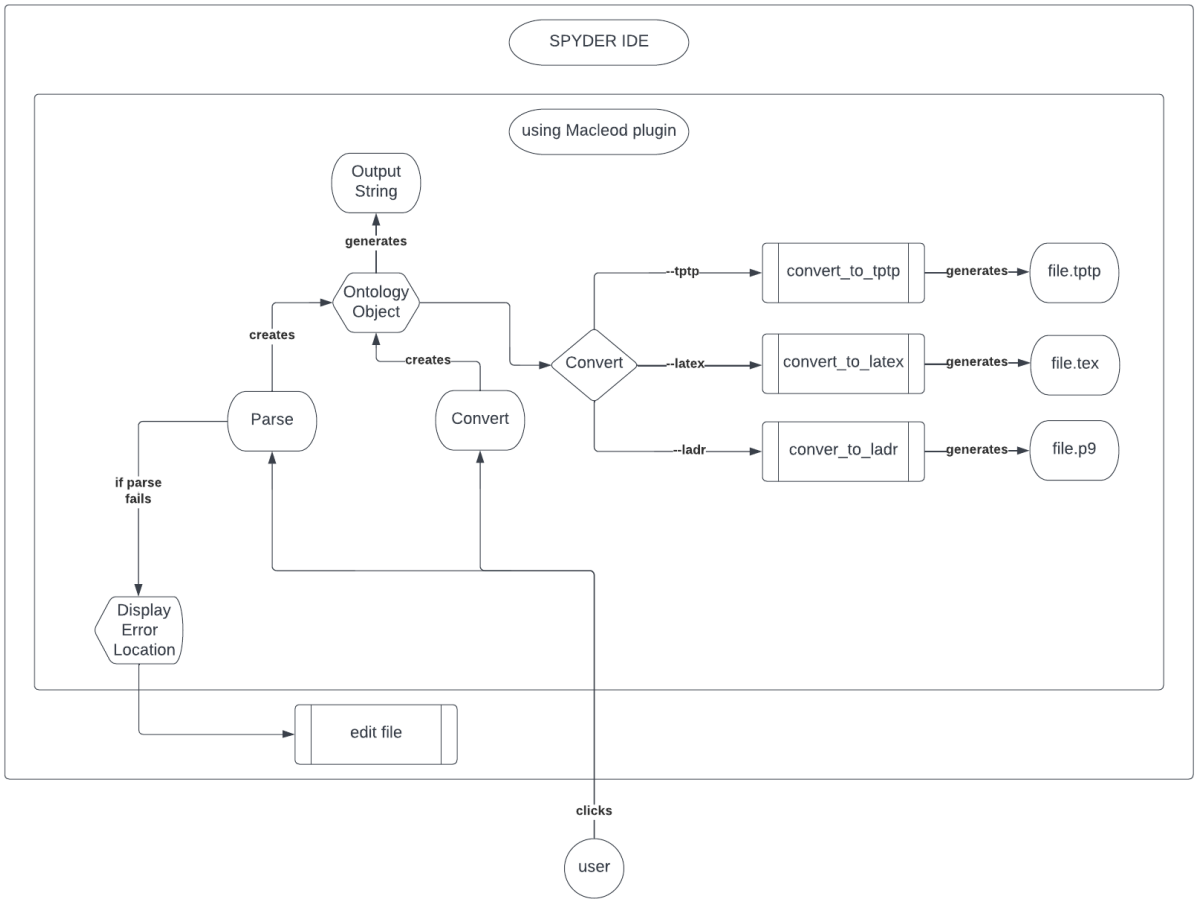
Fig. 1: Data Pipeline of Macleod Library: This diagram documents the flow of data through the Macleod system, from the parser to the eventual output.

Macleod is currently built on Python. All of the relevant scripts from Fig. 1 exist, though the parser is presently incomplete. The work of RCD is twofold on the parser. Firstly is to package up Macleod into a Python library via Poetry that is then downloadable by Pip. Secondly is to ensure completeness of the parser.

Fig. 2 demonstrates how the Macleod plugin will interface with the Spyder IDE. The Macleod plugin will have the same general functionality as the Macleod library. The parse and parse and convert options will be handled via buttons displayed on the IDE. The parse

and convert then leads to a separate menu where the language can be specified. However, if the parse fails at either button press, we will use Spyder's error checking to display where the error is in the CLIF file that caused the parse to fail. The user will then be able to edit the file using Spyder's normal editor.

Fig. 2: IDE



Architecture Diagram of Macleod: This diagram documents how the user input is processed by the system depending on which button the user clicks.

2.2 Decomposition Description

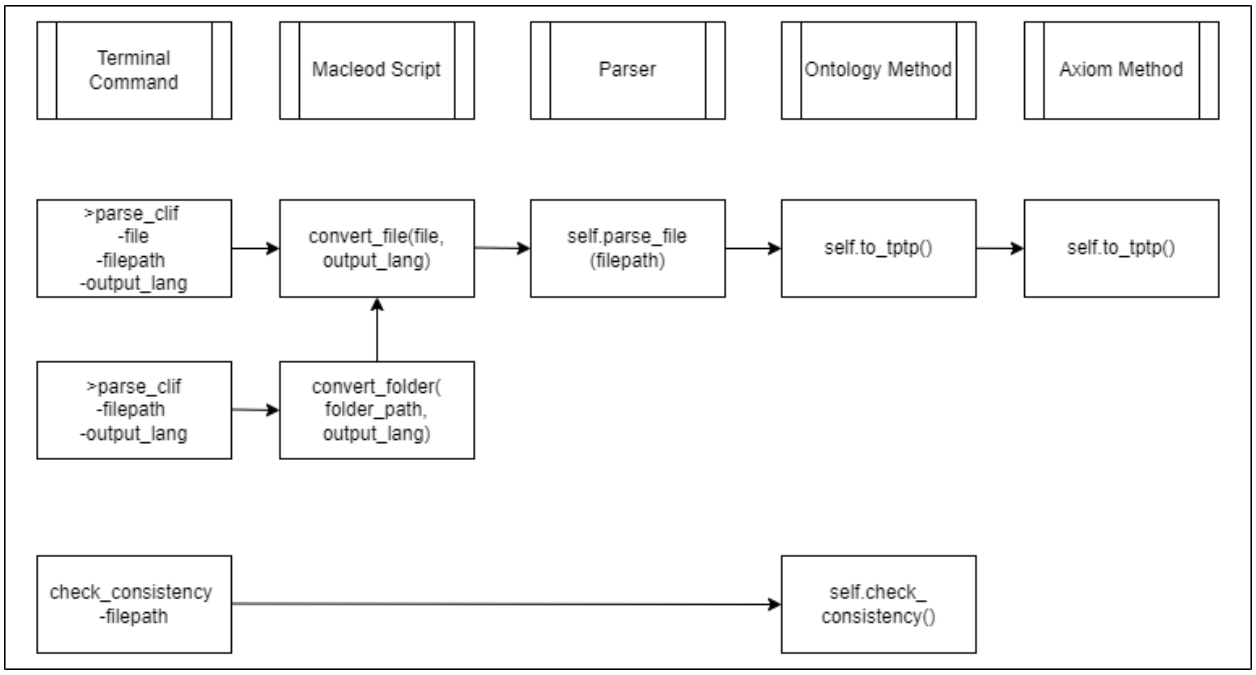


Fig. 3: Outline of Macleod function call order: This diagram describes which functions call which other functions in the parsing and translating processes.

Below are descriptions of the scripts the system makes available to users, the parameters of each of these scripts, and the functions called by each of them.

Scripts:

- parse_clif [file or directory] [filepath] [format] [optional]
    - parse_clif will call either convert_file or convert_folder, which iterates through the files and calls convert_file on each of them.
    - convert_file will create an Ontology object for the file if one does not already exist, and call the parser to read the CLIF file.
    - Once the CLIF file has been read and exists in terms of formula, axiom, and ontology objects, the convert_file function will simply call the proper translation function, according to which format to put the output in, which will be a method of the Ontology object.
    - The translation functions simply iterate through each axiom and have each one translate themselves, adding them to an output at the end.
- check_consistency [filepath] [optional]
    - check_consistency will assemble an ontology object from the input, and then call the check_consistency() method of that object.
    - The check_consistency method simply allows the ontology object to call a theorem prover on itself.

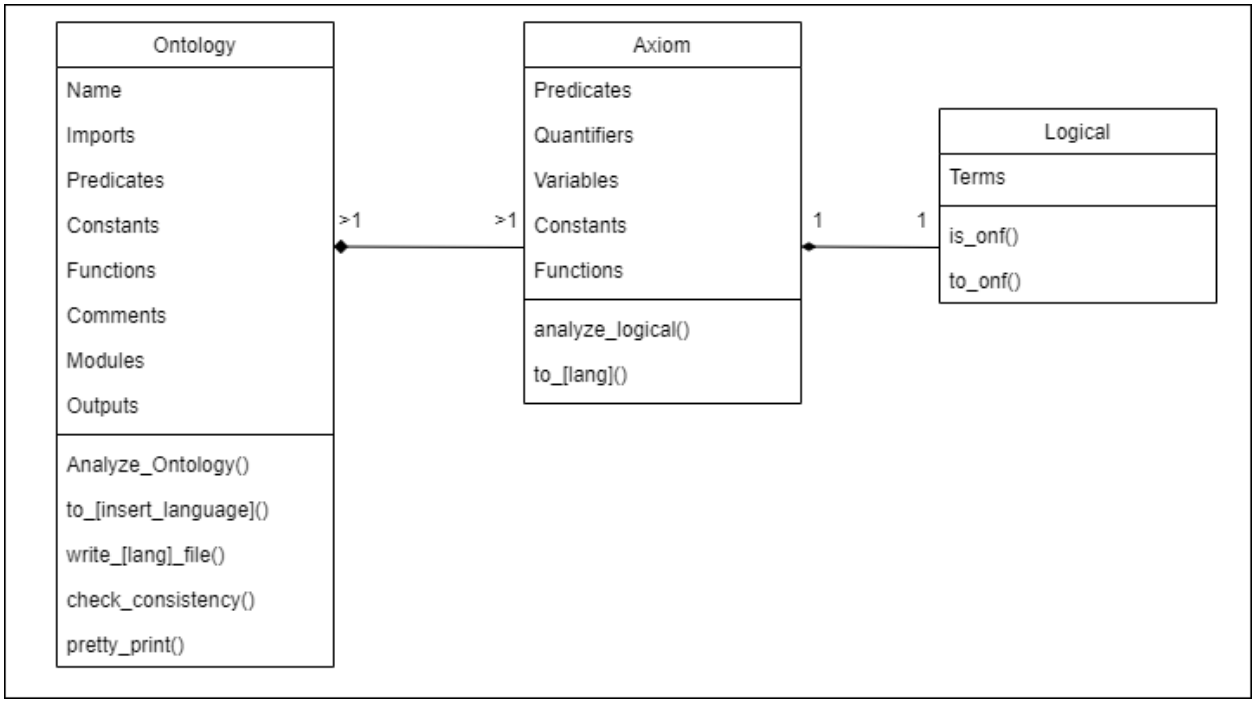


Fig. 4: Class Diagram of Macleod Objects: Describes the fields and methods of the Ontology, Axiom, and Logical Formula classes.

Three main classes are used in the system: Ontologies, Axioms, and Logical statements.

Each formula object contains an array of logical symbols, each represented by an object of a different class, though each of those objects contains very little other than a name. Each formula object is passed in as a field of an axiom object.

The axiom class will make use of its fields to keep far more detailed information about the formulas and what is in it than the formula object itself, and will be able to translate itself into different logical file formats.

Each Ontology object contains any number of these axiom objects, and will maintain a list of predicates, constants, etc. contained within its axioms for convenience, as well as being able to translate itself into different logical file formats, keep those translations should they be needed again, call theorem provers upon itself, and print itself out in the terminal for readability.

3. Persistent Data Design

This section is to display the data that our system will use to run or produce.

3.1 Database Descriptions

Databases are not applicable to the CLIF Parser.

3.2 File Descriptions

The system has some files that are persistent, including logging files and the outputs the system produces. The system will produce a TPTP or vampire output, which can be sent to COLORE, or the Common Logic Ontology Repository. This is done so the system can keep track of what outputs are consistent.

4 Requirements Matrix

In this section each functional requirement of the system is matched with the system component that will satisfy that requirement. The functional requirements are listed by name, number, and use case of each requirement. The system components are listed by either the script that is used in the system or the name of any tools that are accessed by the system.

System Component	Functional Requirement	Number	Use Case
parser.py	Identify Quantified Variables (and their scope and use)	FR-1	The system will identify variables in a given logical statement
parser.py	Identify Predicates	FR-2	The system will identify predicates and function symbols in a inputted logical statement
parser.py	Identify Statements from file	FR-3	The system can read a file and identify logical statements written inside

parser.py	Identify Syntactical Errors	FR-4	The system will identify syntax errors in the input CLIF file and open a debugger so they can be addressed.
clif_to_tptp.py	Take CLIF and output TPTP	FR-5	The system should be able to take a CLIF file as an input and give a TPTP file as an output
clif_to_ladr.py	Take CLIF file and produce LADR output	FR-6	The system should be able to take in a CLIF file and produce LADR output
clif_to_owl.py	Extract OWL approximation from CLIF ontology/module	FR-7	The system should be able to extract an OWL (Web Ontology Language) approximation from a CLIF ontology or module
check_consistency_new.py	Verify the logical consistency of a CLIF ontology or module	FR-8	The system shall be able to verify the logical consistency of a CLIF ontology/module

Theorem provers accessed by system: Prover9 Vampire	Prove theorems that encode intended consequences (e.g. properties of concepts and relations) of ontologies/modules	FR-9	The system should have theorem proving capabilities that can encode the intended consequences of ontologies/modules given to the system. Examples of intended consequences are properties of concepts and relations or competency questions.

Table 1: Requirement Matrix of CLIF Parser: In this matrix, functional requirements are enumerated and the components responsible for completion of the requirements are described.

Appendix B-1

This appendix details the expectations that RCD shall uphold to the client upon completion of this document, and how future changes to this document shall be made.

RCD and the client, upon the signing of the document, are agreeing that this SDD contains a compilation of the architecture necessary for the CLIF Parser. RCD and the client agree that this architecture is to be developed over the course of the Fall 2022 and Spring 2023 University of Maine semesters. The team, RCD, agrees that this architecture is meant to be agile and flexible in nature, so if the need arises, the requirements may change in accordance with the client's wishes.

Any changes made to this document must be approved by all members of RCD and the client via signatures to an additional appendix wherein the changes are enumerated and detailed. Changes to this document include, but are not limited to, the shifting of architectural design as to be in accordance with the Capstone requirements for the University of Maine course this project is managed through. The signing of this appendix consents all members of RCD and the client that this structure of implementing changes is acceptable.

Name:	Signature:	Date:
Torsten Hahmann	_____	__/__/__
Jake Emerson	_____	__/__/__
Matthew Brown	_____	__/__/__
Gunnar Eastman	_____	__/__/__
Jesiah Harris	_____	__/__/__
Shea Keegan	_____	__/__/__
Eli Story	_____	__/__/__

Customer Comments:

Appendix B-2

This appendix will contain the agreement that all members of RCD have read and consent to the document in its entirety.

Through signing this appendix, we, as the members of RCD, agree that we have reviewed this document fully, we agree to the formatting of this document, and agree to the content that is located within this SDD. Each member of RCD may have minor disagreements with certain parts of this document, though by signing below, we agree that there are not any major points of contention within this SRS. We have all agreed to these terms and placed our signatures below.

Name:	Signature:	Date:
Matthew Brown Comments:	_____	11/09/22
Gunnar Eastman Comments:	_____	__/__/__
Jesiah Harris Comments:	_____	__/__/__
Shea Keegan Comments:	_____	__/__/__
Eli Story Comments:	_____	__/__/__

Appendix B-3

This appendix will outline the approximate contributions of each of the team members of RCD to the completion of this SDD.

Matthew Brown

- Created the Architecture Design Diagram and wrote the description
- Formatted the document
- Worked on all sections
- Contributed 32.5% of the document

Gunnar Eastman

- Created the document
- Created the Decomposition Diagram and wrote the description
- Worked on the entire document
- Contributed 32.5% of the document

Jesiah Harris

- Wrote the requirement matrix
- Worked on §4
- Contributed 20% of the document

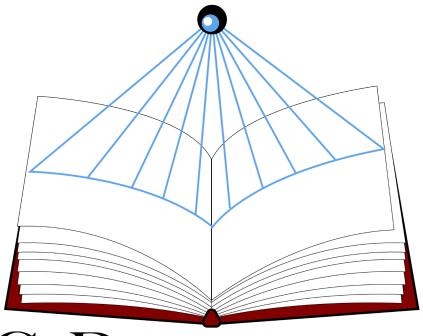
Shea Keegan

- Wrote the Persistent Data Description
- Worked on §3
- Contributed 10% of the document

Eli Story

- Sister team review
- Formatted the document
- Contributed 5% of the document

Appendix C - UIDD



R.C. DEVELOPMENT

User Interface Design Document
CLIF Parser, designed for Torsten Hahmann and Jake Emerson

Designed by Reading Club Development:
Matthew Brown, Gunnar Eastman, Jesiah Harris, Shea Keegan, Eli Story

Version 1.0
Nov. 29, 2022

Table of Contents

1. Introduction

In this section of this User Interface Design Document (UIDD), the project will be introduced along with the purpose of the UIDD, the references used and compiled by Reading Club Development (RCD), the purpose of the product, and the scope of the product.

This is a capstone project for Dr. Torsten Hahmann and Jake Emerson, in partial fulfillment of the Computer Science BS degree for the University of Maine. This UIDD will detail the user interface that will be produced to interact with the product and document the consent of the team members and the client.

1.1 Purpose of This Document

This UIDD is meant to expand upon the user interface details for the Common Logic Interchange Format (CLIF) Parser that RCD has been tasked with developing. The intended readership of this document consists of the client and the RCD team so as to effectively develop the proposed CLIF Parser.

1.2 References

Macleod, Dr. Torsten Hahmann, GitHub, 2022, https://github.com/thahmann/macleod.

SDD, Reading Club Development, 2022. https://docs.google.com/document/d/1H77pVpVR7mhu8ciDFAjY1WjnKQC3SNy9Po1MRIDwRcE/edit

SRS, Reading Club Development, 2022. https://docs.google.com/document/d/1vr3CD5g3azf6OBhu2w5QEy9lcAv1LxDzvFqTvZ7-CwI/edit

2. User Interface Standards

This section will outline the standards used when developing the user interface, to ensure consistency. It will describe what information will be available to the user, in which windows, and how users will interact with the system. This section will also outline how errors will be presented to the user.

The user interface will be composed of 5 sections: file navigation, file editor, console, error

list, and toolbar. Each section will be available to the user at all times while using the program even if a certain section is not being utilized at a given time.

2.1 File Navigation

On the left side of the screen there will be a pane the height of the program window that will contain the file navigation hierarchy. The user will be able to navigate through the hierarchy by clicking a folder to see its contents. Clicking on a file will open that file in the file editor window so long as the file type is supported. Consistency checks will be run on a file as it is being opened.

2.2 File Editor

In the middle of the program window, there will be a pane that will display a file that was chosen from the file navigation pane. In this editor pane, the user will be able to make edits to the file directly. If Macleod encounters a syntactical error while parsing a CLIF file, it will open that file in this pane so that the error may be fixed.

2.3 Console

At the bottom of the program window, there will be a pane that will contain a console. This console will be a place where the logs that may be printed during the runtime of a program will be displayed.

2.4 Error List

On the left side of the program window there will be a pane that contains a list of parsing errors that the program occurred during run time. This list will allow the user to quickly navigate to the location in the file where the error occurred. Each error will list the line it was encountered on and any other information Macleod was able to discern.

2.5 Toolbar

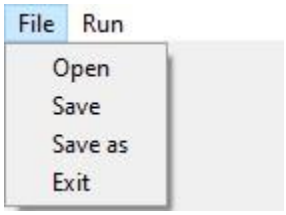
At the top of the program window, running the length of the program window, there will be a toolbar. In this toolbar the user will be able to select the language they would like the file currently open in the file editing pane to be parsed in. There will also be the option to run the parser on said file. In the toolbar there will be a button which will allow the user to save the edits made to the file that is actively open in the file editor. There will also be a button

present to run consistency checks on a currently open file. The toolbar will also be a place where the other actions will be placed as development continues and user actions are created.

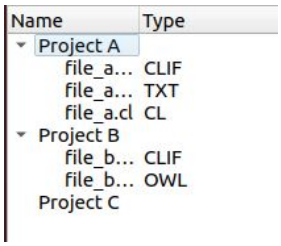
3. User Interface Walkthrough

Present in this section is a navigation diagram, illustrating how, from beginning to end, a user would translate a CLIF file using the system's user interface. Afterwards, all other windows will be illustrated and described.

3.1 Navigation Diagram



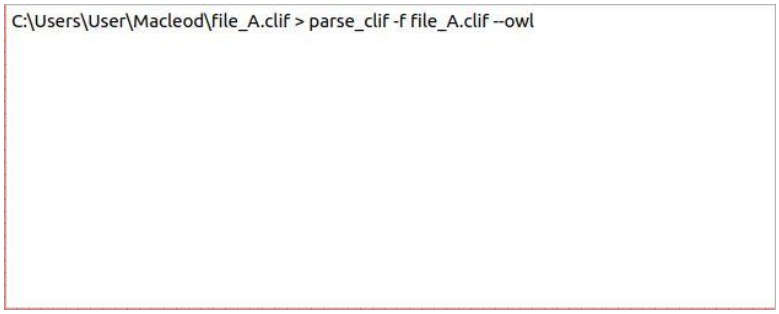
Users will be able to begin a project and open a CLIF file into the editor through the "File" tool in the toolbar. Once a project has been opened, the project will be displayed in a tree-like structure in the project window on the left of the editor.



CLIF files that have been opened by a user will be available for editing and debugging in the text editor as is pictured here:

```
/*******************************************************************************
 * Copyright (c) University of Toronto and others. All rights reserved.
 * The content of this file is licensed under the Creative Commons Attribution-
 * ShareAlike 4.0 Unported license. The legal text of this license can be
 * found at http://creativecommons.org/licenses/by-sa/4.0/legalcode.
 *
 * Contributors:
 *   Torsten Hahmann - initial implementation
 *******************************************************************************/

(cl:text

(cl:ttl http://colore.oor.net/mereotopology/rcc_basic.clif)

(cl:comment 'Basic axioms of the Region Connection Calculus that allows finite models (RCC-4b and RCC8 removed)')

(cl:comment 'RCC-P: Parthood')

(forall (x y)
        (iff
                (p x y)
                (forall (z)
                        (if
                                (c z x)
                                (c z y) )))
)


(cl:comment 'RCC-PP: Proper Parthood')

(forall (x y)
        (iff
                (pp x y)
                (and
                        (p x y)
                        (not (p y x)))))


(cl:comment 'RCC-O: Overlap')

(forall (x y)
        (iff
                (o x y)
                (exists (z)
                        (and
                                (p z x)
                                (p z y) )))
)


(cl:comment 'RCC-EC: External connection')

(forall (x y)
        (iff
                (ec x y)
                (and
                        (c x y)
                        (not (o x y)) ))
)


(cl:comment 'RCC-NTPP: Non-tangential parthood')

(forall (x y)
        (iff
                (ntpp x y)
                (and
                        (pp x y)
                        (not (exists (z)
```

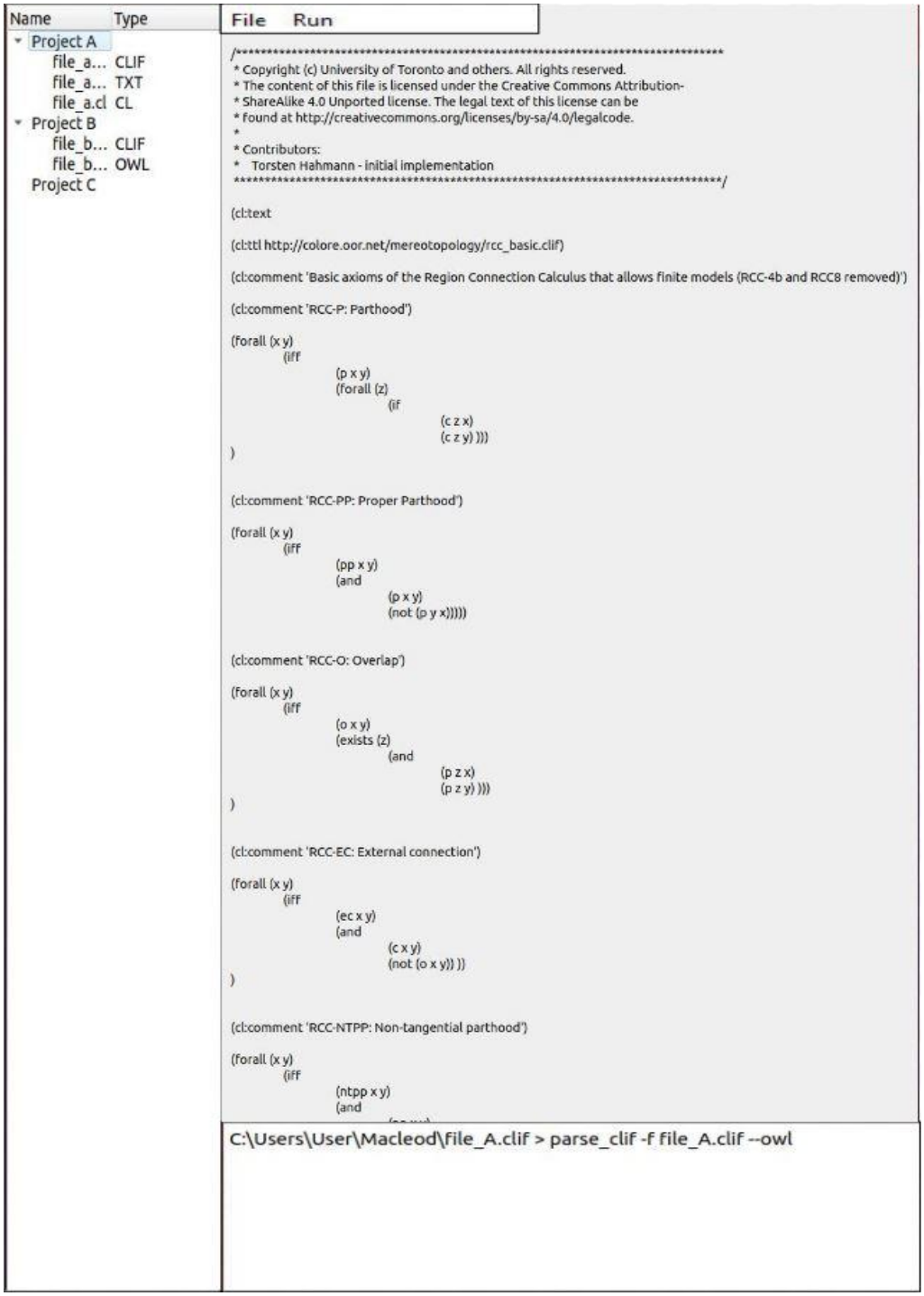
Users will be able to run various commands such as parse, translate, etc through both the “Run” function on the toolbar (to be expanded upon later in development) as well as by running Macleod scripts/commands in the console.



The user interface will have the option for users to use the console to run scripts and view output, and a second tab will be available to view an error list that contains a list of parsing errors that the program occurred during run time.

3.2 Remaining Windows

An example of the combined user interface can be seen below:



4. Data Validation

This section will detail what types of data Macleod will and will not accept as input.

4.1 Input via GUI

The File Editor will be able to open text-like files, including .clif, .cl, and .txt.

When using the GUI to parse a file, the GUI will request a filepath, which will be a plaintext string that must end in either a slash, to indicate a folder, or ".clif", to indicate an individual file. In the case of a CLIF file, Macleod will parse that file and all files it imports, though in the case of a folder, Macleod will parse all CLIF files in that folder.

The GUI will ask for an output filename, though if none is provided, it shall make its own based on the input filename. The output filename must be a string that does not end in a slash or a file extension, as Macleod will automatically add the file extension to the end of the filename. If no output name is provided, Macleod will create a folder called "conversions" in the same folder as the input and put the output into the conversions folder.

The GUI will also provide selectable options to translate the file into TPTP, OWL, LaTeX, and LADR formats. These options will not be mutually exclusive, should a user wish to translate a file to all of the previously listed formats.

Finally, the GUI will provide an option to include axioms in the import closure of the CLIF file. This will simply be a checkbox which will be checked by default where, if checked, the parsing will be performed as though the –resolve parameter was included. The user may uncheck it if they wish to not include the –resolve parameter.

4.2 Input via Terminal

Given that using Macleod via the terminal mostly works at the time of writing, it will remain largely unchanged.

The parse_clif command will still be the main function used to translate CLIF files, and will require only a filename, either ending in .clif to signify a single CLIF file, or in a slash, to signify a folder. Ideally we will be able to remove the -f parameter and be able to detect the .clif at the end of the filename to identify an input of one file.

Optional parameters for this command will be a language parameter: either –tptp, –owl, or –ladr, to signify the format to translate the CLIF file into. If the language parameter is not provided, Macleod will simply parse and print the CLIF file as it currently does.

The final optional parameter will be the –resolve parameter, whose functionality will remain unchanged.

Appendix C-1

This appendix details the expectations that RCD shall uphold to the client upon completion of this document, and how future changes to this document shall be made.

RCD and the client, upon the signing of the document, are agreeing that this UIDD contains a compilation of the user interface (UI) necessary for the CLIF Parser. RCD and the client agree that this UI is to be developed over the course of the Fall 2022 and Spring 2023 University of Maine semesters. The team, RCD, agrees that this UI is meant to be agile and flexible in nature, so if the need arises, the requirements may change in accordance with the client's wishes.

Any changes made to this document must be approved by all members of RCD and the client via signatures to an additional appendix wherein the changes are enumerated and detailed. Changes to this document include, but are not limited to, the shifting of architectural design as to be in accordance with the Capstone requirements for the University of Maine course this project is managed through. The signing of this appendix consents all members of RCD and the client that this structure of implementing changes is acceptable.

Name:	Signature:	Date:
Torsten Hahmann	_____	__/__/__
Jake Emerson	_____	__/__/__
Matthew Brown	_____	__/__/__
Gunnar Eastman	_____	__/__/__
Jesiah Harris	_____	__/__/__
Shea Keegan	_____	__/__/__
Eli Story	_____	__/__/__

Customer Comments:

Appendix C-2

This appendix will contain the agreement that all members of RCD have read and consent to the document in its entirety.

Through signing this appendix, we, as the members of RCD, agree that we have reviewed this document fully, we agree to the formatting of this document, and agree to the content that is located within this UIDD. Each member of RCD may have minor disagreements with certain parts of this document, though by signing below, we agree that there are not any major points of contention within this SRS. We have all agreed to these terms and placed our signatures below.

Name:	Signature:	Date:
Matthew Brown Comments:	_____	__/__/__
Gunnar Eastman Comments:	_____	__/__/__
Jesiah Harris Comments:	_____	__/__/__
Shea Keegan Comments:	_____	__/__/__
Eli Story Comments:	_____	__/__/__

Appendix C-3

This appendix will outline the approximate contributions of each of the team members of RCD to the completion of this SDD.

Matthew Brown
- Conducted sister team review
- Edited and formatted document
- Contributed 20% of the document

Gunnar Eastman
- Wrote Section 1: Introduction
- Wrote Section 4: Data Validation.
- Wrote the introduction for Section 2: User Interface Standards, edited the rest of the section.
- Contributed 30% of the document

Jesiah Harris
- Wrote the User Interface Walkthrough
- Contributed 22.5% of the document

Shea Keegan
- Wrote the User Interface Standards
- Contributed 5% of the document

Eli Story
- Aided in management
- Wrote Section 2
- Contributed 22.5% of the document