

A beginner-friendly environment for exploring error messages in the Clojure programming language.

Tristan Kalvoda, Elena Machkasova, Jaydon Stanislawski, and John Walbran

University of Minnesota, Morris

Midwest Instruction and Computing Symposium, April 2025

Outline

- 1 Overview of Clojure and its Error Messages
 - Clojure Language and Syntax
 - Clojure Error Messages
- 2 Babel Project
 - Motivation and Goals
 - Overview and Usage
- 3 Interactive Visualizations with Morse
 - Morse Viewers
 - Prototype Viewer
- 4 Current State of the Project and Future Work
 - Future Work

Clojure Language and Syntax

What is Clojure?

- Clojure is a part of the Lisp language family
 - prefix notation (operators before operands)
 - expressions are surrounded by parentheses

Example: `(/ 9 3)` denotes 9 divided by 3

- Clojure is an interpreted language
- It's implemented in Java and runs on the Java Virtual Machine (JVM) interpreter

Clojure code → Java code → JVM bytecode → executed on JVM

Clojure Language and Syntax

Clojure's REPL

- Interactive environment for code evaluation
- Read → Evaluate → Print → Loop (REPL)

```
Clojure 1.11.4
user=> (count [-1 0 1])
3
user=>
```

Clojure's Error Messages

Clojure Exceptions

- All exceptions in Clojure are Java exceptions
- Clojure syntax errors will also result in an exception

Error Messages

- Generate when an exception occurs
- Provide error type, cause, and location

Clojure's Error Messages

Anatomy of a Clojure Error Message

```
=> (/ 9 0)
```

```
Execution error (ArithmeticException) at user/eval1  
(REPL:1).
```

Divide by zero

- `ArithmeticException`: The type of error that occurred
- `user/eval1 (REPL:1)`: The location where the error happened (in this case, REPL, line 1)
- `Divide by zero`: The description of the error's cause

Motivation

- UMN Morris has been teaching functional languages first (Scheme, Racket) to students for over 30 years
- Historically done at other major institutions, including MIT
- This practice has many pedagogical benefits
- Clojure is good to learn due to its popularity and widespread industry use, but has flaws in its presentation of error messages

Goals of Babel

- Designed to be an interactive, beginner-friendly tool for understanding error data in Clojure
- Simplifies error messages to be more intuitive, removing jargon and clutter
- Utilizes tools to make Clojure errors more descriptive and detailed for formatting messages

Current State of Babel

- Our group began working after error replacement was largely achieved
- Present goals include exploring more interactive, IDE-like tools that integrate with Babel
- Need the capacity to support testing and usability
- Lots of refactoring work is being done to support this, as well as refining Babel itself

Motivation and Goals

Example

Consider the error produced by the form below. What does it mean?

```
=> (count 1)
```

```
Execution error (UnsupportedOperationException) at  
user/eval1529 (REPL:1).
```

```
count not supported on this type: Long
```

On Clojure *spec*

Babel is based on an existing Clojure library called *spec*:

- Allows specifying requirements on arguments of functions
- Requirements are predicates: can check argument types, count, values, etc.
- We bind specs to core Clojure functions
- If requirements aren't met then a *spec* error happens
- Error reports from *spec* are more detailed than native Clojure error messages

Overview

Babel tool:

- Replaces native Clojure error messages
- Messages produced by Babel broadly fall into two types:

Spec errors:

- Babel uses *spec* to specify conditions on Clojure functions
- *spec* provides information to make error messages more precise

Non-spec errors:

- When *spec* cannot be provided, e.g. syntax errors, regex is used to identify error messages
- We use a dictionary to rephrase these errors

Usage

- Launching a REPL server in the Babel repository allows the tool to “hook” to it
- Babel can report errors on a loaded Clojure file
- Babel catches Clojure errors, including our *spec* reports
- Error messages are replaced with modified ones
- Currently only supports REPL, not IDE

Setup

```
user=> (count 1)
Execution error (UnsupportedOperationException) at user/eval1531 (REPL:1).
count not supported on this type: Long
```

Figure: A default error on the `(count)` function, caused by a mistake that a student typically makes.

Setup

```
babel.middleware=> (count 1)
Function count does not allow a number as an argument in this position.

In Clojure interactive session on line 1.
Call sequence:
[Clojure interactive session (repl)]
```

Figure: The error message on the same form produced by Babel, containing a description of the potential problem in plain English.

Morse

- We would like to expand error message handling from REPL strings to interactive visual environments

Morse

- We would like to expand error message handling from REPL strings to interactive visual environments
- Joe Lane (Nubank) suggested a new tool called *Morse*

Morse

- We would like to expand error message handling from REPL strings to interactive visual environments
- Joe Lane (Nubank) suggested a new tool called *Morse*
- Morse is a third party data visualization and navigation tool

Morse

- We would like to expand error message handling from REPL strings to interactive visual environments
- Joe Lane (Nubank) suggested a new tool called *Morse*
- Morse is a third party data visualization and navigation tool
- Morse provides a customizable set of viewers for different data structures, like text, maps, or HTML content

Morse

- We would like to expand error message handling from REPL strings to interactive visual environments
- Joe Lane (Nubank) suggested a new tool called *Morse*
- Morse is a third party data visualization and navigation tool
- Morse provides a customizable set of viewers for different data structures, like text, maps, or HTML content
- We are developing a custom set of viewers for error messages

Morse

- We would like to expand error message handling from REPL strings to interactive visual environments
- Joe Lane (Nubank) suggested a new tool called *Morse*
- Morse is a third party data visualization and navigation tool
- Morse provides a customizable set of viewers for different data structures, like text, maps, or HTML content
- We are developing a custom set of viewers for error messages
- We need to label different parts of error message for different viewers

Morse Default

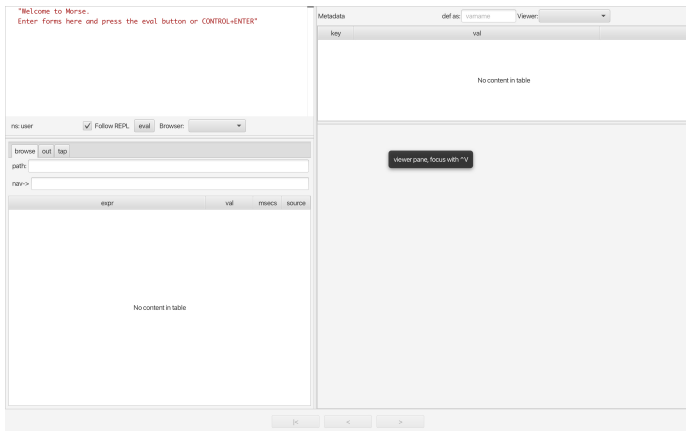


Figure: The default view of the Morse tool.

Morse Lists

The screenshot shows the Morse viewer interface. On the left, there's a control panel with buttons for 'ns: user', 'Follow REPL', 'eval', and 'Browser: repl/eval-history'. Below these are 'browse', 'out', and 'tap' buttons, a 'path: [0]' field, and a 'nav->' field. A table displays the results of an evaluation:

expr	val	msecs	source
(map inc [1 2 3])	(2 3 4)	1	Morse 2

On the right, the 'Metadata' section shows 'def as: varname' and a 'Viewer' dropdown set to 'repl/coll'. Below this is a table with 'idx' and 'val' columns:

idx	val
0	2
1	3
2	4

At the bottom of the interface are navigation buttons: '<|<', '<', '>', and '>|>'.

Figure: Morse viewer for vector data. Shows the result of the form `(map inc [1 2 3])`.

Integrating with Morse

- Babel is specialized to integrate within a REPL, so it doesn't expose the right hooks for integrating with external tools.

Integrating with Morse

- Babel is specialized to integrate within a REPL, so it doesn't expose the right hooks for integrating with external tools.
- In order to connect to a Morse session, we introduce a new session layer that exposes communication hooks for other tools.

Integrating with Morse

- Babel is specialized to integrate within a REPL, so it doesn't expose the right hooks for integrating with external tools.
- In order to connect to a Morse session, we introduce a new session layer that exposes communication hooks for other tools.
- `:init` Behavior on startup, launches a new Morse session connected to the current REPL.

Integrating with Morse

- Babel is specialized to integrate within a REPL, so it doesn't expose the right hooks for integrating with external tools.
- In order to connect to a Morse session, we introduce a new session layer that exposes communication hooks for other tools.
- `:init` Behavior on startup, launches a new Morse session connected to the current REPL.
- `:eval` Behavior on form evaluation. Stores the command and sends it to REPL and Morse.

Integrating with Morse

- Babel is specialized to integrate within a REPL, so it doesn't expose the right hooks for integrating with external tools.
- In order to connect to a Morse session, we introduce a new session layer that exposes communication hooks for other tools.
- `:init` Behavior on startup, launches a new Morse session connected to the current REPL.
- `:eval` Behavior on form evaluation. Stores the command and sends it to REPL and Morse.
- `:caught` Behavior on caught exception. Processes the error in Babel, and changes the `Exception←:wantString` processing to instead create a vector of labelled pairs.

Our Morse Viewer

- We used the existing WebView structure within Morse to render error messages.

Our Morse Viewer

- We used the existing WebView structure within Morse to render error messages.
- We use HTML to add color coding for important terms, and mono-space fonts to highlight code chunks within the message.

Our Morse Viewer

- We used the existing WebView structure within Morse to render error messages.
- We use HTML to add color coding for important terms, and mono-space fonts to highlight code chunks within the message.
- The exact design of the final viewers are still a work in progress.

```
[user=> (even? 1 2)  
Execution error (ArityException) at user/eval2044 (REPL:1).  
Wrong number of args (2) passed to: clojure.core/even?
```

Figure: The output for the form `(even? 1 2)` in default Clojure.


```
babel.middleware=> (even? 1 2)  
Wrong number of arguments in (even? 1 2): the function even? expects one  
argument but was given two arguments.  
In Clojure interactive session on line 1.  
Call sequence:  
[Clojure interactive session (repl)]
```

Figure: The output for the form `(even? 1 2)` in Babel through the REPL.

The screenshot shows the Morse web viewer interface. On the left is a sidebar with a 'Follow REF' checkbox and a table with columns 'expr' and 'val'. The main area has a header with 'Metadata', 'def as: vaname', and 'Viewer: :rebl/testWebView'. Below this, a message states: 'Your code had an error in *Clojure interactive session* on line 1.' A code editor shows the expression `(even? 1 2)`. Below the editor, an error message reads: 'Wrong number of arguments in `(even? 1 2)`: the function `even?` expects one argument but was given two arguments. In *Clojure interactive session* on line 1.' A link to the documentation for the `even?` function is provided: https://clojuredocs.org/clojure.core/even_q. At the bottom of the main area are navigation buttons: back, forward, and search.

... ☒ Follow REF Metadata def as: vaname Viewer: :rebl/testWebView

browse out tap

p... [0]

na...

expr	val
error-map	{:form (even? ...
(even? 1 2)	{:via [{:type cl...

Your code had an error in *Clojure interactive session* on line 1.

```
(even? 1 2)
```

Wrong number of arguments in `(even? 1 2)`: the function `even?` expects one argument but was given two arguments. In *Clojure interactive session* on line 1.

Documentation for the reported function.
https://clojuredocs.org/clojure.core/even_q

< < > >

Figure: The output for the form `(even? 1 2)` in Babel with Morse.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.
- We can connect Morse to a REPL session, and have mirroring form evaluation.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.
- We can connect Morse to a REPL session, and have mirroring form evaluation.
- Most of the work this year was spent structuring things for integration with Morse viewers.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.
- We can connect Morse to a REPL session, and have mirroring form evaluation.
- Most of the work this year was spent structuring things for integration with Morse viewers.
- The introduction of the error labeling and prototyping this was pivotal in enabling data formatting.

Current State of the Project

- We have existing error messages without labels for many common errors of core functions.
- We can connect Morse to a REPL session, and have mirroring form evaluation.
- Most of the work this year was spent structuring things for integration with Morse viewers.
- The introduction of the error labeling and prototyping this was pivotal in enabling data formatting.
- We currently have a small number of error messages labeled for demonstration purposes.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.
- Add hover text to viewers for specific terms to add definitions and supplementary information to the presented error message.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.
- Add hover text to viewers for specific terms to add definitions and supplementary information to the presented error message.
- Develop Morse viewers for other information, such as the stack trace, and full java error messages.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.
- Add hover text to viewers for specific terms to add definitions and supplementary information to the presented error message.
- Develop Morse viewers for other information, such as the stack trace, and full java error messages.
- We plan to run a usability study about our developments after we have greater feature coverage.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.
- Add hover text to viewers for specific terms to add definitions and supplementary information to the presented error message.
- Develop Morse viewers for other information, such as the stack trace, and full java error messages.
- We plan to run a usability study about our developments after we have greater feature coverage.
- We are going to use the results to guide further design.

Future Work

- Expand data labeling to all Babel error messages, expanding our ability to use Morse viewers.
- Add hover text to viewers for specific terms to add definitions and supplementary information to the presented error message.
- Develop Morse viewers for other information, such as the stack trace, and full java error messages.
- We plan to run a usability study about our developments after we have greater feature coverage.
- We are going to use the results to guide further design.
- We hope to explore IDE (VSCode) integration for possible work-flow refinements.

Acknowledgements

This work was supported in part by Morris Academic Partnership (MAP) and UMN Undergraduate Research Opportunity (UROP).

We thank Joe Lane for introducing us to Morse tools and for numerous helpful discussions.

Discussion

Questions?