

Clay MUD Client Documentation

A 90s MUD Client Written Today

Clay Development Team

2026-01-28

Contents

Introduction	13
What is Clay?	13
Key Features	13
Terminal User Interface (TUI)	13
Multi-World Management	13
Automation & Scripting	13
Remote Access	14
Advanced Features	14
Supported Platforms	14
Architecture Overview	14
Installation	15
Pre-built Binaries	15
Building from Source	15
Prerequisites	15
Linux (Recommended: musl build)	15
Linux with GUI Support	16
macOS	16
Termux (Android)	16
Windows (via WSL)	17
Feature Flags	17
Verifying the Installation	17
Command-Line Options	18
Quick Start	19
First Launch	19
Creating Your First World	19
Connecting to a World	21
Basic Navigation	21
Screen Layout	21
Separator Bar Components	21
Sending Commands	21
Essential Keyboard Shortcuts	22
Quick Command Reference	22
Next Steps	22

Interface Overview	23
Screen Layout	23
Output Area	23
Separator Bar	23
Input Area	24
More-Mode Pausing	24
Scrollbar Navigation	24
Themes	24
Colored Square Emoji	25
Display Width Handling	25
Commands	26
General Commands	26
/help	26
/quit	26
World Management	26
/worlds	26
/connections (or /l)	27
/disconnect (or /dc)	27
Sending Commands	27
/send	27
Settings Commands	27
/setup	27
/web	28
/actions	28
Utility Commands	28
/reload	28
/testmusic	28
/notify	28
Command Completion	29
TinyFugue Commands	30
Variables	30
#set / #unset	30
#let	30
#setenv	30
#listvar	30
Output Commands	30
#echo	30
#send	31
#beep	31
#quote	31
Expressions	31
#expr	31
#test	31
#eval	31
Expression Operators	31
Built-in Functions	32

Control Flow	32
#if / #elseif / #else / #endif	32
#while / #done	32
#for / #done	33
#break	33
Macros (Triggers)	33
#def	33
#undef / #undefn / #undeft	34
#list	34
#purge	34
Hooks	34
Key Bindings	35
#bind / #unbind	35
File Operations	35
#load	35
#save	35
#lcd	35
Variable Substitution	35
Examples	35
Auto-heal Trigger	35
Connect Hook	36
Conditional Response	36
Loop Example	36
Keyboard Shortcuts	37
World Switching	37
Input Area	37
Output Scrollback	38
More-Mode (When Paused)	38
General	38
Popup Controls (All Popups)	38
Filter Popup (F4)	39
Help Popup (F1)	39
World Selector (/worlds)	39
Actions List (/actions)	40
Confirmation Dialogs	40
Remote GUI Client Additional Shortcuts	40
Settings	41
Global Settings (/setup)	41
Available Options	41
World Switching Modes	42
Temperature Conversion	42
Per-World Settings (/worlds -e)	42
Connection Settings	42
Authentication Settings	42
Keep-Alive Settings	43
Other Settings	43

Web Settings (/web)	44
WebSocket Server (Secure)	44
WebSocket Server (Non-Secure)	45
HTTP/HTTPS Web Interface	45
Settings Persistence	45
Example ~/.clay.dat	45
Actions (Triggers)	47
Opening the Actions Editor	47
Action List Controls	47
Creating an Action	48
Action Fields	48
Match Types	48
Regex (Regular Expression)	48
Wildcard (Glob-style)	48
Capture Groups	49
Regex Capture Groups	49
Wildcard Capture Groups	49
Command Examples	49
Basic Auto-Response	49
Multi-Command Action	49
Notification Action	50
Gagging (Hiding Lines)	50
Combined Gag and Command	50
Manual Invocation	50
Example Manual Action	51
F8 Pattern Highlighting	51
World-Specific Actions	51
Best Practices	51
Common Action Recipes	51
Auto-Heal When Low	51
Reply to Tells	52
Channel Logger	52
Combat Spam Filter	52
Multi-World System	53
Understanding Worlds	53
World Switching	53
Active World Cycling (Up/Down)	53
All World Cycling (Shift+Up/Down)	53
Activity-Based Switching (Escape+w or Alt+w)	53
World Switching Mode	53
Activity Indicators	54
Separator Bar	54
Unseen Line Tracking	54
Managing Connections	54
Listing Connections	54
Connecting to Worlds	54

Disconnecting	54
Per-World Features	55
Output Buffers	55
Prompts	55
Logging	55
Encoding	55
Cross-Interface Sync	55
Tips for Multi-World Usage	55
Web Interface	57
Setup	57
1. Configure WebSocket Server	57
2. Enable HTTP Server	57
3. Access the Web Interface	57
Features	58
Full MUD Client	58
Toolbar	58
Hamburger Menu	58
World Selector	58
Connected Worlds List	58
Actions Editor	59
Keyboard Shortcuts	59
Mobile Support	59
Layout Adjustments	59
Touch Controls	59
Visibility Handling	60
Security	60
Password Protection	60
Allow List / Whitelisting	60
TLS/SSL	60
Cross-Interface Sync	60
Troubleshooting	61
Can't Connect	61
Connection Drops	61
Display Issues	61
Remote GUI Client	62
Building	62
Running	62
Login Screen	62
Interface Overview	63
World Tabs	63
Output Area	63
Input Field	63
Status Bar	63
Hamburger Menu	63
Keyboard Shortcuts	63
World Switching	63

Input Area	64
Output	64
Display	64
Menu Shortcuts	64
Filter Popup	64
Debug Selection	65
Features	65
Color Support	65
Colored Square Emoji	65
Word Wrapping	65
ANSI Music	65
Themes	65
Synchronization	65
Troubleshooting	66
Build Errors	66
Connection Issues	66
Display Issues	66
No Sound	66
Remote Console Client	67
Running	67
Use Cases	67
Interface	67
Key Differences from Main Console	68
No Direct Connections	68
Local World Switching	68
Synchronized State	68
Keyboard Shortcuts	68
Special Commands	68
Menu Popup	69
Popups	69
Authentication	69
Building	69
Example Setup	70
Server Side (Master)	70
Client Side (Remote)	70
Tips	70
SSH Forwarding	70
Multiple Views	70
Screen/Tmux	70
Troubleshooting	71
Connection Refused	71
Authentication Failed	71
Display Issues	71
Sync Issues	71
Telnet Features	72
Automatic Negotiation	72

Supported Telnet Options	72
SGA (Suppress Go Ahead)	72
TTYTYPE (Terminal Type)	72
EOR (End of Record)	72
NAWS (Negotiate About Window Size)	73
Prompt Detection	73
How It Works	73
Prompt Handling	73
Auto-Login Prompt Detection	73
Keepalive	73
When Sent	73
Keepalive Types	74
Custom Keepalive	74
Timing Fields	74
Line Buffering	74
Safe Splitting	74
Partial Lines	75
Telnet Sequences	75
Common Sequences	75
IAC Values	75
Troubleshooting	75
No Prompt Detected	75
Wrong Terminal Type	75
Disconnected for Idling	76
Garbled Output	76
ANSI Music	77
What is ANSI Music?	77
Format	77
Music String Syntax	77
Notes	77
Octave	78
Tempo and Duration	78
Example	78
Configuration	78
Enable ANSI Music	78
Test Audio	78
Playback	78
Console	78
Web Interface	79
Remote GUI	79
Building with Audio	79
Linux	79
macOS	79
Troubleshooting	79
No Sound in Web Interface	79
No Sound in GUI	79
Music Sounds Wrong	80

Music Not Playing	80
Technical Details	80
Sequence Detection	80
Processing	80
Web Audio Implementation	80
Frequency Calculation	80
Hot Reload	81
How It Works	81
Triggering Reload	81
/reload Command	81
Keyboard Shortcut	81
External Signal	82
Updated Binary Detection	82
Limitations	82
TLS/SSL Connections	82
State Compatibility	82
Auto-Login	82
Message Suppression	82
Use Cases	83
Apply Code Changes	83
Deploy Bug Fixes	83
Configuration Changes	83
State File	83
Reload vs Restart	83
Troubleshooting	84
Reload Fails	84
Connections Lost After Reload	84
State Incompatibility	84
SIGUSR1 Not Working	84
Technical Details	84
State Serialization	84
Socket Preservation	84
Process Replacement	85
TLS Proxy	86
The Problem	86
The Solution	86
Configuration	86
Enable TLS Proxy	86
Behavior	86
How It Works	87
On TLS Connect	87
On Hot Reload	87
On Disconnect	87
Implementation Details	87
Functions	87
Stream Types	87

Saved State	87
Socket Path	88
Health Monitoring	88
Fallback Behavior	88
When to Use	88
Enable TLS Proxy	88
Skip TLS Proxy	88
Resource Usage	88
Platform Support	88
Troubleshooting	89
TLS Connection Lost After Reload	89
Proxy Not Spawning	89
Socket Permission Errors	89
Zombie Proxy Processes	89
Example Session	89
Spell Checking	91
Dictionary	91
Location	91
Dictionary Format	91
Installing Dictionary	91
Configuration	91
Enable/Disable	91
How It Works	92
Word Checking	92
Visual Feedback	92
Caching	92
Suggestions	92
Suggestion Algorithm	92
Contraction Support	92
Examples	93
Typing Flow	93
Using Suggestions	93
Cycling Suggestions	93
Limitations	93
Not Checked	93
Dictionary Limitations	93
Performance	94
Integration	94
Works With	94
Doesn't Work With	94
Technical Details	94
SpellChecker Struct	94
Levenshtein Distance	94
Word Boundaries	94
Troubleshooting	94
No Spell Checking	94
Too Many False Positives	95

Suggestions Not Working	95
Missing Dictionary	95
Android/Termux	96
Installation	96
Install Termux	96
Install Rust	96
Build Clay	96
Pre-built Binary	96
Limitations	97
Not Available	97
What Works	97
Android App	97
Installing the App	97
Notifications	97
Foreground Service	98
Tips for Termux	98
Storage Setup	98
Keyboard	98
Session Management	98
Battery Optimization	98
Wake Lock	98
Building Tips	99
Memory Usage	99
Storage Space	99
Build Time	99
Troubleshooting	99
Connection Issues	99
Display Issues	99
Performance	99
Build Failures	99
Settings Not Saving	100
Recommended Setup	100
Troubleshooting	101
Connection Problems	101
Can't Connect to MUD	101
Connection Drops	101
SSL/TLS Handshake Fails	101
Display Issues	101
Garbled Output	101
Colors Wrong	102
Screen Corruption	102
Wide Characters Display Wrong	102
Input Problems	102
Keys Not Working	102
Input Lag	102
Cursor Position Wrong	102

WebSocket/Web Interface	102
Can't Connect to Web Interface	102
Web Interface Shows "Disconnected"	103
Authentication Fails	103
Hot Reload	103
Reload Fails	103
Connections Lost After Reload	103
State Incompatible	103
Performance	103
High Memory Usage	103
High CPU Usage	104
Slow Startup	104
Settings	104
Settings Not Saving	104
Settings Corrupted	104
Build Problems	104
Musl Build Fails	104
GUI Build Fails	104
Missing Dependencies	105
Recovery	105
Complete Freeze	105
Lost All Settings	105
Crash on Startup	105
Appendices	106
A. Character Encodings	106
UTF-8	106
Latin1 (ISO-8859-1)	106
Fansi (CP437)	106
B. WebSocket Protocol	106
Message Types (Client → Server)	106
Message Types (Server → Client)	106
Authentication Flow	107
Password Hashing	107
C. Configuration File Format	107
Global Section	107
World Sections	108
Value Types	108
Encoding Values	108
Auto-Login Types	109
Keepalive Types	109
World Switch Modes	109
D. Command Reference	109
Slash Commands (/)	109
TF Commands (#)	109
E. ANSI Color Codes	110
Standard Colors (0-7)	110
Bright Colors (8-15)	110

256-Color Mode	110
True Color (24-bit)	110
F. Telnet Protocol Reference	111
IAC Commands	111
Option Codes	111

Introduction

What is Clay?

Clay is a modern terminal-based MUD (Multi-User Dungeon) client that combines the nostalgic feel of classic MUD clients with contemporary features and robust architecture. Built with Rust for performance and safety, Clay offers:

- **Multiple simultaneous world connections** - Connect to several MUDs at once
- **SSL/TLS support** - Secure connections to modern MUD servers
- **Rich ANSI color support** - Full 256-color and true color rendering
- **Web and GUI interfaces** - Access your MUD sessions from anywhere
- **TinyFugue compatibility** - Familiar scripting for TF veterans
- **Hot reload** - Update the client without losing connections

Key Features

Terminal User Interface (TUI)

- Clean, responsive interface built with ratatui/crossterm
- Unlimited scrollback buffer
- More-mode pausing for reading long output
- Spell checking with suggestions
- Command history and completion

Multi-World Management

- Independent output buffers per world
- Activity indicators for background worlds
- Quick world switching with keyboard shortcuts
- Per-world settings (encoding, auto-login, logging)

Automation & Scripting

- Actions/triggers with regex or wildcard patterns
- Capture group substitution in commands
- TinyFugue-compatible macro system
- Hooks for connect/disconnect events

Remote Access

- WebSocket server for remote clients
- Browser-based web interface
- Native GUI client (egui)
- Remote console client

Advanced Features

- Telnet protocol negotiation (SGA, TTYPE, NAWS, EOR)
- ANSI music playback
- Hot reload with connection preservation
- TLS proxy for SSL connection persistence
- File logging per world

Supported Platforms

Platform	Status	Notes
Linux x86_64	Full	Primary development platform
Linux ARM64	Full	Tested on Raspberry Pi
macOS (Intel)	Full	Native builds
macOS (Apple Silicon)	Full	Native ARM64 builds
Windows (WSL)	Full	Via Windows Subsystem for Linux
Android (Termux)	Partial	Some features unavailable

Architecture Overview

Clay is built on an async architecture using Tokio:

Main Event Loop

```
|
+-- Terminal Events (keyboard, resize)
+-- WebSocket Server (remote clients)
+-- World Tasks (one per connection)
|
v
```

App State (central state manager)

Each connected world has: - A reader task for incoming data - A writer task for outgoing commands - Independent output buffer and scroll state - Per-world settings and connection state

Installation

Pre-built Binaries

Pre-built binaries are available for common platforms. Download the appropriate binary for your system and make it executable:

```
# Linux x86_64 (static musl build - works on any Linux)
chmod +x clay-linux-x86_64-musl
./clay-linux-x86_64-musl

# Linux ARM64 (Termux)
chmod +x clay-linux-aarch64
./clay-linux-aarch64
```

Building from Source

Prerequisites

You need Rust installed. If you don't have it:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source ~/.cargo/env
```

Linux (Recommended: musl build)

The musl build produces a fully static binary that works on any Linux system regardless of glibc version:

```
# Install musl target (one-time setup)
rustup target add x86_64-unknown-linux-musl

# Build
cargo build --target x86_64-unknown-linux-musl \
  --no-default-features --features rustls-backend

# Binary location
./target/x86_64-unknown-linux-musl/debug/clay
```

Why musl instead of glibc?

- glibc static builds cause SIGFPE crashes during DNS resolution
- glibc's NSS requires dynamic loading even in static builds
- musl handles DNS resolution properly in fully static binaries

Why rustls instead of native-tls?

- native-tls requires OpenSSL, which needs cross-compilation setup for musl
- rustls is pure Rust and works seamlessly with musl builds

Linux with GUI Support

To build with the remote GUI client feature:

```
# Install dependencies (Debian/Ubuntu)
sudo apt install libxcb-render0-dev libxcb-shape0-dev libxcb-xfixes0-dev

# Build with GUI
cargo build --features remote-gui

# Build with GUI and audio support
sudo apt install libasound2-dev
cargo build --features remote-gui-audio
```

macOS

macOS builds use the native toolchain (no musl needed):

```
# Terminal client only
cargo build --no-default-features --features rustls-backend

# With GUI support
cargo build --features remote-gui

# With GUI and audio
cargo build --features remote-gui-audio

# Binary location
./target/debug/clay
```

Works on both Intel (x86_64) and Apple Silicon (aarch64) Macs.

Termux (Android)

```
# Install Rust in Termux
pkg install rust

# Build (no GUI features available)
```

```
cargo build --no-default-features --features rustls-backend
```

```
# Binary location
./target/debug/clay
```

Termux Limitations:

- Hot reload not available (exec() limited on Android)
- TLS proxy not available
- Process suspension (Ctrl+Z) not available
- Remote GUI client not available

Windows (via WSL)

Clay runs in Windows Subsystem for Linux:

```
# In WSL terminal
rustup target add x86_64-unknown-linux-musl
cargo build --target x86_64-unknown-linux-musl \
  --no-default-features --features rustls-backend
```

Feature Flags

Feature	Description
rustls-backend	Use rustls for TLS (recommended for musl)
native-tls-backend	Use native TLS (requires OpenSSL)
remote-gui	Build remote GUI client (requires display)
remote-gui-audio	GUI with ANSI music playback

Verifying the Installation

```
# Run Clay
./clay

# Check version
./clay --version

# Show help
./clay --help
```

On first run, Clay creates a settings file at `~/.clay.dat` and displays a colorful splash screen with quick command hints.

Command-Line Options

Option	Description
-v, --version	Show version and exit
--conf=<path>	Use alternate config file instead of ~/.clay.dat
--gui	Start with GUI interface (requires remote-gui feature)
--console	Start with console interface (default)
--remote=host:port	Connect to remote Clay instance
--console=host:port	Console mode connecting to remote instance
-D	Daemon mode (background server only)

Example using alternate config:

```
./clay --conf=/path/to/my-config.dat
```

Quick Start

First Launch

When you first start Clay, you'll see a colorful ASCII art splash screen:



Figure 1: Clay Startup Screen

The splash screen shows the Clay logo with the tagline “A 90s MUD client written today” and quick command hints.

Creating Your First World

1. Type `/worlds` to open the World Selector popup:
2. Press A or click “Add” to create a new world
3. Fill in the world settings:
 - **World name:** A friendly name (e.g., “MyMUD”)
 - **Hostname:** The server address (e.g., “mud.example.com”)

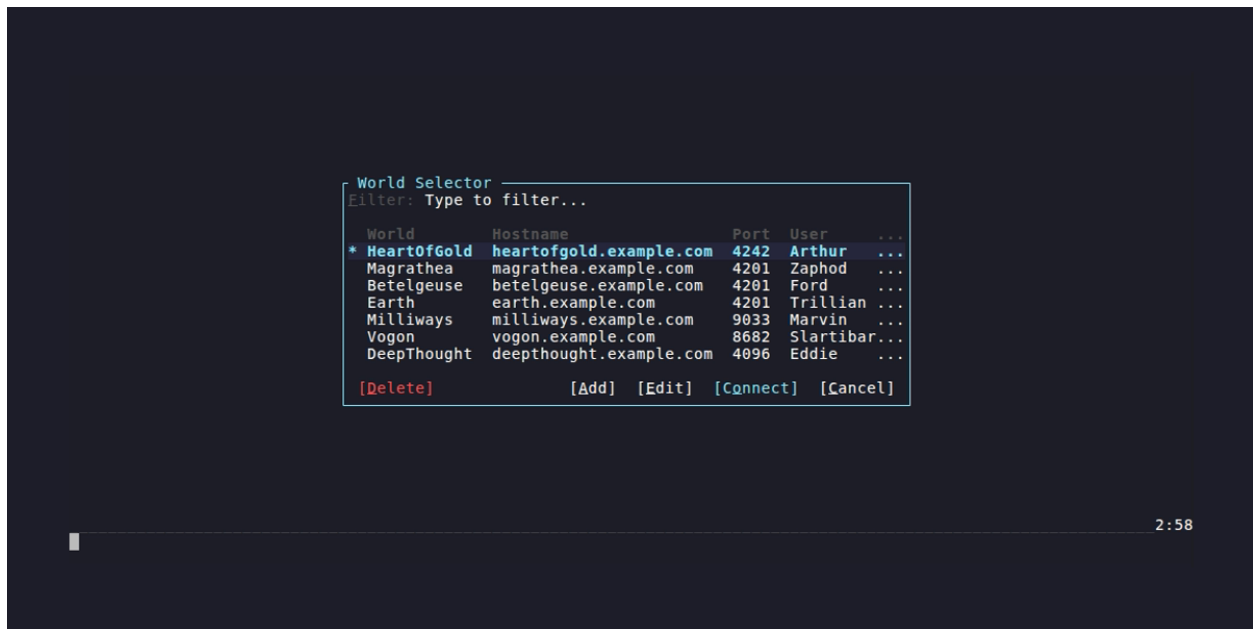


Figure 2: World Selector

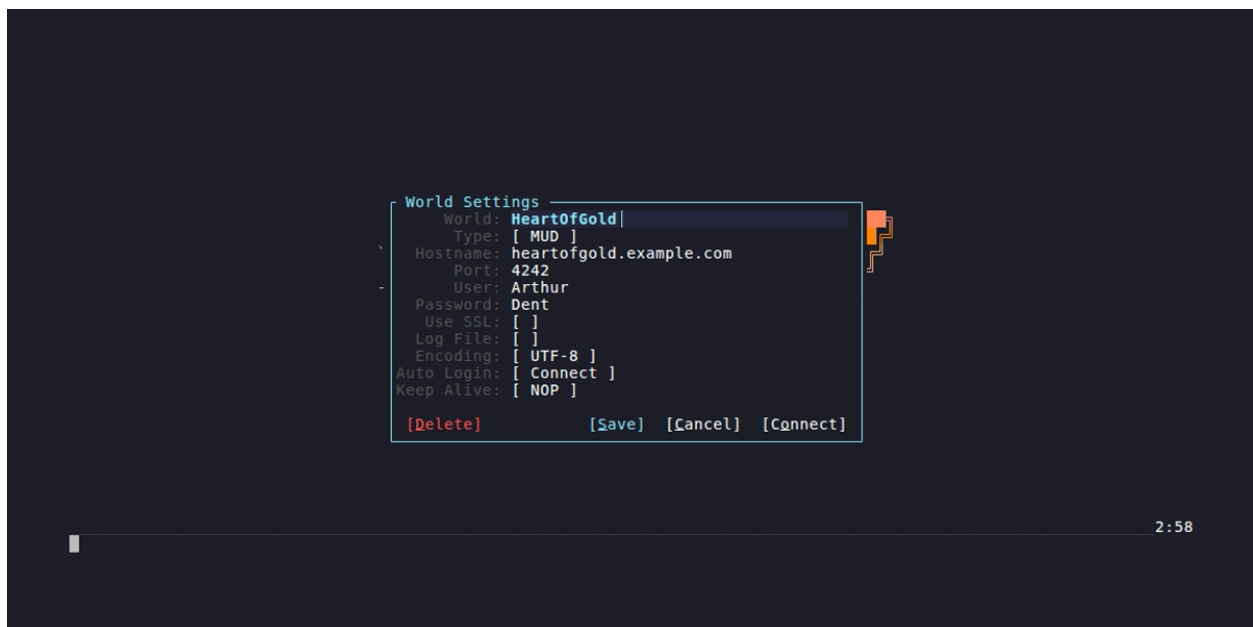


Figure 3: World Editor

- **Port:** The server port (e.g., 4000)
 - **Use SSL:** Enable if the server supports TLS
 - Optional: **User** and **Password** for auto-login
4. Press Enter on “Connect” or press 0 to save and connect

Connecting to a World

Once configured, there are several ways to connect:

```
/worlds MyMUD          # Connect to a world by name
/worlds -e MyMUD       # Edit a world's settings
/worlds -l MyMUD       # Connect without auto-login
```

Or use the World Selector (/worlds) and press Enter on your world.

Basic Navigation

Screen Layout

The interface has three main areas stacked vertically:

Output Area (top, largest) Displays MUD text with ANSI colors. Unlimited scroll-back.

Separator Bar (one line) Shows status, world name, activity count, and time.

Input Area (bottom, 1-15 lines) Where you type commands. Shows server prompt if detected.

Example separator bar:

```
More: 1234 * HeartOfGold      (Activity: 2)           14:30
```

Separator Bar Components

Position	Content	Description
Left	More: XXXX	Pending lines when paused
Left	Hist: XXXX	Lines scrolled back in history
Center-left	* WorldName	Connected world indicator
Center	(Activity: N)	Worlds with unseen output
Right	HH:MM	Current time

Sending Commands

Simply type your command and press Enter. For example:

```
look
north
```

say Hello, world!

Commands are sent to the currently selected world.

Essential Keyboard Shortcuts

Keys	Action
Up/Down	Switch between active worlds
PageUp/PageDown	Scroll output history
Tab	Release one screenful when paused
Ctrl+P/N	Navigate command history
Ctrl+U	Clear input line
F1	Open help popup
/quit	Exit Clay

Quick Command Reference

Command	Description
/help	Show help popup
/worlds	Open world selector
/worlds -e	Edit current world
/connections or /l	List connected worlds
/disconnect or /dc	Disconnect current world
/setup	Open global settings
/actions	Open actions/triggers editor
/quit	Exit Clay

Next Steps

- Read the **Interface Overview** chapter to understand all screen elements
- Check **Commands** for the full command reference
- Set up **Actions** to automate responses to MUD output
- Configure **Settings** for spell checking, themes, and more

Interface Overview

Screen Layout

Clay's terminal interface is divided into three main areas, stacked vertically:

1. **Output Area** - Takes most of the screen, displays MUD text
2. **Separator Bar** - Single line with status information
3. **Input Area** - Bottom section for typing commands (1-15 lines)

Example separator bar format:

_____	* HeartOfGold	(Activity: 2)	14:30
[status]	[world name]	[activity]	[time]

When paused or scrolled back:

More: 1234	* HeartOfGold	(Activity: 2)	14:30
Hist: 500	* HeartOfGold	(Activity: 2)	14:30

Output Area

The output area displays text from the MUD server:

- **ANSI color support:** Full 256-color and 24-bit true color
- **Unlimited scrollbar:** Output is stored in memory (grows with available RAM)
- **Word wrapping:** Long lines wrap intelligently, preserving ANSI codes
- **MUD tags:** Optional display of channel tags and timestamps (toggle with F2)

Separator Bar

The separator bar (made of underscores) contains several indicators:

Position	Component	Description
Left (10 chars)	Status	More: XXXX when paused, Hist: XXXX when scrolled, underscores otherwise
After status	Connection	Green ball (●) + world name when connected
Center	Activity	(Activity: X) count of worlds with unseen output

Position	Component	Description
Right	Time	Current time in HH:MM format (cyan)

Status indicator formatting: - Numbers up to 9999 shown as-is - 10000+ formatted as "10K", "999K" - 1000000+ shown as "Alot"

Input Area

The input area is where you type commands:

- **Prompt display:** Server prompts (detected via telnet GA/EOR) shown in cyan
- **Multi-line support:** Resize with Alt+Up/Down (1-15 lines)
- **Cursor:** Standard text cursor with left/right movement
- **Spell checking:** Misspelled words highlighted in red

More-Mode Pausing

When a lot of output arrives at once, Clay pauses to let you read. The separator bar displays More: XXXX (where XXXX is the number of pending lines) in red, indicating that output is being held back. This prevents fast-scrolling text from flying past before you can read it.

Trigger conditions: - Automatic: After (screen height - 2) lines of output without user input - Manual: When you scroll up with PageUp

Controls when paused: - Tab - Release one screenful of pending lines - PageDown - Release all pending and scroll to bottom - Escape then j - Jump to end, release all pending - Enter - Sends your command but does NOT release pending lines

Pending line counter: - Shows More: XXXX in the separator bar - Counts lines waiting to be displayed

Scrollback Navigation

Use PageUp/PageDown to scroll through output history:

- **PageUp:** Scroll back in history (enables more-pause mode)
- **PageDown:** Scroll forward (unpauses if you reach the bottom)
- **Escape+j:** Jump to the bottom and release all pending

When scrolled back, the separator shows Hist: XXXX indicating lines from bottom.

Themes

Clay supports light and dark themes for both console and GUI:

Console themes (set in /setup): - Dark (default): Light text on dark background - Light: Dark text on light background

GUI themes (set in /setup): - Dark (default): Dark mode interface - Light: Light mode interface

Colored Square Emoji

Clay renders colored square emoji (🟩🟦🟧🟨🟪🟫🟬🟭) with proper colors:

- **Console:** Converted to ANSI true-color block characters (█)
- **Web/GUI:** Native rendering with correct colors

This ensures consistent color display across all interfaces.

Display Width Handling

Clay correctly handles: - **Wide characters:** CJK characters, emoji (2 columns wide)
- **Zero-width characters:** Combining marks, zero-width spaces - **Mixed content:** Cursor positioning works correctly with any mix

Commands

All Clay commands start with /. These are client commands, not sent to the MUD server.

General Commands

/help

Opens the help popup with quick reference information.

/help

Controls in help popup: - Up/Down - Scroll content - PageUp/PageDown - Scroll faster
- Enter or Esc - Close popup

/quit

Exit Clay and disconnect all worlds.

/quit

You can also press Ctrl+C twice within 15 seconds to quit.

World Management

/worlds

Open the World Selector popup or manage worlds.

```
/worlds                # Open world selector popup
/worlds <name>          # Connect to world (create if doesn't exist)
/worlds -e [name]       # Edit world settings (current if no name)
/worlds -l <name>       # Connect without auto-login
```

World Selector Controls: - Up/Down - Navigate list - Enter - Connect to selected world - A - Add new world - E - Edit selected world - Tab - Cycle through buttons - Type to filter worlds

/connections (or /l)

List all connected worlds in a table format.

```
/connections  
/l
```

Output columns:

Column	Description
World	World name (* = current)
Unseen	Count of unseen lines
LastSend	Time since last command sent
LastRecv	Time since last data received
LastNOP	Time since last NOP keepalive
NextNOP	Time until next NOP

/disconnect (or /dc)

Disconnect the current world and close its log file.

```
/disconnect  
/dc
```

Sending Commands

/send

Send text to one or more worlds.

```
/send <text>                # Send to current world  
/send -w<world> <text>      # Send to specific world  
/send -W <text>              # Send to ALL connected worlds  
/send -n <text>              # Send without newline (no CR/LF)
```

Examples:

```
/send look                  # Send "look" to current world  
/send -wMyMUD say hello    # Send "say hello" to MyMUD  
/send -W ooc I'm here!     # Broadcast to all worlds
```

Settings Commands

/setup

Open the Global Settings popup.

```
/setup
```

Available settings: - More mode - Enable/disable more-style pausing - Spell check - Enable/disable spell checking - Temp convert - Auto-convert temperatures in input - World Switching - Cycling behavior (Unseen First, Alphabetical) - Show tags - Show/hide MUD tags (also F2) - Input height - Default input area height (1-15) - Console Theme - Dark or Light - GUI Theme - Dark or Light - TLS Proxy - Enable TLS connection preservation - ANSI Music - Enable music playback

/web

Open Web Settings popup for HTTP/WebSocket configuration.

/web

See the **Web Interface** chapter for details.

/actions

Open the Actions editor to create triggers.

/actions

See the **Actions** chapter for details.

Utility Commands

/reload

Hot reload the client with a new binary while preserving connections.

/reload

Also triggered by: - Ctrl+R keyboard shortcut - SIGUSR1 signal (kill -USR1 \$(pgrep clay))

See the **Hot Reload** chapter for details.

/testmusic

Play a test ANSI music sequence (C-D-E-F-G) to verify audio.

/testmusic

Requires ANSI Music enabled in /setup and a connected web/GUI client.

/notify

Send a notification to the Android app.

/notify <message>

Useful in action commands:

/notify Someone is paging you!

Command Completion

When input starts with /, press Tab to cycle through matching commands:

- Matches internal commands (/help, /disconnect, etc.)
- Matches manual actions (actions with empty patterns)
- Case-insensitive matching
- Arguments after the command are preserved

Example:

```
/wo[Tab]          -> /worlds  
/worlds -e[Tab] -> cycles through other /w commands
```

TinyFugue Commands

Clay includes a TinyFugue compatibility layer using the # prefix. This allows TF veterans to use familiar commands and scripting patterns.

Variables

#set / #unset

Set or remove global variables.

```
#set varname value      # Set variable
#unset varname          # Remove variable
```

#let

Set a local variable (within macro scope).

```
#let temp_value 100
```

#setenv

Export a variable to the environment.

```
#setenv MY_VAR
```

#listvar

List variables matching a pattern.

```
#listvar                # List all variables
#listvar hp*            # List variables starting with "hp"
```

Output Commands

#echo

Display a local message (not sent to MUD). Supports variable substitution.

```
#echo Hello, world!  
#echo Your HP is %{hp}
```

#send

Send text to the MUD.

```
#send look  
#send -w MyMUD say hello    # Send to specific world
```

#beep

Play terminal bell.

```
#beep
```

#quote

Send text without variable substitution.

```
#quote This %{var} stays literal
```

Expressions

#expr

Evaluate and display an expression result.

```
#expr 5 + 3          # Displays: 8  
#expr strlen("hello") # Displays: 5
```

#test

Evaluate expression as boolean (returns 0 or 1).

```
#test 5 > 3          # Returns: 1  
#test hp < 50        # Returns: 1 or 0
```

#eval

Evaluate expression and execute result as command.

```
#set cmd "look"  
#eval cmd          # Executes: look
```

Expression Operators

Category	Operators
Arithmetic	+ - * / %
Comparison	== != < > <= >=
Logical	& \ !
Regex	=~ !~
Ternary	? :

Built-in Functions

Function	Description
strlen(s)	String length
substr(s, start, len)	Substring
strcat(s1, s2)	Concatenate strings
tolower(s)	Lowercase
toupper(s)	Uppercase
rand()	Random number
time()	Current time
abs(n)	Absolute value
min(a, b)	Minimum
max(a, b)	Maximum

Control Flow

#if / #elseif / #else / #endif

Conditional execution.

```
# Single-line
#if (hp < 50) cast heal
```

```
# Multi-line
#if (hp < 25)
    cast 'cure critical'
#elif (hp < 50)
    cast heal
#else
    #echo HP is fine
#endif
```

#while / #done

While loop.

```
#while (count < 10)
    #echo Count: %{count}
```

```
#set count %{count} + 1
#done
```

#for / #done

For loop.

```
#for i 1 10
#echo Number: %{i}
#done
```

```
#for i 1 10 2          # Step by 2
#echo Odd: %{i}
#done
```

#break

Exit loop early.

```
#while (1)
#if (done) #break
#done
```

Macros (Triggers)

#def

Define a macro with optional trigger pattern.

```
#def name = command body
```

With trigger pattern

```
#def -t"pattern" name = command
```

With options

```
#def -t"* tells you: *" -mglob reply_tell = say Thanks, $1!
```

Options:

Option	Description
-t"pattern"	Trigger pattern
-mtype	Match type: simple, glob, regexp
-p priority	Execution priority (higher = first)
-F	Fall-through (continue checking other triggers)
-1	One-shot (delete after firing)
-n count	Fire only N times
-ag	Gag (suppress) matched line
-ah	Highlight matched line

Option	Description
-ab	Bold
-au	Underline
-E"expr"	Conditional expression
-c chance	Probability (0.0-1.0)
-w world	Restrict to specific world
-h event	Hook event
-b"key"	Key binding

#undef / #undefn / #undeft

Remove macros.

```
#undef name          # Remove by name
#undefn pattern      # Remove matching name pattern
#undeft pattern      # Remove matching trigger pattern
```

#list

List defined macros.

```
#list                # List all
#list heal*          # List matching pattern
```

#purge

Remove all macros (or matching pattern).

```
#purge               # Remove all
#purge temp_*        # Remove matching pattern
```

Hooks

Define macros that fire on events:

```
#def -hCONNECT auto_look = look
#def -hDISCONNECT goodbye = #echo Disconnected!
```

Available events: - CONNECT - When connected to a world - DISCONNECT - When disconnected - LOGIN - After auto-login completes - PROMPT - When prompt is received - SEND - Before command is sent - ACTIVITY - When activity occurs in background world - WORLD - When world changes - RESIZE - When terminal resizes - LOAD - When script is loaded - REDEF - When macro is redefined - BACKGROUND - When world goes to background

Key Bindings

#bind / #unbind

Bind keys to commands.

```
#bind F5 = cast heal  
#unbind F5
```

Key names: - F1 - F12 - ^A - ^Z (Ctrl+letter) - @a - @z (Alt+letter) - PgUp, PgDn, Home, End, Insert, Delete

File Operations

#load

Load a TF script file.

```
#load scripts/my_triggers.tf
```

#save

Save macros to a file.

```
#save macros_backup.tf
```

#lcd

Change local directory.

```
#lcd /home/user/mud
```

Variable Substitution

Use %{varname} or %varname in commands:

```
#set target orc  
#send kill %{target}
```

Special variables: - %1 - %9 - Positional parameters from trigger match - %* - All positional parameters - %L - Text left of match - %R - Text right of match - %% - Literal percent sign

Examples

Auto-heal Trigger

```
#def -t"Your health: *" -mglob heal_check = \  
  #if ({1} < 50) cast heal
```

Connect Hook

```
#def -hCONNECT auto_look = look
```

Conditional Response

```
#def -t"* tells you: *" -mglob tell_response = \  
  #if ("{1}" =~ "friend") say Hi {1}!
```

Loop Example

```
#def train_all = \  
  #for i 1 5; \  
    train str; \  
  #done
```

Keyboard Shortcuts

World Switching

Keys	Action
Up / Down	Cycle through active worlds (connected OR with unseen output)
Shift+Up / Shift+Down	Cycle through all worlds (including disconnected)
Escape then w	Switch to world with activity
Alt+w	Switch to world with activity (same as Escape+w)

Activity priority: Oldest pending lines → Unseen output → Previous world

Input Area

Keys	Action
Left / Right	Move cursor
Ctrl+B / Ctrl+F	Move cursor (alternative)
Ctrl+Up / Ctrl+Down	Move cursor up/down in multi-line input
Alt+Up / Alt+Down	Resize input area (1-15 lines)
Ctrl+U	Clear entire input
Ctrl+W	Delete word before cursor
Ctrl+A	Jump to start of line
Home	Jump to start of line
End	Jump to end of line
Ctrl+P	Previous command in history
Ctrl+N	Next command in history
Ctrl+Q	Spell suggestions / cycle and replace
Tab	Command completion (when input starts with / or #)
Enter	Send command

Output Scrollback

Keys	Action
PageUp	Scroll back in history (enables more-pause)
PageDown	Scroll forward (unpauses if at bottom)
Tab	Release one screenful when paused; scroll down when viewing history
Escape then j	Jump to end, release all pending lines

More-Mode (When Paused)

Keys	Action
Tab	Release one screenful of pending lines
PageDown	Release all pending and scroll to bottom
Escape then j	Jump to end, release all pending
Enter	Send command (does NOT release pending)

General

Keys	Action
F1	Open help popup
F2	Toggle MUD tag display (show/hide channel tags and timestamps)
F4	Open filter popup to search output
F8	Toggle action pattern highlighting
Ctrl+C	Press twice within 15 seconds to quit
Ctrl+L	Redraw screen (filters out client-generated output)
Ctrl+R	Hot reload (same as /reload)
Ctrl+Z	Suspend process (use fg to resume)

Popup Controls (All Pops)

Keys	Action
Up / Down	Navigate between fields
Tab / Shift+Tab	Cycle through buttons only
Left / Right	Navigate between buttons; change select/toggle values
Enter	Edit text field / Toggle option / Activate button

Keys	Action
Space	Toggle boolean / Cycle options
Esc	Close popup or cancel text edit

Button shortcuts: Letters are highlighted in button labels (e.g., **S**ave, **C**ancel, **D**elete)

Filter Popup (F4)

Keys	Action
Type text	Filter output to matching lines
Backspace / Delete	Edit filter text
Left / Right	Move cursor in filter text
Home / End	Jump to start/end of filter
PageUp / PageDown	Scroll through filtered results
Esc / F4	Close filter and restore normal view

Help Popup (F1)

Keys	Action
Up / Down	Scroll one line
PageUp / PageDown	Scroll multiple lines
0	Highlight Ok button
Enter / Esc	Close popup

World Selector (/worlds)

Keys	Action
Up / Down	Navigate world list
Tab / Shift+Tab	Cycle between list and buttons
Enter	Connect to selected world / Activate button
Left / Right	Move between buttons
A	Add new world
E	Edit selected world
/	Focus filter box
Esc	Close popup

Actions List (/actions)

Keys	Action
Up / Down	Navigate action list
Space	Toggle enable/disable selected action
Enter	Edit selected action
Tab	Cycle between list and buttons
A	Add new action
E	Edit selected action
D	Delete selected action
C	Cancel/close
F or /	Focus filter box
Esc	Close popup

Confirmation Dialogs

Keys	Action
Left / Right / Up / Down / Tab	Toggle between Yes/No
Y	Select Yes
N	Select No
Enter	Confirm selection
Esc	Cancel and close

Remote GUI Client Additional Shortcuts

Keys	Action
Ctrl+L	Open World List popup
Ctrl+E	Open World Editor for current world
Ctrl+S	Open Setup popup
Ctrl+O	Connect current world
Ctrl+D	Disconnect current world

Settings

Clay has three levels of settings: global settings, per-world settings, and web/server settings.

Global Settings (/setup)

Open with /setup command:



Figure 4: Global Settings

Available Options

Setting	Description	Default
More mode	Enable more-style pausing	On
Spell check	Enable spell checking	On
Temp convert	Auto-convert temperatures in input	Off
World Switching	World cycling behavior	Unseen First

Setting	Description	Default
Show tags	Show MUD tags at line start	Off
Input height	Default input area height (1-15)	3
Console Theme	Terminal color scheme	Dark
GUI Theme	Remote GUI color scheme	Dark
TLS Proxy	Preserve TLS connections across reload	Off
ANSI Music	Enable ANSI music playback	Off

World Switching Modes

Unseen First: - If any OTHER world has unseen output, switch to the world that received unseen output first (oldest) - Otherwise, switch alphabetically

Alphabetical: - Always switch to the alphabetically next world by name - Wraps from last world back to first

Temperature Conversion

When enabled (and F2/Show tags is on), temperatures typed in the input area are auto-converted: - Type 32F → Shows 32F (0C) - Type 100C → Shows 100C (212F)

Useful for international MUD players.

Per-World Settings (/worlds -e)

Open with /worlds -e or edit from World Selector:

Connection Settings

Setting	Description
World name	Display name for this world
Hostname	Server address (e.g., mud.example.com)
Port	Server port (e.g., 4000)
Use SSL	Enable TLS/SSL connection

Authentication Settings

Setting	Description
User	Username for auto-login
Password	Password for auto-login (plaintext)
Auto login	Login method (see below)

Auto-login types:

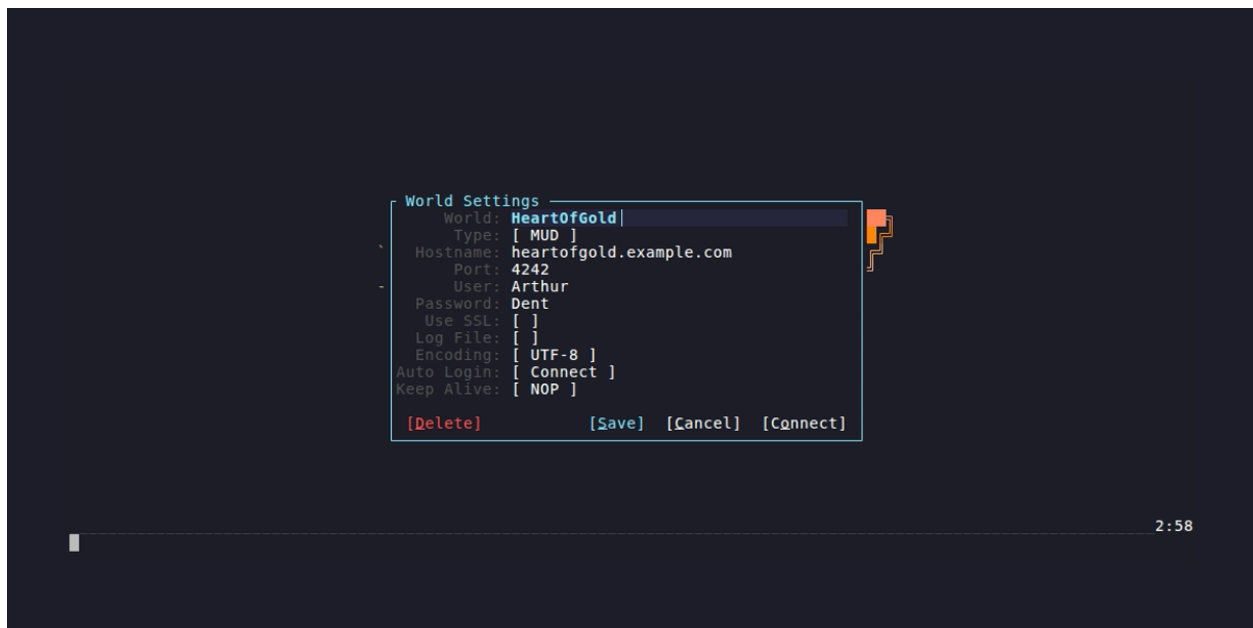


Figure 5: World Editor

Type	Behavior
Connect	Sends connect <user> <password> after 500ms
Prompt	Sends username on first telnet prompt, password on second
MOO_prompt	Like Prompt, but sends username again on third prompt

Auto-login only triggers if BOTH username AND password are configured.

Keep-Alive Settings

Setting	Description
Keep alive	Keepalive type (NOP, Custom, Generic)
Keep alive cmd	Custom command (when type is Custom)

Keepalive types:

Type	Behavior
NOP	Sends telnet NOP command (IAC NOP) every 5 minutes
Custom	Sends your custom command
Generic	Sends help commands ##_idler_message_<rand>_###

Other Settings

Setting	Description
Log file	Path to output log file (append mode)
Encoding	Character encoding (UTF-8, Latin1, Fansi)

Encoding types:

Type	Description
UTF-8	Standard UTF-8 (default)
Latin1	ISO-8859-1 for older MUDs
Fansi	CP437-like with box drawing characters

Web Settings (/web)

Open with /web command:

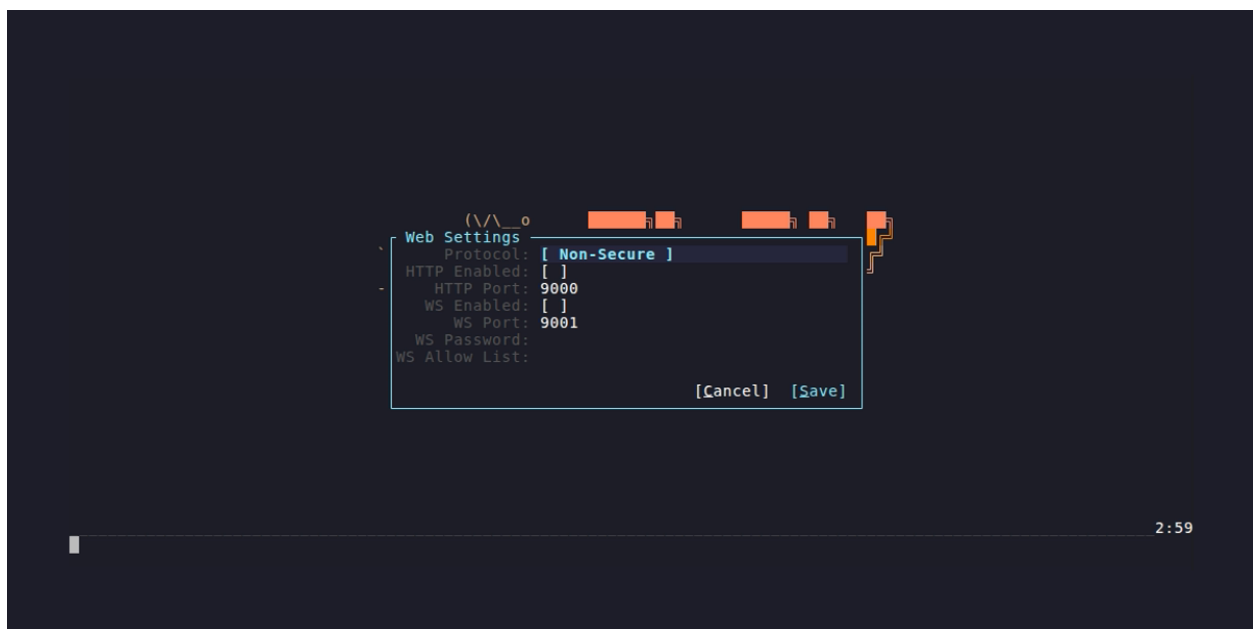


Figure 6: Web Settings

WebSocket Server (Secure)

Setting	Description	Default
WS enabled	Enable secure WebSocket server	Off
WS port	Port for wss:// connections	9002
WS password	Authentication password	(required)
WS Allow List	CSV of IPs that can be whitelisted	(empty)

Setting	Description	Default
TLS Cert File	Path to TLS certificate	(required for TLS)
TLS Key File	Path to TLS private key	(required for TLS)
WS Use TLS	Enable TLS for WebSocket	Off

WebSocket Server (Non-Secure)

Setting	Description	Default
WS Nonsecure	Enable non-secure WebSocket	Off
WS NS port	Port for ws:// connections	9003

HTTP/HTTPS Web Interface

Setting	Description	Default
HTTP enabled	Enable HTTP web server	Off
HTTP port	Port for HTTP	9000
HTTPS enabled	Enable HTTPS web server	Off
HTTPS port	Port for HTTPS	9001

Note: HTTP automatically starts the non-secure WebSocket server if needed.

Settings Persistence

Settings are automatically saved to `~/.clay.dat`:

- Global settings saved when closing the Settings popup
- World settings saved when closing the World Editor
- File format is INI-like with `[global]` and `[world:name]` sections

Example `~/.clay.dat`

```
[global]
more_mode_enabled=true
spell_check_enabled=true
world_switch_mode=unseen_first
websocket_enabled=false
websocket_port=9002

[world:MyMUD]
hostname=mud.example.com
port=4000
user=myname
```

```
password=secret  
use_ssl=true  
encoding=utf8  
auto_connect_type=connect
```

Actions (Triggers)

Actions are automated triggers that match incoming MUD output and execute commands. They're essential for automating repetitive tasks like healing, responding to tells, or filtering spam.

Opening the Actions Editor

/actions



Figure 7: Actions List

Action List Controls

Keys	Action
Up/Down	Navigate list
Space	Toggle enable/disable

Keys	Action
Enter	Edit selected action
A	Add new action
E	Edit selected action
D	Delete selected action
/ or F	Focus filter box
Esc	Close popup

Creating an Action

Press A to add a new action:

Action Fields

Field	Description
Name	Action name (also used for manual invocation)
World	Restrict to specific world (empty = all worlds)
Match Type	Regexp or Wildcard
Pattern	Trigger pattern (empty = manual-only)
Command	Commands to execute (multiline, semicolon-separated)
Enabled	Whether the action is active

Match Types

Regexp (Regular Expression)

Full regex syntax for precise pattern matching:

Pattern	Matches
^You say	Lines starting with "You say"
tells you:	Lines containing "tells you:"
HP: (\d+)	Captures the number after "HP:"
^\[.*\]	Lines starting with bracketed text

Wildcard (Glob-style)

Simple pattern matching where: - * matches any sequence of characters - ? matches any single character - * and \? match literal asterisk/question mark

Pattern	Matches
tells you	Any line containing "tells you"

Pattern	Matches
* pages you:*	Lines with "pages you:" anywhere
You feel hungry*	Lines starting with "You feel hungry"

Capture Groups

When a pattern matches, you can use captured text in commands:

Variable	Description
\$0	The entire matched text
\$1 - \$9	Captured groups from the pattern

Regex Capture Groups

Use parentheses to create groups:

```
Pattern: ^(\w+) tells you: (.*)$
Input:  "Bob tells you: Hello!"
$1 = "Bob"
$2 = "Hello!"
```

Wildcard Capture Groups

Each * and ? becomes a capture group automatically:

```
Pattern: * tells you: *
Input:  "Bob tells you: Hello!"
$1 = "Bob"
$2 = "Hello!"
```

Command Examples

Basic Auto-Response

```
Name: tell_thanks
Pattern: * tells you: *
Command: tell $1 Thanks for the message!
```

Multi-Command Action

Separate commands with semicolons:

```
Name: heal_self
Pattern: Your health drops to *
Command: cast heal;drink potion;say Ouch!
```



Figure 8: Action Editor

Notification Action

Name: page_alert
 Pattern: *pages you*
 Command: /notify Page received: \$0

Gagging (Hiding Lines)

Use /gag in the command to hide matched lines:

Name: hide_spam
 Pattern: You hear a loud noise*
 Command: /gag

Gagging behavior: - Gagged lines are hidden from normal display - Gagged lines are still stored (visible with F2/Show tags) - Useful for filtering spam while preserving history

Combined Gag and Command

Name: quiet_combat
 Pattern: *misses you*
 Command: /gag;#set misses %{misses}+1

Manual Invocation

Actions can be invoked manually by typing /actionname:

```
/heal_self          # Run the heal_self action
/greet Bob          # Run greet with $1="Bob"
```

For manual actions: - \$1-\$9 are space-separated arguments - \$* is all arguments combined - Actions with empty patterns are manual-only

Example Manual Action

Name: greet
Pattern: (empty)
Command: bow \$1;say Hello, \$1!
Usage: /greet Alice sends “bow Alice” and “say Hello, Alice!”

F8 Pattern Highlighting

Press F8 to highlight lines matching any action pattern:

- Matched lines get a dark background color
- Useful for debugging action patterns
- Commands are NOT executed in highlight mode

World-Specific Actions

Set the World field to restrict an action to one world:

Name: mymud_heal
World: MyMUD
Pattern: You are bleeding
Command: bandage

This action only triggers for output from the “MyMUD” world.

Best Practices

1. **Test patterns first:** Use F8 highlighting to verify matches
2. **Use specific patterns:** Avoid overly broad patterns like *
3. **Order by specificity:** More specific patterns should have higher priority
4. **Gag carefully:** You can always see gagged lines with F2
5. **Name descriptively:** Action names become manual commands

Common Action Recipes

Auto-Heal When Low

Name: auto_heal
Pattern: HP: (\d+)/

Match: Regexp
Command: #if (\$1 < 50) cast heal

Reply to Tells

Name: afk_reply
Pattern: * tells you: *
Command: tell \$1 I'm AFK, back soon!

Channel Logger

Name: log_ooc
Pattern: \[OOC\] *
Match: Regexp
Command: #echo [Logged] \$0

Combat Spam Filter

Name: filter_misses
Pattern: * misses *
Command: /gag

Multi-World System

Clay supports multiple simultaneous MUD connections, each with independent state and settings.

Understanding Worlds

Each world has: - Independent output buffer (unlimited scrollback) - Independent scroll position - Independent connection state - Per-world settings (encoding, auto-login, logging) - Unseen line counter for background activity

World Switching

Active World Cycling (Up/Down)

The Up and Down arrow keys cycle through **active** worlds: - Connected worlds - Disconnected worlds with unseen output

Disconnected worlds without unseen lines are skipped.

All World Cycling (Shift+Up/Down)

Shift+Up and Shift+Down cycle through ALL worlds, including disconnected ones without unseen output.

Activity-Based Switching (Escape+w or Alt+w)

Switches to the world with activity, using this priority: 1. World with oldest pending lines (paused output) 2. World with unseen output (oldest first) 3. Previous world

World Switching Mode

Configure in /setup:

Unseen First (default): - If any OTHER world has unseen output, switch to the one that received unseen output first - Otherwise, switch alphabetically

Alphabetical: - Always switch to the alphabetically next world - Wraps from last to first

Activity Indicators

Separator Bar

The separator bar shows activity at a glance:

More: 1234 ☐ CurrentWorld__ (Activity: 2) _____ 14:30

- **More: XXXX:** Pending lines when current world is paused
- **Hist: XXXX:** Lines scrolled back in history
- ☐ **WorldName:** Currently viewing this connected world
- **(Activity: 2):** Two OTHER worlds have unseen output

Unseen Line Tracking

When output arrives for a non-current world: - unseen_lines counter increments - first_unseen_at timestamp is set (for “Unseen First” mode)

When you switch to a world: - Its unseen count is cleared - All clients are notified (console, web, GUI stay in sync)

Managing Connections

Listing Connections

Use /connections or /l to see all connected worlds:

/connections

Output:

World	Unseen	LastSend	LastRecv	LastNOP	NextNOP
*MyMUD		1m 30s	5s	4m 30s	30s
OtherMUD	15	10m	30s	5m	idle

Columns: - * marks the current world - Unseen: Count of unseen lines (empty if 0) - LastSend: Time since last command you sent - LastRecv: Time since last data received - LastNOP: Time since last keepalive sent - NextNOP: Time until next keepalive

Connecting to Worlds

```
/worlds MyMUD      # Switch to and connect
/worlds -l MyMUD    # Connect without auto-login
```

Or use the World Selector (/worlds) and press Enter.

Disconnecting

```
/disconnect        # Disconnect current world
/dc                # Short form
```

The world remains in your list for reconnection.

Per-World Features

Output Buffers

Each world maintains its own output buffer: - Unlimited size (grows with available memory) - Independent scroll position per world - Independent more-mode pause state

Prompts

Telnet prompts (GA/EOR) are per-world: - Stored in the world's state - Only displayed when viewing that world - Cleared when you send a command

Logging

Configure per-world in the World Editor: - Log file opened on connect (append mode) - All received output written to log - Log file closed on disconnect

Encoding

Each world can have different character encoding: - UTF-8 (default) - Latin1 (ISO-8859-1) - Fansi (CP437-like)

Cross-Interface Sync

World state syncs across all interfaces:

Event	Console	Web	GUI
Output arrives	Shows in buffer	Broadcast	Broadcast
World switched	Clears unseen	Notified	Notified
Connect/Disconnect	Updates status	Notified	Notified

Each interface can independently: - View different worlds - Have different scroll positions - Be at different pause states

But they all see the same underlying data.

Tips for Multi-World Usage

1. **Use activity switching:** Escape+w quickly jumps to worlds that need attention
2. **Watch the Activity indicator:** The separator bar shows how many worlds have unseen output
3. **Configure World Switching mode:**
 - “Unseen First” if you prioritize responding to activity

- “Alphabetical” for predictable navigation
4. **Use world-specific actions:** Set the World field in actions to avoid triggers firing in the wrong context
 5. **Name worlds clearly:** Names are used for navigation and action filtering

Web Interface

Clay includes a browser-based client that connects via WebSocket to control your MUD sessions from anywhere.

Setup

1. Configure WebSocket Server

Open /web settings:

/web

Set up either secure (recommended) or non-secure WebSocket:

Secure WebSocket (wss://): - Enable "WS enabled" - Set "WS port" (default: 9002)
- Set "WS password" (required) - Optionally configure TLS certificate/key

Non-Secure WebSocket (ws://): - Enable "WS Nonsecure" - Set "WS NS port" (default: 9003)

2. Enable HTTP Server

Still in /web:

For HTTP (ws://): - Enable "HTTP enabled" - Set "HTTP port" (default: 9000)

For HTTPS (wss://): - Enable "HTTPS enabled" - Set "HTTPS port" (default: 9001) - Requires TLS cert/key configured

3. Access the Web Interface


Open in your browser: - HTTP: `http://your-server:9000` - HTTPS: `https://your-server:9001`

Enter your WebSocket password to authenticate.

Features

Full MUD Client

The web interface provides a complete MUD experience:

- **ANSI color rendering:** Xubuntu Dark palette, 256-color and true color
- **Shade character blending:**  rendered with proper color blending
- **Clickable URLs:** Links are cyan, underlined, open in new tab
- **More-mode pausing:** Tab to release, same as console
- **Command history:** Ctrl+P/N navigation
- **Multiple worlds:** Full world switching support

Toolbar

The toolbar at the top provides quick access:

Left side: - Hamburger menu (≡) - PgUp button - PgDn button

Right side: - ▲ Previous world - ▼ Next world

Font slider: Adjust text size

Hamburger Menu

Click the hamburger icon for:

Option	Description
Worlds List	Show connected worlds
World Selector	Open world selector popup
Actions	Open actions editor
Settings	(Android) Open server settings
Toggle Tags	Show/hide MUD tags (F2)
Toggle Highlight	Show action pattern matches (F8)
Resync	Request full state refresh
Clay Server	(Android) Disconnect and reconfigure

World Selector

Access via hamburger menu or /worlds command:

- Filter worlds by name/hostname
- Arrow keys to navigate
- Enter to switch
- Shows connection status

Connected Worlds List

Access via hamburger menu or /connections:

- Shows all connected worlds
- Unseen count per world
- Arrow keys to navigate
- Enter to switch

Actions Editor

Access via hamburger menu or /actions:

- Full action editing capability
- Same interface as console
- Create, edit, delete actions

Keyboard Shortcuts

Keys	Action
Up/Down	Switch between active worlds
PageUp/PageDown	Scroll output
Tab	Release screenful when paused; scroll down otherwise
Escape+j	Jump to end
Ctrl+P/N	Command history
Ctrl+U	Clear input
Ctrl+W	Delete word
Ctrl+A	Move to start of line
Alt+Up/Down	Resize input area
F2	Toggle MUD tags
F4	Open filter popup
F8	Toggle action highlighting
Enter	Send command
Escape	Close popup

Mobile Support

The web interface is optimized for mobile devices:

Layout Adjustments

- Fixed toolbar stays visible during scrolling
- Uses 100dvh for proper mobile viewport
- Smooth scrolling on iOS
- Proper keyboard handling

Touch Controls

- Tap toolbar buttons for common actions

- Swipe to scroll output
- Long-press for text selection

Visibility Handling

- Auto-resync when tab becomes visible
- Handles sleep/wake properly
- Reconnects if connection dropped

Security

Password Protection

- WebSocket password required for all connections
- Password hashed with SHA-256 before transmission
- Empty password disables the server

Allow List / Whitelisting

Configure “WS Allow List” in /web as a CSV of IP addresses:

192.168.1.100,192.168.1.101

Whitelisting behavior:

1. Client from allow-list IP connects → must authenticate with password
2. After successful auth → that IP is whitelisted
3. Future connections from that IP → auto-authenticated
4. Different allow-list IP authenticates → previous whitelist cleared
5. Non-allow-list IPs must always use password

Use case: Authenticate once from home, then reconnect without password. Moving locations automatically requires re-authentication.

TLS/SSL

For secure connections:

1. Obtain TLS certificate and key
2. Configure paths in /web:
 - TLS Cert File
 - TLS Key File
3. Enable “WS Use TLS”
4. Use HTTPS and wss:// URLs

Cross-Interface Sync

The web interface stays synchronized with console and GUI:

Event	Behavior
Output arrives	All clients receive it
World switched	All clients notified
Unseen cleared	Broadcast to all clients
Activity count	Broadcast when changed

Each client can independently: - View different worlds - Have different scroll positions
 - Use different tag visibility

Troubleshooting

Can't Connect

1. Verify server is running (check Clay console)
2. Check firewall allows the port
3. Verify password is correct
4. Try non-secure WebSocket first (ws://)

Connection Drops

1. Check network stability
2. Enable keepalive in world settings
3. Use "Resync" from hamburger menu

Display Issues

1. Try different browser
2. Clear browser cache
3. Use "Resync" to refresh state

Remote GUI Client

Clay includes a native graphical client built with egui that connects to a running Clay instance via WebSocket.

Building

The remote GUI requires display libraries and is built with the `remote-gui` feature:

```
# Linux (requires X11 or Wayland dev libraries)
sudo apt install libxcb-render0-dev libxcb-shape0-dev libxcb-xfixes0-dev
cargo build --features remote-gui

# With audio support (for ANSI music)
sudo apt install libasound2-dev
cargo build --features remote-gui-audio

# macOS (no extra dependencies needed)
cargo build --features remote-gui
```

Note: The `remote-gui` feature cannot be built in headless environments.

Running

```
./clay --remote=hostname:port
```

Examples:

```
./clay --remote=localhost:9002      # Local secure WebSocket
./clay --remote=mud.server.com:9002 # Remote server
```

Login Screen

On launch, you'll see the login screen with the Clay logo:

- Enter the WebSocket password
- Click "Connect" or press Enter

- If whitelisted (see Web Interface chapter), auto-connects

Interface Overview

World Tabs

Tabs at the top show all worlds: - ● Connected (filled circle) - ○ Disconnected (empty circle) - Click tab to switch worlds

Output Area

The main area displays MUD output: - Full ANSI color support - Scrollable with mouse wheel or PageUp/PageDown - Clickable URLs (underlined, opens browser)

Input Field

At the bottom: - Server prompt displayed in input area - Type commands and press Enter - Supports multi-line input

Status Bar

Shows connection state and current world info.

Hamburger Menu

Click the menu icon for:

Option	Description
Worlds List	Show connected worlds (Ctrl+L)
World Selector	Open world selector popup
World Editor	Edit current world (Ctrl+E)
Setup	Open settings (Ctrl+S)
Font	Adjust font size
Toggle Tags	Show/hide MUD tags (F2)
Toggle Highlight	Action pattern highlighting (F8)
Resync	Request full state refresh

Keyboard Shortcuts

World Switching

Keys	Action
Up/Down	Cycle through active worlds
Shift+Up/Down	Cycle through all worlds

Input Area

Keys	Action
Ctrl+U	Clear input
Ctrl+W	Delete word
Ctrl+A	Move to start
Ctrl+P/N	Command history
Ctrl+Up/Down	Move cursor in multi-line input
Alt+Up/Down	Resize input area
Enter	Send command

Output

Keys	Action
PageUp/PageDown	Scroll output

Display

Keys	Action
F2	Toggle MUD tags
F4	Open filter popup
F8	Toggle action highlighting
Esc	Close filter popup

Menu Shortcuts

Keys	Action
Ctrl+L	Open World List
Ctrl+E	Edit current world
Ctrl+S	Open Setup
Ctrl+O	Connect current world
Ctrl+D	Disconnect current world

Filter Popup

Press F4 to open the filter popup:

- Type text to filter output (case-insensitive)
- Only matching lines are shown
- ANSI codes stripped for matching, preserved in display
- Esc or F4 to close

Debug Selection

For troubleshooting ANSI color issues:

1. Highlight text in the output area
2. Right-click to open context menu
3. Select "Debug Selection"
4. A popup shows the raw text with escape codes visible
5. ESC character shown as <esc>
6. Copy button available

Features

Color Support

- Full 256-color palette
- 24-bit true color
- Xubuntu Dark color scheme

Colored Square Emoji

Colored square emoji (🟩🟦🟨🟧🟪🟫) are rendered as colored rectangles for consistent appearance.

Word Wrapping

Long words break at sensible points: - [] () , \ / - & = ? - Preserves URLs and filenames

ANSI Music

With remote-gui-audio feature: - ANSI music sequences are played - Uses rodio library - Requires ALSA on Linux

Themes

Configure GUI theme in /setup: - **Dark:** Dark background, light text (default) - **Light:** Light background, dark text

Synchronization

The GUI stays synchronized with console and web clients:

- All output is shared
- Unseen counts sync
- Activity indicators match

- World switching is GUI-local (doesn't affect other clients)

Troubleshooting

Build Errors

“Could not find X11 libraries”

```
sudo apt install libxcb-render0-dev libxcb-shape0-dev libxcb-xfixes0-dev
```

“Could not find ALSA” (for audio)

```
sudo apt install libasound2-dev
```

Connection Issues

1. Verify Clay console is running with WebSocket enabled
2. Check the port matches your /web configuration
3. Try connecting with the web interface first to verify password

Display Issues

1. Ensure X11 or Wayland is running
2. Check DISPLAY environment variable
3. Try different theme (dark vs light)

No Sound

1. Verify remote-gui-audio feature was enabled at build
2. Check ANSI Music is enabled in /setup
3. Use /testmusic to verify audio works

Remote Console Client

Clay includes a remote console client that provides the full terminal interface while connecting to a master Clay instance via WebSocket.

Running

```
./clay --console=hostname:port
```

Examples:

```
./clay --console=localhost:9002      # Local secure WebSocket  
./clay --console=mud.server.com:9002 # Remote server
```

No special build features required - works with the standard musl build.

Use Cases

- Access your MUD sessions from another terminal/SSH
- Run Clay on a server, connect from anywhere
- Have multiple terminal views of the same sessions

Interface

The remote console provides the identical interface to the main console:

- Full terminal UI with ratatui/crossterm
- All popup dialogs (help, menu, settings, world selector, etc.)
- Output scrollbar with PageUp/PageDown
- More-mode pausing
- World switching
- Command history
- Spell checking (if enabled on master)

Key Differences from Main Console

No Direct Connections

- All MUD connections go through the master instance
- Commands are forwarded to the master
- Output is received via WebSocket

Local World Switching

- Switching worlds only affects your view
- Doesn't change what the master or other clients see
- Each remote console can view different worlds

Synchronized State

- Output history is shared across all clients
- Unseen counts sync when any client views a world
- Settings changes affect all clients

Keyboard Shortcuts

All standard console shortcuts work:

Keys	Action
Up/Down	Switch active worlds
Shift+Up/Down	Switch all worlds
PageUp/PageDown	Scroll output
Tab	More-mode release / scroll
Escape+j	Jump to end
Ctrl+P/N	Command history
Ctrl+U	Clear input
F1	Help popup
F2	Toggle MUD tags
F4	Filter popup
F8	Action highlighting
Ctrl+L	Redraw screen

Special Commands

Command	Description
/menu	Open hamburger menu popup
/version	Display version information

Menu Popup

Access with /menu or hamburger icon:

Option	Description
Worlds List	Connected worlds
World Selector	All worlds
World Editor	Edit current world
Setup	Global settings
Web	Web settings
Actions	Actions editor
Toggle Tags	Show/hide MUD tags
Toggle Highlight	Action highlighting
Resync	Refresh from master

Popups

All popup dialogs work in remote console mode:

- **Help** (F1): Scrollable help content
- **Settings** (/setup): Global settings
- **Web Settings** (/web): WebSocket/HTTP configuration
- **World Selector** (/worlds): Browse and switch worlds
- **World Editor** (/worlds -e): Edit world settings
- **Actions** (/actions): Edit triggers
- **Filter** (F4): Search output

Popups use the unified popup system with consistent controls.

Authentication

Uses the same authentication as web interface:

1. Connect to master's WebSocket
2. Enter password (or auto-connect if whitelisted)
3. Receive full state sync
4. Begin using the client

Building

No special features needed:

```
# Standard musl build works
cargo build --target x86_64-unknown-linux-musl \
  --no-default-features --features rustls-backend
```

Example Setup

Server Side (Master)

```
# Start Clay normally
./clay

# Enable WebSocket in /web settings:
# - WS enabled: On
# - WS port: 9002
# - WS password: your_password
```

Client Side (Remote)

```
# Connect from anywhere
./clay --console=your-server.com:9002

# Enter password when prompted
```

Tips

SSH Forwarding

For secure access without TLS:

```
# On client machine
ssh -L 9002:localhost:9002 your-server

# Then connect locally
./clay --console=localhost:9002
```

Multiple Views

Run multiple remote consoles to: - Monitor different worlds simultaneously - Have different scroll positions - Use different tag visibility settings

Screen/Tmux

Combine with screen or tmux for persistent sessions:

```
# On server
tmux new -s clay
./clay

# From anywhere
ssh your-server -t tmux attach -t clay
```

Troubleshooting

Connection Refused

1. Verify master Clay is running
2. Check WebSocket is enabled in /web
3. Verify port matches

Authentication Failed

1. Check password is correct
2. Verify allow list configuration if using whitelisting

Display Issues

1. Check TERM environment variable
2. Try Ctrl+L to redraw
3. Verify terminal supports colors

Sync Issues

1. Use /menu → Resync to refresh state
2. Check network connection
3. Restart remote console

Telnet Features

Clay includes comprehensive telnet protocol support for proper MUD server communication.

Automatic Negotiation

Clay automatically handles telnet negotiation: - Detects telnet mode when IAC sequences are received - Responds appropriately to server requests - Strips telnet sequences from displayed output

Supported Telnet Options

Option	Code	Description
SGA	3	Suppress Go Ahead
TTYTYPE	24	Terminal Type
EOR	25	End of Record
NAWS	31	Negotiate About Window Size

SGA (Suppress Go Ahead)

Accepts server's WILL SGA with DO SGA. Most modern MUDs use SGA for character-at-a-time mode.

TTYTYPE (Terminal Type)

Reports terminal type to the server: - Uses TERM environment variable (e.g., "xterm-256color") - Falls back to "ANSI" if TERM is not set - Responds to SB TTYTYPE SEND with SB TTYTYPE IS <terminal>

EOR (End of Record)

Alternative prompt marker, treated the same as GA: - When received, text from last newline is identified as prompt - Prompt is displayed at the start of input area

NAWS (Negotiate About Window Size)

Reports window dimensions to the server: - Sends smallest width × height across all connected clients - Updates sent when terminal resizes - Updates sent when web/GUI client dimensions change - Dimensions tracked per-world, reset on disconnect

Prompt Detection

Clay detects prompts using telnet GA (Go Ahead) or EOR (End of Record):

How It Works

1. Server sends output ending with IAC GA or IAC EOR
2. Text after the last newline is identified as the prompt
3. Prompt is stored per-world
4. Prompt is displayed at the start of the input area (cyan)
5. Prompt is NOT shown in output area

Prompt Handling

- Trailing spaces are normalized: stripped, then one space added
- ANSI codes in prompts are preserved
- Cursor positioning uses visible prompt length
- Prompt cleared when user sends a command

Auto-Login Prompt Detection

When Auto Login is set to “Prompt” or “MOO_prompt”: - First telnet prompt: Username sent automatically - Second telnet prompt: Password sent automatically - MOO_prompt third prompt: Username sent again

Prompts that are auto-answered are cleared and not displayed.

Keepalive

Configurable keepalive prevents idle disconnection:

When Sent

- Every 5 minutes of inactivity (no data sent)
- Only when in telnet mode
- Per-world timing

Keepalive Types

Configure in World Settings:

Type	Behavior
NOP	Sends telnet NOP command (IAC NOP)
Custom	Sends your custom command
Generic	Sends help commands <code>##_idler_message_<rand>_###</code>

Custom Keepalive

Set “Keep alive” to “Custom” and configure “Keep alive cmd”:

```
# Example custom keepalive  
look
```

The generic option sends a command that: - Works on most MUDs (help commands) - Includes random text to avoid pattern detection - Produces minimal server output

Timing Fields

Track connection activity:

Field	Description
<code>last_send_time</code>	When user last sent a command
<code>last_receive_time</code>	When data was last received

These are: - Initialized on connect - Reset after /reload - Used to calculate keepalive timing - Shown in /connections output

Line Buffering

Clay properly buffers incoming data:

Safe Splitting

- `find_safe_split_point()` checks for incomplete sequences
- ANSI CSI sequences not split mid-sequence
- Telnet commands not split mid-command
- Remaining buffer flushed on connection close

Partial Lines

Lines without trailing newlines (e.g., prompts) are handled specially: - Displayed immediately - `partial_line` tracks incomplete lines - When more data arrives, the line is updated in-place - Prevents duplicate lines from TCP read splitting

Telnet Sequences

Common Sequences

Sequence	Name	Purpose
IAC GA	Go Ahead	End of output, prompt follows
IAC EOR	End of Record	Alternative prompt marker
IAC NOP	No Operation	Keepalive
IAC WILL x	Will	Server offers option x
IAC WONT x	Won't	Server refuses option x
IAC DO x	Do	Client should enable option x
IAC DONT x	Don't	Client should disable option x
IAC SB ... IAC SE	Subnegotiation	Option-specific data

IAC Values

- IAC: 255 (0xFF)
- WILL: 251
- WONT: 252
- DO: 253
- DONT: 254
- SB: 250
- SE: 240
- GA: 249
- EOR: 239
- NOP: 241

Troubleshooting

No Prompt Detected

1. Verify MUD server sends GA or EOR
2. Check telnet mode is active (server sends IAC sequences)
3. Some MUDs require explicit telnet negotiation

Wrong Terminal Type

1. Check TERM environment variable
2. Set explicitly: `TERM=xterm-256color ./clay`

Disconnected for Idling

1. Enable keepalive in World Settings
2. Try different keepalive type
3. Reduce the 5-minute interval may require code change

Garbled Output

1. Check character encoding in World Settings
2. Try Latin1 for older MUDs
3. Try Fansi for BBS-style output

ANSI Music

Clay supports ANSI music sequences, a feature from BBS-era computing that allows servers to send simple melodies.

What is ANSI Music?

ANSI music uses escape sequences to define melodies that play on the client's speaker. The format originated with PC BBS software in the 1990s and uses a syntax similar to the BASIC PLAY command.

Format

ANSI music sequences have this structure:

```
ESC [ M <music_string> Ctrl-N
```

Or with modifiers:

```
ESC [ MF <music_string> Ctrl-N    # Foreground (blocks output)
ESC [ MB <music_string> Ctrl-N    # Background (concurrent)
ESC [ N <music_string> Ctrl-N     # Alternative format
```

Where: - ESC is the escape character (0x1B) - Ctrl-N (0x0E) marks the end of the sequence

Music String Syntax

The music string uses BASIC PLAY command notation:

Notes

Command	Description
A-G	Play note A through G
# or +	Sharp (follows note)
-	Flat (follows note)
.	Dotted note (1.5x duration)

Command	Description
P or R	Pause/rest

Octave

Command	Description
O<n>	Set octave (0-6, default 4)
>	Increase octave
<	Decrease octave

Tempo and Duration

Command	Description
T<n>	Set tempo (32-255 BPM, default 120)
L<n>	Set default note length (1=whole, 4=quarter, etc.)

Example

T120 L4 04 CDEFGAB>C

Plays C major scale at 120 BPM, quarter notes, starting at octave 4.

Configuration

Enable ANSI Music

1. Open /setup
2. Enable "ANSI Music"

Test Audio

Use the test command to verify audio works:

/testmusic

This plays a simple C-D-E-F-G sequence.

Playback

Console

The console itself cannot play audio (no speaker access from terminal). Music sequences are: - Extracted from output - Stripped from display - Forwarded to connected web/GUI clients

Web Interface

Uses Web Audio API: - Square wave oscillator (PC speaker simulation) - Plays through browser audio - May require user interaction to start (browser autoplay policy)

Remote GUI

Requires remote-gui-audio feature:

```
# Build with audio support  
sudo apt install libasound2-dev # Linux only  
cargo build --features remote-gui-audio
```

Uses rodio library with: - ALSA backend on Linux - CoreAudio on macOS

Building with Audio

Linux

```
# Install ALSA development libraries  
sudo apt install libasound2-dev  
  
# Build with audio  
cargo build --features remote-gui-audio
```

macOS

No extra dependencies needed:

```
cargo build --features remote-gui-audio
```

Audio uses CoreAudio automatically.

Troubleshooting

No Sound in Web Interface

1. Check browser allows audio (click somewhere first)
2. Verify ANSI Music is enabled in /setup
3. Check browser console for errors
4. Try different browser

No Sound in GUI

1. Verify built with remote-gui-audio feature
2. Check ANSI Music is enabled in /setup
3. Verify ALSA is working: `aplay /usr/share/sounds/alsa/Front_Center.wav`

4. Check PulseAudio/PipeWire is running

Music Sounds Wrong

1. ANSI music is limited to PC speaker frequencies
2. Complex sequences may not sound as intended
3. Timing may vary based on system load

Music Not Playing

1. Verify MUD actually sends ANSI music sequences
2. Check sequences are properly formatted
3. Use /testmusic to verify client audio works

Technical Details

Sequence Detection

Clay detects ANSI music by looking for: 1. ESC [followed by M or N 2. Optional modifier (F or B) 3. Music string content 4. Terminating Ctrl-N (0x0E)

Processing

1. Music sequences extracted during output processing
2. Sequences forwarded to web/GUI clients via WebSocket
3. Display shows output with music sequences removed

Web Audio Implementation

```
// Simplified example  
const oscillator = audioContext.createOscillator();  
oscillator.type = 'square';  
oscillator.frequency.setValueAtTime(frequency, time);  
oscillator.start(time);  
oscillator.stop(time + duration);
```

Frequency Calculation

Note frequencies calculated from: - Base: A4 = 440 Hz - Formula: $\text{freq} = 440 * 2^{((\text{note}-A4)/12)}$

Hot Reload

Clay supports hot reloading - replacing the running binary with a new version while preserving active connections.

How It Works

The hot reload process:

1. **Save state:** Complete application state is serialized
 - Output buffers and pending lines
 - Scroll positions
 - Per-world settings (encoding, auto-login type, etc.)
 - Connection file descriptors
2. **Prepare sockets:** FD_CLOEXEC flag is cleared on socket file descriptors so they survive exec
3. **Execute:** `exec()` replaces the current process with the new binary
4. **Restore state:** New process detects reload mode and restores everything
5. **Reconstruct connections:** TCP sockets are rebuilt from preserved file descriptors
6. **Cleanup:** Inconsistent states are fixed
 - Worlds without working command channels marked disconnected
 - Pending lines cleared for disconnected worlds
 - Pause state cleared for disconnected worlds

Triggering Reload

/reload Command

/reload

Keyboard Shortcut

Press Ctrl+R

External Signal

Send SIGUSR1 to the process:

```
kill -USR1 $(pgrep clay)
```

Useful for: - Automated deployment - Scripts - CI/CD pipelines

Updated Binary Detection

On Linux, when you rebuild Clay while it's running: - `/proc/self/exe` shows the path with " (deleted)" suffix - The reload logic strips this suffix to find the new binary

This allows seamless workflow: 1. Make code changes 2. Run `cargo build` 3. Type `/reload` or press `Ctrl+R` 4. New code is active, connections preserved

Limitations

TLS/SSL Connections

Without TLS Proxy: - TLS connections cannot be preserved - TLS state (session keys, IVs, sequence numbers) exists only in process memory - `exec()` destroys this state even though the TCP socket survives - TLS worlds will need manual reconnection after reload

With TLS Proxy: - Enable "TLS Proxy" in `/setup` - TLS connections are preserved across reload - See **TLS Proxy** chapter for details

State Compatibility

The new binary must be compatible with the saved state format. Major version changes may break reload compatibility.

Auto-Login

Restored connections have auto-login disabled: - Prevents duplicate login attempts - Only fresh connections trigger auto-login

Message Suppression

During reload, success messages are suppressed to reduce noise: - WebSocket/HTTP/HTTPS server startup (only shown on failure) - Binary path message (only shown on failure)

Warnings and errors are always shown.

Use Cases

Apply Code Changes

Develop and test without losing sessions:

```
# Terminal 1: Running Clay
./clay

# Terminal 2: Make changes
vim src/main.rs
cargo build

# Terminal 1: Reload
/reload
# Changes active, still connected!
```

Deploy Bug Fixes

Fix a bug without disconnecting users:

1. Build new binary
2. Send SIGUSR1: `kill -USR1 $(pgrep clay)`
3. Fix is live

Configuration Changes

Some settings require reload to take effect. Instead of restarting:

```
/reload
```

State File

During reload, state is temporarily saved to `~/.clay.reload`: - Contains serialized application state - Socket file descriptors - World configurations - Output buffers

This file is: - Created during reload - Read by the new process - Automatically cleaned up

Reload vs Restart

Aspect	Reload	Restart
TCP connections	Preserved	Lost
TLS connections	Requires TLS Proxy	Lost
Output history	Preserved	Lost
Scroll position	Preserved	Reset
Settings	Preserved	Reloaded from file

Aspect	Reload	Restart
Memory usage	Cleaned	Fresh start

Troubleshooting

Reload Fails

1. Check binary exists and is executable
2. Verify sufficient disk space for state file
3. Check file permissions on ~/.clay.reload

Connections Lost After Reload

1. For TLS: Enable TLS Proxy
2. For TCP: Check network didn't drop during reload
3. Verify world cleanup didn't mark healthy connections as dead

State Incompatibility

If reload fails due to state format changes: 1. Disconnect all worlds 2. Save important settings manually 3. Restart normally 4. Reconnect worlds

SIGUSR1 Not Working

1. Verify correct PID: `pgrep clay`
2. Check signal permissions
3. Ensure Clay is the foreground process (not backgrounded with wrong signal handling)

Technical Details

State Serialization

State is serialized using a binary format including: - Version marker for compatibility - Output line vectors - Pending line vectors - Scroll offsets - World configurations - Socket file descriptors (as integers)

Socket Preservation

Before `exec()`:

```
// Clear close-on-exec flag
fcntl(fd, F_SETFD, 0);
```

After `exec()`:

```
// Reconstruct socket from fd  
let socket = TcpStream::from_raw_fd(fd);
```

Process Replacement

Uses `execve()` to replace the process image: - Same PID maintained - Same parent process - File descriptors preserved (with cleared `FD_CLOEXEC`) - Memory completely replaced

TLS Proxy

The TLS Proxy feature allows TLS/SSL connections to be preserved across hot reloads.

The Problem

TLS encryption state includes: - Session keys - Initialization vectors (IVs) - Sequence numbers - Other cryptographic state

This state exists only in process memory. When `exec()` replaces the process during hot reload, this memory is destroyed - even though the underlying TCP socket can survive.

Without TLS Proxy, TLS connections must be manually reconnected after every reload.

The Solution

A forked child process handles TLS, communicating with the main process via Unix socket:

MUD Server <--TLS--> TLS Proxy (child) <--Unix Socket--> Main Client

The proxy process: - Survives `exec()` (separate process) - Maintains TLS state - Relays data between TLS connection and Unix socket - Reconnects to main client after reload

Configuration

Enable TLS Proxy

1. Open /setup
2. Toggle "TLS Proxy" to On

Behavior

When enabled: - New TLS connections spawn a proxy process - Proxy handles TLS termination - Main client communicates via Unix socket - On reload, new main process reconnects to existing proxies

How It Works

On TLS Connect

1. Main client forks a child process
2. Child establishes TLS connection to MUD
3. Child creates Unix socket at `/tmp/clay-tls-<pid>-<world_name>.sock`
4. Main client connects to Unix socket
5. Child relays: TLS ↔ Unix socket

On Hot Reload

1. Main client saves proxy PID and socket path per world
2. `exec()` replaces main process
3. New main process reads saved proxy info
4. Reconnects to existing proxy via Unix socket
5. TLS connection continues uninterrupted

On Disconnect

1. Main client closes Unix socket connection
2. Proxy detects disconnect
3. Proxy closes TLS connection and exits

Implementation Details

Functions

Function	Purpose
<code>spawn_tls_proxy()</code>	Forks child, establishes TLS
<code>run_tls_proxy()</code>	Child main loop (relay)

Stream Types

```
enum StreamReader {  
    Plain(TcpStream),  
    Tls(TlsStream),  
    Proxy(UnixStream), // When using TLS proxy  
}
```

Saved State

During reload, per-world: - `proxy_pid`: Process ID of proxy child - `proxy_socket_path`: Path to Unix socket

Socket Path

Format: /tmp/clay-tls-<main_pid>-<world_name>.sock

Example: /tmp/clay-tls-12345-MyMUD.sock

Health Monitoring

The main client monitors proxy health: - Detects if proxy process dies - Marks world as disconnected on proxy death - Cleans up Unix socket

Fallback Behavior

If proxy spawn fails: - Falls back to direct TLS connection - Connection works but won't survive reload - Warning logged

When to Use

Enable TLS Proxy

- You use TLS connections AND
- You want to preserve them across hot reload AND
- You reload frequently (development, automated deploys)

Skip TLS Proxy

- Only use non-TLS connections
- Rarely use hot reload
- Minimal resource usage is priority
- Running on Termux (not available)

Resource Usage

Each TLS proxy uses: - One child process - One Unix socket - One TLS connection - Minimal memory (relay only)

Proxies are lightweight but add process count.

Platform Support

Platform	TLS Proxy Support
Linux	Full
macOS	Full

Platform	TLS Proxy Support
Windows (WSL)	Full
Termux/Android	Not available

Termux doesn't support TLS Proxy because: - `exec()` is limited on Android - Signal handling is restricted

Troubleshooting

TLS Connection Lost After Reload

1. Verify TLS Proxy is enabled in `/setup`
2. Check proxy process is running: `ps aux | grep clay`
3. Verify Unix socket exists: `ls /tmp/clay-tls-*`

Proxy Not Spawning

1. Check disk space for socket file
2. Verify `/tmp` is writable
3. Check process limit not reached

Socket Permission Errors

1. Check `/tmp` permissions
2. Verify socket file is accessible
3. May need to clean up stale sockets

Zombie Proxy Processes

If proxy processes remain after Clay exits:

```
pskill -f "clay.*tls.*proxy"
```

Or clean up sockets:

```
rm /tmp/clay-tls-*.sock
```

Example Session

```
# Start Clay
./clay

# Connect to TLS world
/worlds -e MyMUD
# Enable Use SSL, save
```

```
# Enable TLS Proxy
/setup
# Enable TLS Proxy, save

# Connect
/worlds MyMUD

# Make code changes, rebuild
# (in another terminal)
cargo build

# Reload - TLS connection preserved!
/reload
```

Spell Checking

Clay includes real-time spell checking with suggestions, using the system dictionary.

Dictionary

Location

Clay looks for the system dictionary at: 1. /usr/share/dict/words 2. /usr/share/dict/american-english (fallback) 3. /usr/share/dict/british-english (fallback)

Dictionary Format

- Plain text file, one word per line
- Case-insensitive matching
- Typically 100,000+ words

Installing Dictionary

If dictionary is not installed:

```
# Debian/Ubuntu
sudo apt install wamerican

# Or British English
sudo apt install wbritish

# Fedora/RHEL
sudo dnf install words

# macOS
# Dictionary exists at /usr/share/dict/words by default
```

Configuration

Enable/Disable

1. Open /setup

2. Toggle “Spell check”

Or with TinyFugue commands:

```
#set spell_check on  
#set spell_check off
```

How It Works

Word Checking

- Words are only checked when “complete” (followed by space/punctuation)
- Words at end of input are NOT checked while typing
- Prevents premature flagging of partial words

Visual Feedback

- Misspelled words highlighted in red
- Highlighting persists until word is re-checked

Caching

Misspelling state is cached between keystrokes: - Prevents flickering when editing - Example: “thiss” → flagged → backspace to “thiss” → stays flagged → backspace to “this” → stays flagged until space, then re-checked

Suggestions

Press Ctrl+Q to get spell suggestions:

1. Place cursor on or after a misspelled word
2. Press Ctrl+Q
3. First suggestion replaces the word
4. Press Ctrl+Q again to cycle through suggestions

Suggestion Algorithm

Uses Levenshtein distance: - Maximum edit distance: 3 - Prefers words of similar length - Sorted by edit distance (closest first)

Contraction Support

Contractions are recognized as valid words: - “didn’t”, “won’t”, “I’m”, “you’re” - Apostrophes between alphabetic characters are part of the word - Special handling for irregular contractions (e.g., “won’t” → “will”)

Examples

Typing Flow

Type: "helo "
^^^^
Red (misspelled)

Type: "hello "
^^^^^
Normal (correct)

Using Suggestions

Type: "recieve "
^^^^^^
Red (misspelled)

Press Ctrl+Q:
"receive "
^^^^^^
Normal (replaced with suggestion)

Cycling Suggestions

Word: "teh"
Ctrl+Q → "the"
Ctrl+Q → "tea"
Ctrl+Q → "ten"
...

Limitations

Not Checked

- Command prefixes (/, #)
- URLs and paths
- Words with numbers
- Very short words (1-2 characters)

Dictionary Limitations

- Technical terms may be flagged
- Proper nouns may be flagged
- MUD-specific vocabulary not included

Performance

- Large dictionaries load quickly
- Suggestions computed on-demand
- Caching minimizes re-checking

Integration

Works With

- Console input
- Web interface input (if enabled)
- GUI input (if enabled)

Doesn't Work With

- Output text (not checked)
- Popup text fields (settings, etc.)

Technical Details

SpellChecker Struct

```
struct SpellChecker {  
    words: HashSet<String>,  
    // Dictionary loaded at startup  
}
```

Levenshtein Distance

Edit distance between two strings: - Insertion: +1 - Deletion: +1 - Substitution: +1

Maximum distance of 3 balances: - Finding reasonable suggestions - Performance (limiting search space)

Word Boundaries

A word is considered complete when followed by: - Space - Tab - Punctuation (. , ! ? ; : etc.) - End of line (when checking mid-input)

Troubleshooting

No Spell Checking

1. Verify enabled in /setup
2. Check dictionary file exists

3. Check dictionary file is readable

Too Many False Positives

1. Consider adding custom dictionary
2. Use /setup to disable if too distracting
3. Technical content may not match dictionary

Suggestions Not Working

1. Verify cursor is on/near misspelled word
2. Check word is actually flagged (red)
3. Try pressing Ctrl+Q multiple times

Missing Dictionary

```
# Check if dictionary exists
ls -la /usr/share/dict/words

# Install if missing
sudo apt install wamerican
```

Android/Termux

Clay compiles and runs on Termux, the Android terminal emulator, bringing MUD access to mobile devices.

Installation

Install Termux

Download Termux from F-Droid (recommended) or Google Play Store.

F-Droid: <https://f-droid.org/packages/com.termux/>

Install Rust

```
pkg update  
pkg install rust
```

Build Clay

```
# Clone the repository  
git clone https://github.com/your-repo/clay  
cd clay  
  
# Build (no GUI features)  
cargo build --no-default-features --features rustls-backend  
  
# Run  
./target/debug/clay
```

Pre-built Binary

If a pre-built ARM64 binary is available:

```
chmod +x clay-linux-aarch64  
./clay-linux-aarch64
```

Limitations

Some features are unavailable on Termux due to Android restrictions:

Not Available

Feature	Reason
Hot reload	<code>exec()</code> is limited on Android
TLS proxy	Requires <code>exec()</code> for reload
Process suspension (Ctrl+Z)	Signal handling restricted
Remote GUI client	No display server
SIGUSR1 reload trigger	Signal handling restricted

What Works

Feature	Status
Core MUD client	Full
Multiple worlds	Full
TLS connections	Direct only (not via proxy)
More-mode pausing	Full
Actions/triggers	Full
Command history	Full
All TUI features	Full
Settings persistence	Full
WebSocket server	Full
Web interface	Full

Android App

Clay has a companion Android app that provides: - Native Android interface - Background WebSocket connection - Push notifications via `/notify` command - Foreground service for persistent connection

Installing the App

The APK can be built from the `android/` directory in the repository.

Notifications

Use `/notify` to send notifications:

```
/notify Someone is paging you!
```

Or in actions:

Name: page_alert
Pattern: *pages you*
Command: /notify Page from \$1

Foreground Service

When authenticated: - Foreground service starts - Shows “Connected to MUD server” notification - Keeps WebSocket alive in background - Allows receiving notifications

Tips for Termux

Storage Setup

```
# Allow access to shared storage
termux-setup-storage
```

Keyboard

- Use Termux:Styling for better fonts
- Consider external keyboard for extended sessions
- Swipe left on keyboard for arrow keys

Session Management

Since hot reload isn't available: - Use Termux sessions for multiple views - Consider tmux for session persistence:

```
pkg install tmux
tmux new -s clay
./clay
# Detach with Ctrl+B then D
# Reattach with: tmux attach -t clay
```

Battery Optimization

- Disable battery optimization for Termux
- Settings → Apps → Termux → Battery → Unrestricted

Wake Lock

Keep Termux running:

```
termux-wake-lock
./clay
# When done:
termux-wake-unlock
```

Building Tips

Memory Usage

Termux has limited memory. If build fails:

```
# Reduce parallel jobs  
cargo build -j 1 --no-default-features --features rustls-backend
```

Storage Space

Rust builds need significant space: - Clean old builds: `cargo clean` - Check space: `df -h`

Build Time

Initial build takes a while on mobile: - Leave device plugged in - Disable sleep during build - Consider building on PC and copying binary

Troubleshooting

Connection Issues

1. Check network connectivity
2. Verify hostname resolution works
3. Try IP address instead of hostname

Display Issues

1. Check terminal size: `stty size`
2. Try different Termux font
3. Resize terminal and press `Ctrl+L`

Performance

1. Close other apps
2. Reduce scrollbar if memory is tight
3. Disable spell checking in `/setup`

Build Failures

1. Update packages: `pkg update && pkg upgrade`
2. Check disk space
3. Try building with less parallelism: `cargo build -j 1`

Settings Not Saving

1. Check Termux has storage permission
2. Verify ~/.clay.dat is writable
3. Check disk isn't full

Recommended Setup

```
# One-time setup
pkg update
pkg install rust tmux
termux-setup-storage

# Clone and build
git clone https://github.com/your-repo/clay
cd clay
cargo build --no-default-features --features rustls-backend

# Run in tmux for persistence
tmux new -s clay
./target/debug/clay

# Later, reattach
tmux attach -t clay
```

Troubleshooting

Common issues and their solutions.

Connection Problems

Can't Connect to MUD

Symptoms: Connection times out or refused

Solutions: 1. Verify hostname and port are correct 2. Check if MUD is online: `telnet hostname port` 3. Check firewall isn't blocking 4. Try IP address instead of hostname 5. Check if SSL is required but not enabled (or vice versa)

Connection Drops

Symptoms: Disconnected after period of inactivity

Solutions: 1. Enable keepalive in World Settings 2. Try different keepalive type (NOP, Generic, Custom) 3. Check if MUD has short idle timeout 4. Verify network is stable

SSL/TLS Handshake Fails

Symptoms: Connection refused or handshake error

Solutions: 1. Verify MUD actually supports SSL on that port 2. Check if MUD uses self-signed certificate (may need configuration) 3. Try without SSL to verify server is reachable 4. Update CA certificates: `sudo update-ca-certificates`

Display Issues

Garbled Output

Symptoms: Strange characters, wrong symbols

Solutions: 1. Check encoding in World Settings (try Latin1, Fansi) 2. Verify TERM environment variable is set correctly 3. Check terminal supports UTF-8 4. Press Ctrl+L to redraw

Colors Wrong

Symptoms: Wrong colors or missing colors

Solutions: 1. Check TERM is set (e.g., xterm-256color) 2. Verify terminal supports colors 3. Try different theme in /setup 4. Check terminal color scheme

Screen Corruption

Symptoms: Text overlapping, wrong positions

Solutions: 1. Press Ctrl+L to redraw 2. Resize terminal window 3. Check terminal size: `stty size` 4. Restart Clay if persistent

Wide Characters Display Wrong

Symptoms: CJK characters, emoji misaligned

Solutions: 1. Ensure terminal uses monospace font with CJK support 2. Check font supports wide characters 3. Try different terminal emulator

Input Problems

Keys Not Working

Symptoms: Function keys, arrows don't work

Solutions: 1. Check TERM environment variable 2. Verify terminal sends correct escape sequences 3. Try different terminal emulator 4. Check for conflicting terminal shortcuts

Input Lag

Symptoms: Typing feels slow

Solutions: 1. Check network latency to MUD 2. Disable spell checking in /setup 3. Verify not running in slow emulator

Cursor Position Wrong

Symptoms: Cursor not where expected

Solutions: 1. Press Ctrl+L to redraw 2. Check for zero-width characters in input 3. Verify font is monospace

WebSocket/Web Interface

Can't Connect to Web Interface

Symptoms: Browser shows connection refused

Solutions: 1. Verify WebSocket server is enabled in /web 2. Check password is set 3. Verify correct port 4. Check firewall allows the port 5. For HTTPS, verify TLS certificate is valid

Web Interface Shows “Disconnected”

Symptoms: Connected then immediately disconnects

Solutions: 1. Check WebSocket password is correct 2. Verify WebSocket port matches 3. Check for HTTPS/HTTP mismatch (ws:// vs wss://) 4. Look for browser console errors

Authentication Fails

Symptoms: Password rejected

Solutions: 1. Verify password matches /web settings 2. Check caps lock 3. Try clearing browser cache 4. If using allow list, verify IP is in list

Hot Reload

Reload Fails

Symptoms: /reload causes crash or disconnect

Solutions: 1. Verify new binary exists 2. Check disk space for state file 3. Ensure ~/.clay.reload is writable 4. For TLS connections, enable TLS Proxy

Connections Lost After Reload

Symptoms: Worlds disconnected after /reload

Solutions: 1. TLS connections: Enable TLS Proxy in /setup 2. TCP connections: Network may have dropped 3. Check reload cleaned up stale connections

State Incompatible

Symptoms: Error about state version

Solutions: 1. Save important settings manually 2. Disconnect all worlds 3. Restart normally (don't reload) 4. Reconnect and reconfigure

Performance

High Memory Usage

Symptoms: Clay using lots of RAM

Solutions: 1. Reduce scrollbar (code change required) 2. Disconnect unused worlds 3. Clear old output with Ctrl+L 4. Restart periodically

High CPU Usage

Symptoms: Clay using CPU when idle

Solutions: 1. Check for runaway triggers 2. Disable ANSI music if not needed 3. Reduce web client refresh rate 4. Check for reconnection loops

Slow Startup

Symptoms: Takes long to start

Solutions: 1. Check dictionary file size (for spell check) 2. Verify network DNS is fast 3. Disable auto-connect temporarily

Settings

Settings Not Saving

Symptoms: Changes lost on restart

Solutions: 1. Check ~/.clay.dat is writable 2. Verify disk space 3. Make sure to press Save in popups 4. Check file permissions

Settings Corrupted

Symptoms: Error loading settings

Solutions: 1. Backup ~/.clay.dat 2. Delete file and restart (recreates with defaults) 3. Manually edit file to fix syntax

Build Problems

Musl Build Fails

Symptoms: Compilation errors with musl target

Solutions: 1. Install musl tools: `sudo apt install musl-tools` 2. Add target: `rustup target add x86_64-unknown-linux-musl` 3. Clean and rebuild: `cargo clean && cargo build`

GUI Build Fails

Symptoms: remote-gui feature won't compile

Solutions: 1. Install X11 libs: `sudo apt install libxcb-render0-dev libxcb-shape0-dev libxcb-xfixes0-dev` 2. For audio: `sudo apt install libasound2-dev` 3. Verify display server is running

Missing Dependencies

Symptoms: Link errors

Solutions: 1. Use `--no-default-features --features rustls-backend` for minimal deps 2. Check platform-specific requirements 3. Update Rust: `rustup update`

Recovery

Complete Freeze

If Clay becomes unresponsive: 1. Try Ctrl+C twice (quit) 2. Send SIGTERM: `kill $(pgrep clay)` 3. Send SIGKILL as last resort: `kill -9 $(pgrep clay)`

Lost All Settings

Restore from backup:

```
cp ~/.clay.dat.backup ~/.clay.dat
```

Or start fresh - Clay creates defaults on startup.

Crash on Startup

1. Rename settings file: `mv ~/.clay.dat ~/.clay.dat.broken`
2. Start Clay (creates fresh settings)
3. Manually migrate important settings

Appendices

A. Character Encodings

UTF-8

Standard Unicode encoding (default): - Full Unicode character support - Variable-width encoding (1-4 bytes per character) - Most compatible with modern systems

Latin1 (ISO-8859-1)

Western European encoding: - 256 characters (single byte) - Direct byte-to-Unicode mapping - Common for older MUDs

Fansi (CP437)

IBM PC character set: - Box drawing characters: ─ │ □ ▣ ▤ ▥ ▦ ▧ ▨ ▩ - Block elements: ▒ ▓ ▔ ▕ ▖ ▗ ▘ ▙ ▚ ▛ ▜ ▝ ▞ ▟ ■ □ ▢ ▣ ▤ ▥ ▦ ▧ ▨ ▩ - Line drawing symbols - Common for BBS-style MUDs

B. WebSocket Protocol

Clay's WebSocket server uses JSON messages for communication.

Message Types (Client → Server)

```
{"type": "AuthRequest", "password_hash": "<sha256>"}
{"type": "SendCommand", "world": "name", "command": "text"}
{"type": "SwitchWorld", "world": "name"}
{"type": "ConnectWorld", "world": "name"}
{"type": "DisconnectWorld", "world": "name"}
{"type": "MarkWorldSeen", "world": "name"}
{"type": "Ping"}
```

Message Types (Server → Client)

```

{"type": "AuthResponse", "success": true}
{"type": "InitialState", "worlds": [...], "settings": {...}}
{"type": "ServerData", "world": "name", "data": "text", "timestamp": 123}
{"type": "WorldConnected", "world": "name"}
{"type": "WorldDisconnected", "world": "name"}
{"type": "WorldSwitched", "world": "name"}
{"type": "PromptUpdate", "world": "name", "prompt": "text"}
{"type": "UnseenCleared", "world": "name"}
{"type": "UnseenUpdate", "world": "name", "count": 5}
{"type": "ActivityUpdate", "count": 2}
{"type": "Pong"}

```

Authentication Flow

1. Client connects to WebSocket
2. If IP is whitelisted, server sends InitialState immediately
3. Otherwise, client sends AuthRequest with SHA-256 hash of password
4. Server responds with AuthResponse
5. On success, server sends InitialState

Password Hashing

```

const hash = await crypto.subtle.digest('SHA-256',
  new TextEncoder().encode(password));
const hashHex = Array.from(new Uint8Array(hash))
  .map(b => b.toString(16).padStart(2, '0'))
  .join('');

```

C. Configuration File Format

Settings are stored in `~/.clay.dat` using INI-like format.

Global Section

```

[global]
more_mode_enabled=true
spell_check_enabled=true
temp_convert_enabled=false
world_switch_mode=unseen_first
show_tags=false
input_height=3
console_theme=dark
gui_theme=dark
tls_proxy_enabled=false

```

```

ansi_music_enabled=false
websocket_enabled=false
websocket_port=9002
websocket_password_hash=<sha256>
websocket_use_tls=false
websocket_cert_file=/path/to/cert.pem
websocket_key_file=/path/to/key.pem
websocket_allow_list=192.168.1.100,192.168.1.101
websocket_nonsecure_enabled=false
websocket_nonsecure_port=9003
http_enabled=false
http_port=9000
https_enabled=false
https_port=9001

```

World Sections

```

[world:MyMUD]
hostname=mud.example.com
port=4000
user=username
password=secret
use_ssl=true
log_file=/home/user/logs/mymud.log
encoding=utf8
auto_connect_type=connect
keep_alive_type=nop
keep_alive_cmd=

```

Value Types

Type	Format	Example
Boolean	true/false	more_mode_enabled=true
Integer	Decimal number	port=4000
String	Plain text	hostname=mud.example.com
Enum	Lowercase identifier	encoding=utf8

Encoding Values

- utf8 - UTF-8 (default)
- latin1 - ISO-8859-1
- fansi - CP437

Auto-Login Types

- connect - Send “connect user password”
- prompt - Send on telnet prompts
- moo_prompt - MOO-style prompts

Keepalive Types

- nop - Telnet NOP
- custom - Custom command
- generic - Generic help command

World Switch Modes

- unseen_first - Priority to worlds with unseen output
- alphabetical - Alphabetical order

D. Command Reference

Slash Commands (/)

Command	Description
/help	Open help popup
/quit	Exit Clay
/worlds	Open world selector
/worlds <name>	Connect to world
/worlds -e [name]	Edit world
/worlds -l <name>	Connect without auto-login
/connections, /l	List connections
/disconnect, /dc	Disconnect current world
/send [opts] <text>	Send text to world(s)
/setup	Open global settings
/web	Open web settings
/actions	Open actions editor
/reload	Hot reload
/testmusic	Test ANSI music
/notify <msg>	Send notification (Android)
/gag	In actions: hide matched line

TF Commands (#)

Command	Description
#set var val	Set variable
#unset var	Remove variable

Command	Description
<code>#let var val</code>	Local variable
<code>#echo msg</code>	Display message
<code>#send text</code>	Send to MUD
<code>#expr expr</code>	Evaluate expression
<code>#test expr</code>	Boolean test
<code>#if ... #endif</code>	Conditional
<code>#while ... #done</code>	While loop
<code>#for ... #done</code>	For loop
<code>#def name = body</code>	Define macro
<code>#undef name</code>	Remove macro
<code>#list [pat]</code>	List macros
<code>#bind key = cmd</code>	Key binding
<code>#load file</code>	Load script
<code>#save file</code>	Save macros

E. ANSI Color Codes

Standard Colors (0-7)

Code	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

Bright Colors (8-15)

Add 8 to standard color number.

256-Color Mode

`ESC[38;5;<n>m` # Foreground

`ESC[48;5;<n>m` # Background

Where n is: - 0-7: Standard colors - 8-15: Bright colors - 16-231: 6x6x6 color cube - 232-255: Grayscale

True Color (24-bit)

`ESC[38;2;<r>;<g>;m` # Foreground

ESC[48;2;<r>;<g>;m # Background

F. Telnet Protocol Reference

IAC Commands

Value	Name	Purpose
255	IAC	Interpret As Command
254	DONT	Refuse option
253	DO	Request option
252	WONT	Refuse to perform
251	WILL	Agree to perform
250	SB	Subnegotiation begin
249	GA	Go Ahead
241	NOP	No Operation
240	SE	Subnegotiation end
239	EOR	End of Record

Option Codes

Value	Name	Description
3	SGA	Suppress Go Ahead
24	TTYTYPE	Terminal Type
25	EOR	End of Record
31	NAWS	Window Size