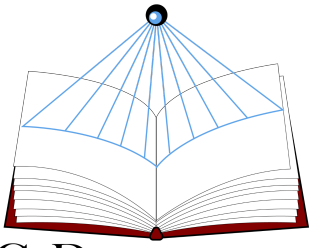
# Code Inspection Report

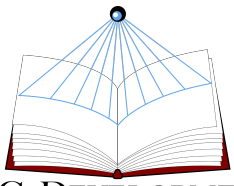
for Macleod: A CLIF Parser, designed for Torsten Hahmann and Jake Emerson



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


**1. Introduction** **2**

1.1 Purpose of This Document 2

1.2 References 2

1.3 Coding and Commenting Conventions 3

1.4 Defect Checklist 3

**2. Code Inspection Process** **4**

2.1 Inspection Approach 5

2.2 Impressions of the Process 6

2.3 Inspection Meetings 6

**3. Modules Inspected** **7**

3.1 Macleod 7

3.2 Macleod-Core 7

3.3 Macleod-IDE 8

**4. Defects** **8**

**Appendix A** **10**

**Appendix B** **11**

**Appendix C** **12**


| **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<br>Comments: | _____ | __/__/__ |
| Gunnar Eastman<br>Comments: | _____ | __/__/__ |
| Jesiah Harris<br>Comments: | _____ | __/__/__ |
| Shea Keegan<br>Comments: | _____ | __/__/__ |
| Eli Story<br>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