Tool Reference

Table of Contents

---

1. Overview

1.1 GUI-Based Design Philosophy
1.2 Shared Behaviors Across Tools

2. Scribe (Lexicon Builder)

2.1 Purpose
2.2 Randomness Source
2.3 Interface Overview
2.4 Inputs
2.5 Configuration Options
2.6 Execution
2.7 Results Dialog
2.8 Lexicon Modification

3. LAT (Lexicon Analysis Tool)

3.1 Purpose
3.2 Interface Overview
3.3 View Modes
3.4 Metrics
3.5 Mnemonic Analysis

4. Engine

4.1 Purpose
4.2 Interface Overview
4.3 Inputs
4.4 Processing Model
4.5 Output Behavior

5. Data Format Notes

5.1 SQLite Dependency
5.2 Lexicon Portability
5.3 Compatibility Considerations

1. Overview

This document provides a reference for all tools included in the Vizini2 toolset, their behavior, and their interaction with VLex data.

1.1 GUI-Based Design Philosophy

The Vizini2 toolset is implemented exclusively as a graphical interface. This design choice reflects both the structural complexity of the system and the intended mode of interaction.

The configuration space of the system includes multiple interdependent inputs, including lexicons, mnemonics, starting indices, and data. Representing these inputs as command-line arguments would require a large number of parameters with non-trivial relationships, resulting in reduced usability and increased likelihood of operator error.

The graphical interface provides:

- Clear visibility into all required inputs and their current state
- Direct manipulation of parameters without reliance on argument syntax
- Immediate feedback when required inputs are missing or invalid
- A structured workflow that reduces ambiguity during operation

In addition to usability considerations, the absence of a command-line interface limits straightforward large-scale automation. While this does not prevent automated use entirely, it increases the effort required to perform high-throughput or brute-force operations.

The GUI is therefore both a usability layer and a practical constraint on misuse, without serving as a primary security mechanism.

1.2 Shared Behaviors Across Tools

All tools in the Vizini2 suite are designed with consistent interaction patterns and interface conventions.

Lexicon Handling

- All tools operate on VLex (Vizini Lexicon) files
- Loaded lexicons display metadata within the interface to confirm state
- In Scribe, loaded lexicons may be modified and re-saved

Mnemonic Parsing

- The Engine and LAT interpret mnemonics using the same parsing rules
- Scribe does not process mnemonics

File Interaction

- All tools use the standard Lazarus file dialog system for opening and saving files
- No custom file management behavior is introduced beyond these dialogs

Interface Consistency

- A shared icon set is used across all tools (e.g., VLex, text, search, analyze, file actions)
- Common visual elements, including image-based buttons, are reused
- Layouts are designed to operate within a minimum resolution of approximately 800×500 pixels

Execution Model

- Tools do not maintain persistent state between sessions
- All required inputs must be explicitly provided for each use

- Errors are limited and primarily arise from missing required inputs or invalid configurations (e.g., empty Active Key conditions in the Engine)

2. Scribe (Lexicon Builder)

2.1 Purpose

Scribe is used to construct and modify VLex (Vizini Lexicon) files. It transforms input text into a structured lexicon by assigning iteration depth and chunk weight to each word.

The resulting lexicon defines how mnemonics resolve into Chunks during transformation. Scribe is therefore the primary tool for shaping the structural behavior of the system.

2.2 Randomness Source

Scribe uses the operating system's cryptographically secure random number generator (CSPRNG) for all stochastic operations.

This randomness is applied when resolving dice math expressions and assigning structural properties during lexicon construction.

2.3 Interface Overview

The interface is organized into distinct functional areas:

- File Selection
    - Choose between creating a new lexicon or using an existing VLex file
    - File selection is performed via image-based buttons
- Input Source
    - Select between loading a text file or entering text manually
    - Manual input is provided through a text area

- Configuration Panel
    - Depth and weight expressions are defined using text inputs
    - Duplicate handling behavior is selected via radio buttons
- Lexicon Metadata
    - Title, author, and notes fields are displayed and editable
    - When loading an existing lexicon, these fields are populated automatically
- Execution Control
    - A dedicated "Run Scribe" button initiates lexicon generation or modification

The layout is designed to keep all required inputs visible simultaneously, minimizing hidden state.

2.4 Inputs

Scribe accepts two forms of input for lexicon construction:

- Text Sources
    - Plain text files or manually entered text
    - Words are extracted without requiring strict formatting
- Dice Math Expressions
    - Used to define depth and weight values
    - Expressions are evaluated at runtime using the system's randomness source

2.5 Configuration Options

Depth

- Defines the number of iterations (Chunks) generated per word
- Accepts dice math expressions

Weight

- Defines the number of S-boxes (internal units) per Chunk
- Accepts dice math expressions

Duplicate Handling

Controls how repeated words in the input source are treated:

- Append - Adds additional entries for repeated words
- Ignore - Skips words that already exist
- Replace - Overwrites existing entries with new values

2.6 Execution

Execution is initiated via the "Run Scribe" button.

During execution:

- A save dialog determines the output location[^1]
- Input text is normalized to lowercase and parsed into individual words
- Depth expressions are evaluated per source word
- Weight expressions are evaluated per generated chunk
- Chunks are generated and assigned to each word
- Duplicate handling rules are applied as configured

[^1]: When editing an existing lexicon, selecting the same file will overwrite it.

2.7 Results Dialog

Upon completion, a results dialog is displayed summarizing the operation:

- Start - Initial state metrics
- Finish - Final state metrics
- Delta - Net change resulting from the operation

This provides a concise view of how the lexicon was affected.

2.8 Lexicon Modification

Scribe supports direct editing of existing lexicons.

- Loading a VLex file populates all metadata into the interface
- Subsequent execution applies changes based on current inputs and configuration

Metadata fields may be edited after loading and will be saved with the lexicon.

When modifying an existing lexicon, changes are not applied until execution is performed.

3. LAT (Lexicon Analysis Tool)

3.1 Purpose

LAT is used to inspect and analyze VLex (Vizini Lexicon) files.

It provides visibility into lexicon structure, including word composition, iteration depth, and weight distribution. In addition, LAT allows mnemonics to be evaluated in a non-destructive context, exposing how they resolve and what structural impact they would have during transformation.

LAT does not perform transformations. It reflects how the Engine interprets data.

3.2 Interface Overview

The interface is divided into three primary functional areas:

Lexicon Selection and Information

- Load a VLex file via the file selection control

- Displays lexicon title, author, and notes

  Word Search
- Input field for querying specified words or special input (*) for an aggregate listing.
- Results are displayed in a structured table format on the right
    - A tabular view that lists word-depth pairs alongside their associated weight

      Mnemonic Analysis
- Input field for testing mnemonics
- Displays how the mnemonic is parsed and interpreted
- Provides weight calculations and structural breakdown

3.3 View Modes

LAT supports two modes of viewing lexicon data:

- Aggregate Mode (*)
    - Displays all words in the lexicon
    - Each word is displayed once with its depth and the cumulative weight of all its chunks
    - This is the default view
    - Activated by entering * in the search field
- Granular Mode
    - Displays only entries matching the specified word(s)
    - Each chunk is shown individually with its respective weight
    - Activated by entering text in the search field

3.4 Metrics

LAT exposes the following structural metrics:

- Iterations

    - The depth associated with each word (number of Chunks)
- Weight
    - The total weight derived from all Chunks associated with a word in Aggregate Mode
    - The weight of individual Chunks in Granular Mode
    - Also displayed at the mnemonic level during analysis

3.5 Mnemonic Analysis

LAT evaluates mnemonics using the same parsing logic as the Engine.

Weight Calculation

- Each word found in the lexicon resolves to one Chunk
- Repeated use of the same word may contribute unique or repeated chunks depending on its Depth
- Each Chunk contributes a weight value
- The total mnemonic weight is calculated as the sum of all contributing Chunk weights
- A full expression of the weight calculation is displayed
- The final computed total is presented separately

Predictive Behavior

- The parsed mnemonic is displayed in its resolved form (e.g., word.depth)
- This reflects how the Engine will interpret the mnemonic during execution

LAT provides a deterministic preview of mnemonic structure without performing any transformation.

4. Engine (Vizini2 Engine)

4.1 Purpose

The Vizini2 Engine performs data transformation using a VLex and a mnemonic-derived Active Key.

It applies a deterministic, stateful process to input data, producing output based on lexicon structure, mnemonic composition, and starting indices. The same process is used for both forward (encryption) and backward (decryption) operations.

4.2 Interface Overview

The interface is organized around required inputs and execution control:

- Mnemonic Input
  - Text field for entering the mnemonic used to construct the Active Key
- Initial Indices
  - Two numeric inputs defining the starting index for each processing pass
- Lexicon Selection
  - An Image-Button is used for loading a VLex file
  - Displays lexicon metadata (title, author, notes) upon loading
- Data Selection
  - An Image-Button is used for selecting the source file
  - Displays file name and size
- Execution Controls
  - Direction selection (Forward / Backward)
  - Run Button to initiate processing
  - Progress bar indicating execution status

All required inputs are visible simultaneously to reduce ambiguity prior to execution.

4.3 Inputs

The Engine requires the following inputs:

- Lexicon
  - A VLex file defining word structure and Chunk composition
- Mnemonic
  - Parsed into an Active Key using the same rules defined in earlier sections
- Starting Indices
  - Independent initial index values for Pass 1 and Pass 2
- Data
  - The source file to be transformed

Execution will not proceed without a VLex or if the mnemonic produces an empty Active Key.

4.4 Processing Model

The Engine uses a two-pass transformation model:

- Pass 1
  - Processes the source data into an intermediate memory buffer
- Pass 2
  - Processes the intermediate buffer back into the primary buffer
- Output
  - The final buffer is written to the selected output file

Each pass operates over the full dataset, and transformations are applied based on the evolving index state.

The index is updated after each step using the value interpreted as the “ciphertext” for that operation.

This corresponds to:

- Output value during forward (encryption) operations
- Input value during backward (decryption) operations

4.5 Output Behavior

- No validation is performed on transformation correctness
- Output is always produced when execution begins

The Engine does not verify:

- Input correctness
- Mnemonic intent
- Lexicon suitability

As a result:

- Incorrect inputs still produce deterministic output
- Successful execution does not imply correct or meaningful results

This behavior reflects the Engine’s role as a transformation system rather than a validation system.

5. Data Format Notes

5.1 SQLite Dependency

VLex files are implemented using SQLite as the underlying storage format.

This provides:

- A structured and queryable data model

- Broad platform compatibility
- Stability through a well-established storage system

No direct interaction with the SQLite layer is required for normal use. All read and write operations are handled by the Vizini2 toolset.

Direct modification outside the toolset is not recommended

WARNING: Direct modification of Chunk BLOB data may:

- Cause application instability or crashes
- Produce irreversible or destructive transformation behavior

Such modifications are unsupported and should be avoided.

5.2 Lexicon Portability

VLex files are portable across systems that support the Vizini2 toolset.

- Lexicons can be copied, shared, and reused without modification
- No system-specific configuration is embedded in the file
- Metadata and structural data are stored together within the same file

Portability is dependent only on compatibility with the SQLite format and the expected schema.

5.3 Compatibility Considerations

Original Vizini "Keyring" `*.keÿ` files are not compatible with the Vizini2 Lexicon `*.VLex` structure.

No formal schema versioning is enforced at this time. Future versions will introduce versioning if necessary and treat the lack of version data as "Lexicon Version 1"

As a result, perfect compatibility between different versions of the toolset is not guaranteed.