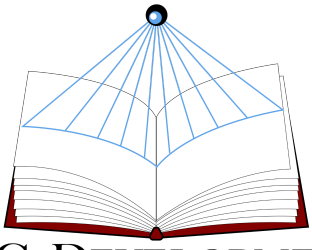
Final Project Report

for the Macleod System, designed for Torsten Hahmann and Jake Emerson



R.C. DEVELOPMENT

Designed by Reading Club Development:
Matthew Brown, Gunnar Eastman, Jesiah Harris, Elijah Story

Version 1.0
May 2, 2023

Executive Summary

The project set forth is to create a Python library, entitled Macleod, from an existing set of scripts, Macleod-Core, which needed to be updated. We were to use the Python Package Index (PyPI) so that Macleod can be installable via pip and be used to parse CLIF files. We were also 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-IDE. 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.

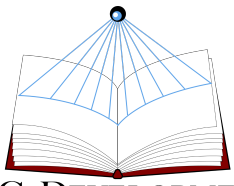
Throughout the 2022-2023 academic year, RCD has updated Macleod-Core to the common CLIF format, expanded the functionality of Macleod-Core to more accurately convert CLIF files into other logical formats, allowed for Macleod-Core to be downloadable and usable from within Python files via a pip library, and created a UI for Macleod-IDE which lacks functionality but is largely demonstrative of the next steps in the Macleod system's development.

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



R.C. DEVELOPMENT

Final Project Report:

Table of Contents

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

Introduction

The Macleod System 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 four seniors from the University of Maine: Matthew Brown, Gunnar Eastman, Jesiah Harris, and Elijah Story.

RCD's efforts break down into three separate parts: Macleod-Core, Macleod-IDE, and Macleod. Macleod-Core consists of prewritten python scripts which take CLIF files, parse them, and convert them to various logical syntaxes. Our task with Macleod-Core was to update and expand the functionality of these scripts to parse and translate more information, and also to update these scripts to the modern CLIF format. Macleod-IDE is built as a plugin for the Spyder IDE, and includes functionality to edit, parse, or convert CLIF files. Macleod is the python library implementation of Macleod-Core which is installable via pip and importable into Python files for easier parsing and converting.

Problem Statement

The field of biology is currently facing a "crisis of reproducibility" according to Jake Emerson. The development of an ontological reader is, for him, paramount due to the congruity of semantics within ontological statements. The CLIF (Common Logic Interchange Format) 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.

Requirements

In this section, the requirements for Macleod are listed along with their completion status and their priorities. We completed eight out of thirteen functional requirements (FR) and seven out of nine non-functional requirements (NFR), though one NFR was actually not required. We derived our requirements through a meeting with our client wherein we elicited as much information as possible, and through our SRS, which specified requirements and the clients agreed upon.

Functional Requirements

Functional requirements are the feature requirements that were the goal for the end product of the Macleod system.

Complete Functional Requirements

FR-2 The system shall identify Predicates.
- Priority 5

FR-3 The system shall identify Statements from file.
- Priority 5

FR-4 The system shall take CLIF and output TPTP.
- Priority 5

FR-5 The system shall take CLIF file and produce LADR output.
- Priority 4

FR-7 The system shall verify the logical consistency of a CLIF ontology or module.
- Priority 5

FR-8 The system shall prove theorems that encode intended consequences (e.g. properties of concepts and relations) of ontologies/modules.
- Priority 3

FR-9 The system shall preserve all comments.
- Priority 3

FR-11 The system shall identify Instructions from file.
- Priority 5

Incomplete Functional Requirements

FR-1 The system shall identify Quantified Variables (and their scope and use).
- Priority 5
- This was not already in the existing codebase, as we had expected.

FR-6 The system shall extract OWL approximation from CLIF ontology/module.
- Priority 4
- There were functions that had not been implemented upon our receiving of the codebase, nor were they implementable by RCD.

FR-10 The system shall bring up an editor to fix syntax errors.
- Priority 3
- This ultimately proved to be an extremely difficult task, and one not able to be completed in our time frame.

FR-12 The system shall provide a button to parse a CLIF file.
- Priority 5
- There is a button, though it does not actually connect to the parse function of Macleod-Core and Macleod.

FR-13 The system shall provide a button to parse and convert a file.
- Priority 5
- There is a button, though it does not actually connect to the parse and convert functions of Macleod-Core and Macleod.

Nonfunctional Requirements

Nonfunctional requirements are the non-feature requirements that were necessary for the security, responsiveness, and accuracy of the project.

Complete Nonfunctional Requirements

NFR-1 The system shall work on Linux, Mac, and Windows.

- Priority 5

NFR-2 The system shall be installable via pip as a python library.

- Priority 5

NFR-3 The GUI will respond to all inputs with at least a “loading” message within 2 seconds 90% of the time.

- Priority 4

NFR-4 The system shall correctly translate at least 95% of input content.

- Priority 4

NFR-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.

- Priority 4

NFR-6 The system shall parse a CLIF file within 3 seconds 90% of the time.

- Priority 4

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

- Priority 5

Incomplete Nonfunctional Requirements

NFR-8 The system itself shall not keep a record of any files it translates.

- Priority 2
- The client ultimately wanted to have a connected logger which would keep track of the parsed files and their success or failure of whether they are parsed and converted.

NFR-9 The system shall be installable as a Spyder plugin.

- Priority 5
- There is a Spyder plugin, though it is not functional.

Design

In this section, the design approaches for the library and the parser are discussed, along with the process RCD followed. Also discussed are false starts, the legacy software RCD interacted with, and the false starts that occurred.

Design Process

The goal of the project was to implement test-driven development, where tests are written and understood, then code is created to satisfy the tests. However, the design process ended up being to write as much code as we can to fulfill requirements, then to tack on the tests after the code had already been written.

The concept of prototype-driven development was also utilized heavily during this project, particularly relating to the IDE. RCD ultimately created three prototypes for the Macleod-IDE, ending on a Spyder plugin as the final goal.

Legacy Software

The entirety of Macleod-Core was legacy offered, which RCD had been given in order to expand and update. This software consisted of scripts to parse and translate CLIF into TPTP, LADR, and LaTeX. The issues with the legacy software was that the IDE and visualizer were entirely defunct, the CLIF conventions were outdated, and the breadth of CLIF conventions had not been entirely covered.

Ultimately, RCD utilized these scripts, expanded on them, and updated them to the modern conventions. It was difficult to understand this code, and caused massive issues with implementing CI/CD. RCD did satisfy the clients wishes with this existing codebase, despite the challenges.

False Starts

As was alluded to earlier, there were several false starts. At first, RCD believed that the task was to simply expand the parser's conventions, and allow it to be Pip installable. Both of these assumptions were incorrect.

Expanding and updating the parser's conventions was only a third of the project. There were also major issues in getting the Macleod-Core installed and operational on all of our computers due to major issues with configuration files. RCD did overcome these issues, and ultimately managed to update the scripts, and allow them to be installable via Pip and usable by functions.

The IDE was where the most false starts occurred. At the beginning, it was unclear what the IDE would be doing, as it was believed that the parser was simply to be used in Python files and via the command line. It was soon realized that an IDE was necessary to edit CLIF files and to more easily parse and translate these files. The first prototype was in QT, and laid the groundwork for the general structure of how the IDE should be built. However, the client later suggested that TKinter should be used, and so the second prototype was built in

TKinter. This Tkinter prototype proved that implementing a terminal and accessing files would be an extremely difficult task given the time frame. Upon this realization, the client requested that a plugin to the pre-existing IDE Spyder be created instead. The end design of Macleod-IDE is built within Spyder, has buttons to parse, convert, and edit CLIF files, but this plugin does not have the functionality of Macleod-Core to execute these functions as of yet.

User Interface Design

A Spyder plugin was decided to be the target GUI environment for the Macleod parser after exploring alternative solutions like QT and TKinter. Ultimately a Spyder plugin was decided upon by the client due to Spyder already in use at Jackson Lab and other areas of science where common logic is used.

Spyder is an open source development environment for the Python programming language. Spyder is designed to assist scientists by offering a large selection of plugins that assist with debugging, analysis, and visualization of data. Spyder also allows for custom plugins to be created and shared among the community. Using Spyder to be the GUI for the Macleod parser cut out a large section of grunt work making a development environment from scratch.

To meet the minimum requirements of the plugin, we created a series of buttons that could utilize the core functionality of Macleod. These buttons trigger functionality that alternatively could be run in the command line. The goal here was to simplify the process of loading a file and running functions against those files especially for those not comfortable with the command line.

Additional requirements were to create a window that allowed any errors that may have occurred when parsing a file, to be displayed in a list fashion. These errors could then be clicked directing the user to the location of the error allowing them to edit the file quickly.

Library Design

There was an existing codebase behind Macleod prior to RCD's involvement in the project, which had fallen into a deprecated state. This codebase is largely what composes the structure of the library. The methodology behind the creation of Macleod was three-fold: we had to update Macleod-Core to be callable via functions, update Macleod-Core to parse the modern CLIF syntax, and handle any errors that came up along the way

Macleod-Core was written as a series of command line scripts. Therefore, when it came time to turn them into a library that could be importable into a python file, it was mostly a matter of turning the command line scripts into callable functions.

We also had the task of updating the underlying functions to accurately reflect the current CLIF syntax, which encompassed both expanding and repairing the existing codebase. There were various conventions, such as comments, that were changed in how they should be represented in CLIF. We also expanded the parser and translators to accept the modern CLIF conventions.

DevOps

In this section we will go over the DevOps of the project. We will cover what the original goal was and what the final result ended up being.

Tools Used

At the beginning of the project it was decided that a certain number of tools would be utilized over the course of the project. To ensure our code was consistent during development, we wanted to use Flake8 and Black across the codebase. For testing the UI elements of the project, we planned to use Selenium. For unit testing, we had decided on unittest since it was already used in the project. To house the code, GitHub was chosen since the legacy code we would be using was already in a GitHub repository. GitHub actions were also selected to be used to automate a number of tasks which will be covered later on.

Due to some of the setbacks mentioned previously, we were unable to utilize all the tools that we originally intended to use. Because of the legacy codebase, we were unable to satisfy Flake8 with the code without a major overhaul. We decided against this since during our CIR, we thought that the code was very clean and understandable from a programming standpoint. We were successful with getting Black to run on the code. Because of the setbacks with developing the UI, we were not able to create tests using Selenium since there was not enough time or content to create such a test. The unittest library was used when creating unit tests and maintaining the existing tests. Finally we were able to utilize GitHub actions to automate just a few of the tasks we had planned. We were able to get Black to be run across the code base on a pull request which we will talk about later on. We wanted the unit tests to be run automatically as well, but again due to the legacy codebase, there were some issues with configuration files causing Macleod-Core from being recognized by the action. We had planned to automate the push of a new pip library version, but decided not to follow through with that since the automation of the tests was

not successful. We did not want to make public untested code so this ensures that manual tests will be considered before a version release.

Testing Plan

As mentioned before, we had planned to create tests then develop code to meet the test requirements. In reality we simply modified the existing tests to work with the new Macleod-Core changes we implemented. The modified tests proved sufficient for ensuring that the core changes to the code did not break existing functionality.

Agile Methods Adopted

For this project we adapted a few agile methodologies. We held standups twice a week at a minimum where we discussed what was accomplished and what needed to be done in the upcoming days. We also held scrum meetings at the beginning of two week sprints where we assigned tasks and updated the requirements. Using the agile methodology was crucial to our development since it allowed us to adapt to the constantly changing requirements and modes of development that have been mentioned previously.

Deliverables

Throughout the development of the Macleod system, we were to submit deliverables to document the practices that we have taken. These deliverables are briefly detailed within this section.

SRS

The System Requirements Specification (SRS) document was created to present and explain the functional and nonfunctional requirements for Macleod. The SRS also describes the development tasks that were undertaken by RCD, as well as the initial goals the team worked towards during development. The purpose and scope of Macleod can be found in the SRS, as well as open issues that were inherited with the project. The entire SRS is available for viewing in Appendix A of this document.

SDD

The System Design Document (SDD) provides an in-depth explanation of the design details for the development of Macleod. This document was intended for use by the client and RCD to provide insight into the system design planning and process. The SDD contains system architecture diagrams and a requirements matrix which maps system components to their relevant functional requirements and use cases. The SDD is available for viewing in Appendix B of this document.

UIDD

The User Interface Design Document (UIDD) goes into detail about Macleod's user interface and its development standards. The user interface standards provide understanding of the guidelines used in creating the user interface, how the user interface is presented to users, as well as how graphical errors are presented to users. The UIDD is provided in this document as Appendix C.

CDRD

The Critical Design Review Document (CDRD) contains an executive summary which defines Macleod, the goals of the project, and the purpose of the software. The CDRD further examines the design, development, and testing processes undertaken by RCD while developing Macleod. All development plans, lessons, and results that were underway at the halfway point of development are also enumerated in the CDRD. The CDRD can be found in its entirety as Appendix D of this document.

CIR

The Code Inspection Review document (CIR) enumerates the details of the code inspection meetings that were conducted by RCD during development of Macleod. The CIR acts as both a quality assurance document for RCD as well as a detailed recounting of the code inspection process. The CIR also contains an exhaustive list of defects documented during code inspection. The CIR is available for viewing in Appendix E of this document.

AM

The Administrator Manual (AM) is a document designed to provide detailed instructions for the installation and management of the Macleod system. The AM contains installation instructions for all three components of Macleod: MacleodCore, Macleod, and MacleodIDE. The AM also presents a list of routine tasks to be performed during the administrative management of Macleod. The AM also contains information for user support, as well as troubleshooting instructions for commonly encountered errors and defects. The AM is provided as Appendix F of this document.

UG

The User Guide (UG) supplies steps on how to use the Macleod system (MacleodCore, Macleod, and MacleodIDE), a summary of relevant documentation, and troubleshooting steps for commonly encountered issues. The UG also contains steps for the installation of the Macleod system as well as recovery procedures for issues the user may encounter. The UG is intended for end users, casual users, and anyone else who wants to use Macleod or

extend its features through further development. The UG is provided in this document as Appendix G.

Conclusions

In this section, discussed are our results, how our results relate to the requirements enumerated within §2, what went right and wrong, and future development that can be taken on the Macleod system.

Results

At the end of the two semester capstone project, a Spyder plugin was created and published to the Pip library for user downloads. The plugin consists of three buttons: Edit, Parse, and Translate. These buttons are currently nonfunctional.

- The Edit button allows the user to edit the file at highlighted locations in the file where errors occurred. This makes the process of debugging a file a lot easier and faster for the user allowing them to parse files faster.
- The Parse button takes the selected file and parses it into an Ontology object. From there it can be translated into a language of the user's choosing.
- The Translate button takes the Ontology object and translates it into the language selected by the user. The user can select a language by using the dropdown menu that appears when clicking the Translate button.

For the parser, we accomplished three major tasks: bring the parser to a working state, add the functionality for parsing comments, and adding the parser to the Pip library.

When we received the project code, there were a lot of problems getting the parser to funcion. When it was functioning, it was producing many errors with the parser. The first task was to clean the code base of old dependency and the former UI implementations.

Once the parser was producing desirable results, we moved on with adding the ability to parse comments that are added to the files which is an extremely important part of the files for scientists to understand the file they are translating.

Finally, we turned the parser into a Pip library that could be downloaded easily via pip in the command line. This is a major improvement to the ease and usability of the parser compared to the previous GitHub download process that came with the project

Relation to Requirements

Ultimately, 15 out of 22 requirements were satisfied. Of the seven non-implemented requirements, three requirements (FR-1, FR-6, and FR-10) were not feasible to develop over the two semester development time due to requiring deeper knowledge than was achievable, and one requirement (NFR-8) the client ultimately did not want us to develop.

What Went Right

Truly, RCD is proud of the development we put into this project. We satisfied the vast majority of achievable requirements, and we believe we produced a product that our client would find valuable. We are especially proud of the Pip-installable Python library and the updating of the parsing and converting scripts as these functionalities satisfied every test file we had access to.

We are also very proud of our interactions with our clients. Not only were they instrumental every step of the way, but also they aided us in development, even with Jake pair programming with Gunnar along the way. We believe that we were professional, courteous, and held the interactions with our clients to a very high standard.

Our team interactions are also a point of pride within the group. We were always truthful with one another, and we produced a product that we are satisfied with.

What Went Wrong

Our biggest issues came with the IDE. Connecting the plugin to the existing scripts presented an insurmountable challenge given the false starts that we had with the development of the IDE. We believe it would not take an exceptionally long amount of development time in order to add these functionalities, though.

Future Development

RCD's development work did not reach as extensively as we had wished due to the limited timeframe of this project, and the complexity of the task at hand. The largest requirement which we could not implement despite our best efforts was the ability for Macleod-IDE to, if it fails a parse, allow the user to edit the file directly in Spyder. We would ideally have the error be displayed either in the terminal, or highlighted within the CLIF file where the error occurred, like any code editor would with syntax errors.

There are also extensive extensions that need to be made to the parser. There are two logical standards which it would be helpful for Macleod to translate CLIF files into: Function Free Prenex Conjunctive Normal Form (FFPCNF) and Web Ontology Language

(OWL). The bases to allow for conversion into these other formats do exist within Macleod, but there are functions that must be called in order for conversions to occur that are not written in the Macleod codebase we had access to manipulate. Further development would include parsing the remaining elements of the CLIF syntax, along with implementing how to convert each missing piece of CLIF syntax into the other languages Macleod can convert into.

Macleod will also necessitate maintenance to ensure it does not become obsolete again. The code base will need to be run against the current CLIF standards year on year, likely using COLORE. Primarily, expansions and maintenance are paramount to Macleod's continual usage.

Acknowledgements

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.

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.

References

[1] Brown, M, et al., CLIF Parser System Requirement Specification, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_SRS_Ver_1.3.pdf.

[2] Brown, M, et al., CLIF Parser System Design Document, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_SDD_Ver_1.1.pdf.

[3] 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.

[4] Brown, M, et al., CLIF Parser Critical Design Review Document, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_CDRD.pdf.

[5] Brown, M, et al., CLIF Parser Code Inspection Report, 2023, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_CIR_Ver_1.1.pdf.

[6] Brown, M, et al., CLIF Parser Code Inspection Report, 2023, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_AM.pdf.

[7] Brown, M, et al., CLIF Parser Code Inspection Report, 2023, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_UG.pdf.

[8] M. Brown, et al., *Reading-Club-Development*, Github, 2023, https://github.com/Reading-Club-Development.

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

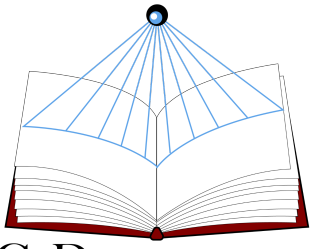
[10] Hahmann, Torsten, *Macleod*, GitHub, 2022, https://github.com/thahmann/macleod.

[11] International Organization for Standardization, *Common Logic (CL): a framework for a family of logic-based languages*, 2018, https://www.iso.org/standard/39175.htm

Appendix A - SRS

System Requirements Specification Document

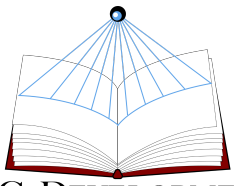
for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



R.C. DEVELOPMENT

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

Version 1.3
Dec. 13, 2022



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.

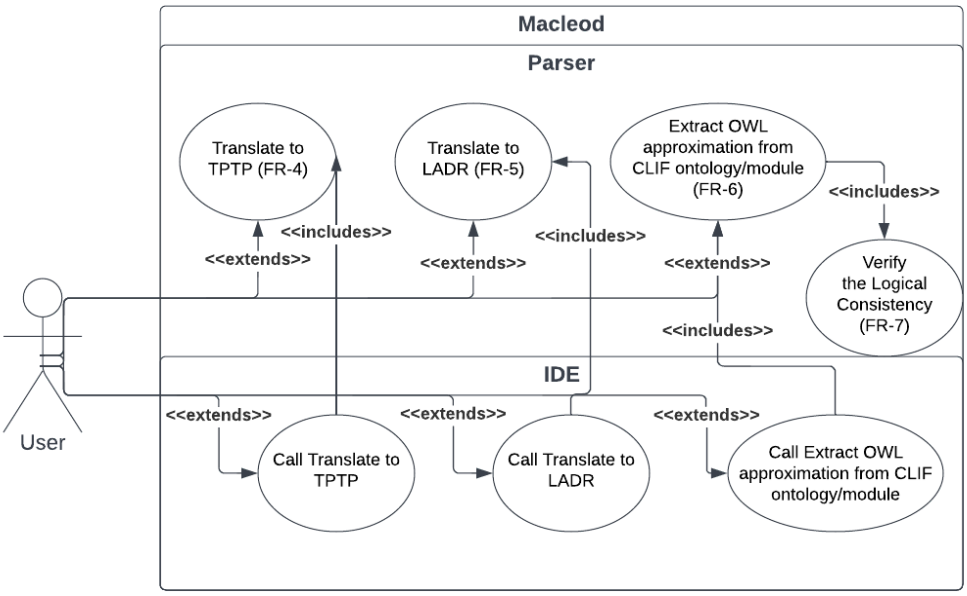


Fig. 1: Primary Use Case Diagram: This diagram depicts how Macleod is to be used as a parser and as an IDE.

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

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 B

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 C

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.

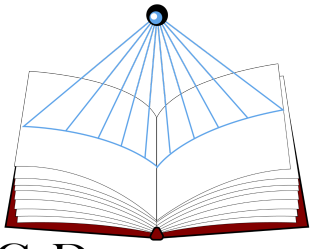
Eli Story

- 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

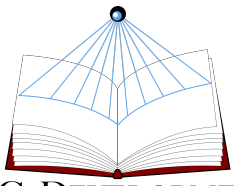
for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



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.

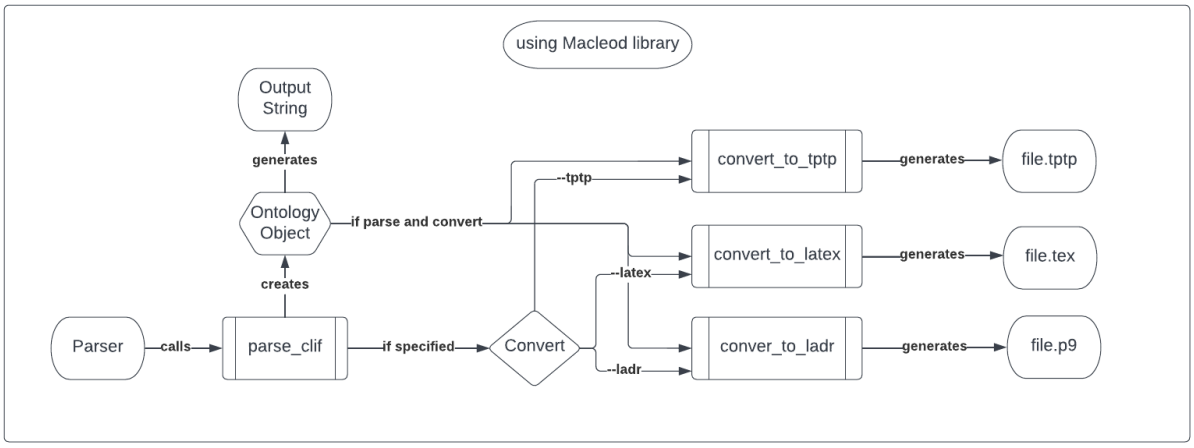


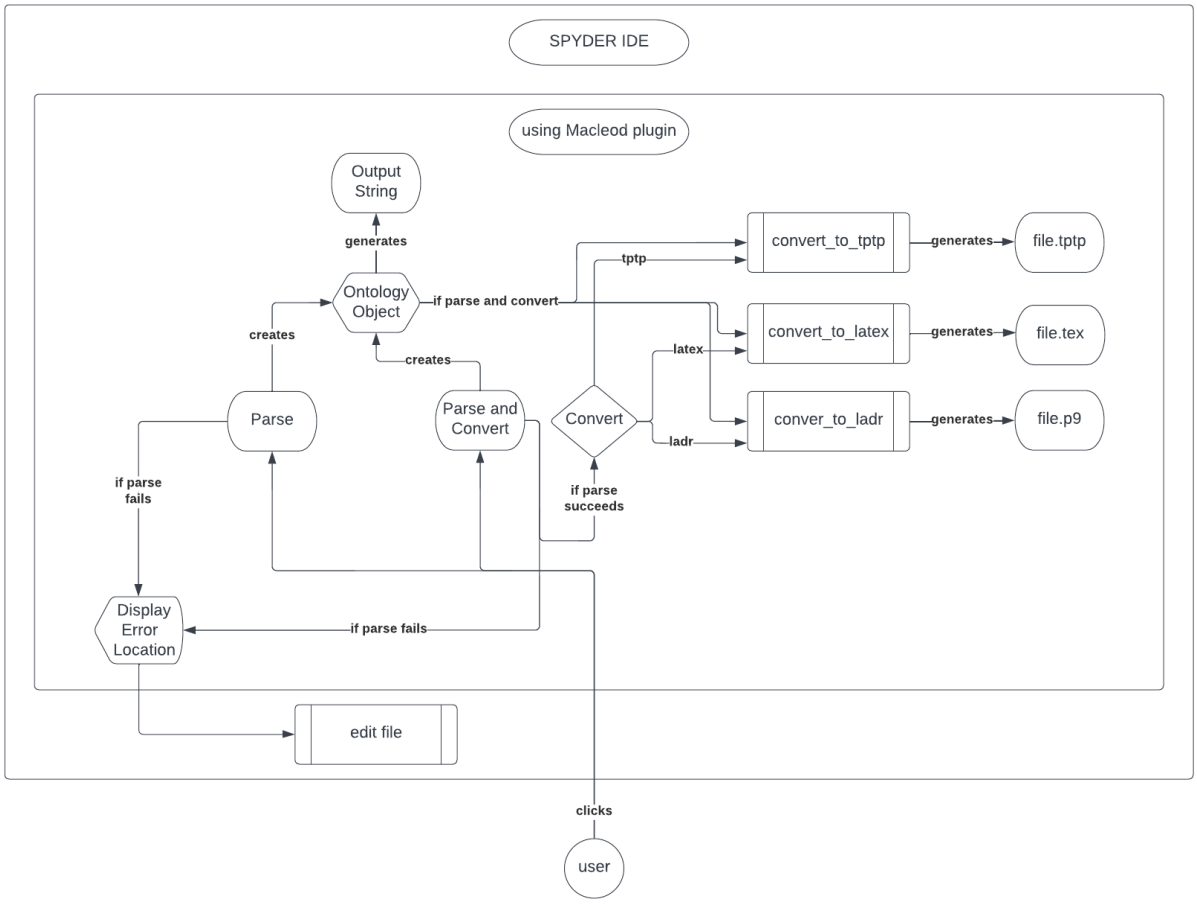
Fig. 1: Data Pipeline of Macleod Library: This diagram documents the flow of data through the Macleod system, from the parser to the eventual output.

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

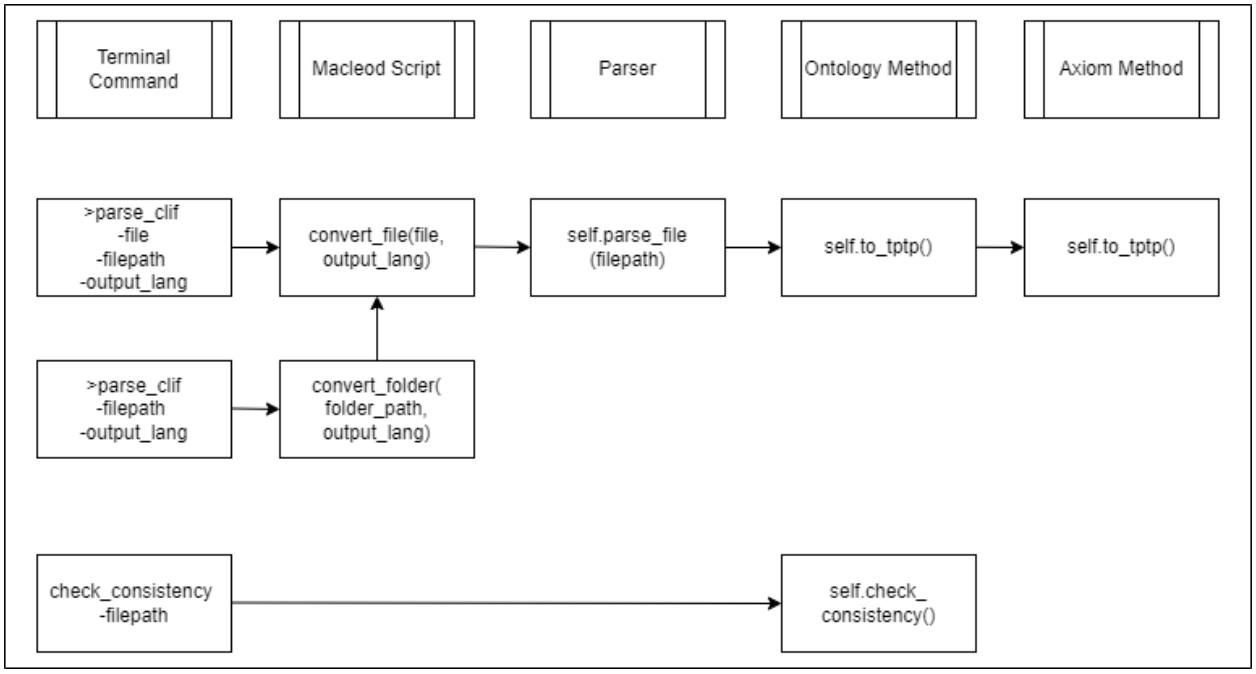


Fig. 3: Outline of Macleod function call order: This diagram describes which functions call which other functions in the parsing and translating processes.

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.

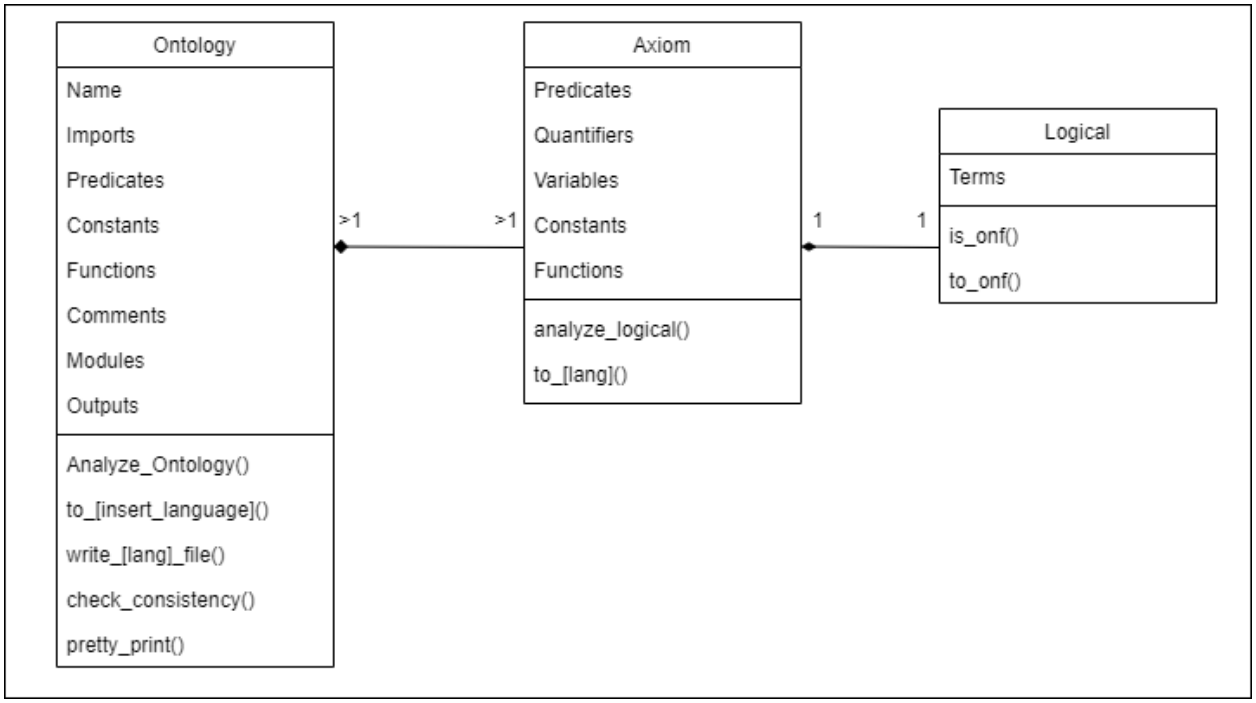


Fig. 4: Class Diagram of Macleod Objects: Describes the fields and methods of the Ontology, Axiom, and Logical Formula classes.

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 A

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

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 C

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

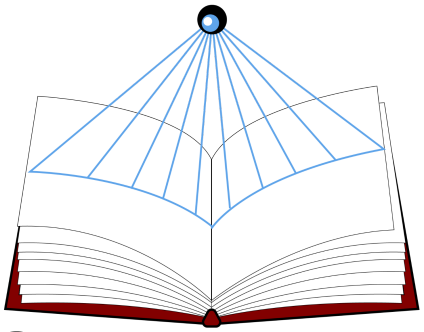
Shea Keegan

- 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

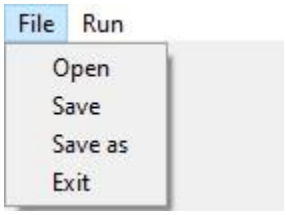
At the top of the program window, running the length of the program window, there will be a toolbar. In this toolbar the user will be able to select the language they would like the file currently open in the file editing pane to be parsed in. There will also be the option to run the parser on said file. 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.

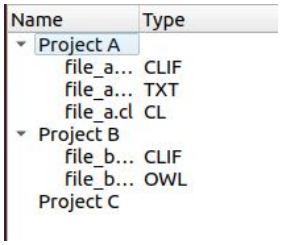
3. User Interface Walkthrough

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



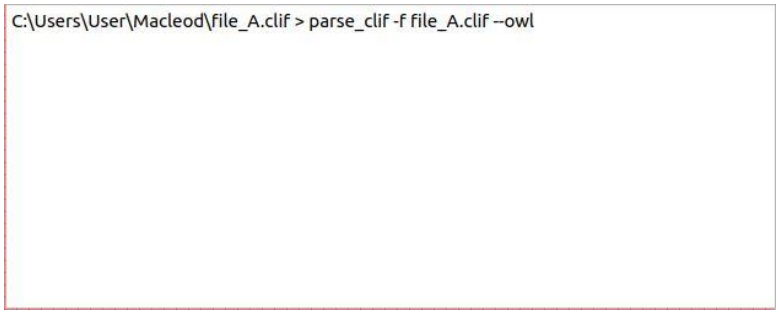
Users will be able to begin a project and open a CLIF file into the editor through the "File" tool in the toolbar. Once a project has been opened, the project will be displayed in a tree-like structure in the project window on the left of the editor.



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)
```

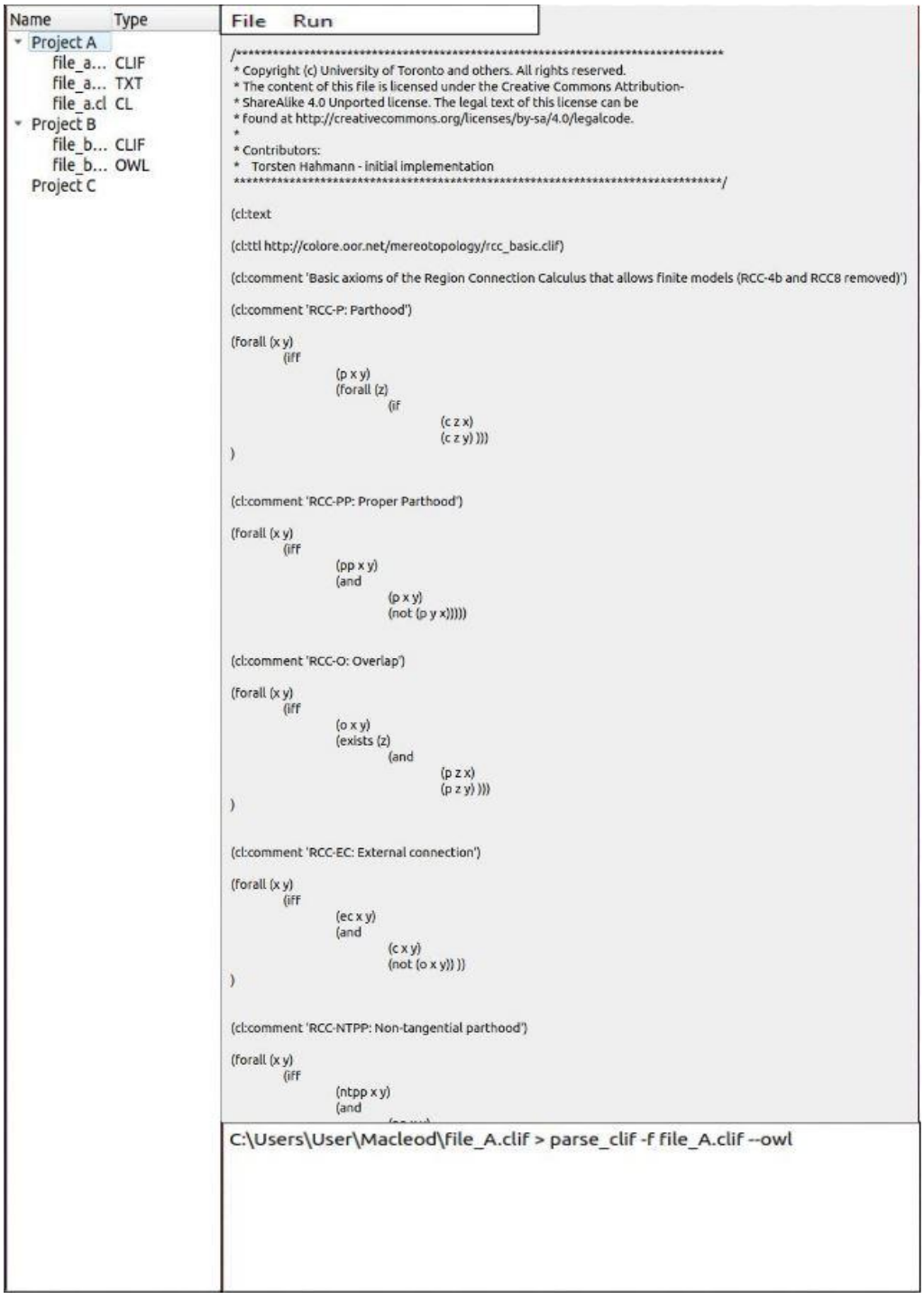
Users will be able to run various commands such as parse, translate, etc through both the “Run” function on the toolbar (to be expanded upon later in development) as well as by running Macleod scripts/commands in the console.



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 A

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		11/29/22
Shea Keegan	_____	__/__/__
Eli Story	_____	__/__/__

Customer Comments:

Appendix B

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:		11/29/22
Shea Keegan Comments:	_____	__/__/__
Eli Story Comments:	_____	__/__/__

Appendix C

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

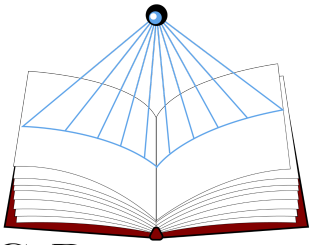
Eli Story

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

Appendix D - CDRD

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

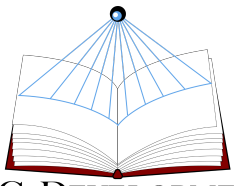
Preface

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.



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.

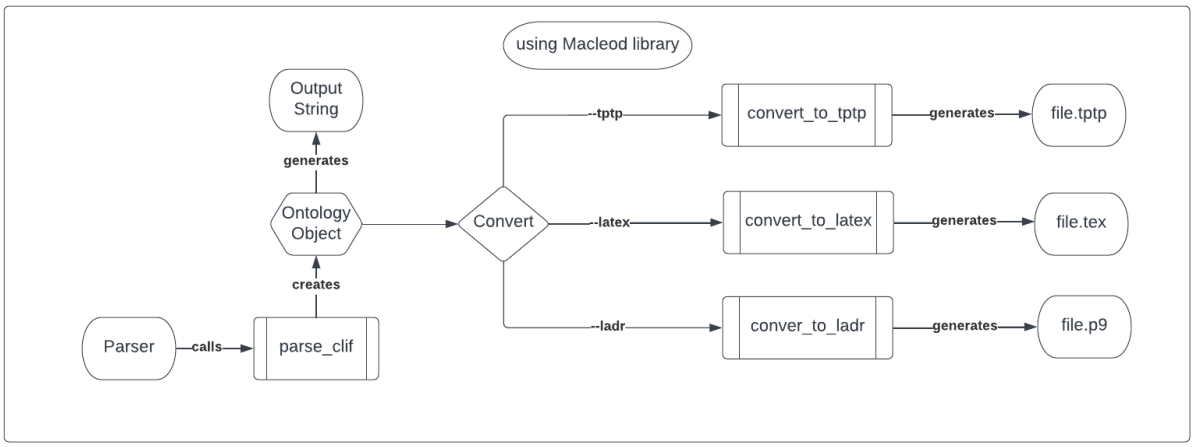


Fig. 1: Data Pipeline of Macleod Library: This diagram documents the flow of data through the Macleod system, from the parser to the eventual output.

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.

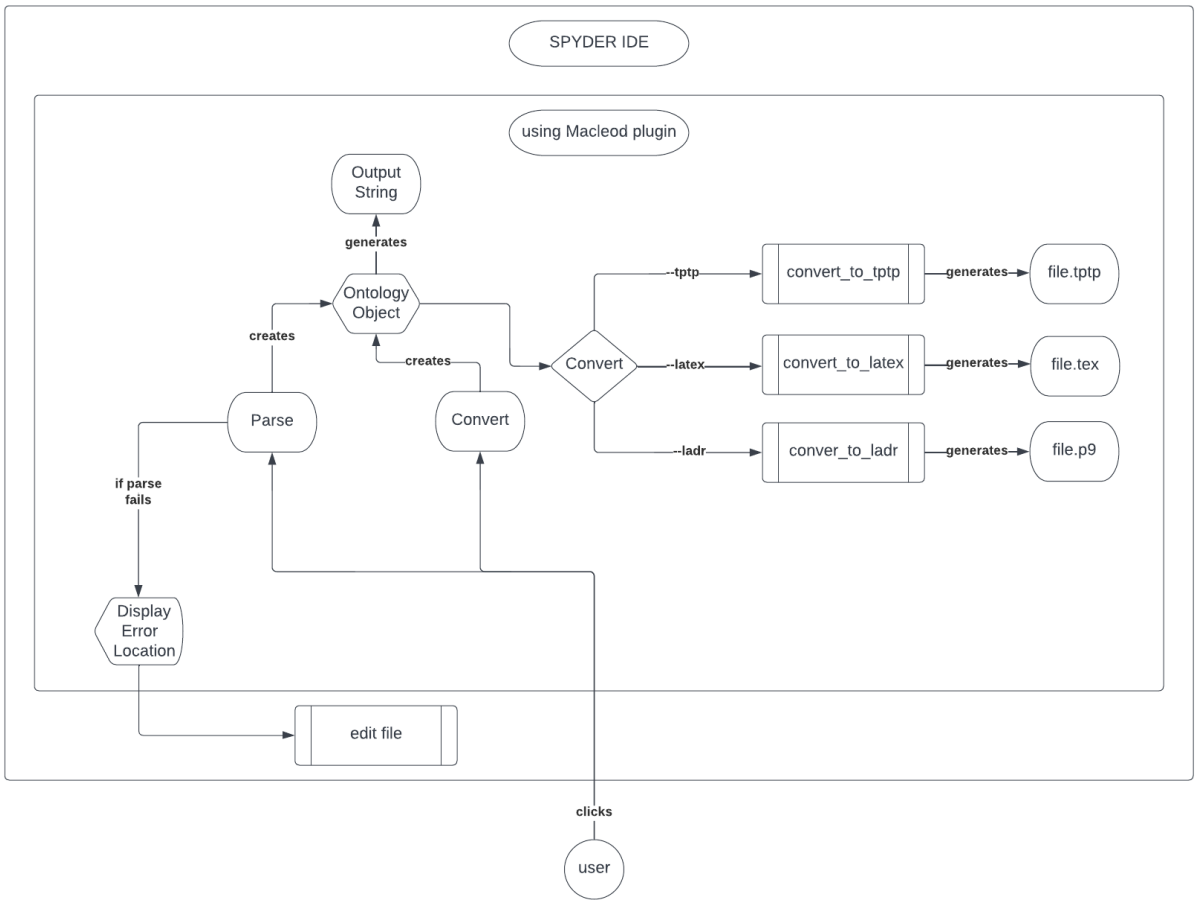


Fig. 2: IDE Architecture Diagram of Macleod: This diagram documents how the user input is processed by the system depending on which button the user clicks.

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.

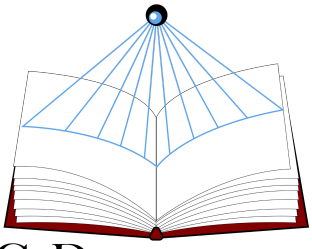
We also want to express our thanks to Dr. Torsten Hahmann for the opportunity to work on this project under his wisdom and tutelage. He is patient with us, and always insightful in his opinions.

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

Appendix E - CIR

Code Inspection Report

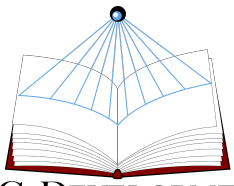
for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



R.C. DEVELOPMENT

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

Version 1.1
Mar 9, 2023



R.C. DEVELOPMENT

Code Inspection Review:

Table of Contents

Date	**Reason for Change**	**Version**
2/23/2023	Initial Creation	1.0
3/9/2023	Updates according to peer review	1.1

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 four seniors from the University of Maine: Matthew Brown, Gunnar Eastman, Jesiah Harris, and Eli Story.

RCD will be responsible for the following products: MacleodCore, pre-existing code that will be used in the terminal; Macleod, code that has been adjusted by RCD that enables the scripts available in MacleodCore to be used as functions in a python library; and MacleodIDE, a plugin for the SpyderIDE that will allow the Macleod functions to have access to a User Interface, for an easy to understand experience.

1.1 Purpose of This Document

The purpose of this Code Inspection Report is to elaborate upon the details of our Code Inspection meetings. Quality Assurance is important to RCD, and this document is being provided in order for others to be able to have confidence in our process.

1.2 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_Ver_1.3.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_Ver_1.1.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.

Brown, M, et al., CLIF Parser Critical Design Review Document, 2022,

https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_CDRD.pdf.

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

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

1.3 Coding and Commenting Conventions

Coding and commenting conventions used within the code written by RCD are based on those that were employed by Dr. Hahmann and Jake Emerson during their development of the original Macleod codebase. Much of the code written by RCD was simply adjusting the pre-existing code so that the scripts written by Dr. Hahmann could be utilized as normal python functions, accepting standard arguments instead of trying to take arguments from the command line. Each function adjusted this way was accompanied by comments explaining both the nature of the function, the return value (if applicable), and descriptions of all arguments. These comments were already present in the code, and were simply moved to a more proper position above the function declaration rather than as an argument being passed to the function that takes information from the command line.

These coding conventions include the formats for names of files, classes, functions, and variables. Names of files within subfolders are intended to be camel case (i.e. camelCase), whereas names of files not within subfolders are intended to be title case, i.e. TitleCase. Names of classes are meant to be title case. Names of functions and variables are meant to be snake case, i.e. snake_case.

1.4 Defect Checklist

Table 1 below explains the different types of errors whose presence was anticipated.

Defect Type #	Defect Type Name	Explanation
00	Variable Name	An RCD member named a new variable improperly
01	Import Confusion	The code tries to import a file that doesn't exist
02	Comment Missing	A function exists and does not have comments explaining what the function does, what the function returns, or what the function requires as arguments.
03	Inadequate	A function exists and has comments, but those

	Comment	comments are incomplete.
04	Function Name	An RCD member named a new function improperly: either formatting wise, or in a way that is not descriptive.
05	Requirement Missing	A library that is listed as a requirement for Macleod was not actually present in the pyproject.toml file.
06	File Name	A file is named not in accordance with other file names.
07	Class Name	An RCD member named a new class improperly
08	Uncertain Return Type	The code calls a function assuming it will give a particular output when multiple output types are possible.
09	Function not Defined	Functions are in use that are labeled by the editor as not being defined.
10	Function Member of None Type	Functions in use labeled by the editor as a member of none type.
11	Typo in String Input requirements	An if statement checks whether the string input is equal to a particular value, though that value is incorrect.
12	Misordering of Inputs	Arguments are passed into a function in the wrong order
13	Incorrect Type	A value was set as the incorrect type, i.e. a string “None” instead of the null value.
14	File type translation error	A member of RCD improperly hard-coded the comment, module declaration, or import statement formatting for TPTP, LADR, LaTeX, or OWL.

2. Code Inspection Process

The purpose of a code inspection is to discover flaws in the codebase. These flaws could be logical errors, design errors, or even style errors. Anything that could cause problems further down the road in the development process. The inspections can happen in a variety of ways involving as few as two people, up to teams of at least seven. In each style of this inspection, the code is walked through and looked at looking for the flaws mentioned

previously. You can also run automated software to assist with tasks such as spell checking, grammar checking, and can even run a linter to assist in the assurance that the code is written in a way that meets style guidelines.

The current codebase is split into several subcategories: there is the code that is written as part of the MacleodIDE plugin, code in the Macleod that is written to provide a python-usable version of MacleodCore, and adjustments made to MacleodCore to facilitate additional functionality as requested by the client. As of 3/9/23, all code has been written, we are simply taking steps to ensure that it passes every test written.

The goals of our code inspections were to ensure not only that the code functions properly, but that it is easy to read and understand, and conforms to all coding conventions and commenting standards.

2.1 Inspection Approach

The first step of the process was to define all the possible types of defects, so that we would know what to look for, and how to describe what we found. We first read through each document that had been edited, namely parserlib.py and check_consistency_lib.py. We noted any defects we found on our own, then enabled a type checker available through VS Code and noted defects that the type checker detected.

We did not follow standard code inspection procedures during our Round Robin Review (RRR) of Macleod and Macleod-Core. Due to the existing complexity of the project, we deemed it best to do both static and dynamic processes when identifying defects within our codebase. The code was run multiple times to identify any path defects between the configuration files, the import system, and the existing file system. Essentially, we walked through the process of importing Macleod using pip and then using the library to parse an existing CLIF file. Every error we encountered we documented.

We did perform static analysis as well. We attempted to get Flake8 running on our Macleod and Macleod-Core, but we were ultimately unsuccessful on that front. We also are using Black for code formatting of Macleod and Macleod-Core, which we have running on a separate branch of our GitHub repository. Black is fully operational on both of these two arms of the project. We also have individually read through the RCD edited code for Macleod, Macleod-Core, and Macleod-IDE in order to visually identify any improper variable, function, or file names, along with picking out any other easily identifiable defects.

Separate from the analysis of Macleod and Macleod-Core was the analysis of Macleod-IDE. During this analysis, we ran the code through a style-checker and we also ran the code to

attempt to identify defects as the code is used. We did ensure that Macleod-IDE was operational and working.

2.2 Impressions of the Process

In general the code inspections were helpful in learning more about the code base and how the different files work together, but for this particular project it was a little more challenging to spot errors or flaws since the members of our team were not the ones that wrote it and knew its inner workings by heart.

One of the things that was not immediately helpful was running Flake8 on the codebase. This resulted in numerous lines that exceeded Flake8's line length max. Once this was overwritten in the setting, it then produced 373 problems with the code such as ambiguous variables, random whitespaces, unused imports and the like. This felt more daunting than helpful. If something like Flake8 were run continuously during development, I could see the benefit, but we did not have that option with this project.

The cleanest of the Macleod modules is by far Macleod-IDE, as Spyder has a strong API that makes it simple and efficient to create plugins, leaving very little room for errors to be made by RCD. In contrast, the edits to Macleod-Core are likely to contain the most errors, as those make up the largest of RCD's efforts.

2.3 Inspection Meetings

Inspection Meeting One:

Date	2/20/23
Location	RCD Discord Server Voice Channel (virtual meeting)
Time started	1:15pm
Time ended	3:15pm
Participants	Matthew Brown, Gunnar Eastman, Eli Story
Roles filled	Author (Gunnar Eastman), Recorder (Gunnar Eastman), Readers (Matthew Brown and Eli Story), Moderator (Matthew Brown)
Code Units Covered:	Macleod (the python library); parserlib.py and check_consistency_lib.py

Inspection Meeting Two:

Date	2/22/23
Location	RCD Discord Server Voice Channel (virtual meeting)
Time started	5:30pm
Time ended	5:45pm
Participants	Jesiah Harris, Gunnar Eastman
Roles filled	Author (Jesiah Harris), Reader (Gunnar Eatman), Recorder (both team members shared the role)
Code Units Covered:	MacleodIDE; container.py, api.py, widgets.py and plugin.py

3. Modules Inspected

As mentioned before, the Macleod project is split into three interconnected modules: Macleod, Macleod-Core, and Macleod-IDE. In this section each module's functionality is detailed.

3.1 Macleod

Macleod was inspected fully on February 20th, 2023.

Macleod is the Pip installable, importable library form of Macleod-Core. The goal of RCD's development of this project is both to properly package Macleod-Core and also to adjust the parse_clif() and check_consistency() functions to accept arguments as normal python functions would, instead of searching for them as though they were being run on the command line. These functions are present in the SDD, and continue in Macleod to fulfill the roles they performed in Macleod-Core.

3.2 Macleod-Core

Macleod-Core was inspected fully on February 20th, 2023.

Macleod-Core is the backbone of Macleod and contains all of the scripts to parse and convert CLIF files. Our work on Macleod-Core was to allow for the functions and scripts to be callable within a python file, and to further extend the capabilities of Macleod-Core by

ensuring that CLIF files translated into other logic file formats keep comments, import instructions, and module declarations. We are also tasked with expanding the coverage of the parser to more fully satisfy all CLIF conventions.

We were not clear on the differences between Macleod and Macleod core at the time of the preparation of the SDD, thus the SDD lists Macleod (and, by extension, Macleod-Core) as simply the Macleod Plug-In and Macleod Library. The key differences are that Macleod-Core consists of the parsing and converting functions, which will then be callable via either Macleod or Macleod-IDE.

3.3 Macleod-IDE

Macleod-IDE was inspected fully on February 22nd, 2023.

Macleod-IDE is a plugin for the Spyder development environment wherein parsing and converting will be made easier via Macleod-Core functionality. The goal is for Macleod-Core to be utilized within Macleod-IDE through buttons that can parse or parse and convert CLIF files. The idea we had for Macleod-IDE on the SDD is congruent with our development of Macleod-IDE to date.

4. Defects

Table 2 below shows the defects that were discovered during the code inspection process. Each defect is numbered, explained, and labeled according to type, which file it was found in, and which module it was found in.

#	Defect Type	Defect Location	Module	Defect Explanation
03	Correctness: Requirement Missing	parsing/parser.py	Macleod	parsing/parser.py tried to import the ply library, which was not present in the requirements.txt file
05	Coding Convention: Function Name	parsinglib.py	Macleod	The function owlType() is camel case where other functions are snake case.
06	Coding Convention: File Name	parsinglib.py	Macleod	All other files not in a subfolder are title case. (except Filemgt, but Torsten wrote that one)
07	Coding Convention: File Name	check_consistency_lib.p	Macleod	All other files not in a subfolder are title case (except Filemgt, but

		y		Torsten wrote that one)
08	Coding Conventions: Class Name	confpage.py	Macleod -IDE	The class name “macleod_ideConfigPage” is somehow both snake and camel case.
09	Coding Conventions: Uncertain Return Type	check_consistency_lib.py	Macleod	There is a line “var1, var2 = consistency(args)” where it is possible for consistency to return None, even though args will cause it to return a tuple.
10	Naming Conventions: Variable Name	check_consistency_lib.py	Macleod	There is a variable named derp, which is not descriptive of its function. It is never referenced.
11	Coding Conventions: Unused Variable	check_consistency_lib.py	Macleod	There is a variable named derp that is never referenced and is therefore unnecessary.
12	Other: Unused Import	dl/owl.py	Macleod	The package functools is imported, but never used.

Appendix A

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 CIR contains a compilation of the code review conducted for the CLIF Parser. The client, in signing this CIR, agrees that the published code has been inspected thoroughly and completely. The team, RCD, agrees that we have thoroughly and completely inspected the code up until the date specified.

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 B

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 C

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

Matthew Brown

- Wrote sections 2.1, 3.2 and 3.3

Gunnar Eastman

- Recorded defects in section 4
- Recorded meeting details in section 2
- Wrote section 3.1
- Adjusted section 1, wrote section 1.3
- Helped with section 2.2
- Helped record possible defects in section 1.4

Jesiah Harris

-

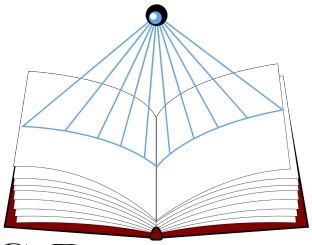
Eli Story

- Wrote section 2.2
- Helped with section 2
- Helped record possible defects in section 1.4

Appendix F - AM

Administrator Manual

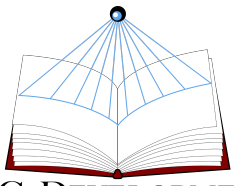
for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



R.C. DEVELOPMENT

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

Version 1.1
April 28, 2023



R.C. DEVELOPMENT

Administrator Manual:

Table of Contents

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 four seniors from the University of Maine: Matthew Brown, Gunnar Eastman, Jesiah Harris, and Elijah Story.

RCD is responsible for the following products: MacleodCore, pre-existing code that will be used in the terminal; Macleod, code that has been adjusted by RCD that enables the scripts available in MacleodCore to be used as functions in a python library; and MacleodIDE, a plugin for the SpyderIDE that will allow the Macleod functions to have access to a User Interface, for an easy to understand experience.

The intended audience of this document consists of those who would, in the future, be making updates to the Macleod software, either to expand its potential even further or to update it in accordance with changes to the CLIF standard, or to changes to the file formats that Macleod can translate to.

In the second section of this document, we review relevant background information and system requirements for each edition of Macleod. In the third section we move into reviewing both how to install and operate Macleod, as well as the various administrative procedures for keeping Macleod up to date.

1.1 Purpose of This Document

The purpose of this Administrator Manual is to elaborate upon how one can install and manage the Macleod system.

1.2 References

[1] Brown, M, et al., CLIF Parser System Requirement Specification, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_SRS_Ver_1.3.pdf.

[2] Brown, M, et al., CLIF Parser System Design Document, 2022,

https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_SDD_Ver_1.1.pdf.

[3] 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.

[4] Brown, M, et al., CLIF Parser Critical Design Review Document, 2022, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_CDRD.pdf.

[5] Brown, M, et al., CLIF Parser Code Inspection Report, 2023, https://github.com/Reading-Club-Development/macleod/blob/master/UMaine%20Capstone%20Documents/RCD_CIR_Ver_1.1.pdf.

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

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

[8] International Organization for Standardization, *Common Logic (CL): a framework for a family of logic-based languages*, 2018, https://www.iso.org/standard/39175.html

2 System Overview

This section details what systems the Macleod system is available on and what Macleod is, especially related to Administration.

2.1 Background

Macleod is entirely a client-side process, so no intensive operations are normally necessary on the part of administrators. The responsibilities of the Administrators are to handle GitHub in accordance with the updating standards relating to the logical conventions involved in the Macleod system. There should not need to be any daily operations on the administration side, nor is daily maintenance necessary.

2.2 Hardware and Software Requirements

In order to use the system, any operating system in the current ecosystem (2023) will work. MacleodIDE requires Spyder 5 and Python 3.7 or later in order to use. MacleodCore also requires Python 3.7 or later. The plugin requires Python version 3.8 or later. The Macleod

system is built to be run from the command line interface, within Python code segments, or within Spyder.

The Macleod system is extremely small, less than 100 megabytes, and is stored client-side.

3. Administrative Procedures

For the Macleod system, the maintenance required is threefold. Firstly, the logic behind the parser and scripts within MacleodCore must be kept up to date with the CLIF standard, along with also ensuring the standards of TPTP, LADR, OWL, and LaTeX are also kept up to date. Secondly, to maintain Macleod, every update to MacleodCore should accompany an update to the Macleod library. The Macleod library must also be kept up to date with Python's package specifications, and the Python programming language. Lastly, MacleodIDE, the Spyder plug-in, must simply not fall into obsolescence as Spyder updates over time.

3.1 Installation

Below we detail how one can install the various releases of Macleod. We will first discuss MacleodCore, which provides commands to be used from the terminal. Secondly, we will overview the installation process of Macleod, which can be installed as a python package. Last is how to install MacleodIDE, a plug-in for the Spyder IDE.

3.1.1 Installing MacleodCore

The installation of MacleodCore follows the same format as is in the Macleod GitHub repository [7]. We did not modify this procedure, but using only MacleodCore without the IDE or library is not recommended by Reading Club Development. The procedure is copied in its entirety here:

It is recommended to create a virtual environment to work out of:

On Windows hosts with Python3 already installed and on the %PATH%

```
# Create a new virtual environment called "ve" (or whatever you want to
name it)
> python -m venv ve
# Activate the newly created virtual environment
> ve/scripts/activate.bat
```

On Linux hosts with Python3 installed, the two commands can be run in one line:

```
>python3 -m venv ve && . ve/bin/activate
```

Once the virtual environment has been created and activated clone this repository and install:

```
# Clone the repository using https or ssh
> git clone https://github.com/thahmann/macleod.git
# Go to the folder that contains the cloned repository,
# e.g. "%USERPROFILE%\GitHub\macleod\" by default on Windows hosts,
# or a location like "/home/git/macleod" on Linux hosts
> cd macleod

# Install macleod and all dependencies via pip
> pip install .

# Optionally you can install the GUI components as well (see more information about the GUI alpha version below)
> pip install .[GUI]
```

As a next step, the configuration files need to be put in place:

On Windows hosts:

```
# Go to your home directory and create a subfolder `macleod'
> cd %USERPROFILE%
> mkdir macleod
> cd macleod
# copy the configuration files from the github folder to this new folder.
> copy "%USERPROFILE%\GitHub\macleod\conf\macleod_win.conf .
> copy "%USERPROFILE%\GitHub\macleod\conf\logging.conf .
```

On Linux hosts:

```
# Go to your home directory and create a subfolder `macleod'
> cd ~
> mkdir macleod
> cd macleod
# copy the configuration files from the github folder to this new folder
> cp "/home/git/macleod/conf/macleod_linux.conf .
> cp "/home/git/macleod/conf/logging.conf .
```

In a final step, add the binaries of the theorem provers and model finders to be used for ontology verification and proving of lemmas to the "macleod" directory and edit the configuration file "maleod_win", "macleod_linux" or "macleod_mac" as necessary. In particular the commands for the utilized theorem provers and model finders must use the

correct paths (if not on the PATH variable) and commands. Likewise, make sure that in *logging.conf* the args parameter of the section *[handler_fHandler]* uses the correct path for your host.

Note: The provers are not needed for translation to TPTP, LADR or for extraction of OWL ontologies.

3.1.2 Installing Macleod

The installation of Macleod is as easy as installing a python library. In order to install the Macleod library, one must simply open the command line and run:

```
> pip install -i https://test.pypi.org/simple/ macleod==0.2
```

If you run the following command:

```
> pip show macleod
```

And get the following result:

```
Name: macleod
Version: 0.2.7
Summary: A simple way to parse clif files and convert them into different ontology file formats
Home-page:
Author:
Author-email: Gunnar Eastman <gunnar.eastman@gmail.com>
License:
Location: [insert long filepath here]
Requires:
Required-by:
```

...then the Macleod python package has been installed correctly.

3.1.3 Installing MacleodIDE

In order to install the MacleodIDE, there are four steps. Firstly, one must install Spyder. The easiest way to install Spyder is with the Anaconda Python distribution, which comes with everything you need to get started in an all-in-one package. Download Spyder from its webpage. For more information, visit Spyder's Installation Guide. Spyder is the home of Macleod-IDE.

Second, Spyder will likely need to be updated. In order to update Spyder using conda, one can run from the command line (or Anaconda prompt on Windows):

```
> conda update anaconda
> conda update spyder
```

If this results in an error or does not update Spyder to the latest version, try:

```
> conda install spyder=5
```

The third step is to install the plugin using pip. The macleod_ide plugin is available via test.pypi.org. The plugin can be installed using pip by running:

```
> pip install -i https://test.pypi.org/simple/ macleod-ide
```

Lastly, it is necessary to set up a development environment. In principle, we could use any Spyder installed within a conda environment according to the instructions given in the installation guide. However, it is recommended to create a new environment which only has Spyder with the minimum dependencies needed for your plugin.

This minimum dependency environment can be installed in the following way:

```
$ conda activate base
$ conda install -c conda-forge mamba
$ mamba create -n spyder-dev -c conda-forge python=3
$ mamba activate spyder-dev
$ mamba install spyder
```

Then, you can launch the virtual environment and run

```
$ spyder
```

...and Spyder should open, your terminal should read: "spyder launched" , then "macleod_ide initialized!"

The plugin is the notepad symbol located at the top right hand corner of the screen.

3.2 Routine Tasks

The administrator must also keep Macleod up to date to the logical standards they wish to use by following the changelog of Macleod. In the event that there is a change to the library or plugin, the user will need to update or install the latest version of MacleodCore. The routine tasks for the Macleod system lie in the user maintaining their own virtual environments and such. In many cases, the user is the administrator because the system does not interface with the Internet, and thus the user is responsible for their local maintenance.

3.2.1 Updating Pypi Version

One task related to the maintenance of Macleod is ensuring that any updates to the source code are pushed to Pypi, so that those installing macleod can receive these updates. The process followed, while also explained here, is very similar to the one provided by python: https://packaging.python.org/en/latest/tutorials/packaging-projects/

The macleod folder, either downloaded from the git repository or via pip, should, for convenience, be placed into a folder that will contain the supplementary files, which we will refer to as the package folder. This is not necessary, but recommended for the sake of organization. The package folder can be any directory that contains the macleod folder.

If downloaded from the git repository, the macleod folder should contain a folder named "dist," a file named "README," and a file named "pyproject.toml." Move these three items into the package folder.

Open pyproject.toml in any text editor, and update the version number. At time of writing, it is present on line 7 of the document, as version 0.2.8. Save and close the file.

Open the terminal and navigate to the package folder, then run the following command:

```
py -m build
```

This will create two files in the dist folder: A .whl file and a .tar.gz file. Ensure that the only files in the dist folder are those that match the newest version number. Then run the following command:

```
py -m twine upload dist/*
```

This will ask for your pypi username and password, then upload the files within the dist folder to PyPi, making them available for distribution via pip.

Note that you may need to run `pip install --upgrade build` and `pip install --upgrade twine` if those libraries have not already been installed on your machine, and that on Linux or MacOS, "py" will need to be replaced with "python3"

3.3 Periodic Administration

For every update to TPTP, LADR, LaTeX, or CLIF standards, the Macleod system will have to be updated to accommodate the shifts in standards. Currently, the system is designed for the 2018 version of CLIF. Updates to the target formats can be performed by editing the

to_tptp(), to_ladr(), or to_latex() functions of each file within the /logical folder. Should the OWL standards change, those changes can be reflected in dl/translation.py.

In the event that the CLIF standards change to make Macleod out of date, changes to update it must be made to the parsing rules in the /parsing/parser.py file.

3.4 User Support

User support can be achieved through either reading the extensive documentation accompanying this project, or by reaching out to Dr. Torsten Hahmann or Jake Emerson directly via emails found within the Macleod GitHub. At time of writing, Dr. Torsten Hahmann can be reached at torsten.hahmann@maine.edu and Jake Emerson at raymond.emerson@maine.edu. If an administrator would need contact with an RCD team member, please contact Matthew Brown at matthewbrownie3@gmail.com with any questions or concerns.

4. Troubleshooting

In this section, known bugs and limitations will be discussed, along with how to deal with potential errors or system failures.

4.1 Dealing with Error Messages and Failures

Error messages should be handled through the verbose error system already included with Python. If these errors cannot be resolved, the best course of action is to either reach out to Dr. Torsten Hahmann, for script errors or errors in logic, or Matthew Brown, for installation process errors.

4.2 Known Bugs and Limitations

The most evident limitation of the system is many places where code was never fully implemented, resulting in many functions that will only raise “not implemented” exceptions. These were present before RCD’s involvement, but are only limited to the translations to function-free prenex conjunctive normal form (ffpcnf) or the Web Ontology Language (OWL).

Appendix A

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 AM contains instructions on how an administrator may maintain the system. The client, in signing this AM, agrees that this document contains how to install and manage Macleod thoroughly and completely. The team, RCD, agrees that this is an extensive and complete record of how to manage and install Macleod.

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. 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	_____	__/__/__
Eli Story	_____	__/__/__

Appendix B

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 AM. 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 AM. We have all agreed to these terms and placed our signatures below.

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

Appendix C

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

Matthew Brown

- Formatted the document
- Applied Sister team feedback
- Wrote the majority of the document

Gunnar Eastman

- Applied sister team feedback
- Wrote a significant amount of the document

Jesiah Harris

- Wrote some of the document

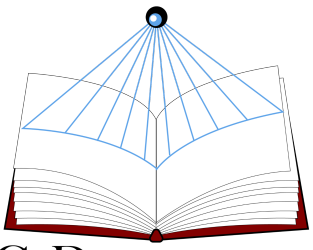
Eli Story

- Wrote some of the document
- Wrote sister team feedback

Appendix G – UG

User Guide

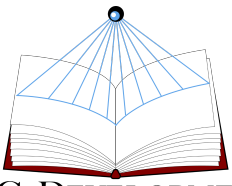
for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



R.C. DEVELOPMENT

Designed by Reading Club Development:
Matthew Brown, Gunnar Eastman, Jesiah Harris, Elijah Story

Version 1.0
Apr 18, 2023



User Guide:
Table of Contents

How to Use this Document

This user guide is constructed as follows:

Section 1 is an introduction to Macleod, its use cases, and its intended users. The introduction also contains information regarding the applicability and purposes of the user guide.

Section 2 provides a general overview of the functionalities available in Macleod and is intended to give readers/users a broad set of expectations on what will be encountered during the use of Macleod.

Section 3 consists of instructions for the use of Macleod and Macleod-IDE. This section is aimed at giving users an understanding of how to operate the system.

Section 4 is a list of references and documentation relevant to the functionalities and capabilities of Macleod.

Section 5 gives a list of common error messages that may be encountered during the use of Macleod, as well as troubleshooting methods and recovery procedures.

Section 6 is a glossary of terms related to Macleod and its intended use cases.

Related Documents

Num	Title	Author	Date	Issue
1	SRS - System Requirements Specification	R.C. Development	10/21/2022	1.3
2	SDD - System Design Document	R.C. Development	11/11/2022	1.1
3	UIDD - User Interface Design Document	R.C. Development	11/30/2022	1.0
4	CIR - Code Inspection Report	R.C. Development	3/10/2023	1.1
5	AM- Administrator Manual	R.C. Development	4/11/2023	1.1

1. Introduction

The CLIF (Common Logic Interchange Format) Parser is a stepping stone in the progress toward a unified Common Logic, which should allow for more properly defined variables, and hopefully slightly mitigating this crisis of reproducibility. Macleod stands for “Macleod Common Logic Environment for Ontology Development.” On a smaller scale, the purpose of Macleod is primarily to parse CLIF files to ensure they are syntactically correct according to CLIF standards, and translate them into other logic-based conventions. In addition to the CLIF Parser, Macleod has access to a GUI, Macleod IDE, in the form of a plugin for the Spyder IDE.

Macleod’s intended audience is specialists in scientific fields, and make use of TPTP, OWL, LADR, LaTeX, or CLIF files, and/or would be able to make use of an Ontology.

Use of the Macleod system assumes that prospective users are familiar to some degree with the use of one of the following: the command line, the Python programming language, or the Spyder IDE.

Applicability Statement

This User Guide is written for use with Macleod version 1.0 and its GUI, Macleod-ide version 0.0.2. Macleod 0.2.8 and its release history are available for viewing and installation here. Macleod-IDE and its release history are available for viewing and installation here.

Purpose

This user guide provides steps on how to use Macleod, a breakdown of related and relevant documentation, as well as a troubleshooting guide for issues commonly encountered when using Macleod. This user guide is intended for end users, casual users, and anyone else who wants to use Macleod or extend its features through further development.

Problem Reporting Instructions

To report any problems encountered during the installation, use, or maintenance of either Macleod or Macleod-IDE, please create an issue on the corresponding GitHub repository. If the problem encountered is urgent or important you may reach out to Dr. Torsten Hahmann or Jake Emerson directly via emails found within the Macleod GitHub. At time of writing, Dr. Torsten Hahmann can be reached at torsten.hahmann@maine.edu and Jake Emerson at raymond.emerson@maine.edu. If an administrator would need contact with an RCD team member, please contact Matthew Brown at matthewbrownie3@gmail.com with any questions or concerns.

2. Overview

Below we describe the current available editions of the Macleod software. Use of MacleodCore assumes a familiarity with terminal commands. Use of Macleod assumes a familiarity with Python. Use of Macleod-IDE assumes a familiarity with the Spyder IDE.

Macleod

Macleod is a common logic parser that takes in CLIF files and creates axioms that are then used to create a file in a different logic-based convention. In essence, it translates one logic language to another.

The user will primarily use the one function `parse_clif()`. This function takes the path to a .clif file and the file type you would like to convert it to as a string. This will take in the file and create the axioms for that file. It will then take the axioms and send it off to the function corresponding to the desired file type to be converted to that file. This will then print out the location of the newly created file.

Macleod-IDE

Macleod-IDE is a Spyder plugin that utilizes the Macleod Pip library to parse files from the GUI of Spyder. The plugin creates a set of buttons that allows the user to select a file and click a button that will translate the file to the desired logical language. When the file is translated, the path to the newly created file will be printed in the terminal located in Spyder.

MacleodCore

MacleodCore is a series of python scripts that are designed to be run from the command line. These scripts have the same functionality as the Macleod python package functions.

3. Instructions

Below are detailed instructions for the installation and use of Macleod the python package, MacleodCore the collection of command-line scripts, and MacleodIDE the Spyder plugin.

Macleod Installation

Because Macleod is a Pip library, installation is very simple. All the user needs to do is run the following command in the terminal:

```
pip install macleod
```

This will install Macleod for use in any file.

Macleod Use

To use Macleod once it is installed, the user can import Macleod into any file using the following import statement:

```
import macleod.scripts.parserlib as macleod
```

In this import statement, the macleod after “as” can be anything the user would like. This is simply the variable that will be used to invoke Macleod functions.

To use Macleod once imported, the user can use the following function call:

```
macleod.parse_clif(“FILEPATH”, “LANGUAGE”)
```

As in the import statement, the macleod is the name the user decided on. Make sure that is the same as the variable name after the “as” in the import statement.

The function `parse_clif()` takes two arguments. The first is the file path, in string form, to the .clif file the user wishes to translate. Please use the full filepath where possible. The second argument is the language the user wishes the output file to be translated to. It also is a string. The following are the languages available at this current time:

- “TPTP”
- “LADR”
- “LaTeX”

The Language argument is optional, and if no argument is provided, the .clif file will be parsed and printed, without being translated.

Macleod-IDE Installation

Macleod-IDE requires Macleod (installation instructions above) and the Spyder IDE to be installed.

To install Spyder:
The easiest way to install Spyder is with the Anaconda Python distribution, which comes with everything you need to get started in an all-in-one package. Download it from its webpage. For more information, visit its Installation Guide.

Update Spyder using conda:
From the command line (or Anaconda prompt on Windows), run:

```
$ conda update anaconda
$ conda update spyder
```

If this results in an error or does not update Spyder to the latest version, try:

```
$ conda install spyder=5
```

Install the plugin using pip:
The macleod_ide plugin is available via pypi.org

The plugin can be installed using pip as follows:

```
$ pip install macleod-ide
```

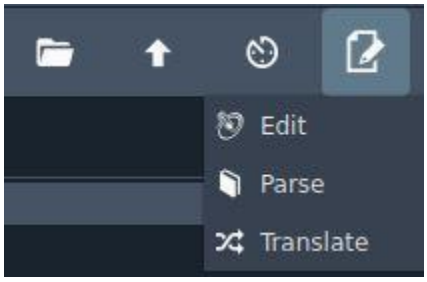
Set up a development environment:
In principle, we could use any Spyder installed within a conda environment according to the instructions given in the installation guide. However, it is recommended to create a new environment which only has Spyder with the minimum dependencies needed for your plugin.

It can be installed in the following way:

```
$ conda activate base
$ conda install -c conda-forge mamba (Recommended by spyder-ide.org)
$ mamba create -n spyder-dev -c conda-forge python=3
$ mamba activate spyder-dev
$ mamba install spyder
```

To use Macleod-IDE:

After installation, Macleod-IDE will be available for use in Spyder. The plugin icon will appear in the top right corner as shown here:



Macleod-IDE offers the following functionalities, though they are nonfunctional at this point in development:

Edit: Creates pop-out window for the loaded CLIF file, highlights errors, and allows for editing of the CLIF file.

Parse: Parses the loaded CLIF file and ensures that it is syntactically correct.

Translate: Parses the loaded CLIF file, ensures that it is syntactically correct, and translates the CLIF file to an indicated other language.

MacleodCore Installation

The installation of MacleodCore has two potential methods, the first follows the same format as is in the Macleod GitHub repository, and involves downloading the code in that repository and installing it. The second involves downloading Macleod via pip, and using that library similarly to the code in the GitHub repository. We did not modify this first procedure, but using only MacleodCore without the IDE or library is not recommended by Reading Club Development. The procedure is copied in its entirety here:

It is recommended to create a virtual environment to work out of. On Windows hosts with Python3 already installed:

```
> python -m venv ve
> ve/scripts/activate.bat
```

On Linux hosts with Python3 installed:

```
>python3 -m venv ve && . ve/bin/activate
```

Once the virtual environment has been created and activated clone this repository and install:

```
> git clone https://github.com/thahmann/macleod.git
> cd macleod
> pip install .
```

In a final step, add the binaries of the theorem provers and model finders to be used for ontology verification and proving of lemmas to the "macleod" directory and edit the configuration file "maleod_win", "macleod_linux" or "macleod_mac" as necessary. In particular the commands for the utilized theorem provers and model finders must use the correct paths (if not on the PATH variable) and commands. Likewise, make sure that in logging.conf the args parameter of the section [handler_fHandler] uses the correct path for your host.

Note: The provers are not needed for translation to TPTP, LADR or for extraction of OWL ontologies.

The second method of installing MacleodCore is to install Macleod via pip, then navigate in the terminal to the location of that python library in the terminal, then run

```
> pip install .
```

The library is likely present under C:\Users\[username]\AppData\Local\Packages\PythonSoftwareFoundation.Python.[long version number]\LocalCache\local-packages\Python[short version number]\site-packages\macleod

Reference Section

In this section are explanations of each terminal command for MacleodCore, as well as each function made available via Macleod.

Macleod

In the Macleod library, there are two functions designed for external use. They will be listed below alongside all of their arguments. '(req.)' denotes a required argument, and '(opt.)' denotes an optional argument.

```
parse_clif(filepath, format="None", OWL_version="Full", enum=False, output=False, resolve=False, nocond=False, base=None, sub=None):
```

parse_clif() is the first of these functions, and will accept the following arguments. Note that the file path is the only required argument.

```
#Returns nothing
#Filepath (req.): A string filepath of the file or folder to be converted
#format (opt.): The format to be converted into. Not case sensitive. If set to
"None" it'll just print the clif file.
#enum (opt.): Do you want it to enumerate axioms? Only relevant for LaTeX output.
#output (opt.): Boolean. Do you want it to write an output file (True) or just
print (False)
#resolve (opt.): do you want it to resolve import statements
#nocond (opt.): do u want it to keep conditionals (True) or switch to
disjunctions (False)
#base (opt.): path to directory with ontology files. Only relevant if resolve is
on
#sub (opt.): path to directory with import files. Only relevant if resolve is on
```

Note that 'format' should be given one of the following values, if it is given a value at all:

- "TPTP"
- "LADR"
- "LaTeX"

```
check_consistency(filepath, method="simple", output=True, resolve=False,
stats=True, nontrivial=False, base=None, sub=None):
```

check_consistency() is the second function provided by Macleod, and exists to help confirm that a given CLIF file is not only syntactically correct, but also logically consistent.

```
#Function to check the consistency of ontologies in the Common Logic Interchange
Format (.clif).'
    #filepath: path to the file or folder of files to be checked.
    #method: string determining how to check
        #"simple" for a simple consistency check
        #"full" for a full recursive consistency check in case the entire
ontology is not probably consistent
        #"module" to check each module individually
    #output: Bool. Do you want it to write to an output file
```

```
    #resolve: Bool. auto resolve imports?
    #stats: bool. Do you want detailed stats (including definitions) about the
ontology?
    #nontrivial: bool. Instantiate all predicates to check for nontrivial
consistency?
    #base: Path to directory containing ontology files (basepath; only relevant
when option --resolve is turned on; can also be set in configuration file
    #sub: String to replace with basepath found in imports, only relevant when
option --resolve is turned on
```

MacleodCore

Once installed, MacleodCore offers scripts that fill similar roles to the two Macleod functions.

On the command line, parse_clif should be constructed thusly:

```
parse_clif -f [FILEPATH] (--[LANG]) (--enum) (-out) (--resolve) (--nocond)
(--base) (--sub) (--ffpcnf) (--clip)
```

-f and the accompanying filepath is again the only required argument. Though an optional argument, if a value is provided, the only accepted inputs for LANG are as follows:

- --owl
- --tptp
- --ladr
- --latex

Below, explanations for individual optional arguments are provided.

```
'--enum', 'Enumerate axioms in LaTeX output'

'--output', 'Write output to file'

'--resolve', 'Automatically resolve imports'

'--nocond', 'Do not use conditionals (only applies to TPTP, LADR and LaTeX
production)'
```

```
'--base', 'Path to directory containing ontology files (basepath; only relevant
when option --resolve is turned on; can also be set in configuration file)'

'--sub', 'String to replace with basepath found in imports, only relevant when
option --resolve is turned on'

'--ffpcnf', 'Automatically convert axioms to function-free prenex conjunctive
normal form (FF-PCNF)'

'--clip', 'Split FF-PCNF axioms across the top level quantifier'
```

The second script is check_consistency, which can be called like so:

```
check_consistency -f [FILEPATH] (--output) (--resolve) (--stats)
(--nontrivial) (--base) (--sub) --[METHOD]
```

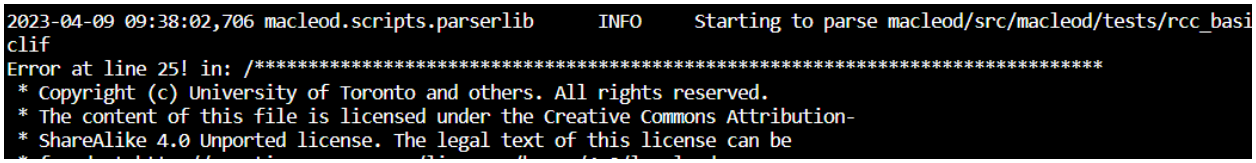
Note that filepath and method are the only two required arguments. Below, each of the optional arguments are elaborated upon:

```
'--output', 'Write output to file'

'--resolve', 'Automatically resolve imports'

'--stats', 'Present detailed statistics (including definitions) about the
ontology'

'--nontrivial', 'Instantiate all predicates to check for nontrivial consistency'

'--base', 'Path to directory containing ontology files (basepath; only relevant
when option --resolve is turned on; can also be set in configuration file)'

'--sub', 'String to replace with basepath found in imports, only relevant when
option --resolve is turned on'

The following are the options for the ‘method’ argument, and are mutually
exclusive:

'--simple', 'Do a simple consistency check'
```

```
'--full', 'Do a full recursive consistency check in case the entire ontology is
not probably consistent'

'--module', 'Check each module individually'
```

Error Messages and Recovery Procedures

It should be the case that the only error messages Macleod produces are related to incorrect input files, in the case that those input files do not correctly implement the CLIF syntax. In the case that there is such an error, the system will note the line at which this error occurs, and will print the beginning of the file being parsed, and the filename.



The recommended solution is to consult the CLIF Standard, provided by the ISO here: here.

Glossary

Axiom - A statement or proposition that is taken to be true to be used in further reasoning and arguments

CLIF - Common Logic Interchange Format. ISO/IEC 24707:2007 defines Common Logic: a first-order logic framework intended for information exchange and transmission. More information available here.

LADR - Library for Automatic Deduction Research, a file format.

Ontology - A collection of classifications and traits used to organize information.

OWL - Web Ontology Language, a file format.

Parse - A form of syntactic analysis. To divide a string into grammatical parts and check the correctness of its syntax based on the rules of a logical language's grammar.

TPTP - Thousand Problem Theorem Prover, a file format.

Appendix A

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 UG contains instructions on how an administrator may maintain the system. The client, in signing this UG, agrees that this document contains how to install and manage Macleod thoroughly and completely. The team, RCD, agrees that this is an extensive and complete record of how to manage and install Macleod.

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. 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	_____	__/__/__
Elijah Story	_____	__/__/__

Appendix B

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 UG. 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 AM. We have all agreed to these terms and placed our signatures below.

Name:	Signature:	Date:
Matthew Brown Comments:	_____	__/__/__
Gunnar Eastman Comments:	_____	__/__/__
Jesiah Harris Comments:	_____	__/__/__
Elijah Story Comments:	_____	__/__/__

Appendix C

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

Matthew Brown

- Formatted the document
- Did some editing of the document

Gunnar Eastman

- Handled the majority of the editing
- Wrote some of the document

Jesiah Harris

- Wrote a good portion of the document
- Did some sister team editing

Elijah Story

- Wrote a good portion of the document
- Conducted sister team review