Module Interface Specification for Live Neuro

Bo Liang

April 13, 2025

1 Revision History

Date	Version	Notes
March 17	1.0	Initial Draft
Apr 11	1.1	Modify

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS

Contents

3 Introduction

This document specifies the interfaces for modules in the Live Neuro system, an interactive neural data visualization tool. It complements the SRS and MG, with full implementation details available at GitHub Repository.

4 Notation

The following table summarizes the primitive data types used by Live Neuro.

Data Type	Notation	Description
String	$\mathbb{S}$	a character string
~~data type dictionary~~ Dict[str, Any]	$\mathbb{D}$	a storage type for KV structures
bool	$\mathbb{B}$	Boolean data type, which can hold one of two values: either true or false
integer	$\mathbb{Z}$	a number without a fractional component in $(-\infty, \infty)$
natural number	$\mathbb{N}$	a number without a fractional component in $[1, \infty)$
real	$\mathbb{R}$	any number in $(-\infty, \infty)$

The specification of Live Neuro uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Live Neuro uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**	**Module ID**
Hardware-Hiding Module	Hardware-Hiding Module	M1
Behaviour-Hiding Module	Input Format Module	M2
	Data Processing Module	M3
	Visualization Module	M4
Software Decision Module	TRF Calculation Module	M5

Table 1: Module Hierarchy

6 MIS of Hardware-Hiding Module(M1)

6.1 Module

M1: OS Abstraction Layer

6.2 Uses

Directly interacts with OS APIs (e.g., file I/O, hardware drivers).

6.3 Syntax

6.3.1 Exported Access Programs

Name	In	Out	Exceptions
readFile	$\mathbb{S}$ (file path)	Sequence	FileNotFoundError
savePlot	plot data	file(.jpeg)	SavePlotError

6.4 Semantics

6.4.1 Environment Variables

File ~~system, display hardware~~system-fs, display hardware-dh.

6.4.2 Assumptions

OS compatibility (Linux/Windows/macOS)

6.4.3 Access Routine Semantics

- readFile(): Reads neural data from disk, returns a sequence of ~~$\mathbb{R}$ numbers~~characters.
- savePlot(): Renders visualization output to screen or file under ../visualization/output.

7 MIS of Input Format Module

7.1 Module

Multi-Format MEG/EEG Data Parser

7.2 Uses

M1 (readFile for raw data loading).

7.3 Syntax

7.3.1 Exported Constants

supported formats = [EDF, FIF~~, BrainVision~~] (supported data formats).
max channels = 256 (maximum allowed channels per dataset).

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
load edf()	$\mathbb{S}$ (EDF file path)	$\mathbb{D}$<$\mathbb{S}$: $\mathbb{R}$[]>	EDFFileError
load fif()	$\mathbb{S}$ (FIF file path)	$\mathbb{D}$<$\mathbb{S}$: $\mathbb{R}$[]>	FIFFileError
load brain-vision()	$\mathbb{S}$ (.vhdr file path)	$\mathbb{D}$<$\mathbb{S}$: $\mathbb{R}$[]>	VHDRFileError

Notation Explanation:

- $\mathbb{R}$[] – a one-dimensional array of real numbers (e.g., one EEG channel)

7.4 Semantics

7.4.1 State Variables

- currentFormat: $\mathbb{S}$(last detected data format, e.g., ~~"FIF")~~FIF(enum type)).
  - ~~metadataCache: $\mathbb{D}$(cached metadata from parsed files).~~ Stores the most recently detected data format.
    FormatType is an **enumerated type** with the following values:
    * FIF
    * EDF

7.4.2 Environment Variables

7.4.3 ~~Access Routine Semantics~~

- ~~load edf(~~**DATA_PATH: String**

  Path to the directory where data files are stored.

  This value can be passed explicitly or set using an environment variable (e.g., LIVE_NEURO_DATA_PATH).

7.4.3 Access Routine Semantics

- **Purpose**: Loads data from an EDF (European Data Format) file and returns EEG time-series data.
- **Input**:
  - filepath: Path to the .edf file as a string.
- **Output**: ~~Output: data: Time-series array of shape~~
  - data: A data Dict[~~channels × samples~~String : array] ~~. Exception: EDFHeaderError: Invalid EDF header structure. Implementation: Uses eelbrain for EDF parsing.~~
- ~~load fif() : Output: data:~~ **Exceptions**:
  - EDFFileError: Raised if the EDF file is malformed or unsupported.
- **Implementation Detail**: Uses mne.io.read_raw_edf()

load_fif(filepath: String) → mne.io.Raw

- **Purpose**: Loads MEG/EEG sensor data from an MNE .fif file. ~~Exception: FIFFileError: FIF fileversion mismatch or missing data tags. Implementation: Relies on mne-python library.~~
- ~~load brainvision(): Output: data: EEG data segmented by markers Exception: VHDRParseError: Inconsistent header fields in.vhdr file .~~ **Input**:
  - filepath: Path to the .fif file as a string.

7.4.4 ~~Local Functions~~

- **Output**:
  - ~~parse edf header(): Extracts EDF header fields and validates integrity.~~ data: A Raw object from the mne library.
- ~~align brainvision files(): Synchronizes.vhdr, .vmrk, and. eeg data.~~**Exceptions**:
  - FIFFileError: Raised for version mismatch, file corruption, or missing tags.
- **Implementation Detail**: Calls mne.io.read_raw_fif().

8 MIS of Data Processing Module

8.1 Module

Statistical Preprocessing and Validation
Primary Function: Validates input data integrity and applies statistical preprocessing to neural signals.

8.2 Uses

- M2(Input Format Module): Receives parsed raw data (e.g., MEG/EEG time-series).
- M5(TRF Calculation Module): Provides preprocessed data for dipole current computation.

8.3 Syntax

8.3.1 Exported Access Programs

Name	In	Out	Exceptions
validate input	$\mathbb{D}$<$\mathbb{S}$: $\mathbb{R}$[]>	$\mathbb{B}$	InvalidDataError
compute statistics	$\mathbb{R}$[]	$\mathbb{D}$<$\mathbb{S}$: $\mathbb{R}$>	NaNError

Although input validation could technically be performed immediately after data is loaded, we intentionally separate the **validation logic** into its own module for reasons of **modularity**, **reusability**, and **maintainability**.

Specifically:

- **Validation is not only for file loading**: In many use cases, data may come from **non-file sources**, such as the **internet**, **APIs**, or **in-memory structures**. Centralizing validation ensures consistency across all entry points.
- **Validation rules evolve**: Scientific or clinical datasets may have **changing standards** or **conditional rules**. Keeping validation separate allows updates to these rules **without modifying the loader** or tightly coupled modules.
- **Cleaner architecture**: Following the **Single Responsibility Principle**, the file-loading module focuses purely on parsing and decoding data formats (e.g., EDF, FIF), while the validation module handles semantic correctness and integrity.
- **Testing and reuse**: A standalone validation module can be tested independently and reused in **preprocessing pipelines**, **live data streaming**, or even **user-uploaded data checks**.

Therefore, validation is performed in a **later module**, after the raw data is loaded, to support broader use cases and maintain long-term flexibility of the system.

8.4 Semantics

8.4.1 State Variables

validatedSignals

- ~~validatedSignals: $\mathbb{D}$<$\mathbb{S}$: $\mathbb{R}$>(cached validated data) .~~ **Initial State**: None
- ~~baselineStats: $\mathbb{D}$<$\mathbb{S}$: $\mathbb{R}$>(mean and std of reference signals).~~ **Updated By**: validate_input(data)

- **Transition**:
  - If data passes validation, validatedSignals ← data
  - If validation fails, validatedSignals remains unchanged

baselineStats

- **Initial State**: None
- **Updated By**: compute_statistics(data)
- **Transition**:
  - Computes mean and standard deviation from data
  - Sets baselineStats ← mean: , std:
  - If data contains invalid values (e.g., NaN), raises NaNError, and baselineStats remains unchanged

8.4.2 Access Routine Semantics

- validate input(raw data: $\mathbb{D}$):

  Validation Steps:

  Check if raw data[data] is of type $\mathbb{R}$[] and non-empty.

  Ensure no NaN or infinite values in the signal.

  Output:

  Returns True if validation passes.

  Exceptions:

  InvalidDataError: Non-numeric values or mismatched channel counts.

- compute statistics(ℝ[]):

  Output:

  mean, std, min, max and other statistical information

  Exceptions:

  NaNError if signal contains invalid values after preprocessing

9 MIS of Visualization Module

9.1 Module

Interactive Neuro-imaging Module

9.2 Uses

M1 (writePlot), M3 (processed data), M5 (TRF results)

9.3 Syntax

9.3.1 Exported Access Programs

Name	In	Out	Exceptions
plot	~~List~~Dict<~~ℝ> (dipole currents),List<~~𝕊,𝕊>~~ (plot type, plot data path), 𝕊(interaction type)	void	Plotting Error

9.4 Semantics

9.4.1 State Variables

activePlots

- ~~activePlots: 𝔻<PlotID, PlotData> (metadata for open plots) .~~ **Initial State**: An empty dictionary
- **Updated By**:
  - ~~linkedViews: Set<PlotID>(plots synchronized via linkPlots) .~~ plot(data, type)
  - update_plot(plotID, newData)

9.4.2 ~~Environment Variables~~

– close_plot(plotID)
~~GPU acceleration (enabled via plotly.graph_objects and nilearn.plotting)~~

9.4.2 ~~Assumptions~~

9.4.2 ~~Access Routine Semantics~~

- ~~renderCorticalMap()~~**Transition**:

  – ~~*Output:* Generates brain activation map using plotly.graph_objects and nilearn.plotting~~
  When a new plot is created via plot, a new entry is added:
  activePlots[plotID] ← PlotData
  – ~~*Exceptions:* Plotting Error~~ **When a plot is updated**, the entry is modified:
  activePlots[plotID] ← updatedPlotData
  – **When a plot is closed**, the entry is removed:
  del activePlots[plotID]

9.4.2 Access Routine: Semantics

9.4.3 ~~Local Functions~~

Syntax:

- ~~_updateLinkedAxes(~~`plot(data: Dict<R,R>, interactionMode: String)` ~~: Propagates axis changes to all linked plots.~~

Valid `plotTypes` values (case-insensitive):

- `"time_series"` – Renders time vs amplitude plots for each channel.
- `"glass_brain"` – Displays brain activity using anatomical projections (e.g., with `nilearn`).
- `"topomap"` – Sensor-space topographic activation maps.

Valid `interactionMode` values:

- `"zoom"` – Enables scroll and drag-based zooming.
- `"drag"` – Enable click and drag plots
- `"click"` – Enable click in data plots

10 Appendix

10.1 Dependencies

- Plotly: Used for interactive HTML5 visualizations (zooming, panning, tooltips).
- Nilearn: Handles neuroimaging-specific rendering (glass brain plots).

10.2 ~~Performance Notes~~

- ~~GPU acceleration is optional but recommended for real-time interaction with large datasets (>10 samples).~~