Euphorix 1.0 (Java 25 VM for OpenEuphoria Syntax) – Developer Manual & API Reference

> 🇩🇪 **Deutsche Version verfügbar:** HANDBUCH.md | 📄 **Overview:** README.md | ⚖️ **License:** GNU GPLv3

> [!NOTE]

> **OpenEuphoria Syntax Compatibility: Euphorix** is an independent, clean-room high-performance Virtual Machine written in Java 25. It natively interprets and executes the **OpenEuphoria 4.x syntax** and standard library, enhanced with modern Java 25 capabilities (Virtual Threads, Go-style Channels, Async/Await Futures, Swing/FlatLaf Desktop GUI, JDBC for SQLite & PostgreSQL, UTF-8, and XML/JSON).

This manual is the complete documentation and technical reference for **Euphorix** (`./euphorix` , `./euphorix-standalone` , and `euphorix.jar`). Backward-compatible aliases such as `./eui-java` and `./eui-standalone` are maintained.

The runtime environment combines the clean, lightweight syntax of OpenEuphoria with modern Java platform capabilities (OpenJDK 25):

- **Virtual Threads (Project Loom)** for scalable servers, asynchronous streaming pipelines, and background processes—completely **without fragile POSIX `fork()`** .
- **Seamless Java Interoperability (`std/java.e`)**: Runtime loading of external JAR files, instantiation of Java classes, static and instance method invocation, field access, and dynamic proxy listeners for callbacks.
- **Native GUI Development with Java Swing & AWT (`std/gui.e`)**: Modern Linux desktop design via **GTK+ Look & Feel** and FlatLaf, crisp **subpixel LCD anti-aliasing**, native Ubuntu typography, and declarative widget definitions.
- **Comprehensive UTF-8 Support**: Consistent handling of non-ASCII characters, German umlauts (`ä` , `ö` , `ü` , `ß`), and worldwide Unicode glyphs across lexer, compiler, bytecode VM, Swing GUI, and I/O.
- **Built-in Enterprise Modules**: JSON & RFC 9535 JSONPath, JSON-Schema (Draft 2020-12), XML DOM with W3C XPath 1.0 & XSD, SQLite 3 & PostgreSQL (JDBC), HTTP/HTTPS client & server, and dual-stack IPv4/IPv6 sockets.
- **C-FFI & Native Raw Memory**: Direct binding of native Linux shared libraries (`.so`) using the JDK 25 Foreign Function & Memory (FFM) API (`java.lang.foreign.*`).
- **Standalone Single-Binary (`./euphorix-standalone`)**: Fully self-contained 53 MB single executable bundling an optimized minimal Java 25 runtime (via `jlink`), requiring **no installed Java** on target systems.

Table of Contents

1. Architecture, Installation, Build & Operations (Java 25 on Linux 64-Bit)

- 1.1 Project Structure & Fat-JAR Architecture
- 1.2 Virtual Threads Architecture (Processes & Concurrency without `fork()`)
- 1.3 Native C Binding (.so) via JDK 25 FFM API
- 1.4 Prerequisites & Maven Build
- 1.5 Managing & Integrating Java Dependencies via `pom.xml`
- 1.6 Self-Contained Single-Binary with Bundled Minimal JRE (`./euphorix-standalone`)
- 1.7 Ahead-of-Time Bytecode Compilation (`.exb`), Executable Binding & Dedicated Runtime (`runtime.jar` / `euphorix-rt`)
- 1.8 Cross-Platform & Microsoft Windows Support (`euphorix.cmd` / `euphorix.bat`)
- 1.9 Developer Workflow with Hot-Reload File Watcher (`--watch` / `-w`)
- 1.10 Linux Daemon Operations & Systemd Units (`tools/make_service.ex`)
- 1.11 Reverse Proxy Operations (lighttpd, Nginx, Caddy)
- 1.12 IDE Integration & Language Server Protocol (LSP) for Apache NetBeans & VS Code

1. Complete Language, Syntax & Standard Reference

- 2.1 Data Types & Type System
- 2.2 Visibility, Namespaces & Includes
- 2.3 Operators, Indexing & Slicing
- 2.4 Control Structures & Loops
- 2.5 Procedures, Functions & Callbacks
- 2.6 Conditional Compilation (`ifdef`)
- 2.7 All Built-in Routines

- 2.8 Complete Reference of All 77 Include Files & Standard Modules (`std/.e` & `euphoria/ .e`)
- 2.8.1 Data Structures, Maps & Algorithms
- 2.8.2 I/O, File System, Console & Operating System
- 2.8.3 Text Processing, Unicode & Localization
- 2.8.4 Mathematics, Statistics & Random Numbers
- 2.8.5 Date, Time & Calendar
- 2.8.6 Networking, Internet, DNS & URLs
- 2.8.7 Data Formats, XML, JSON & Databases
- 2.8.8 Desktop GUI & Web Frontend Frameworks
- 2.8.9 Java Interoperability, C-FFI & Machine Memory
- 2.8.10 Modern Concurrency, Threads & Multitasking
- 2.8.11 Error Handling, Testing & Utilities
- 2.8.12 Windows Platform-Specific Modules
- 2.8.13 Euphoria Internals, Debugger & Syntax Tokenizer
- 2.8.14 Artificial Intelligence, MCP, LLM & RAG

1. Java Interoperability & JAR Integration (`std/java.e`)

- 3.1 Loading External JAR Files (`java:load_jar`)
- 3.2 Class Resolution & Object Instantiation
- 3.3 Method Invocation (Static & Instance)
- 3.4 Reading and Writing Fields
- 3.5 Automatic Type Matching & Conversions
- 3.6 Event Listeners & Callbacks (`java:create_listener`)
- 3.7 Subprocess Streaming Pipes with Virtual Threads

1. Graphical User Interfaces with Swing & AWT (`std/gui.e`)

- 4.1 Linux Desktop Integration & Theme Selection (`gui:set_theme`)
- 4.2 Typography & Subpixel LCD Anti-Aliasing
- 4.3 Window Management (`window` , `show` , `wait_window`)
- 4.4 Layout Managers with Gap Control (`hgap` / `vgap`)
- 4.5 Widget Component Reference
- 4.6 Properties, Styling & Padding
- 4.7 Standard Dialogs (Message, Input, Confirm, FileDialog)
- 4.8 Complete Demo Application (`demo/gui_demo.ex`)

1. UTF-8 String Processing (`std/utf8.e` & VM Engine)

- 5.1 Comprehensive UTF-8 Integration in Java 25 Interpreter
- 5.2 Exact Unicode Operations (`std/utf8.e`)

1. JSON & RFC 9535 JSONPath (`std/json.e`)

- 6.1 JSON Parsing & Serialization (`std/json.e`)
- 6.2 RFC 9535 JSONPath Engine with Path Queries

1. JSON-Schema Draft 2020-12 (`std/json_schema.e`)

- 7.1 Declarative Schema Validation (Draft 2020-12)

1. Document & Tabular Formats: XML, CSV, Excel & PDF (`std/xml.e` , `std/csv.e` , `std/excel.e` , `std/pdf_report.e`)

- 8.1 XML DOM, W3C XPath 1.0 & XSD Validation (`std/xml.e`)
- 8.2 RFC 4180 High-Performance CSV Engine (`std/csv.e`)
- 8.3 Native Microsoft Excel OpenXML Engine (`std/excel.e`)
- 8.4 Standard PDF 1.4 Business Report Generator (`std/pdf_report.e`)

1. SQL Databases: SQLite 3 (`std/sqlite.e`) & PostgreSQL (`std/pgsql.e`)

- 9.1 Embedded SQLite 3 Database (`std/sqlite.e`)
- 9.2 Enterprise PostgreSQL Database (`std/pgsql.e`)

1. Web Servers, Microservices, Security & Realtime (`std/http.e` , `std/auth.e` , `std/session.e` , `std/sse.e`)

- 10.1 HTTP & HTTPS Client (`std/http.e`)
- 10.2 Embedded Web Server & REST Routing (`std/http.e`)
- 10.3 Enterprise Security: PBKDF2 & JWT HS256 (`std/auth.e`)
- 10.4 HTTP Session & Cookie Management (`std/session.e`)
- 10.5 Realtime Server-Sent Events (SSE) Push (`std/sse.e`)

1. Dual-Stack Networking & Sockets (`std/socket.e` & `std/net/dns.e`)

- 11.1 Dual-Stack IPv4 & IPv6 Sockets (`std/socket.e`)
- 11.2 DNS Name Resolution & URLs (`std/net/dns.e` , `std/net/url.e`)

1. Demos, Examples & Execution

- 12.1 Overview of Practical Demos
- 12.2 Running Demonstration Programs

1. Modern Concurrency: Virtual Threads, Async/Await & Channels (`std/thread.e` , `std/channel.e` , `std/task.e` , `std/cron.e`)

- 13.1 Approach 1: Classic Tasks with Virtual Threads (`task:create_thread`)
- 13.2 Approach 2: Channels / Go-Style Concurrency (`std/channel.e`)
- 13.3 Approach 3: Async / Await Futures (`std/thread.e`)
- 13.4 Approach 4: Background Tasks & Cron Scheduler (`std/cron.e`)

1. Web Applications with SAP UI5 Web Components 2.29 (`std/ui5.e`)

- 14.1 Architecture & Fiori Horizon Design
- 14.2 Component Suite & Declaration
- 14.3 Reactive Server Bridge & Live DOM Updates (`/__ui5_action`)
- 14.4 Complete Application Example (`examples/07_sapui5_webcomponents_demo.ex`)

1. Self-Contained High-Density Web-UI Suite (`std/vsui.e`)

- 15.1 Architecture & Philosophy: The Fusion of HTMX and Vaadin
- 15.2 The Complete 22-Component Suite (`html/lib2/components/`)
- 15.3 Integrated Asset Server (`/lib2/` , `/css/` , `/img/*`)
- 15.4 Reactive Event Bridge with `vsevent` & DOM Patches
- 15.5 Pixel-Exact Positioning & Window Centering (`ldom.setXYWH` , `opt_xywh`)
- 15.6 Commercial Accounting Precision & German Number Formatting (`vsnumber.js`)
- 15.7 Mobile Responsiveness & Touch Ergonomics (`vs-mobile.css`)
- 15.8 Form Validation & Error Tooltips (`vsui:field_error`)
- 15.9 Drag & Drop File Upload Dropzone (`vsui:upload`)
- 15.10 Native Responsive SVG Charts (`vsui:chart`)
- 15.11 Complete Application Examples & Demos (`demo/vsui_demo.ex` , `examples/08_vsui_cockpit_demo.ex` , `examples/11_vsui_charts_und_upload_demo.ex`)

1. Artificial Intelligence: Autonomous Agents, Model Context Protocol (MCP), RAG & LLMs (`std/mcp.e` , `std/rag.e` , `std/llm.e`)

- 16.1 Architecture: The Interaction between LLM, MCP, and RAG
- 16.2 When LLM + MCP Suffice Alone (and When RAG Is Needed)
- 16.3 Model Context Protocol (MCP) – Server & Client (`std/mcp.e`)
- 16.4 Universal LLM Client (`std/llm.e`)
- 16.5 Retrieval-Augmented Generation with SQLite FTS5 & Vectors (`std/rag.e`)
- 16.6 Complete Autonomous AI Agent (`examples/15_autonomous_agent_mcp_rag_demo.ex`)

---

1. Architecture, Installation, Build & Operations (Java 25 on Linux 64-Bit)

1.1 Project Structure & Fat-JAR Architecture

The interpreter builds as a **standalone Fat-JAR (`euphorix.jar`)**. When built with Maven, all dependencies (JDBC drivers, Jackson JSON, FlatLaf look-and-feel, embedded standard library) are shaded into a single archive:

```
prj/intpret/
├── euphorix (eui-java, eux) <- Shell launcher with optimized JVM parameters
├── euphorix.jar (eui.jar)   <- Standalone Java 25 Fat-JAR (19.4 MB shaded)
├── euphorix-standalone      <- Standalone 53 MB single executable (bundled minimal JRE)
├── euphorix.cmd / .bat      <- Windows command launchers
├── pom.xml                  <- Maven project descriptor for JDK 25 & Shade plugin
├── eu_include/              <- Standard library (std/*.e, euphoria/*.e)
│   ├── std/
│   │   ├── gui.e            <- Swing & AWT Desktop GUI API
│   │   ├── java.e           <- Java Interoperability, JAR Loader & Virtual Thread Pipes
│   │   ├── utf8.e           <- Unicode & UTF-8 string manipulation
│   │   ├── json.e           <- JSON & RFC 9535 JSONPath
│   │   ├── json_schema.e    <- JSON-Schema Draft 2020-12
│   │   ├── xml.e            <- XML DOM, XPath 1.0, XSD
│   │   ├── sqlite.e         <- SQLite 3 API
│   │   ├── pgsql.e          <- PostgreSQL API
│   │   ├── http.e           <- HTTP/HTTPS Client & Server
│   │   └── ...              <- sequence.e, math.e, regex.e, socket.e, etc.
├── java/src/main/java/eu/   <- Java 25 source code
│   ├── Main.java            <- CLI entry point, UTF-8 & AWT property initialization
│   ├── Lexer.java           <- Lexer with native UTF-8 multi-byte encoding
│   ├── Compiler.java        <- Single-pass bytecode compiler & constant folder
│   ├── Bytecode.java        <- 3-address register instruction set (OpCodes)
│   ├── VM.java              <- Register VM with virtual threads & engine dispatch
│   ├── Value.java           <- 48-bit Int, IEEE-754 Atom & Copy-on-Write Sequences
│   ├── JavaInterop.java     <- Reflection, JAR ClassLoader, dynamic proxy & pipes
│   └── DataEngines.java     <- JSON, XPath & schema validation engines
├── tools/
│   └── build_standalone.sh  <- jlink packaging tool for standalone executable
└── demo/
    └── gui_demo.ex          <- Interactive GTK+ Swing GUI demo application
```

The `./euphorix` Launcher Script

The launcher script (`./euphorix` with aliases `./eui-java` and `./eux`) provides optimized Linux 64-bit JVM parameters:

```
#!/usr/bin/env bash
DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
JAR="$DIR/euphorix.jar"
exec /home/volkix/prg/jdk25/bin/java \
    --enable-native-access=ALL-UNNAMED \
    -Dfile.encoding=UTF-8 \
    -Dawt.useSystemAAFontSettings=lcd \
    -Dswing.aatext=true \
    -Dsun.java2d.renderer=sun.java2d.marlin.MarlinRenderingEngine \
    -Djdk.internal.httpclient.disableHostnameVerification=true \
    -jar "$JAR" "$@"
```

1.2 Virtual Threads Architecture (Processes & Concurrency without `fork()`)

In C programs, `fork()` clones the address space of the process, which is problematic and prone to deadlocks in multithreaded environments.

The Java 25 Edition utilizes **Virtual Threads (Project Loom)** exclusively:

1. **Subprocess Pipes (`pipe_open` , `pipe_read` , `pipe_write` , `pipe_close`)**:

Subprocesses launch asynchronously via `java.lang.ProcessBuilder` . Dedicated virtual threads continuously read `stdout` and `stderr` into lock-free `LinkedBlockingQueue` buffers. This eliminates pipeline stalls and deadlocks without blocking the Euphoria interpreter thread.

1. **HTTP/HTTPS Servers**:

Incoming connections to `http:server_create` dispatch across a virtual thread executor. Thousands of concurrent connections consume minimal operating system resources.

1. **Re-entrant Event Callbacks**:

Swing and background threads safely invoke Euphoria callback routines via `VM.invokeRoutine()`.

1.3 Native C Binding (.so) via JDK 25 FFM API

Standard OpenEuphoria C-FFI routines (`open_dll`, `define_c_func`, `define_c_proc`, `c_func`, `c_proc`, `allocate`, `free`, `peek`, *`poke`*) interface with native Linux shared objects through the **JDK 25 Foreign Function & Memory (FFM) API**:

- Shared libraries (`.so`) load via `SymbolLookup.libraryLookup(Path, Arena)`.
- Function invocations execute through `Linker.nativeLinker().downcallHandle()` using native ABI descriptors (`ValueLayout.JAVA_INT`, `JAVA_LONG`, `ADDRESS`, `JAVA_DOUBLE`).
- Off-heap memory is allocated via `MemorySegment.allocateNative()`.

This enables direct calls into Linux system libraries (e.g. `libc.so.6`, `libm.so.6`, `libcrypto.so`) without any C compiler.

1.4 Prerequisites & Maven Build

- **Operating System**: Linux 64-Bit (`x86_64`)
- **Java SDK**: OpenJDK 25 (at `/home/volkix/prg/jdk25` or configured via `JAVA_HOME`)
- **Build Tool**: Apache Maven (`mvn`)

```
# Build the Fat-JAR (euphorix.jar)
JAVA_HOME=/home/volkix/prg/jdk25 mvn clean package -DskipTests

# Execute an OpenEuphoria script
./euphorix demo/gui_demo.ex

# Run unit test suites (Java 25 Runner)
./tools/run_all_tests ./euphorix tests/t_gui.e tests/t_java_interop.e tests/t_concurrency.ex
```

1.5 Managing & Integrating Java Dependencies via `pom.xml`

Managing dependencies through `pom.xml` makes millions of Java libraries instantly accessible in OpenEuphoria without classpath configuration.

How Fat-JAR Dependency Management Works:

1. **Central Declaration**: Libraries declared in `pom.xml` are downloaded automatically from Maven Central, along with all transitive dependencies.
2. **Automatic Shading**: The `maven-shade-plugin` unpacks all classes and bundles them directly into `euphorix.jar`.
3. **Immediate Euphoria Availability**: Every class becomes accessible in Euphoria using `java:new_object()` and `java:call_static()`.

Example: Adding a New Java Library (3 Steps)

To add libraries such as **JFreeChart** (data visualization), **Apache POI** (Excel/Office documents), or **ZXing** (barcodes):

1. **Add dependency to `pom.xml`**:

```
<dependency>
    <groupId>org.jfree</groupId>
    <artifactId>jfreechart</artifactId>
    <version>1.5.4</version>
</dependency>
```

1. **Repackage the Fat-JAR**:

```
JAVA_HOME=/home/volkix/prg/jdk25 mvn package -DskipTests
```

1. **Use directly in Euphoria**:

```
include std/java.e as java

atom chart = java:call_static("org.jfree.chart.ChartFactory", "createBarChart", {
    "Revenue 2026", "Category", "EUR", dataset
})
```

Pre-configured Enterprise Dependencies in `pom.xml`:

- **SQLite JDBC** (`org.xerial:sqlite-jdbc`): Native SQLite 3 driver with bundled C engine for Linux x86_64.
- **PostgreSQL JDBC** (`org.postgresql:postgresql`): Full PostgreSQL driver supporting SCRAM-SHA-256 and TLS.
- **Jackson Databind** (`com.fasterxml.jackson.core:jackson-databind`): Fast JSON and RFC 9535 JSONPath engine.
- **FlatLaf Modern Look & Feel** (`com.formdev:flatlaf`): Clean Light and Dark modern UI themes.
- **SLF4J-NOP** (`org.slf4j:slf4j-nop`): Silent logging backend for clean CLI output.

1.6 Self-Contained Single-Binary with Bundled Minimal JRE (`./euphorix-standalone`)

The entire interpreter, including a **stripped-down 60 MB Java 25 runtime**, can be packaged into a **single, standalone 53 MB executable**:

```
# Build ./euphorix-standalone (approx. 53 MB xz-compressed)
make standalone
# or directly:
./tools/build_standalone.sh ./euphorix-standalone
```

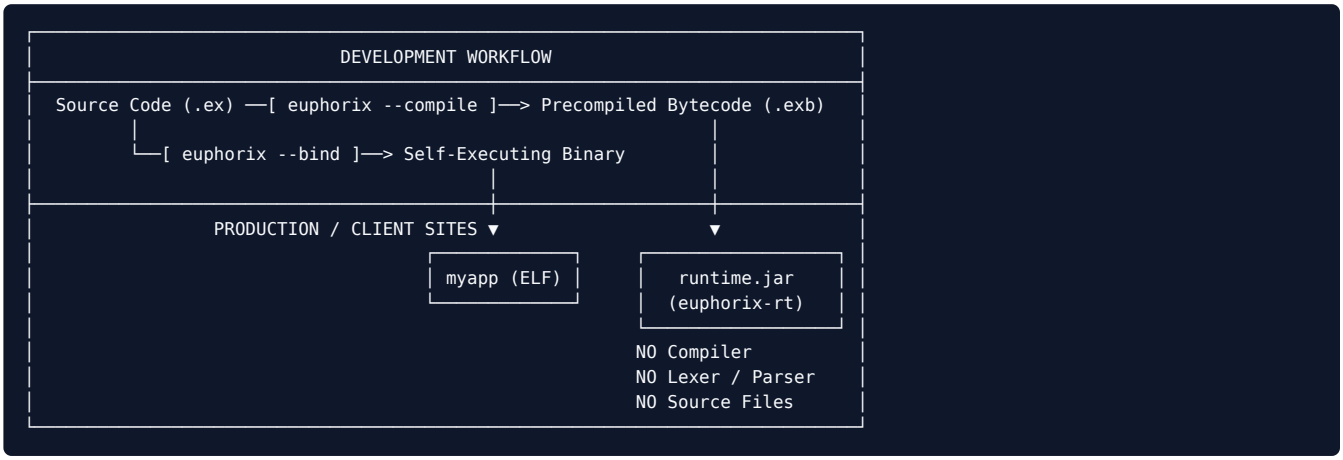
How the Standalone Executable Operates:

1. **`jlink` (Java Linker)** extracts only required modules (`java.base`, `java.desktop`, `java.sql`, `java.net.http`, `java.xml`, etc.), removing debug symbols, manual pages, and unused modules.
2. The `tools/build_standalone.sh` script bundles the mini-JRE and `euphorix.jar` into an archive with an auto-extracting shell header.
3. **Execution**:

- The file `euphorix-standalone` (with alias `eui-standalone`) can be copied to any Linux 64-bit machine (e.g. to `/usr/local/bin/euphorix`).
- **No Java runtime or JDK is required on the target machine!**
- On initial launch, the runtime extracts to `~/.cache/euphorix/java25-runtime/<hash>/`. Subsequent launches execute directly from cache in milliseconds.
- **Advantage over GraalVM**: Full support for runtime JAR loading (`java:load_jar`), Swing & GTK+ desktop GUIs, virtual threads, and HotSpot JIT optimization without closed-world limitations.

1.7 Ahead-of-Time Bytecode Compilation (`.exb`), Executable Binding & Dedicated Runtime (`runtime.jar` / `euphorix-rt`)

For commercial organizations and enterprise deployments looking to **protect proprietary Euphoria source code (`.ex`, `.e`)** and eliminate startup latency, Euphorix provides a complete Ahead-of-Time (AOT) bytecode compiler and dedicated runtime ecosystem:



1. Ahead-of-Time Bytecode Compilation (`-b` / `--compile`)

Compiles Euphoria source files into an optimized standalone binary bytecode archive (`.exb`), resolving and bundling all included standard modules:

```
# Compile app.ex to app.exb:
./euphorix --compile app.ex -o app.exb
# or shorthand:
./euphorix -b app.ex -o app.exb
```

- **Binary Format:** High-density GZIP-compressed binary format starting with magic identifier `EXB1` (`0x45584231`). Encodes register instructions, constant pools, routine signatures, and static type metadata.
- **Intellectual Property Protection:** Source code files (`.ex`, `.e`) are never distributed to client systems.

2. Dedicated Bytecode Runtime (`runtime.jar` / `./euphorix-rt`)

In addition to the full compiler/interpreter fat-JAR (`euphorix.jar`), the build process (`make runtime` or `make java`) produces a stripped, minimal runtime JAR:

```
# Execute precompiled bytecode with minimal runtime:
java -jar runtime.jar app.exb [args...]

# Or directly using the CLI wrapper script:
./euphorix-rt app.exb [args...]
# or alias:
./runtime app.exb
```

- **Stripped of Compiler and Sources:** `runtime.jar` completely excludes `Compiler.class` , `Lexer .class` , `Token.class` , `TokenType .class` , and the entire `/eu_include/**` standard library source tree.
- **Security & Hardening:** End users and operators cannot modify or compile arbitrary code on target production servers.
- **Helpful Diagnostics:** If an uncompiled `.ex` file is mistakenly passed to the runtime, it aborts cleanly with instructions on how to compile it.

3. Startup Performance Benchmarks (`--time` / `-t`)

Ahead-of-Time bytecode skips source lexing, AST construction, and include resolution entirely:

Mode	Command	Startup / Load Time	VM Execution	Total Wall-Clock
Interpreter (`.ex`)	`./euphorix -t tutorial/10_web_und_gui.ex`	**285 – 419 ms**	~280 ms	~570 – 700 ms
Bytecode (`.exb`)	`java -jar runtime.jar -t tutorial/10_web_und_gui.exb`	**24 – 41 ms**	~240 ms	**~270 – 290 ms**

> 🚀 **Result:** Bytecode startup is **7x to 10x faster**, cutting total execution time in half.

4. Self-Executing Bound Binaries (`--bind`)

Creates a standalone executable Linux binary with embedded bytecode payload:

```
./euphorix --bind my_service.ex -o my_service
chmod +x my_service
./my_service --port 8080
```

5. Self-Contained Standalone Single-Binary (`tools/build_standalone.sh --app`)

The premier distribution model bundling the precompiled application with a minimal 60 MB Java 25 JRE:

```
./tools/build_standalone.sh --app my_service.ex -o my_service-standalone
```

- Requires **zero pre-installed Java** and **zero external dependencies**.
- Contains **no source code**—only the precompiled `app.exb` , `runtime.jar` , and static VS-UI assets (`html/`).

1.8 Cross-Platform & Microsoft Windows Support (`euphorix.cmd` / `euphorix.bat`)

Thanks to the pure Java 25 bytecode engine, `euphorix.jar` runs out of the box on **Microsoft Windows** (10, 11, Server) and **macOS** with zero code changes:

- **Dynamic OS Detection**: At compile time, flags such as `WINDOWS` , `WIN32` or `LINUX` , `UNIX` are resolved dynamically via `System.getProperty("os.name")` . The built-in `platform()` returns `2` (`WIN32`) on Windows.
- **Paths & Environment Variables**: `EUINC` automatically splits paths using `File.pathSeparator` (semicolon `;` on Windows, colon `:` on Linux), fully respecting Windows drive letters (`C:\...`).
- **Shell & Process Execution**: Shell commands (`system()` , `JavaInterop.pipeExec`) invoke `cmd.exe /c` automatically on Windows and `/bin/sh -c` on Unix.
- **Windows Launchers**: Convenient command-line wrappers euphorix.cmd and euphorix.bat are provided in the root directory:

```
  euphorix.cmd my_script.ex
```

1.9 Developer Workflow with Hot-Reload File Watcher (`--watch` / `-w`)

When building web APIs, services, or desktop UIs, manual application restarts after each code modification slow down productivity. Passing `--watch` (or short `-w`) directs Euphorix to monitor the script directory and all related files in real time:

```
# Start your script with live file watching
./euphorix --watch my_server.ex

# Or short form:
./euphorix -w my_server.ex
```

- **Under the Hood:** Powered by the JDK 25 `WatchService`, recursively capturing file change events (`ENTRY_CREATE`, `ENTRY_MODIFY`, `ENTRY_DELETE` on `.ex` and `.e` files).
- **Smart Debounce:** Rapid consecutive edits are debounced over a 200ms window to prevent race conditions during multi-file saves.
- **Instant Automatic Reload:** When a change is saved, the running process is gracefully recycled and replaced with the updated source code immediately.

1.10 Linux Daemon Operations & Systemd Units (`tools/make_service.ex`)

For production deployments under Linux, the integrated CLI generator `tools/make_service.ex` produces hardened systemd unit definitions:

```
# Usage:
./euphorix tools/make_service.ex <script.ex> [service-name] [description] [user]

# Example:
./euphorix tools/make_service.ex examples/05_rest_api_microservice.ex euphorix-api "Euphorix Microservice" volkix
```

The tool produces a production-grade systemd service unit (e.g. `euphorix-api.service`) with:

- **Automatic Recovery (`Restart=always` , `RestartSec=5`)**: Recovers automatically from transient errors or unexpected termination.
- **Dedicated User & Working Directory**: Runs unprivileged under the specified service account.
- **Journald Logging**: Standard and error output stream directly into `systemd-journald` .
- **Resource Optimization**: Configures `LimitNOFILE=65536` for high-throughput connection concurrency.

The generator also outputs the exact deployment commands:

```
sudo cp euphorix-api.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable --now euphorix-api
sudo journalctl -u euphorix-api -f
```

1.11 Reverse Proxy Operations (lighttpd, Nginx, Caddy)

For production deployments of web applications and interactive dashboards (std/vsui.e), running the Euphorix service behind a lightweight reverse proxy is standard best practice.

> [!TIP]

> **100% Self-Contained:** When using the 52 MB single-binary (`tools/build_standalone.sh --app app.ex`), the embedded HTTP server, all 22 VS-UI web components, CSS, and JS files are **packaged directly inside the executable**. No separate directories or files need to be deployed to the host server!

lighttpd Configuration (`/etc/lighttpd/conf-available/99-euphorix.conf`)

```
server.modules += ( "mod_proxy" )

$HTTP["host"] == "cockpit.my-company.com" {
    # Forward all dynamic requests and web assets to Euphorix:
    proxy.server = ( "" => (
        ( "host" => "127.0.0.1", "port" => 8089 )
    ))
    proxy.header = (
        "map-host-request" => ( "-" => "$0" )
    )
}
```

Nginx Configuration (`/etc/nginx/sites-available/euphorix.conf`)

```
server {
    listen 80;
    server_name cockpit.my-company.com;

    location / {
        proxy_pass http://127.0.0.1:8089;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_buffering off; # Required for reactive VS-UI updates and Server-Sent Events (SSE)
    }
}
```

1.12 IDE Integration & Language Server Protocol (LSP) for Apache NetBeans & VS Code

To provide a first-class developer experience with error detection and code intelligence, Euphorix includes a complete **Language Server** complying with the **LSP 3.17 specification**:

```
./euphorix --lsp
```

The language server operates over standard I/O (JSON-RPC 2.0) and integrates seamlessly into Apache NetBeans and VS Code.

Supported Capabilities

1. **Real-Time Diagnostics (Syntax Checking):**

- On every change, the compiler performs in-memory analysis and reports errors immediately with line, column, and diagnostic message.
- Red squiggly lines appear in the editor and automatically clear upon fixing the issue.

1. **Code Completion (IntelliSense & Snippets):**

- **Keywords & Snippets:** Typing `if`, `for`, `while`, `function`, `procedure`, `switch`, etc., provides template expansions.
- **Built-in Functions:** Auto-completes core functions (`puts`, `printf`, `length`, `append`, etc.) with signatures.
- **Module Namespaces:** Triggered by `:` (e.g. `vsui:`, `map:`, `json:`, `sequence:`, `io:`, `math:`, `sqlite:`, `http:`), listing all exported procedures/functions.
- **Local Symbols:** Auto-completes routines and constants declared within the active file.

1. **Hover Documentation:**

- Hovering over keywords or routines displays Markdown documentation with signatures and parameter explanations.

1. **Go-to-Definition (`F12` / `Ctrl + Click`):**

- Navigates directly to routine or type declarations.

1. **Document Symbols & Outline:**

- Displays the hierarchical structure of routines and types in the IDE outline panel.

Apache NetBeans Integration (NetBeans 12, 19, 25, 31+)

Apache NetBeans provides built-in support for generic Language Servers (`org-netbeans-modules-lsp-client.jar`).

Automated Setup (Recommended)

Run the included configuration script:

```
./tools/setup_netbeans_lsp.sh
```

This writes the LSP configuration directly into NetBeans preferences (`~/.netbeans/<version>/config/Preferences/org/netbeans/modules/lsp/client.properties`).

Manual Configuration via NetBeans UI

1. Open **Tools** -> **Options** -> **Editor** -> **Language Servers** tab.
2. Click **Add...** and enter:

- **Name:** `Euphorix (OpenEuphoria)`
- **File Extensions:** `ex, e, ew, eu`
- **MIME-Type:** `text/x-euphoria`
- **Server Command:** `/home/volkix/prj/intpret/euphorix --lsp`

1. Click **Apply** and **OK**.

Opening any `.ex` or `.e` file will automatically start the language server.

Visual Studio Code Integration

Extension files are provided in editors/vscode/.

Option A: Install Extension Folder

Copy `editors/vscode` to your VS Code extensions folder:

```
# Linux / macOS:
cp -r editors/vscode ~/.vscode/extensions/euphorix-lsp

# Windows:
xcopy /E /I editors\vscode %USERPROFILE%\.vscode\extensions\euphorix-lsp
```

Option B: Generic LSP Client Extension

If using the **Generic LSP Client** extension in VS Code, add to `settings.json`:

```
{
  "generic-lsp.server": {
    "command": "/home/volkix/prj/intpret/euphorix",
    "args": ["--lsp"],
    "languages": ["euphoria"]
  }
}
```

---

2. Complete Language, Syntax & Standard Reference

2.1 Data Types & Type System

OpenEuphoria features a clean, hierarchical type system:

```
       object (any value: number or sequence)
      /      \
   atom      sequence (nested list: {1, "text", {2, 3}})
    |
integer (48-bit signed integer)
```

- **`integer`**: 48-bit signed integers in range `[-140737488355328, +140737488355327]`.
- **`atom`**: Any numerical value—integer or 64-bit IEEE-754 floating-point (`double`).
- **`sequence`**: Ordered, **1-based** dynamic array (`{}`). Strings are sequences of byte values.

Multiline Strings & Raw Strings (Backticks & Triple-Quotes)

Euphorix supports three convenient string literal styles:

1. **Classic Strings (`"..."`)**: Standard single-line strings with escape sequences (`\n`, `\t`, `\"`, `\\`).
2. **Backtick Strings (`` `...` ``)**: Multiline raw strings. Double quotes (`"`) do **not** need escaping. Perfect for JSON, SQL, and HTML:

```
sequence json_doc = `
{
    "name": "Schulze Maschinenbau",
    "status": "Tier-A",
    "turnover": 142500.00
}
`
```

1. **Triple-Quote Strings (`"""..."""`)**: Multiline raw text blocks (familiar from Python/Kotlin). Line breaks and embedded quotes are preserved literally:

```
sequence query = """
SELECT kdnr, name, turnover
FROM customers
WHERE status = 'Tier-A'
"""
```

- **`object`**: Universal type capable of holding an `atom` or a `sequence`.

Variables, Constants & Enums

```
integer count = 42
atom price = 19.95
sequence name = "Euphoria"
constant VAT = 0.19, CURRENCY = "EUR"
enum STATUS_READY = 1, STATUS_ACTIVE, STATUS_DONE
```

Custom Types (`type ... end type`)

```
type percentage(object x)
    return integer(x) and x >= 0 and x <= 100
end type

percentage discount = 15   -- Validated dynamically upon assignment
```

2.2 Visibility, Namespaces & Includes

Keyword	Visibility
(none)	File-local (`local`).
`export`	Accessible only to files that directly include this file.
`public`	Accessible to includers (and re-exported by public includes).
`global`	Visible program-wide across all files.
`override`	Selectively overrides a built-in or global routine.

```
-- In mylib.e:
namespace mylib
public function double_val(atom x)
    return x * 2
end function

-- In main program:
include mylib.e as m
? m:double_val(21)   -- Output: 42
```

2.3 Operators, Indexing & Slicing

Indexing and Slicing (1-based)

- `s[1]` : First element
- `s[$]` : Last element (`$` represents `length(s)`)
- `s[$-1]` : Second to last element
- `s[2..4]` : Sub-sequence slice from index 2 to 4
- `s[2..$-1]` : Slice excluding first and last elements

Concatenation & Element-wise Operations

- `&` : Flattens and concatenates sequences (`"Hello " & "World"`).
- Arithmetic operators (`+` , `-` , `*` , `/`) apply element-wise across sequences:

`{10, 20} * 2` yields `{20, 40}` .

- **Equality Testing**: The `=` operator performs element-wise comparisons. To test whether two sequences or strings are identical as a whole, use **`equal(a, b)`** !

2.4 Control Structures & Loops

```
-- If statement
if condition then
    action1()
elsif alt_condition then
    action2()
else
    action3()
end if

-- Switch statement
switch cmd do
    case "start", "run" then
        start_service()
        break
    case "stop" then
        stop_service()
        break
    case else
        unknown_cmd()
end switch

-- For loop
for i = 1 to 10 by 2 do
    ? i
end for

-- While loop
while is_running do
    process_next()
end while

-- Loop-until loop
loop do
    step()
    until is_done
```

2.5 Procedures, Functions & Callbacks

- **`procedure`**: Executes code without returning a value.
- **`function`**: Returns a value using `return <val>`.
- **Default Parameters**: Parameters can have default values (`function f(integer a, integer b = 10)`).
- **Multi-assignment (Destructuring)**:

```
  function coords()
      return {12.5, 48.2}
  end function
  atom x, y
  {x, y} = coords()
```

- **Routine IDs & Dynamic Invocation**:

`integer rid = routine_id("my_function")`

`object res = call_func(rid, {arg1, arg2})`

2.6 Conditional Compilation (`ifdef`)

```
ifdef LINUX then
    constant PATH_SEP = "/"
elsifdef WINDOWS then
    constant PATH_SEP = "\\"
end ifdef
```

Predefined symbols: `EU4`, `EU4_0`, `EU4_1`, `UNIX`, `LINUX`, `BITS64`, `LONG64`, `JAVA`, `JAVA25`, `CONSOLE`.

2.7 All Built-in Routines

Type Checking & Comparisons

- `integer(x)` : `1` if integer, else `0` .
- `atom(x)` : `1` if number (`integer` or `double`), else `0` .
- `sequence(x)` : `1` if sequence/string, else `0` .
- `object(x)` : `1` if initialized.
- `equal(a, b)` : Deep equality test (`1` or `0`).
- `compare(a, b)` : Lexicographical comparison (`-1` , `0` , `+1`).

Sequence & String Operations

- `length(s)` : Number of elements / bytes in `s` .
- `append(s, x)` : Appends `x` as a single element.
- `prepend(s, x)` : Prepends `x` as a single element.
- `repeat(item, count)` : Creates a sequence of `count` identical items.
- `find(needle, haystack, start=1)` : Finds element index in sequence (`0` if not found).
- `match(sub, haystack, start=1)` : Finds substring index in sequence (`0` if not found).
- `insert(s, item, index)` / `splice(s, items, index)` : Inserts element or slice.
- `remove(s, start, stop=start)` : Removes slice range.
- `replace(s, rep, start, stop=start)` : Replaces slice range with `rep` .

Console & File I/O

- `puts(fn, text)` : Writes string to file channel (`1` = STDOUT, `2` = STDERR).
- `printf(fn, fmt, values)` : Formatted output (`%d` , `%f` , `%s` , `%x`).
- `sprintf(fmt, values)` : Formatted string creation.
- `gets(fn)` : Reads line from file channel (or `-1` on EOF).
- `getc(fn)` : Reads single byte (or `-1` on EOF).
- `open(path, mode)` : Opens file (`"r"` , `"w"` , `"a"` , `"rb"` , `"wb"`).
- `close(fn)` : Closes file.
- `seek(fn, pos)` / `where(fn)` : Moves or queries file pointer position.

2.8 Complete Reference of All 74 Include Files & Standard Modules (`std/.e` & `euphoria/ .e`)

This section documents every single include file in the Euphorix standard library, its exact purpose, and all public procedures (`public procedure`) and functions (`public function`).

Category Overview:

- 2.8.1 Data Structures, Maps & Algorithms (10 Modules)
- 2.8.2 I/O, File System, Console & Operating System (8 Modules)
- 2.8.3 Text Processing, Unicode & Localization (10 Modules)
- 2.8.4 Mathematics, Statistics & Random Numbers (5 Modules)
- 2.8.5 Date, Time & Calendar (1 Modules)
- 2.8.6 Networking, Internet, DNS & URLs (5 Modules)
- 2.8.7 Data Formats, XML, JSON & Databases (6 Modules)
- 2.8.8 Desktop GUI & Web Frontend Frameworks (6 Modules)
- 2.8.9 Java Interoperability, C-FFI & Machine Memory (7 Modules)
- 2.8.10 Modern Concurrency, Threads & Multitasking (3 Modules)
- 2.8.11 Error Handling, Testing & Utilities (3 Modules)
- 2.8.12 Windows Platform-Specific Modules (3 Modules)
- 2.8.13 Euphoria Internals, Debugger & Syntax Tokenizer (7 Modules)

2.8.1 Data Structures, Maps & Algorithms

`std/sequence.e` (Namespace: `stdseq`)

- **Purpose:** Comprehensive sequence manipulation: multidimensional slicing, transformations, filtering, flattening, combining, and columnar operations.

- **Public Functions & Procedures (39):**
- `public function binop_ok(object a, object b)` : Checks whether two objects can perform a sequence operation together.
- `public function fetch(sequence source, sequence indexes)` : Retrieves an element nested arbitrarily deep into a sequence.
- `public function store(sequence target, sequence indexes, object x)` : Stores something at a location nested arbitrarily deep into a sequence.
- `public function valid_index(sequence st, object x)` : Checks whether an index exists on a sequence.
- `public function rotate(sequence source, integer shift, integer start=1, integer stop=length(source))` : Rotates a slice of a sequence.
- `public function columnize(sequence source, object cols = {}, object defval = 0)` : Converts a set of sub sequences into a set of "columns.".
- `public function apply(sequence source, integer rid, object userdata = {})` : Applies a function to every element of a sequence returning a new sequence of the same size.
- `public function mapping(object source_arg, sequence from_set, sequence to_set, integer one_level = 0)` : Changes each item from `source_arg` found in `from_set` into the corresponding item in `to_set`.
- `public function reverse(object target, integer pFrom = 1, integer pTo = 0)` : Reverses the order of elements in a sequence.
- `public function shuffle(object seq)` : Shuffles the elements of a sequence.
- `public function series(object start, object increment, integer count = 2, integer op = '+')` : Returns a new sequence built as a series from a given object.
- `public function repeat_pattern(object pattern, integer count)` : Returns a periodic sequence, given a pattern and a count.
- `public function pad_head(object target, integer size, object ch=' ')` : Pads the beginning of a sequence with an object so as to meet a minimum length condition.
- `public function pad_tail(object target, integer size, object ch=' ')` : Pads the end of a sequence with an object so as to meet a minimum length condition.
- `public function add_item(object needle, sequence haystack, integer pOrder = 1)` : Adds an item to the sequence if its not already there. If it already exists in the list, the list is returned unchanged.
- `public function remove_item(object needle, sequence haystack)` : Removes an item from the sequence.
- `public function mid(sequence source, atom start, atom len)` : Returns a slice of a sequence, given by a starting point and a length.
- `public function slice(sequence source, atom start = 1, atom stop = 0)` : Returns a portion of the supplied sequence.
- `public function vslice(sequence source, atom colno, object error_control=0)` : Performs a vertical slice on a nested sequence.
- `public function patch(sequence target, sequence source, integer start, object filler = ' ')` : Changes a sequence slice, possibly with padding.
- `public function remove_all(object needle, sequence haystack)` : Removes all occurrences of some object from a sequence.
- `public function retain_all(object needles, sequence haystack)` : Keeps all occurrences of a set of objects from a sequence and removes all others.
- `public function filter(sequence source, object rid, object userdata = {}, object rangetype = "")` : Filters a sequence based on a user supplied comparator function.
- `public function extract(sequence source, sequence indexes)` : Picks out from a sequence a set of elements according to the supplied set of indexes.
- `public function project(sequence source, sequence coords)` : Creates a list of sequences based on selected elements from sequences in the source.
- `public function split( sequence st, object delim=' ', integer no_empty = 0, integer limit=0)` : Splits a sequence on separator delimiters into a number of sub-sequences.
- `public function split_any(sequence source, object delim=", \t|", integer limit=0, integer no_empty=0)` : Splits a sequence by any of the separators in the list of delimiters. If `limit` is `> 0` then limit the number of tokens that will be split to limit.
- `public function join(sequence items, object delim=" ")` : Joins sequences together using a delimiter.
- `public function breakup(sequence source, object size, integer style = BK_LEN)` : Breaks up a sequence into multiple sequences of a given length.
- `public function flatten(sequence pSource, object pDelim={})` : Removes all nesting from a sequence, while retaining all its atoms.
- `public function pivot(object data_p, object pivot_p = 0)` : Returns a sequence of three sub-sequences. The sub-sequences contain all the elements less than the supplied pivot value, equal to the pivot, and greater than the pivot.
- `public function build_list( sequence source, object transformer, integer singleton = 1, object user_data = {})` : Implements "List Comprehension" or building a list based on the contents of another list.
- `public function transform( sequence source_data, object transformer_rids)` : Transforms the input sequence by using one or more user-supplied transformers.
- `public function transmute(sequence source_data, sequence current_items, sequence new_items, integer start=1, integer limit = length(source_data))` : Replaces all instances of any element from the `current_items` sequence that occur in the `source_data` sequence with the corresponding item from the `new_items` sequence.
- `public function sim_index(sequence A, sequence B)` : Calculates the similarity between two sequences.

- `public function remove_subseq( sequence source_list, object alt_value = SEQ_NOALT)` : Removes all sub-sequences from the supplied sequence, optionally replacing them with a supplied alternative value. One common use is to remove all strings from a mixed set of numbers and strings.
- `public function remove_dups(sequence source_data, integer proc_option = RD_PRESORTED)` : Removes duplicate elements.
- `public function combine(sequence source_data, integer proc_option = COMBINE_SORTED)` : Combines all the sub-sequences into a single, optionally sorted, list.
- `public function minsize(object source_data, integer min_size = floor(length(source_data) * 1.5), object new_data = 0)` : Ensures that the supplied sequence is at least the supplied minimum length.

`std/map.e` (Namespace: `map`)

- **Purpose:** High-performance key-value hash map data structures with dynamic rehashing, iterators, filtering, and transformation functions.
- **Public Functions & Procedures (26):**
- `public function calc_hash(object key_p, integer max_hash_p)` : Calculates a Hashing value from the supplied data.
- `public function threshold(integer new_value_p = 0)` : Deprecated.
- `public function type_of(map the_map_p)` : Deprecated.
- `public procedure rehash( map the_map_p, integer requested_size_p = 0 )` : Changes the width (that is the number of buckets) of a map.
- `public function new( integer initial_size_p = DEFAULT_SIZE )` : Creates a new map data structure.
- `public function new_extra(object the_map_p, integer initial_size_p = 8)` : Returns either the supplied map or a new map.
- `public function compare(map map_1_p, map map_2_p, integer scope_p = 'd')` : Compares two maps to test equality.
- `public function has( map the_map_p, object key )` : Checks whether map has a given key.
- `public function get( map the_map_p, object key, object default = 0 )` : Retrieves the value associated to a key in a map.
- `public function nested_get( map the_map_p, sequence the_keys_p, object default_value_p = 0)` : Returns the value given a nested key.
- `public procedure put( map the_map_p, object key, object val, object op = PUT, object deprecated = 0 )` : Adds or updates an entry on a map.
- `public procedure nested_put( map the_map_p, sequence the_keys_p, object the_value_p, integer operation_p = PUT, object deprecated_trigger_p = 0)` : Adds or updates an entry on a map.
- `public procedure remove( map the_map_p, object key )` : Removes an entry with given key from a map.
- `public procedure clear( map the_map_p )` : Removes all entries in a map.
- `public function size( map the_map_p )` : Returns the number of entries in a map.
- `public function statistics(map the_map_p)` : Retrieves characteristics of a map.
- `public function keys( map the_map_p, integer sorted_result = 0 )` : Returns all keys in a map.
- `public function values( map the_map, object keys=0, object default_values=0 )` : Returns values, without their keys, from a map.
- `public function pairs( map the_map, integer sorted_result = 0 )` : Returns all key:value pairs in a map.
- `public procedure optimize(map the_map_p, integer deprecated_max_p = 0, atom deprecated_grow_p = 0)` : Rehashes a map to increase performance. This procedure is deprecated in favor of `rehash`.
- `public function load_map(object input_file_name)` : Loads a map from a file.
- `public function save_map(map the_map_, object file_name_p, integer type_ = SM_TEXT)` : Saves a map to a file.
- `public function copy(map source_map, object dest_map=0, integer put_operation = PUT)` : Duplicates a map.
- `public function new_from_kvpairs(sequence kv_pairs)` : Converts a set of key:value pairs to a map.
- `public function new_from_string(sequence kv_string)` : Converts a set of key:value pairs contained in a string to a map.
- `public function for_each(map source_map, integer user_rid, object user_data = 0, integer in_sorted_order = 0, integer signal_boundary = 0)` : Calls a user-defined routine for each of the items in a map.

`std/stack.e` (Namespace: `stack`)

- **Purpose:** Generic LIFO and FIFO stack data structures supporting push, pop, peek, duplicate, rotate, and capacity management operations.
- **Public Functions & Procedures (14):**
- `public function new(integer typ = FILO)` : Creates a new stack.
- `public function is_empty(stack sk)` : Determines whether a stack is empty.
- `public function size(stack sk)` : Returns how many elements a stack has.
- `public function at(stack sk, integer idx = 1)` : Fetches a value from the stack without removing it from the stack.
- `public procedure push(stack sk, object value)` : Adds something to a stack.
- `public function top(stack sk)` : Retrieve the top element on a stack.
- `public function last(stack sk)` : Retrieves the end element on a stack.
- `public function pop(stack sk, integer idx = 1)` : Removes an object from a stack.
- `public function peek_top(stack sk, integer idx = 1)` : Gets an object, relative to the top, from a stack.
- `public function peek_end(stack sk, integer idx = 1)` : Gets an object, relative to the end, from a stack.

- `public procedure swap(stack sk)` : Swaps the top two elements of a stack.
- `public procedure dup(stack sk)` : Repeats the top element of a stack.
- `public procedure set(stack sk, object val, integer idx = 1)` : Updates a value on the stack.
- `public procedure clear(stack sk)` : Wipes out a stack.

`std/sort.e` (Namespace: `stdsort`)

- **Purpose:** Highly optimized sorting algorithms (merge sort, insertion sort) supporting ascending/descending order and custom comparison callback functions.
- **Public Functions & Procedures (5):**
- `public function sort(sequence x, integer order = ASCENDING)` : Sorts the elements of a sequence into ascending order.
- `public function custom_sort(integer custom_compare, sequence x, object data = {}, integer order = NORMAL_ORDER)` : Sorts the elements of a sequence according to a user-defined order.
- `public function sort_columns(sequence x, sequence column_list)` : Sorts the rows in a sequence according to a user-defined column order.
- `public function merge(sequence a, sequence b, integer compfunc = -1, object userdata = "")` : Merges two pre-sorted sequences into a single sequence.
- `public function insertion_sort(sequence s, object e = "", integer compfunc = -1, object userdata = "")` : Sorts a sequence and optionally another object together.

`std/search.e` (Namespace: `search`)

- **Purpose:** Comprehensive sequence and string searching routines including find_all, match_all, prefix/suffix tests (begins, ends), binary search, and element replacement.
- **Public Functions & Procedures (18):**
- `public function find_any(object needles, sequence haystack, integer start=1)` : Finds any element from a list inside a sequence. Returns the location of the first hit.
- `public function match_any(sequence needles, sequence haystack, integer start=1)` : Determines if any element from `needles` is in `haystack`.
- `public function find_each(sequence needles, sequence haystack, integer start=1)` : Finds all instances of any element from the needle sequence that occur in the haystack sequence. Returns a list of indexes.
- `public function find_all(object needle, sequence haystack, integer start=1)` : Finds all occurrences of an object inside a sequence, starting at some specified point.
- `public function find_all_but( object needle, sequence haystack, integer start = 1 )` : Finds all non-occurrences of an object inside a sequence, starting at some specified point.
- `public function find_nested(object needle, sequence haystack, integer flags=0, integer rtn_id=types:NO_ROUTINE_ID)` : Finds any object (among a list) in a sequence of arbitrary shape at arbitrary nesting.
- `public function rfind(object needle, sequence haystack, integer start=length(haystack))` : Finds a needle in a haystack in reverse order.
- `public function find_replace(object needle, sequence haystack, object replacement, integer max=0)` : Finds a `needle` in the `haystack`, and replaces all or upto `max` occurrences with `replacement`.
- `public function match_replace(object needle, sequence haystack, object replacement, integer max=0)` : Finds a "needle" in a "haystack", and replace any, or only the first few, occurrences with a replacement.
- `public function binary_search(object needle, sequence haystack, integer start_point = 1, integer end_point = 0)` : Finds a "needle" in an ordered "haystack". Start and end point can be given for the search.
- `public function match_all(sequence needle, sequence haystack, integer start=1)` : Matches all items of haystack in needle.
- `public function rmatch(sequence needle, sequence haystack, integer start=length(haystack))` : Tries to match a needle against some slice of a haystack in reverse order.
- `public function begins(object sub_text, sequence full_text)` : Tests whether a sequence is the head of another one.
- `public function ends(object sub_text, sequence full_text)` : Tests whether a sequence ends another one.
- `public function is_in_range(object item, sequence range_limits, sequence boundries = "[]")` : Tests to see if the `item` is in a range of values supplied by `range_limits`.
- `public function is_in_list(object item, sequence list)` : Tests to see if the `item` is in a list of values supplied by `list`.
- `public function lookup(object find_item, sequence source_list, sequence target_list, object def_value = 0)` : Returns the corresponding element from the target list if the supplied item is in the source list.
- `public function vlookup(object find_item, sequence grid_data, integer source_col, integer target_col, object def_value = 0)` : Returns the corresponding element from the target column if the supplied item is in a source grid column.

`std/serialize.e` (Namespace: `serialize`)

- **Purpose:** Binary serialization and deserialization of arbitrary, deeply nested Euphoria objects for persistent storage or socket transmission.
- **Public Functions & Procedures (4):**
- `public function deserialize(object sdata, integer pos = 1)` : Converts a serialized object in to a standard Euphoria object.

- `public function serialize(object x)` : Converts a standard Euphoria object in to a serialized version of it.
- `public function dump(sequence data, sequence filename)` : Saves a Euphoria object to disk in a binary format.
- `public function load(sequence filename)` : Restores a Euphoria object that has been saved to disk by `dump` .

`std/hash.e` (Namespace: `stdhash`)

- **Purpose:** Definition of hashing algorithm types and enumeration constants utilized by the native built-in `hash(source, algo)` engine.
- **Notes & Key Constants:** Defines hash algorithms `HSIEH30` , `HSIEH32` , `ADLER32` , `FLETCHER32` , `MD5` , and `SHA256` . Executed via the built-in function `hash(object source, atom algo)` .

`std/types.e` (Namespace: `types`)

- **Purpose:** Extended data type validators, character set classifications (ASCII, alphabetic, punctuation, whitespace), and boolean definitions.
- **Public Functions & Procedures (4):**
- `public function char_test(object test_data, sequence char_set)` : Determines whether one or more characters are in a given character set.
- `public procedure set_default_charsets()` : Sets all the defined character sets to their default definitions. Example 1: <eucode> set_default_charsets() </eucode>.
- `public function get_charsets()` : Gets the definition for each of the defined character sets.
- `public procedure set_charsets(sequence charset_list)` : Sets the definition for one or more defined character sets.

`std/flags.e` (Namespace: `flags`)

- **Purpose:** Bitwise flag manipulation, bitmask analysis, and human-readable string representation of option bit flags.
- **Public Functions & Procedures (2):**
- `public function which_bit( object theValue )` : Tests if the supplied value has only a single bit on in its representation.
- `public function flags_to_string(object flag_bits, sequence flag_names, integer expand_flags = 0 )` : Returns a list of strings that represent the human-readable identities of the supplied flag or flags.

`std/primes.e` (Namespace: `primes`)

- **Purpose:** Prime number generation using the Sieve of Eratosthenes, primality testing, and next-prime calculation.
- **Public Functions & Procedures (3):**
- `public function calc_primes(integer approx_limit, atom time_limit_p = 10)` : Returns all the prime numbers below a threshold, with a cap on computation time.
- `public function next_prime(integer n, object fail_signal_p = -1, atom time_out_p = 1)` : Returns the next prime number on or after the supplied number.
- `public function prime_list(integer top_prime_p = 0)` : Returns a list of prime numbers.

2.8.2 I/O, File System, Console & Operating System

`std/io.e` (Namespace: `io`)

- **Purpose:** Comprehensive file and stream I/O: full-file read/write (`read_file` , `write_file`), line-based reading, buffered binary integer I/O, and file locking.
- **Public Functions & Procedures (19):**
- `public function get_bytes(integer fn, integer n)` : Reads the next bytes from a file.
- `public function get_integer32(integer fh)` : Reads the next four bytes from a file and returns them as a single integer.
- `public function get_integer16(integer fh)` : Reads the next two bytes from a file and returns them as a single integer.
- `public procedure put_integer32(integer fh, atom val)` : Writes the supplied integer as four bytes to a file.
- `public procedure put_integer16(integer fh, atom val)` : Writes the supplied integer as two bytes to a file.
- `public function get_dstring(integer fh, integer delim = 0)` : Read a delimited byte string from an opened file.
- `public function seek(file_number fn, file_position pos)` : Seek (move) to any byte position in a file.
- `public function where(file_number fn)` : Retrieves the current file position for an opened file or device.
- `public procedure flush(file_number fn)` : Forces writing any buffered data to an open file or device.
- `public function lock_file(file_number fn, lock_type t, byte_range r={})` : Locks a file so access is restricted.
- `public procedure unlock_file(file_number fn, byte_range r={})` : Unlock (a portion of) an open file.
- `public function read_lines(object file)` : Reads the contents of a file as a sequence of lines.
- `public function process_lines(object file, integer proc, object user_data = 0)` : Processes the contents of a file, one line at a time.
- `public function write_lines(object file, sequence lines)` : Write a sequence of lines to a file.
- `public function append_lines(sequence file, sequence lines)` : Appends a sequence of lines to a file.

- `public function read_file(object file, integer as_text = BINARY_MODE)` : Reads the contents of a file as a single sequence of bytes.
- `public function write_file(object file, sequence data, integer as_text = BINARY_MODE)` : Write a sequence of bytes to a file.
- `public procedure writef(object fm, object data={}, object fn = 1, object data_not_string = 0)` : Writes formatted text to a file.
- `public procedure writefln(object fm, object data={}, object fn = 1, object data_not_string = 0)` : Writes formatted text to a file, ensuring that a new line is also output.

`std/filesys.e` (Namespace: `filesys`)

- **Purpose:** Cross-platform file system operations: directory listings (`dir`), directory traversal (`walk_dir`), copy, move, rename, delete, and path parsing.
- **Public Functions & Procedures (37):**
- `public function dir(sequence name)` : Returns directory information for the specified file or directory.
- `public function current_dir()` : Return the name of the current working directory.
- `public function chdir(sequence newdir)` : Sets a new value for the current directory.
- `public function walk_dir(sequence path_name, object your_function, integer scan_subdirs = types:FALSE, object dir_source = types:NO_ROUTINE_ID)` : Generalized Directory Walker.
- `public function create_directory(sequence name, integer mode=448, integer mkparent = 1)` : Creates a new directory.
- `public function create_file(sequence name)` : Create a new file.
- `public function delete_file(sequence name)` : Deletes a file.
- `public function curdir(integer drive_id = 0)` : Returns the current directory, with a trailing SLASH.
- `public function init_curdir()` : Returns the original current directory.
- `public function clear_directory(sequence path, integer recurse = 1)` : Clears (deletes) a directory of all files, but retaining sub-directories.
- `public function remove_directory(sequence dir_name, integer force=0)` : Removes a directory.
- `public function pathinfo(sequence path, integer std_slash = 0)` : Parses a fully qualified pathname.
- `public function dirname(sequence path, integer pcd = 0)` : Returns the directory name of a fully qualified filename.
- `public function pathname(sequence path)` : Returns the directory name of a fully qualified filename.
- `public function filename(sequence path)` : Returns the file name portion of a fully qualified filename.
- `public function filebase(sequence path)` : Returns the base filename of path.
- `public function fileext(sequence path)` : Returns the file extension of a fully qualified filename.
- `public function driveid(sequence path)` : Returns the drive letter of the path on *Windows* platforms.
- `public function defaultext( sequence path, sequence defext)` : Returns the supplied filepath with the supplied extension, if the filepath does not have an extension already.
- `public function absolute_path(sequence filename)` : Determines if the supplied string is an absolute path or a relative path.
- `public function canonical_path(sequence path_in, integer directory_given = 0, case_flagset_type case_flags = AS_IS)` : Returns the full path and file name of the supplied file name.
- `public function abbreviate_path(sequence orig_path, sequence base_paths = {})` : Returns a path string to the supplied file which is shorter than the given path string.
- `public function split_path(sequence fname)` : Split a filename into path segments.
- `public function join_path(sequence path_elements)` : Join multiple path segments into a single path/filename.
- `public function file_type(sequence filename)` : Gets the type of a file.
- `public function file_exists(object name)` : Checks to see if a file exists.
- `public function file_timestamp(sequence fname)` : Gets the timestamp of the file.
- `public function copy_file(sequence src, sequence dest, integer overwrite = 0)` : Copies a file.
- `public function move_file(sequence src, sequence dest, integer overwrite = 0)` : Moves a file to a new location or directory.
- `public function rename_file(sequence old_name, sequence new_name, integer overwrite = 0)` : Renames a file or directory on the file system.
- `public function file_length(sequence filename)` : Returns the size of a file.
- `public function locate_file(sequence filename, sequence search_list = {}, sequence subdir = {})` : Locates a file by looking in a set of directories for it.
- `public function disk_size(object disk_path)` : Returns some information about a disk drive.
- `public function disk_metrics(object disk_path)` : Returns disk space metrics (total size, available space) for path.
- `public function dir_size(sequence dir_path, integer count_all = 0)` : Returns the amount of space used by a directory.
- `public function temp_file(sequence temp_location = "", sequence temp_prefix = "", sequence temp_extn = "_T_", integer reserve_temp = 0)` : Returns a file name that can be used as a temporary file.
- `public function checksum(sequence filename, integer size = 4, integer usename = 0, integer return_text = 0)` : Returns a checksum value for the specified file.

`std/os.e` (Namespace: `os`)

- **Purpose:** Operating system interaction: environment variables (`setenv` , `getenv`), process ID query (`get_pid`), OS platform detection, and system execution.
- **Public Functions & Procedures (7):**
- `public function instance()` : Returns `hInstance` on *Windows* and Process ID (pid) on *Unix*.
- `public function get_pid()` : Returns the ID of the current Process (pid).
- `public function uname()` : Retrieves the name of the host OS.
- `public function is_win_nt()` : Reports whether the host system is a newer Windows version (NT/2K/XP/Vista).
- `public function setenv(sequence name, sequence val, integer overwrite=1)` : Sets an environment variable.
- `public function unsetenv(sequence env)` : Unsets an environment variable.
- `public procedure sleep(atom t)` : Suspend thread execution for `t` seconds.

`std/cmdline.e` (Namespace: `cmdline`)

- **Purpose:** Declarative command-line argument and option parsing with automatic validation, default values, and help screen generation (`show_help`).
- **Public Functions & Procedures (4):**
- `public procedure show_help(sequence opts, object add_help_rid=-1, sequence cmds = command_line(), object parse_options = {})` : Shows the help message for the given opts.
- `public function cmd_parse(sequence opts, object parse_options = {}, sequence cmds = command_line())` : Parses command line options and optionally calls procedures based on these options.
- `public function build_commandline(sequence cmds)` : Returns a text string based on the set of supplied strings.
- `public function parse_commandline(sequence cmdline)` : Parses a command line string breaking it into a sequence of command line options.

`std/console.e` (Namespace: `console`)

- **Purpose:** Terminal and console control: screen dimensions, ANSI/console colors, cursor positioning, non-blocking key presses, and screen text buffering.
- **Public Functions & Procedures (20):**
- `public function has_console()` : Determines if the process has a console (terminal) window.
- `public function key_codes(object codes = 0)` : Gets and sets the keyboard codes used internally by Euphoria.
- `public function set_keycodes( object kcfile )` : Changes the default codes returned by the keyboard.
- `public procedure allow_break( types:boolean b)` : Sets the behavior of Control+C and Control+Break keys.
- `public function check_break()` : Returns the number of Control+C and Control+Break key presses.
- `public function wait_key()` : Waits for user to press a key, unless any is pending, and returns key code.
- `public procedure any_key(sequence prompt="Press Any Key to continue...", integer con = 1)` : Displays a prompt to the user and waits for any key.
- `public procedure maybe_any_key(sequence prompt="Press Any Key to continue...", integer con = 1)` : Displays a prompt to the user and waits for any key. *Only* if the user is running under a GUI environment.
- `public function prompt_number(sequence prompt, sequence range={})` : Displays a prompts then waits for the user to enter a number, which is returned.
- `public function prompt_string(sequence prompt)` : Prompts the user to enter a string of text.
- `public function get_screen_char(positive_atom line, positive_atom column, integer fgbg = 0)` : Gets the value and attribute of the character at a given screen location.
- `public procedure put_screen_char(positive_atom line, positive_atom column, sequence char_attr)` : Stores and displays a sequence of characters with attributes at a given location.
- `public function attr_to_colors(integer attr_code)` : Converts an attribute code to its foreground and background color components.
- `public function colors_to_attr(object fgbg, integer bg = 0)` : Converts a foreground and background color set to its attribute code format.
- `public procedure display_text_image(text_point xy, sequence text)` : Displays a text image in any text mode.
- `public function save_text_image(text_point top_left, text_point bottom_right)` : Copies a rectangular block of text out of screen memory.
- `public function text_rows(positive_int rows)` : Sets the number of lines on a text-mode screen.
- `public procedure cursor(integer style)` : Selects a style of cursor.
- `public procedure free_console()` : Frees (deletes) any console window associated with your program.
- `public procedure display( object data_in, object args = 1, integer finalnl = -918_273_645)` : Displays the supplied data on the console screen at the current cursor position.

`std/pipeio.e` (Namespace: `pipeio`)

- **Purpose:** Subprocess piping and stream redirection (STDIN, STDOUT, STDERR) powered by Java 25 Virtual Threads without POSIX `fork()` .
- **Public Functions & Procedures (7):**

- `public function create()` : Creates pipes for inter-process communication. Provided for backwards compatibility with legacy OpenEuphoria syntax: `object p = exec(cmd, create())` .
- `public function exec(sequence cmd, sequence pipe = {})` : Executes command with inter-process communication pipes.
- `public function read(atom fd, integer bytes)` : Reads up to `bytes` from pipe handle `fd` .
- `public function write(atom fd, sequence str)` : Writes `str` to pipe handle `fd` .
- `public function close(atom fd)` : Closes handle `fd` .
- `public procedure kill(process p, atom signal = 15)` : Closes pipes and terminates process `p` .
- `public function error_no()` : Gets error no from last call to a pipe function.

`std/get.e` (Namespace: `stdget`)

- **Purpose:** Parsing of Euphoria literal representations, numbers, and structured expressions from strings or open file streams (`value` , `get`).
- **Public Functions & Procedures (3):**
- `public function get(integer file, integer offset=0, integer answer=GET_SHORT_ANSWER)` : Reads from an open file a human-readable string of characters representing a Euphoria object. Converts the string into the numeric value of that object.
- `public function value(sequence st, integer start_point=1, integer answer=GET_SHORT_ANSWER)` : Reads, from a string, a human-readable string of characters representing a Euphoria object. Converts the string into the numeric value of that object.
- `public function defaulted_value(object st, object def, integer start_point=1)` : Calls the `value` function and returns the resulting value on success or the default default on failure.

`std/pretty.e` (Namespace: `pretty`)

- **Purpose:** Structured, indented, human-readable pretty-printing and formatting of arbitrary complex nested sequences.
- **Public Functions & Procedures (2):**
- `public procedure pretty_print(integer fn, object x, sequence options = PRETTY_DEFAULT )` : Prints an object to a file or device using braces `{ , , , }` , indentation, and multiple lines to show the structure.
- `public function pretty_sprint(object x, sequence options = PRETTY_DEFAULT )` : Formats an object using braces { , , , }, indentation, and multiple lines to show the structure.

2.8.3 Text Processing, Unicode & Localization

`std/text.e` (Namespace: `text`)

- **Purpose:** Extended string and text manipulation: whitespace trimming (`trim`), padding (`pad_head` , `pad_tail`), casing, and template formatting.
- **Public Functions & Procedures (15):**
- `public function sprint(object x)` : Returns the representation of any Euphoria object as a string of characters.
- `public function trim_head(sequence source, object what=" \t\r\n", integer ret_index = 0)` : Trims all items in the supplied set from the leftmost (start or head) of a sequence.
- `public function trim_tail(sequence source, object what=" \t\r\n", integer ret_index = 0)` : Trims all items in the supplied set from the rightmost (end or tail) of a sequence.
- `public function trim(sequence source, object what=" \t\r\n", integer ret_index = 0)` : Trims all items in the supplied set from both the left end (head/start) and right end (tail/end) of a sequence.
- `public procedure set_encoding_properties(sequence en = "", sequence lc = "", sequence uc = "")` : Sets the table of lowercase and uppercase characters that is used by `lower` and `upper` .
- `public function get_encoding_properties( )` : Gets the table of lowercase and uppercase characters that is used by `lower` and `upper` .
- `public function lower(object x)` : Converts all characters of a string or character to lower case.
- `public function upper(object x)` : Converts an atom or sequence to upper case.
- `public function proper(sequence x)` : Converts a text sequence to capitalized words.
- `public function keyvalues(sequence source, object pair_delim = ";,", object kv_delim = ":=", object quotes = "\"'` ", object whitespace = " \t\n\r", integer haskeys = 1)`: Converts a string containing Key/Value pairs into a set of sequences, one per K/V pair.
- `public function escape(sequence s, sequence what="\"")` : Escapes special characters in a string.
- `public function quote( sequence text_in, object quote_pair = {"\"", "\""}, integer esc = -1, t_text sp = "" )` : Returns a quoted version of the first argument.
- `public function dequote(sequence text_in, object quote_pairs = {{"\"", "\""}}, integer esc = -1)` : Removes 'quotation' text from the argument.
- `public function format(sequence format_pattern, object arg_list = {})` : Formats string using pattern templates (alias for formatting output).
- `public function wrap(sequence content, integer width = 78, sequence wrap_with = "\n", sequence wrap_at = " \t")` : Wraps text to a column width.

`std/utf8.e` (Namespace: `utf8`)

- **Purpose:** Native Java 25 UTF-8 Unicode engine: codepoint validation, Unicode string length, codepoint slicing, and encoding conversions.
- **Public Functions & Procedures (16):**
- `public function valid(sequence s)` : Return `1` if `s` is a valid UTF-8 byte sequence, `0` otherwise.
- `public function len(sequence s)` : Return the number of Unicode characters (codepoints) in UTF-8 string `s`.
- `public function char_at(sequence s, integer idx)` : Return the `idx`-th Unicode character (1-based) as a UTF-8 string (e.g. `"ö"`).
- `public function slice(sequence s, integer start_cp, integer end_cp)` : Extract a UTF-8 substring from character position `start_cp` to `end_cp` (1-based, inclusive). Never splits multi-byte UTF-8 characters.
- `public function to_codepoints(sequence s)` : Decode a UTF-8 byte string into a sequence of Unicode codepoint integers (UTF-32), e.g. `"Größe"` -> `{71, 114, 246, 223, 101}`.
- `public function from_codepoints(object cp_seq)` : Encode a sequence of Unicode codepoint integers (or a single codepoint atom) into a UTF-8 string.
- `public function upper(sequence s)` : Convert a UTF-8 string to uppercase (supporting Unicode characters like `ä -> Ä`, `ö -> Ö`, `ü -> Ü`).
- `public function lower(sequence s)` : Convert a UTF-8 string to lowercase (supporting Unicode characters like `Ä -> ä`, `Ö -> ö`, `Ü -> ü`).
- `public function find(sequence needle, sequence haystack, integer start_cp = 1)` : Find the 1-based Unicode character (codepoint) position of `needle` inside `haystack`, starting at character `start_cp` (default `1`). Returns `0` if not found.
- `public function byte_length(sequence s)` : Return the number of UTF-8 bytes in `s` (alias for `length(s)`).
- `public function strip_bom(sequence s)` : Remove a leading UTF-8 BOM (`{239, 187, 191}`) from `s` if present.
- `public function getc_utf8(integer fn)` : Read a single complete UTF-8 character (1 to 4 bytes) from open file handle `fn`. Returns the UTF-8 character as a string (e.g. `"€"` or `"ä"`), or `-1` at EOF.
- `public function getc_codepoint(integer fn)` : Read a single complete Unicode codepoint integer (e.g. `8364` for `'€'`) from open file handle `fn`, or `-1` at EOF.
- `public function read_file(sequence path, integer remove_bom = 1)` : Read an entire UTF-8 file into a string. Automatically removes a leading UTF-8 BOM when `remove_bom = 1` (the default). Returns `-1` if the file cannot be opened.
- `public function read_lines(sequence path, integer remove_bom = 1)` : Read all lines from a UTF-8 file as a sequence of UTF-8 strings (without trailing `r` or `n`). Automatically removes a leading UTF-8 BOM on the first line when `remove_bom = 1` (default). Returns `-1` if the file cannot be opened.
- `public function write_file(sequence path, sequence content, integer add_bom = 0)` : Write a UTF-8 string `content` to `path`. Optionally prepends a UTF-8 BOM when `add_bom = 1`. Returns `1` on success, `0` on failure.

`std/regex.e` (Namespace: `regex`)

- **Purpose:** Regular expression engine (Perl and Java compatible): pattern compilation, matching, search, replace, and substring tokenization.
- **Public Functions & Procedures (17):**
- `public function option_spec_to_string(option_spec o)` : Converts an option spec to a string. This can be useful for debugging what options were passed in. Without it you have to convert a number to hex and lookup the constants in the source code.
- `public function error_to_string(integer i)` : Converts an regex error to a string. This can be useful for debugging and even something rough to give to the user incase of a regex failure. It is preferable to a number.
- `public function new(string pattern, option_spec options=DEFAULT)` : Returns an allocated regular expression.
- `public function error_message(object re)` : Returns a text based error message.
- `public function escape(string s)` : Escapes special regular expression characters that may be entered into a search string from user input.
- `public function get_ovector_size(regex ex, integer maxsize=0)` : Returns the number of capturing subpatterns (the ovector size) for a regex.
- `public function find(regex re, string haystack, integer from=1, option_spec options=DEFAULT, integer size = get_ovector_size(re, 30))` : Returns the first match of `re` in `haystack`. You can optionally start at the position `from`.
- `public function find_all(regex re, string haystack, integer from=1, option_spec options=DEFAULT, integer size = get_ovector_size(re, 30))` : Returns all matches of `re` in `haystack` optionally starting at the sequence position `from`.
- `public function has_match(regex re, string haystack, integer from=1, option_spec options=DEFAULT)` : Determines if `re` matches any portion of `haystack`.
- `public function is_match(regex re, string haystack, integer from=1, option_spec options=DEFAULT)` : Determines if the entire `haystack` matches `re`.
- `public function matches(regex re, string haystack, integer from=1, option_spec options=DEFAULT)` : Gets the matched text only.
- `public function all_matches(regex re, string haystack, integer from=1, option_spec options=DEFAULT)` : Gets the text of all matches.
- `public function split(regex re, string text, integer from=1, option_spec options=DEFAULT)` : Splits a string based on a regex as a delimiter.
- `public function split_limit(regex re, string text, integer limit=0, integer from=1, option_spec options=DEFAULT)` : Splits string by regex pattern up to a maximum number of parts.

- `public function find_replace(regex ex, string text, sequence replacement, integer from=1, option_spec options=DEFAULT)` : Replaces all matches of a regex with the replacement text.
- `public function find_replace_limit(regex ex, string text, sequence replacement, integer limit, integer from=1, option_spec options=DEFAULT)` : Replaces up to `limit` matches of `ex` in `text` except when `limit` is 0. When `limit` is 0, this routine replaces all of the matches.
- `public function find_replace_callback(regex ex, string text, integer rid, integer limit=0, integer from=1, option_spec options=DEFAULT)` : Finds and then replaces text that is processed by a call back function.

`std/wildcard.e` (Namespace: `wildcard`)

- **Purpose:** Simple wildcard pattern matching (`*` and `?`) for filenames, paths, and search filters.
- **Public Functions & Procedures (1):**
- `public function is_match(sequence pattern, sequence string)` : Determines whether a string matches a pattern. The pattern may contain `*` and `?` wildcards.

`std/base64.e` (Namespace: `base64`)

- **Purpose:** RFC 4648 compliant Base64 encoding and decoding for binary data, text strings, and attachments with optional line wrapping.
- **Public Functions & Procedures (2):**
- `public function encode(sequence in, integer wrap_column = 0)` : Encodes to base64.
- `public function decode(sequence in)` : Decodes from base64.

`std/convert.e` (Namespace: `convert`)

- **Purpose:** Low-level data conversion: integers to bytes/bits, IEEE 754 float representation (32/64/80 bit), hexadecimal strings, and numeric formats.
- **Public Functions & Procedures (15):**
- `public function int_to_bytes(atom x, integer size = 4 )` : Converts an atom that represents an integer to a sequence of 4 bytes.
- `public function bytes_to_int(sequence s)` : Converts a sequence of at most 4 bytes into an atom.
- `public function int_to_bits(atom x, integer nbits = 32)` : Extracts the lower bits from an integer.
- `public function bits_to_int(sequence bits)` : Converts a sequence of bits to an atom that has no fractional part.
- `public function atom_to_float64(atom a)` : Converts an atom to a sequence of 8 bytes in IEEE 64-bit format.
- `public function atom_to_float80(atom a)` : **.
- `public function float80_to_atom( sequence bytes )` : **.
- `public function atom_to_float32(atom a)` : Converts an atom to a sequence of 4 bytes in IEEE 32-bit format.
- `public function float64_to_atom(sequence_8 ieee64)` : Converts a sequence of 8 bytes in IEEE 64-bit format to an atom.
- `public function float32_to_atom(sequence_4 ieee32)` : Converts a sequence of 4 bytes in IEEE 32-bit format to an atom.
- `public function hex_text(sequence text)` : Converts a text representation of a hexadecimal number to an atom.
- `public function set_decimal_mark(integer new_mark)` : Gets, and possibly sets, the decimal mark that `to_number` uses.
- `public function to_number( sequence text_in, integer return_bad_pos = 0)` : Converts the text into a number.
- `public function to_integer(object data_in, integer def_value = 0)` : Converts an object into a integer.
- `public function to_string(object data_in, integer string_quote = 0, integer embed_string_quote = '"')` : Converts an object into a text string.

`std/locale.e` (Namespace: `locale`)

- **Purpose:** Internationalization (i18n) and localization: loading language dictionaries, managing active locales, and localized message retrieval.
- **Public Functions & Procedures (13):**
- `public procedure set_lang_path(object pp)` : Sets the language path.
- `public function get_lang_path()` : Gets the language path.
- `public function lang_load(sequence filename)` : Loads a language file.
- `public procedure set_def_lang( object langmap )` : Sets the default language (translation) map.
- `public function get_def_lang( )` : Gets the default language (translation) map.
- `public function translate(sequence word, object langmap = 0, object defval="", integer mode = 0)` : Translates a word, using the current language file.
- `public function trsprintf(sequence fmt, sequence data, object langmap = 0)` : Returns a formatted string with automatic translation performed on the parameters.
- `public function set(sequence new_locale)` : Sets the computer locale, and possibly loads an appropriate translation file.
- `public function get()` : Gets the current locale string.
- `public function money(object amount)` : Converts an amount of currency into a string representing that amount.
- `public function number(object num)` : Converts a number into a string representing that number.
- `public function datetime(sequence fmt, datetime:datetime dtm)` : Formats a date according to current locale.

- `public function get_text(integer MsgNum, sequence LocalQuals = {}, sequence DBBase = "teksto")` : Gets the text associated with the message number in the requested locale.

`std/localeconv.e` (Namespace: `localconv`)

- **Purpose:** Canonical locale naming conventions, ISO language/territory codes, and cross-conversion between Unix and Windows locale formats.
- **Public Functions & Procedures (3):**
- `public function canonical(sequence new_locale)` : Get canonical name for a locale.
- `public function decanonical(sequence new_locale)` : Gets the translation of a locale string for current platform.
- `public function canon2win(sequence new_locale)` : Gets the translation of a canoncial locale string for the *Windows* platform.

`std/lcid.e` (Namespace: `lcid`)

- **Purpose:** Windows Locale Identifier (LCID) mapping tables and lookups for platform-specific regional settings.
- **Public Functions & Procedures (1):**
- `public function get_lcid(sequence name)` : -- Returns: -- # The `lcid` which is of type LCID in C/C++. --.

`std/scinot.e` (Namespace: `scinot`)

- **Purpose:** Scientific exponential notation: parsing and converting exponential strings (`1.23e-04`) into exact atoms and formatted representations.
- **Public Functions & Procedures (4):**
- `public function bits_to_bytes( sequence bits )` : Takes a sequence of bits (all elements either 0 or 1) and converts it into a sequence of bytes.
- `public function bytes_to_bits( sequence bytes )` : Converts a sequence of bytes (all elements integers between 0 and 255) and converts it into a sequence of bits.
- `public function scientific_to_float( sequence s, floating_point fp = NATIVE )` : Takes a string reprepresentation of a number in scientific notation and the requested precision (DOUBLE or EXTENDED) and returns a sequence of bytes in the raw format of an IEEE 754 double or extended precision floating point number. This value can be passed to the euphoria library function, `` `float64_to_atom or float80_to_atom` ``, respectively.
- `public function scientific_to_atom( sequence s, floating_point fp = NATIVE )` : Takes a string reprepresentation of a number in scientific notation and returns an atom.

2.8.4 Mathematics, Statistics & Random Numbers

`std/math.e` (Namespace: `math`)

- **Purpose:** Comprehensive mathematical routines: trigonometry, hyperbolic functions, logarithms, roots, rounding, modulo, GCD, and LCM.
- **Public Functions & Procedures (38):**
- `public function abs(object a)` : Returns the absolute value of numbers.
- `public function sign(object a)` : Returns -1, 0 or 1 for each element according to it being negative, zero or positive.
- `public function larger_of(object objA, object objB)` : Returns the larger of two objects.
- `public function smaller_of(object objA, object objB)` : Returns the smaller of two objects.
- `public function max(object a)` : Computes the maximum value among all the argument's elements.
- `public function min(object a)` : Computes the minimum value among all the argument's elements.
- `public function ensure_in_range(object item, sequence range_limits)` : Ensures that the `item` is in a range of values supplied by inclusive `range_limits` .
- `public function ensure_in_list(object item, sequence list, integer default=1)` : Ensures that the `item` is in a list of values supplied by `list` .
- `public function mod(object x, object y)` : Computes the remainder of the division of two objects using floored division.
- `public function trunc(object x)` : Returns the integer portion of a number.
- `public function frac(object x)` : Returns the fractional portion of a number.
- `public function intdiv(object a, object b)` : Returns an integral division of two objects.
- `public function ceil(object a)` : Computes the next integer equal or greater than the argument.
- `public function round(object a, object precision=1)` : Returns the argument's elements rounded to some precision.
- `public function arccos(trig_range x)` : Returns an angle given its cosine.
- `public function arcsin(trig_range x)` : Returns an angle given its sine.
- `public function atan2(atom y, atom x)` : Calculate the arctangent of a ratio.
- `public function rad2deg (object x)` : Converts an angle measured in radians to an angle measured in degrees.
- `public function deg2rad (object x)` : Converts an angle measured in degrees to an angle measured in radians.
- `public function log10(object x1)` : Returns the base 10 logarithm of a number.
- `public function exp(atom x)` : Computes some power of E.

- `public function fib(integer i)` : Computes the nth Fibonacci Number.
- `public function cosh(object a)` : Computes the hyperbolic cosine of an object.
- `public function sinh(object a)` : Computes the hyperbolic sine of an object.
- `public function tanh(object a)` : Computes the hyperbolic tangent of an object.
- `public function arcsinh(object a)` : Computes the reverse hyperbolic sine of an object.
- `public function arccosh(not_below_1 a)` : Computes the reverse hyperbolic cosine of an object.
- `public function arctanh(abs_below_1 a)` : Computes the reverse hyperbolic tangent of an object.
- `public function sum(object a)` : Computes the sum of all atoms in the argument, no matter how deeply nested.
- `public function product(object a)` : Computes the product of all the atom in the argument, no matter how deeply nested.
- `public function or_all (object a)` : Or's together all atoms in the argument, no matter how deeply nested.
- `public function shift_bits(object source_number, integer shift_distance)` : Moves the bits in the input value by the specified distance.
- `public function rotate_bits(object source_number, integer shift_distance)` : Rotates the bits in the input value by the specified distance.
- `public function gcd(atom p, atom q)` : Returns the greater common divisor of two atoms.
- `public function approx(object p, object q, atom epsilon = 0.005)` : Compares two (sets of) numbers based on approximate equality.
- `public function powof2(object p)` : Tests for power of 2.
- `public function is_even(integer test_integer)` : Tests if the supplied integer is a even or odd number.
- `public function is_even_obj(object test_object)` : Tests if the supplied Euphoria object is even or odd.

`std/mathcons.e` (Namespace: `mathcons`)

- **Purpose:** Mathematical natural constants and limits for scientific calculations.
- **Notes & Key Constants:** Defines standard constants: `PI`, `HALFPI`, `QUARTPI`, `TWOPI`, `E`, `SQRT2`, `SQRT3`, `LN2`, `LN10`, `DEG_TO_RAD`, `RAD_TO_DEG`. Contains typed constants rather than routines.

`std/stats.e` (Namespace: `stats`)

- **Purpose:** Statistical analysis routines: mean, median, mode, variance, standard deviation, min/max metrics, and percentiles.
- **Public Functions & Procedures (20):**
- `public function small(sequence data_set, integer ordinal_idx)` : Determines the k-th smallest value from the supplied set of numbers.
- `public function largest(object data_set)` : Returns the largest of the data points that are atoms.
- `public function smallest(object data_set)` : Returns the smallest of the data points.
- `public function range(object data_set)` : Determines a number of *range* statistics for the data set.
- `public function stdev(sequence data_set, object subseq_opt = ST_ALLNUM, integer population_type = ST_SAMPLE)` : Returns the standard deviation based on the population.
- `public function avedev(sequence data_set, object subseq_opt = ST_ALLNUM, integer population_type = ST_SAMPLE)` : Returns the average of the absolute deviations of data points from their mean.
- `public function sum(object data_set, object subseq_opt = ST_ALLNUM)` : Returns the sum of all the atoms in an object.
- `public function count(object data_set, object subseq_opt = ST_ALLNUM)` : Returns the count of all the atoms in an object.
- `public function average(object data_set, object subseq_opt = ST_ALLNUM)` : Returns the average (mean) of the data points.
- `public function geomean(object data_set, object subseq_opt = ST_ALLNUM)` : Returns the geometric mean of the atoms in a sequence.
- `public function harmean(sequence data_set, object subseq_opt = ST_ALLNUM)` : Returns the harmonic mean of the atoms in a sequence.
- `public function movavg(object data_set, object period_delta)` : Returns the average (mean) of the data points for overlaping periods. This can be either a simple or weighted moving average.
- `public function emovavg(object data_set, atom smoothing_factor)` : Returns the exponential moving average of a set of data points.
- `public function median(object data_set, object subseq_opt = ST_ALLNUM)` : Returns the mid point of the data points.
- `public function raw_frequency(object data_set, object subseq_opt = ST_ALLNUM)` : Returns the frequency of each unique item in the data set.
- `public function mode(sequence data_set, object subseq_opt = ST_ALLNUM)` : Returns the most frequent point(s) of the data set.
- `public function central_moment(sequence data_set, object datum, integer order_mag = 1, object subseq_opt = ST_ALLNUM)` : Returns the distance between a supplied value and the mean, to some supplied order of magnitude. This is used to get a measure of the *shape* of a data set.
- `public function sum_central_moments(object data_set, integer order_mag = 1, object subseq_opt = ST_ALLNUM)` : Returns sum of the central moments of each item in a data set.
- `public function skewness(object data_set, object subseq_opt = ST_ALLNUM)` : Returns a measure of the asymmetry of a data set. Usually the data_set is a probablity distribution but it can be anything. This value is used to assess how suitable the data set is in representing the required analysis. It can help detect if there are too many extreme values in the data set.
- `public function kurtosis(object data_set, object subseq_opt = ST_ALLNUM)` : Returns a measure of the spread of values in a dataset when compared to a *normal* probability curve.

`std/rand.e` (Namespace: `random`)

- **Purpose:** Random number generators: pseudo-random numbers, cryptographically secure streams, range bounded randoms, and sequence shuffling.
- **Public Functions & Procedures (8):**
- `public function rand_range(atom lo, atom hi)` : Returns a random integer from a specified inclusive integer range.
- `public function rnd()` : Returns a random floating point number in the range 0 to 1.
- `public function rnd_1()` : Returns a random floating point number in the range 0 to less than 1.
- `public procedure set_rand(object seed)` : Resets the random number generator.
- `public function get_rand()` : Retrieves the current values of the random generator's seeds.
- `public function chance(atom my_limit, atom top_limit = 100)` : Simulates the probability of a desired outcome.
- `public function roll(object desired, integer sides = 6)` : Simulates the probability of a dice throw.
- `public function sample(sequence population, integer sample_size, integer sampling_method = 0)` : Selects a set of random samples from a population set.

`std/fenv.e` (Namespace: `fenv`)

- **Purpose:** Floating-point environment management: querying, raising, and clearing arithmetic exception flags (divide-by-zero, overflow).
- **Public Functions & Procedures (3):**
- `public function clear(object e_p)` : Clears a single exception or a sequence of exceptions.
- `public function raise(object e_p)` : Manually raises floating-point exception flags (FE_DIVBYZERO, FE_OVERFLOW, etc.).
- `public function test(object e_p)` : Tests whether specific floating-point exception flags have occurred.

2.8.5 Date, Time & Calendar

`std/datetime.e` (Namespace: `datetime`)

- **Purpose:** Comprehensive date and time handling: current timestamps (`now`), ISO 8601 formatting, date arithmetic (`add` , `subtract`), differences, and parsing.
- **Public Functions & Procedures (17):**
- `public function from_date(sequence src)` : Converts a sequence formatted according to the built-in `date` function to a valid datetime sequence.
- `public function now()` : Creates a new datetime value initialized with the current date and time.
- `public function now_gmt()` : Create a new datetime value that falls into the Greenwich Mean Time (GMT) timezone.
- `public function new(integer year=0, integer month=0, integer day=0, integer hour=0, integer minute=0, atom second=0)` : Creates a new datetime value.
- `public function new_time(integer hour, integer minute, atom second)` : Creates a new datetime value with a date of zeros.
- `public function weeks_day(datetime dt)` : Gets the day of week of the datetime `dt` .
- `public function years_day(datetime dt)` : Gets the Julian day of year of the supplied date.
- `public function is_leap_year(datetime dt)` : Determines if `dt` falls within leap year.
- `public function days_in_month(datetime dt)` : Returns the number of days in the month of `dt` .
- `public function days_in_year(datetime dt)` : Returns the number of days in the year of `dt` .
- `public function to_unix(datetime dt)` : Converts a datetime value to the *Unix* numeric format (seconds since `EPOCH_1970`).
- `public function from_unix(atom unix)` : Creates a datetime value from the *Unix* numeric format (seconds since EPOCH).
- `public function format(datetime d, sequence pattern = "%Y-%m-%d %H:%M:%S")` : Formats the date according to the format pattern string.
- `public function parse(sequence val, sequence fmt="%Y-%m-%d %H:%M:%S", integer yylower = -80)` : Parses a datetime string according to the given format.
- `public function add(datetime dt, object qty, integer interval)` : Adds a number of *intervals* to a datetime.
- `public function subtract(datetime dt, atom qty, integer interval)` : Subtracts a number of *intervals* to a base datetime.
- `public function diff(datetime dt1, datetime dt2)` : Computes the difference, in seconds, between two dates.

2.8.6 Networking, Internet, DNS & URLs

`std/socket.e` (Namespace: `sockets`)

- **Purpose:** Dual-stack IPv4/IPv6 socket API: creating TCP/UDP server and client sockets, non-blocking I/O, event multiplexing (`select`), and timeout controls.
- **Public Functions & Procedures (18):**
- `public function error_code()` : Gets the error code.
- `public function create(integer family, integer sock_type, integer protocol)` : Creates a new socket.

- `public function close(socket sock)` : Closes a socket.
- `public function shutdown(socket sock, atom method=SD_BOTH)` : Partially or fully close a socket.
- `public function select(object sockets_read, object sockets_write, object sockets_err, integer timeout=0, integer timeout_micro=0)` : Determines the read, write and error status of one or more sockets.
- `public function send(socket sock, sequence data, atom flags=0)` : Sends TCP data to a socket connected remotely.
- `public function receive(socket sock, atom flags=0)` : Receives data from a bound socket.
- `public function get_option(socket sock, integer level, integer optname)` : Gets options for a socket.
- `public function set_option(socket sock, integer level, integer optname, object val)` : Sets options for a socket.
- `public function connect(socket sock, sequence address, integer port=-1)` : Establishes an outgoing connection to a remote computer. Only works with TCP sockets.
- `public function bind(socket sock, sequence address, integer port=-1)` : Joins a socket to a specific local internet address and port so later calls only need to provide the socket.
- `public function listen(socket sock, integer backlog)` : Starts monitoring a connection. Only works with TCP sockets.
- `public function accept(socket sock)` : Produces a new socket for an incoming connection.
- `public function send_to(socket sock, sequence data, sequence address, integer port=-1, atom flags=0)` : Sends a UDP packet to a given socket.
- `public function receive_from(socket sock, atom flags=0)` : Receives a UDP packet from a given socket.
- `public function service_by_name(sequence name, object protocol=0)` : Gets service information by name.
- `public function service_by_port(integer port, object protocol=0)` : Gets service information by port number.
- `public function info(integer Type)` : Gets constant definitions from the backend.

`std/http.e` (Namespace: `http`)

- **Purpose:** Modern HTTP/HTTPS client and multi-threaded server powered by Java Virtual Threads: REST methods (GET, POST, PUT, DELETE), TLS/SSL, headers, and routing.
- **Public Functions & Procedures (20):**
- `public function request(sequence method, sequence url, sequence body = "", sequence headers = {}, integer timeout_ms = 10000, integer verify_ssl = 1)` : Execute a general HTTP/HTTPS request via Java HttpClient. Returns `{status_code, body_string, headers_list}` .
- `public function get(sequence url, sequence headers = {}, integer verify_ssl = 1)` : Perform an HTTP/HTTPS GET request.
- `public function post(sequence url, sequence body, sequence headers = {}, integer verify_ssl = 1)` : Perform an HTTP/HTTPS POST request.
- `public function post_json(sequence url, sequence json_body, sequence headers = {}, integer verify_ssl = 1)` : Perform an HTTP/HTTPS POST request with `Content-Type: application/json` .
- `public function put(sequence url, sequence body, sequence headers = {}, integer verify_ssl = 1)` : Perform an HTTP/HTTPS PUT request.
- `public function delete_req(sequence url, sequence headers = {}, integer verify_ssl = 1)` : Perform an HTTP/HTTPS DELETE request.
- `public function header_get(sequence headers, sequence name, sequence default_val = "")` : Case-insensitive lookup of a header value in a `{ {key, val}, ... }` header sequence.
- `public function server_create(sequence host = "127.0.0.1", integer port = 8080)` : Create a plain HTTP server listening on `host:port` . Returns `server_id` (> 0) on success, or `< 0` on error.
- `public function server_create_tls(sequence host, integer port, sequence cert_pem, sequence key_pem)` : Create an HTTPS (TLS) server listening on `host:port` using a PEM certificate and private key. Returns `server_id` (> 0) on success, or `< 0` on error.
- `public procedure route(integer srv_id, sequence method, sequence path, integer handler_rid)` : Register a route handler function `function handler(sequence req)` for `(srv_id, method, path)` . Setting `method = ""` *matches any HTTP method. Setting* `path = " "` or `path = "/prefix/*"` supports wildcard and prefix routing.
- `public procedure clear_routes(integer srv_id = 0)` : Remove all registered routes for `srv_id` (or all servers if `srv_id = 0`).
- `public function response(integer status, sequence body, sequence content_type = "text/plain; charset=utf-8", sequence extra_headers = {})` : Construct a server response tuple `{status, body, content_type, extra_headers}` .
- `public function response_json(integer status, sequence json_body, sequence extra_headers = {})` : Construct a JSON server response (`application/json; charset=utf-8`).
- `public function response_html(integer status, sequence html_body, sequence extra_headers = {})` : Construct an HTML server response (`text/html; charset=utf-8`).
- `public function accept_request(integer srv_id, integer timeout_ms = 5000)` : Wait up to `timeout_ms` for an incoming HTTP/HTTPS request on `srv_id` . Returns `0` on timeout, `< 0` on error, or `{conn_id, method, path, query, body, headers, client_ip}` .
- `public function reply(integer conn_id, sequence resp)` : Send a response tuple `{status, body, content_type, extra_headers}` to `conn_id` and close the connection.
- `public function handle_one(integer srv_id, integer timeout_ms = 5000)` : Accept and dispatch a single HTTP/HTTPS request using registered routes (`http:route`). Returns `1` if a request was processed, `0` on timeout, or `< 0` on error.

- `public procedure serve_forever(integer srv_id)` : Blocking server loop that continuously accepts and dispatches requests on `srv_id`.
- `public function server_background(integer srv_id, integer max_requests = 0)` : Start the HTTP/HTTPS server `srv_id` in a non-blocking background worker process. If `max_requests > 0`, the worker exits automatically after serving `max_requests`. Returns the worker PID (`> 0`) to the caller.
- `public function server_close(integer srv_id, integer bg_pid = 0)` : Close an HTTP/HTTPS server socket and free its TLS context (if any). If `bg_pid > 0`, also terminates and reaps the background server worker process.

`std/net/dns.e` (Namespace: `dns`)

- **Purpose:** Domain Name System (DNS) resolution: forward and reverse lookups of hostnames and IP addresses (IPv4 and IPv6).
- **Public Functions & Procedures (2):**
- `public function host_by_name(sequence name)` : Get the host information by name.
- `public function host_by_addr(sequence address)` : Get the host information by address.

`std/net/url.e` (Namespace: `url`)

- **Purpose:** URL manipulation according to RFC 3986: parsing into scheme, host, port, path, query parameters, and URL percent encoding/decoding.
- **Public Functions & Procedures (4):**
- `public function parse_querystring(object query_string)` : Parse a query string into a map.
- `public function parse(sequence url, integer querystring_also=0)` : Parse a URL returning its various elements.
- `public function encode(sequence what, sequence spacecode="+")` : Converts all non-alphanumeric characters in a string to their percent-sign hexadecimal representation, or plus sign for spaces.
- `public function decode(sequence what)` : Convert all encoded entities to their decoded counter parts.

`std/net/common.e` (Namespace: `common`)

- **Purpose:** Common internet utilities: IP address validation, socket helper routines, and address string formatting.
- **Public Functions & Procedures (3):**
- `public function is_inetaddr(sequence address)` : Checks if x is an IP address in the form (#.#.#.#[:#]).
- `public function parse_ip_address(sequence address, integer port = -1)` : Converts a text "address:port" into {"address", port} format.
- `public function parse_url(sequence url)` : Parse a common URL. Currently supported URLs are http(s), ftp(s), gopher(s) and mailto.

2.8.7 Data Formats, XML, JSON & Databases

`std/json.e` (Namespace: `json`)

- **Purpose:** High-performance JSON parser, builder, serializer, and RFC 9535 JSONPath engine powered by Jackson and Java 25.
- **Public Functions & Procedures (22):**
- `public function null()` : Creates a typed JSON null node (`JSON_NULL`).
- `public function bool(integer b)` : Creates a typed JSON boolean node (`JSON_TRUE` or `JSON_FALSE`).
- `public function str(sequence s)` : Creates an explicit typed JSON string node.
- `public function array(sequence items = {})` : Creates a typed JSON array from a sequence of elements.
- `public function object_new(sequence pairs = {})` : Creates an empty or pre-populated JSON key-value object.
- `public function type_of(object v)` : Determines the JSON node type (`JSON_OBJECT` , `JSON_ARRAY` , etc.).
- `public function is_null(object v)` : Checks whether the object represents a JSON null value.
- `public function is_bool(object v)` : Checks whether the object is a JSON boolean value.
- `public function is_object(object v)` : Checks whether the object is a JSON key-value object.
- `public function is_array(object v)` : Checks whether the object is a JSON array.
- `public function unwrap(object v)` : Recursively unrolls internal JSON nodes into native Euphoria sequences and atoms.
- `public function parse(sequence json_str)` : Parse a JSON string and return `{ok, doc_or_error}` .
- `public function decode(sequence json_str)` : Parse a JSON string and return the parsed document directly (or `{0, err}` on error).
- `public function stringify(object val, integer indent = 0)` : Serialize a JSON document or Euphoria value to a JSON string. Set `indent > 0` (e.g. `2`) for pretty-printed output.
- `public function encode(object val, integer indent = 0)` : Serializes a Euphoria data structure into a valid JSON string with optional indentation.
- `public function get_node(object doc, object path, object default_val = 0)` : Returns the raw internal JSON node at a path.
- `public function get(object doc, object path, object default_val = 0)` : Navigate a JSON document using a key, an integer index, a path list (`{1, "name"}`), or a dot/bracket string path (e.g. `"user.address.city"` or `"items[1].price"`), automatically unwrapping primitive JSON strings, booleans, and numbers.
- `public function has(object doc, sequence path)` : Checks whether a key exists in a JSON object.

- `public function set(sequence obj_doc, sequence key, object val)` : Set or update a property `key` on a JSON object and return the updated object.
- `public function keys(object obj_doc)` : Returns all keys of a JSON object as a sequence.
- `public function size(object doc)` : Returns the number of elements in a JSON object or array.
- `public function jsonpath(object doc_or_str, sequence expr)` : Executes a JSONPath query on the JSON document (alias for `json_path`).

`std/json_schema.e` (Namespace: `json_schema`)

- **Purpose:** JSON document validation against JSON Schema Draft 2020-12 specifications with precise violation reporting.
- **Public Functions & Procedures (3):**
- `public function validate(object instance_or_str, object schema_or_str)` : Validate `instance` (parsed JSON node or JSON string) against `schema` (parsed JSON node or JSON string). Returns `{ is_valid (1 or 0), { {json_pointer_path, message}, ... } }`.
- `public function is_valid(object instance_or_str, object schema_or_str)` : Return `1` if `instance_or_str` conforms to `schema_or_str`, or `0` otherwise.
- `public function format_errors(sequence validation_res)` : Format all validation errors from `validate()` into human-readable `"<path>: <message>"` strings.

`std/xml.e` (Namespace: `xml`)

- **Purpose:** W3C-compliant XML DOM parser, element builder, W3C XPath 1.0 query evaluator, and XSD schema validator.
- **Public Functions & Procedures (20):**
- `public function parse(sequence xml_str)` : Parses an XML string into a hierarchical W3C XML DOM tree.
- `public function decode(sequence xml_str)` : Decodes XML string into DOM tree (alias for `parse`).
- `public function element(sequence tag_name, sequence attrs = {}, sequence children = {}, sequence text_val = "")` : Programmatically creates a new XML element node with tag, attributes, and children.
- `public function stringify(sequence node, integer indent = 0)` : Serializes DOM node tree into well-formed XML string with indentation.
- `public function to_string(sequence node, integer indent = 0)` : Serializes an XML DOM tree into well-formed XML string (alias for `stringify`).
- `public function is_element(object node)` : Checks if node is an XML element.
- `public function tag(sequence node)` : Retrieves tag name of an XML element node.
- `public function text(sequence node)` : Retrieves aggregated text content of an XML node.
- `public function attrs(sequence node)` : Retrieves attributes of an XML element as a sequence of key-value pairs.
- `public function children(sequence node)` : Retrieves child element nodes of an XML element.
- `public function attr(sequence node, sequence name, sequence default_val = "")` : Retrieves value of a specific attribute from an XML node.
- `public function set_attr(sequence node, sequence name, sequence val)` : Sets or updates an attribute on an XML element node.
- `public function add_child(sequence node, sequence child)` : Appends a child node to an XML element.
- `public function find_all(sequence node, sequence tag_or_path)` : Finds all direct child elements matching tag name.
- `public function find_first(sequence node, sequence tag_or_path, object default_val = 0)` : Finds first child element matching tag name.
- `public function xpath(object node_or_xml, sequence xpath_expr)` : Evaluates a W3C XPath 1.0 expression against the DOM tree.
- `public function validate_xsd(object node_or_xml, sequence xsd_str)` : Validates XML document against an XSD schema file.
- `public function is_valid_xsd(object node_or_xml, sequence xsd_str)` : Validates XML document against an XSD schema (1 = valid, 0 = invalid).
- `public function validate_relaxng(object node_or_xml, sequence rng_str)` : Validates XML document against a RelaxNG schema.
- `public function is_valid_relaxng(object node_or_xml, sequence rng_str)` : Validates XML document against a RelaxNG schema.

`std/sqlite.e` (Namespace: `sqlite`)

- **Purpose:** Embedded SQLite 3 database driver via JDBC: ACID transactions, parameterized prepared statements, query iterators, and DDL.
- **Public Functions & Procedures (15):**
- `public function is_null(object val)` : Return `1` if `val` represents an SQL `NULL` value.
- `public function open(sequence path = ":memory:")` : Open an SQLite database file (or `":memory:"` by default). Returns `db_id (> 0)` on success, or `-1` on error.
- `public function close(integer db_id)` : Close an open SQLite database connection.
- `public function exec(integer db_id, sequence sql, sequence params = {})` : Execute DDL/DML SQL statement(s) (`CREATE TABLE`, `INSERT`, `UPDATE`, `DELETE`) with optional `?` parameter bindings. Returns `1` on success, `0` on error.
- `public function query_raw(integer db_id, sequence sql, sequence params = {})` : Execute a `SELECT` query and return `{ok, column_names, rows}`.
- `public function query(integer db_id, sequence sql, sequence params = {})` : Execute a `SELECT` query and return the list of rows `{ {col1, col2, ...}, ... }`.

- `public function query_one(integer db_id, sequence sql, sequence params = {}, object default_val = {})` : Execute a `SELECT` query and return the first row, or `default_val` if empty.
- `public function query_value(integer db_id, sequence sql, sequence params = {}, object default_val = 0)` : Execute a `SELECT` query and return the single scalar value of the first column of the first row (e.g. `SELECT COUNT(*) FROM users`), or `default_val` if no row is returned.
- `public function query_assoc(integer db_id, sequence sql, sequence params = {})` : Execute a `SELECT` query and return a list of JSON-compatible associative objects (`json:JSON_OBJECT`), allowing access via `json:get(row, "col_name")` or direct serialization via `json:stringify(rows)` .
- `public function last_insert_id(integer db_id)` : Return the `ROWID` of the most recent successful `INSERT` on `db_id` .
- `public function changes(integer db_id)` : Return the number of rows modified, inserted, or deleted by the most recent statement.
- `public function errmsg(integer db_id = 0)` : Return the last SQLite error message string.
- `public function begin(integer db_id)` : Transaction helpers: `begin(db)` , `commit(db)` , `rollback(db)` .
- `public function commit(integer db_id)` : Commits the active transaction to the SQLite database file.
- `public function rollback(integer db_id)` : Rolls back uncommitted changes in the active SQLite transaction.

`std/pgsql.e` (Namespace: `pgsql`)

- **Purpose:** Enterprise PostgreSQL database client via JDBC: connection pooling, prepared statements, transactions, and native data conversions.
- **Public Functions & Procedures (19):**
- `public function is_available()` : Return `1` if PostgreSQL support is available in the runtime, `0` otherwise.
- `public function is_null(object val)` : Return `1` if `val` represents an SQL `NULL` value.
- `public function open(sequence conninfo = "")` : Connect to a PostgreSQL server using a connection string or URI (e.g. `"host=127.0.0.1 port=5432 dbname=mydb user=postgres password=secret"` or `"postgresql://user:pass@127.0.0.1:5432/mydb"`). Returns `db_id (> 0)` on success, or `0` on failure (check `pgsql:errmsg()`).
- `public function connect(sequence conninfo = "")` : Alias for `pgsql:open(conninfo)` .
- `public function close(integer db_id)` : Close an open PostgreSQL connection.
- `public function exec(integer db_id, sequence sql, sequence params = {})` : Execute DDL/DML SQL statement(s) (`CREATE TABLE` , `INSERT` , `UPDATE` , `DELETE`) with optional `$1, $2` or `?` parameter bindings. Returns `1` on success, `0` on error.
- `public function query_raw(integer db_id, sequence sql, sequence params = {})` : Execute a `SELECT` (or `INSERT ... RETURNING`) query and return `{ok, column_names, rows}` .
- `public function query(integer db_id, sequence sql, sequence params = {})` : Execute a `SELECT` (or `RETURNING`) query and return the list of rows `{ {col1, col2, ...}, ... }` .
- `public function query_one(integer db_id, sequence sql, sequence params = {}, object default_val = {})` : Execute a `SELECT` query and return the first row, or `default_val` if empty.
- `public function query_value(integer db_id, sequence sql, sequence params = {}, object default_val = 0)` : Execute a `SELECT` query and return the single scalar value of the first column of the first row (e.g. `SELECT COUNT(*) FROM users` or `INSERT ... RETURNING id`), or `default_val` if no row is returned.
- `public function query_assoc(integer db_id, sequence sql, sequence params = {})` : Execute a `SELECT` query and return a list of JSON-compatible associative objects (`json:JSON_OBJECT`), allowing access via `json:get(row, "col_name")` or direct serialization via `json:stringify(json:array(rows))` .
- `public function changes(integer db_id)` : Return the number of rows affected (`INSERT` , `UPDATE` , `DELETE`) by the most recent statement.
- `public function server_version(integer db_id)` : Return the PostgreSQL server version integer (e.g. `160004` for PostgreSQL 16.4).
- `public function errmsg(integer db_id = 0)` : Return the last PostgreSQL error message string.
- `public function escape_literal(integer db_id, sequence val)` : Safely escape a string literal (`'...'`) using `PQescapeLiteral` .
- `public function escape_identifier(integer db_id, sequence ident)` : Safely escape an SQL identifier (`"..."` , e.g. table or column name) using `PQescapeIdentifier` .
- `public function begin(integer db_id)` : Transaction helpers: `begin(db)` , `commit(db)` , `rollback(db)` .
- `public function commit(integer db_id)` : Commits the current transaction on the PostgreSQL database connection.
- `public function rollback(integer db_id)` : Rolls back the current transaction on the PostgreSQL database connection.

`std/eds.e` (Namespace: `eds`)

- **Purpose:** Euphoria Database System (EDS): Lightweight, self-contained hierarchical NoSQL key-value database engine with tables and records.
- **Public Functions & Procedures (30):**
- `public function db_get_errors(integer clearing = 1)` : Fetches the most recent set of errors recorded by the library.
- `public procedure db_dump(object file_id, integer low_level_too = 0)` : Prints the current database in readable form to file `fn` .
- `public procedure check_free_list()` : Detects corruption of the free list in a Euphoria database.
- `public function db_connect(sequence dbalias, sequence path="", sequence dboptions = {})` : Defines a symbolic name for a database and its default attributes.
- `public function db_create(sequence path, integer lock_method = DB_LOCK_NO, integer init_tables = DEF_INIT_TABLES, integer init_free = DEF_INIT_FREE )` : Creates a new database given a file path and a lock method.

- `public function db_open(sequence path, integer lock_method = DB_LOCK_NO)` : Opens an existing Euphoria database.
- `public function db_select(sequence path, integer lock_method = -1)` : Chooses a new, already open, database to be the current database.
- `public procedure db_close()` : Unlocks and closes the current database.
- `public function db_select_table(sequence name)` : On success, the table with name given by name becomes the current table.
- `public function db_current_table()` : Gets the name of currently selected table.
- `public function db_create_table(sequence name, integer init_records = DEF_INIT_RECORDS)` : Creates a new table within the current database.
- `public procedure db_delete_table(sequence name)` : Deletes a table in the current database.
- `public procedure db_clear_table(sequence name, integer init_records = DEF_INIT_RECORDS)` : Clears a table of all its records, in the current database.
- `public procedure db_rename_table(sequence name, sequence new_name)` : Renames a table in the current database.
- `public function db_table_list()` : Lists all tables in the current database.
- `public function db_find_key(object key, object table_name=current_table_name)` : Finds the record in the current table with supplied key.
- `public function db_insert(object key, object data, object table_name=current_table_name)` : Inserts a new record into the current table.
- `public procedure db_delete_record(integer key_location, object table_name=current_table_name)` : Deletes record number `key_location` from the current table.
- `public procedure db_replace_data(integer key_location, object data, object table_name=current_table_name)` : Replaces, the current table, the data portion of a record with new data.
- `public function db_table_size(object table_name=current_table_name)` : Gets the size (number of records) of the default table.
- `public function db_record_data(integer key_location, object table_name=current_table_name)` : Returns the data in a record queried by position.
- `public function db_fetch_record(object key, object table_name=current_table_name)` : Returns the data for the record with supplied key.
- `public function db_record_key(integer key_location, object table_name=current_table_name)` : Returns the key of a record given an index.
- `public function db_compress()` : Compresses the current database.
- `public function db_current ()` : Gets name of currently selected database.
- `public procedure db_cache_clear()` : Forces the database index cache to be cleared.
- `public function db_set_caching(atom new_setting)` : Sets the key cache behavior.
- `public procedure db_replace_recid(integer recid, object data)` : Replaces, in the current database, the data portion of a record with new data.
- `public function db_record_recid(integer recid)` : Returns the key and data in a record queried by `recid` .
- `public function db_get_recid(object key, object table_name=current_table_name)` : Returns the unique record identifier (`recid`) value for the record.

2.8.8 Desktop GUI & Web Frontend Frameworks

`std/gui.e` (Namespace: `gui`)

- **Purpose:** Native Java Swing & FlatLaf desktop GUI toolkit: GTK+/Nord/Dark themes, subpixel LCD antialiasing, controls, dialogs, and event listeners.
- **Public Functions & Procedures (36):**
- `public procedure init_gui()` : Initializes the GUI subsystem, applying the native Linux Look and Feel (GTK/System).
- `public procedure set_theme(sequence theme_name)` : Sets the UI theme / Look and Feel. Supported values for theme_name: # "gtk" : Native Linux GTK+ Look and Feel # "flatlaf" : Modern FlatLaf Light # "flatlaf-light" : Modern FlatLaf Light # "flatlaf-dark" : Modern FlatLaf Dark # "nimbus" : Modern Nimbus # "system" : Host System Look and Feel.
- `public function window(sequence title = "Euphoria", integer width = 600, integer height = 400, integer on_close = CLOSE_ON_EXIT)` : Creates a top-level application window (JFrame).
- `public procedure show(atom widget)` : Shows a window or component on screen.
- `public procedure hide(atom widget)` : Hides a window or component.
- `public procedure close_window(atom widget)` : Destroys and disposes a window and releases its graphics resources.
- `public procedure pack(atom win)` : Resizes window to fit the preferred size and layout of its subcomponents.
- `public function button(sequence text, integer on_click_rid = -1)` : Creates a button widget.
- `public procedure on_click(atom widget, integer on_click_rid)` : Attaches an on-click action listener to a button or interactive component.
- `public function text_label(sequence text = "", integer align = ALIGN_LEFT)` : Creates a text label widget.
- `public function create_label(sequence text = "", integer align = ALIGN_LEFT)` : Creates a text label (JLabel) with specified text and alignment.
- `public function text_field(sequence initial_text = "", integer columns = 20)` : Creates a single-line text input field.

- `public function text_area(sequence initial_text = "", integer rows = 10, integer cols = 30)` : Creates a multi-line text area.
- `public function scroll_pane(atom view_widget)` : Wraps a component in a scroll pane with horizontal and vertical scrollbars.
- `public function check_box(sequence text, integer checked = 0, integer on_change_rid = -1)` : Creates a checkbox.
- `public function radio_button(sequence text, integer selected = 0)` : Creates a radio button.
- `public function combo_box(sequence items = {})` : Creates a dropdown selection combo box.
- `public function panel(integer layout_type = FLOW_LAYOUT, integer rows = 0, integer cols = 0, integer hgap = 0, integer vgap = 0)` : Creates an empty panel container.
- `public function progress_bar(integer min_val = 0, integer max_val = 100)` : Creates a progress bar.
- `public function slider(integer min_val = 0, integer max_val = 100, integer init_val = 50)` : Creates a slider.
- `public procedure add(atom container, atom child, object constraint = 0)` : Adds a child component into a container (window, panel).
- `public procedure set_layout(atom container, integer layout_type, integer rows = 0, integer cols = 0, integer hgap = 0, integer vgap = 0)` : Sets the layout manager of a container.
- `public procedure set_text(atom widget, sequence text)` : Sets the text of a widget (label, button, text field, text area).
- `public function get_text(atom widget)` : Returns the text of a widget.
- `public procedure set_value(atom widget, integer val)` : Sets the integer value of a widget (slider, progress bar).
- `public function get_value(atom widget)` : Returns the integer value of a widget (slider, progress bar).
- `public procedure set_padding(atom widget, integer top, integer left, integer bottom, integer right)` : Sets an empty border (padding) on a widget in pixels.
- `public procedure set_font(atom widget, sequence name = "Ubuntu", integer size = 13, integer is_bold = 0)` : Sets the font of a widget.
- `public procedure set_enabled(atom widget, integer enabled)` : Enables or disables a component.
- `public function is_selected(atom widget)` : Returns whether a checkbox or radio button is selected.
- `public procedure set_selected(atom widget, integer sel)` : Selects or deselects a checkbox or radio button.
- `public procedure msgbox(sequence title, sequence message, integer msg_type = MSG_INFO)` : Displays a modal informational/error message dialog.
- `public function input_box(sequence title, sequence message)` : Displays a modal text input dialog and returns the entered string (or "" on cancel).
- `public function confirm_box(sequence title, sequence message)` : Displays a modal YES/NO confirmation dialog. Returns 1 for YES, 0 for NO/Cancel.
- `public function file_dialog(sequence title = "Datei auswählen", integer is_save = 0, sequence cur_dir = "")` : Displays a native file selection dialog.
- `public procedure wait_window(atom win)` : Blocks execution until the specified window is closed.

`std/ui5.e` (Namespace: `ui5`)

- **Purpose:** Enterprise web frontend framework based on SAP UI5 Web Components 2.29: Fiori Horizon theme, reactive DOM patches, and integrated HTTP server.
- **Public Functions & Procedures (65):**
- `public function page(sequence title, object content, sequence options = {})` : Generates full HTML5 document with Fiori Horizon theme and reactive bridge.
- `public function title(sequence text, sequence level = TITLE_H2, sequence options = {})` : Creates a heading element (`ui5-title`) from H1 to H6 level.
- `public function ui_label(sequence text, sequence for_id = "", integer required = 0, sequence options = {})` : Creates a standard label element (`ui5-label`).
- `public function form_label(sequence text, sequence for_id = "", integer required = 0, sequence options = {})` : Creates a form-oriented UI5 label (`ui5-label`) with required field support.
- `public function link(sequence text, sequence href = "#", sequence design = LINK_DEFAULT, sequence options = {})` : Creates a hyperlink (`ui5-link`) with configurable design options.
- `public function text_block(sequence text, sequence options = {})` : Creates a typographical text block or paragraph.
- `public function shellbar(sequence primary_title, sequence secondary_title = "", sequence options = {})` : Creates an enterprise header bar (`ui5-shellbar`) with avatar and actions.
- `public function breadcrumbs(sequence items, sequence options = {})` : Creates a breadcrumb navigation trail (`ui5-breadcrumbs`).
- `public function bar(object start_content, object middle_content = {}, object end_content = {}, sequence design = BAR_HEADER, sequence options = {})` : Creates an action bar (`ui5-bar`) for header, subheader, or footer areas.
- `public function card(sequence title, object content, sequence options = {})` : Creates a Fiori card (`ui5-card`) with header, status icon, and content.
- `public function panel(sequence header_text, object content, integer collapsed = 0, sequence options = {})` : Creates a collapsible panel container (`ui5-panel`).
- `public function tab_container(sequence tabs, sequence options = {})` : Creates a tab navigation container (`ui5-tabcontainer`).

- `public function table(sequence columns, sequence rows, sequence options = {})` : Creates a responsive table (`ui5-table`) with column headers and rows.
- `public function table_from_query(sequence headers, sequence query_rows, sequence options = {})` : Generates a complete Fiori table directly from an SQL query result.
- `public function list(sequence items, sequence options = {})` : Creates an interactive list (`ui5-list`).
- `public function tree(sequence items, sequence options = {})` : Creates a hierarchical tree structure (`ui5-tree`).
- `public function button(sequence id, sequence text, sequence options = {})` : Creates a button (`ui5-button`) with design (Default, Emphasized, Positive, Negative).
- `public function toggle_button(sequence id, sequence text, integer pressed = 0, sequence options = {})` : Creates a toggle button (`ui5-toggle-button`).
- `public function menu_button(sequence id, sequence text, sequence options = {})` : Creates a menu button (`ui5-menu-button`).
- `public function segmented_button(sequence id, sequence items, sequence options = {})` : Creates a segmented button group (`ui5-segmented-button`).
- `public function input(sequence id, sequence value = "", sequence options = {})` : Creates a single-line input field (`ui5-input`).
- `public function step_input(sequence id, atom value = 0, atom min_val = 0, atom max_val = 100, atom step_val = 1, sequence options = {})` : Creates a numeric step input (`ui5-step-input`) with plus/minus steppers.
- `public function slider(sequence id, atom value = 0, atom min_val = 0, atom max_val = 100, atom step_val = 1, sequence options = {})` : Creates a numeric slider (`ui5-slider`).
- `public function range_slider(sequence id, atom start_val = 20, atom end_val = 80, atom min_val = 0, atom max_val = 100, atom step_val = 1, sequence options = {})` : Creates a range slider (`ui5-range-slider`) with two thumbs.
- `public function rating_indicator(sequence id, atom value = 3, integer max_val = 5, sequence options = {})` : Creates a star rating indicator (`ui5-rating-indicator`).
- `public function textarea(sequence id, sequence value = "", sequence options = {})` : Creates a multi-line text area (`ui5-textarea`).
- `public function checkbox(sequence id, sequence text, integer checked = 0, sequence options = {})` : Creates a checkbox (`ui5-checkbox`).
- `public function radio_button(sequence id, sequence name, sequence text, sequence value = "", integer checked = 0, sequence options = {})` : Creates a radio button (`ui5-radio-button`).
- `public function switch_btn(sequence id, sequence text_on = "Ein", sequence text_off = "Aus", integer checked = 0, sequence options = {})` : Creates a graphical toggle switch (`ui5-switch`).
- `public function select(sequence id, sequence options_list, sequence options = {})` : Creates a drop-down select control (`ui5-select`).
- `public function combobox(sequence id, sequence value = "", sequence items = {}, sequence options = {})` : Creates a filterable combo box (`ui5-combobox`).
- `public function multi_combobox(sequence id, sequence items = {}, sequence options = {})` : Creates a multi-select combo box (`ui5-multi-combobox`) with token display.
- `public function datepicker(sequence id, sequence value = "", sequence options = {})` : Creates a date picker (`ui5-datepicker`) with calendar popup.
- `public function time_picker(sequence id, sequence value = "", sequence options = {})` : Creates a time picker (`ui5-time-picker`).
- `public function datetime_picker(sequence id, sequence value = "", sequence options = {})` : Creates a combined date-time picker (`ui5-datetime-picker`).
- `public function daterange_picker(sequence id, sequence value = "", sequence options = {})` : Creates a date range picker (`ui5-daterange-picker`).
- `public function file_uploader(sequence id, sequence placeholder = "Datei auswählen...", sequence options = {})` : Creates a file upload button (`ui5-file-uploader`).
- `public function color_picker(sequence id, sequence color = "#0070f2", sequence options = {})` : Creates a color picker (`ui5-color-picker`).
- `public function dialog(sequence id, sequence header_text, object content, object footer_buttons = {}, sequence options = {})` : Creates a modal dialog (`ui5-dialog`).
- `public function popover(sequence id, sequence header_text, object content, object footer_buttons = {}, sequence options = {})` : Creates a floating popover overlay (`ui5-popover`).
- `public function toast(sequence id, sequence text = "", integer duration = 3000)` : Creates a toast notification element (`ui5-toast`).
- `public function badge(sequence text, integer color_scheme = 1, sequence options = {})` : Creates a status badge (`ui5-badge`).
- `public function tag(sequence text, integer color_scheme = 1, sequence options = {})` : Creates an interactive tag (`ui5-tag`).
- `public function icon(sequence name, sequence options = {})` : Embeds an SAP icon (`ui5-icon`).
- `public function avatar(sequence initials_or_src, sequence size = AVATAR_M, sequence shape = AVATAR_CIRCLE, sequence options = {})` : Creates an avatar or initials element (`ui5-avatar`).
- `public function busy_indicator(integer active = 1, sequence size = BUSY_MEDIUM, sequence text = "", object content = {}, sequence options = {})` : Creates a loading spinner (`ui5-busy-indicator`).
- `public function progress_indicator(integer value_pct, sequence display_value = "", sequence state = STATE_NONE, sequence options = {})` : Creates a progress bar (`ui5-progress-indicator`).

- `public function message_strip(sequence text, sequence design = "Information", sequence options = {})` : Creates an alert banner (`ui5-message-strip`).
- `public function timeline(sequence items, sequence options = {})` : Creates an activity timeline (`ui5-timeline`).
- `public function illustrated_message(sequence name = "NoData", sequence title_text = "Keine Daten vorhanden", sequence subtitle_text = "", sequence options = {})` : Creates an empty state vector illustration (`ui5-illustrated-message`).
- `public function patch(sequence id, sequence html)` : 5. Reactive Patch & Bridge Helpers.
- `public function append_html(sequence id, sequence html)` : Creates a DOM patch instruction appending raw HTML to element `id` .
- `public function set_val(sequence id, sequence value)` : Creates a patch instruction setting the `value` property of field `id` .
- `public function set_value_state(sequence id, sequence state, sequence message = "")` : Sets the visual validation state (`None` , `Error` , `Warning` , `Success` , `Information`).
- `public function toast_show(sequence id, sequence text = "")` : Triggers a toast notification popup (`ui5-toast`) on the client.
- `public function open_dialog(sequence id)` : Opens a modal dialog window on the client UI.
- `public function close_dialog(sequence id)` : Closes an active modal dialog window on the client UI.
- `public function redirect(sequence url)` : Instructs the client browser to navigate to a new URL.
- `public function updates(sequence patches_list)` : Packages a list of patch instructions into an event response structure.
- `public function val(sequence ctx, sequence key, sequence default_val = "")` : Extracts the client-submitted form value for key `id` from the event context.
- `public function app(sequence title, integer port = 8080, sequence theme = THEME_HORIZON, sequence version = DEFAULT_VERSION, sequence cdn = DEFAULT_CDN_BASE)` : Generates full SPA application container.
- `public procedure page_route(integer app_id, sequence path, integer handler_rid)` : Associates a URL path with a page rendering routine.
- `public procedure action(integer app_id, sequence action_name, integer handler_rid)` : Registers a handler routine for UI event interactions.
- `public function init_server(integer app_id)` : Initializes the embedded HTTP server with port and page metadata.
- `public procedure run(integer app_id)` : Starts the reactive SAP UI5 application on the configured port.

`std/vsui.e` (Namespace: `vsui`)

- **Purpose:** Self-contained high-density ERP web framework with local assets (`html/`): sticky-header DataGrid, vsevent bridge, reactive DOM patches, SVG charts, upload, and mobile responsiveness.
- **Public Functions & Procedures (47):**
- `public function page(sequence title, object content, sequence options = {})` : Generates high-density HTML container with VS-UI CSS (`globalblauklein.css` , `vs.css`), `vs-mobile.css` , and ES6 modules.
- `public function button(sequence id, sequence text, sequence options = {})` : Creates a high-density button (`.vs-button`) with event forwarding.
- `public function input(sequence id, sequence value = "", sequence options = {})` : Creates a compact text input (`.vs-input`).
- `public function numinput(sequence id, atom value = 0, sequence options = {})` : Creates a numeric input field (`type="number"`).
- `public function dateinput(sequence id, sequence value = "", sequence options = {})` : Creates an HTML5 date input (`type="date"`).
- `public function timeinput(sequence id, sequence value = "", sequence options = {})` : Creates an HTML5 time input (`type="time"`).
- `public function check(sequence id, sequence label_text, integer checked = 0, sequence options = {})` : Creates a checkbox (`.vs-check`).
- `public function textarea(sequence id, sequence value = "", integer rows = 4, sequence options = {})` : Creates a compact textarea (`.vs-input`).
- `public function combo(sequence id, sequence options_list, sequence selected_val = "", sequence options = {})` : Creates a dropdown select box (`.vs-combo`).
- `public function ui_label(sequence text, sequence for_id = "", sequence options = {})` : Creates a form label (`.vs-label`).
- `public function group(sequence legend_text, object content, sequence options = {})` : Creates a fieldset grouping frame (`.vs-group`).
- `public function panel(sequence id, sequence title, object content, sequence options = {})` : Creates a high-density panel container (`.vs-panel`).
- `public function collapse(sequence id, sequence sections, integer open_first = 1, sequence options = {})` : Creates collapsible sections (`<details>` / `<summary>`).
- `public function grid(sequence id, sequence columns, sequence rows, sequence options = {})` : Creates the high-density DataGrid with sticky header, live search, and automatic mobile card-view.
- `public function tabs(sequence id, sequence tabs_list, sequence options = {})` : Creates tab navigation with dynamic panel switching and horizontal mobile swipe.
- `public function modal(sequence id, sequence title, object content, object footer_buttons = {}, sequence options = {})` : Creates a modal dialog popup with dark backdrop (fullscreen on mobile).
- `public function splitter(sequence id, object left_content, object right_content, integer split_pct = 50, sequence options = {})` : Creates a 2-column layout container with adjustable ratio (collapses to column on mobile).

- `public function upload(sequence id, sequence target_url = "/api/upload", sequence options = {})` : Creates a modern drag-and-drop file upload dropzone with progress bar.
- `public function chart(sequence id, sequence chart_type, sequence labels, sequence datasets, sequence options = {})` : Generates a responsive server-rendered vector SVG chart (`"bar"` , `"line"` , `"pie"` , `"doughnut"`).
- `public function chart_set_data(sequence id, sequence chart_type, sequence labels, sequence datasets)` : Generates a live DOM patch instruction updating chart datasets smoothly without reload.
- `public function field_error(sequence id, sequence error_msg)` : Highlights an input field in red and displays an explanatory error tooltip below it.
- `public function field_clear_error(sequence id)` : Clears error styling and error tooltip from a form field.
- `public function validate_required(sequence ctx, sequence field_id, sequence field_label = "")` : Verifies whether a mandatory input is filled in request context.
- `public function validate_email(sequence email_str)` : Validates email format (`user@domain.tld`).
- `public function validate_number(sequence num_str, atom min_val = -1e9, atom max_val = 1e9)` : Validates numeric input within min/max bounds.
- `public function patch(sequence id, sequence html)` : Creates a DOM patch instruction replacing content of element.
- `public function append_html(sequence id, sequence html)` : Creates a DOM patch instruction appending HTML to target element.
- `public function set_val(sequence id, sequence value)` : Creates a patch instruction updating an input field value on the client (invoking `.setValue()`).
- `public function grid_set_data(sequence id, sequence data)` : Updates grid or table data rows via JSON patch without page or HTML re-rendering.
- `public function set_focus(sequence id)` : Sets keyboard focus to the target element.
- `public function set_enabled(sequence id, integer is_enabled)` : Enables or disables an input or button element reactively.
- `public function set_visible(sequence id, integer is_visible)` : Toggles element visibility (`display: none` vs `''`).
- `public function set_css(sequence id, sequence prop, sequence val)` : Sets inline CSS property directly on target element.
- `public function set_class(sequence id, sequence class_name, sequence action = "add")` : Adds, removes, or toggles a CSS class on an element.
- `public function clear_form(sequence container_id = "")` : Clears all form fields within a container element.
- `public function toast_show(sequence text, sequence toast_type = TOAST_INFO, integer duration = 3000)` : Displays a high-density toast message on the client.
- `public function modal_open(sequence id)` : Opens a modal popup dialog.
- `public function modal_close(sequence id)` : Closes an open modal popup dialog.
- `public function redirect(sequence url)` : Instructs the browser to redirect to another URL.
- `public function updates(sequence patches_list)` : Serializes patch instructions for the `vsevent` reactive bridge.
- `public function val(sequence ctx, sequence key, sequence default_val = "")` : Extracts submitted form value from request context (invoking `.getValue()`).
- `public function app(sequence title, integer port = DEFAULT_PORT, sequence static_dir = DEFAULT_STATIC_DIR)` : Creates a complete high-density ERP application layout.
- `public procedure page_route(integer app_id, sequence path, integer handler_rid)` : Maps an HTTP route path to a page generator function.
- `public procedure on_event(integer app_id, sequence event_name, integer handler_rid)` : Binds a custom event name to an event handler routine.
- `public procedure on_click(integer app_id, sequence button_id, integer handler_rid)` : Binds a button click action to an event handler routine.
- `public function init_server(integer app_id)` : Initializes the VS-UI server with asset root directory.
- `public procedure run(integer app_id)` : Starts the high-density ERP web application server.

`std/graphics.e` (Namespace: `graphics`)

- **Purpose:** Cross-platform text-mode graphics: foreground/background colors, cursor coordinates, and text wrapping.
- **Public Functions & Procedures (7):**
- `public function get_position()` : Returns the current line and column position of the cursor.
- `public procedure text_color(color c)` : Sets the foreground text color.
- `public procedure bk_color(color c)` : Sets the background color to one of the sixteen standard colors.
- `public function console_colors(sequence colorset = {})` : Sets the codes for the colors used in `text_color` and `bk_color` .
- `public procedure wrap(object on = 1)` : Determines whether text will wrap when hitting the rightmost column.
- `public procedure scroll(integer amount, console:positive_int top_line, console:positive_int bottom_line)` : Scrolls a region of text on the screen.
- `public function graphics_mode(object m = -1)` : Attempts to set up a new graphics mode.

`std/graphcst.e` (Namespace: `graphcst`)

- **Purpose:** Video mode constants, color palette definitions, and screen configuration structures.
- **Public Functions & Procedures (1):**
- `public function video_config()` : Returns a description of the current video configuration.

`std/image.e` (Namespace: `image`)

- **Purpose:** Bitmap image handling: reading and writing uncompressed Windows bitmap (.BMP) files and color palette mapping.
- **Public Functions & Procedures (2):**
- `public function read_bitmap(sequence file_name)` : Reads a bitmap (`.BMP`) file into a 2-d sequence of sequences (image).
- `public function save_bitmap(two_seq palette_n_image, sequence file_name)` : Create a `.BMP` bitmap file, given a palette and a 2-d sequence of sequences of colors.

2.8.9 Java Interoperability, C-FFI & Machine Memory

`std/java.e` (Namespace: `java`)

- **Purpose:** Java 25 interoperability bridge: dynamic JAR loading (`load_jar`), class instantiation, static/instance method invocation, and callback proxies.
- **Public Functions & Procedures (9):**
- `public function load_jar(sequence jar_path)` : Loads a .jar archive into the runtime classpath dynamically.
- `public function find_class(sequence class_name)` : Resolves a Java class by its fully-qualified name or simple name in java.lang.
- `public function new_object(object class_or_name, sequence args = {})` : Instantiates a new Java object using the best matching constructor.
- `public function call_static(object class_or_name, sequence method_name, sequence args = {})` : Invokes a static method on a Java class.
- `public function call_method(atom obj_handle, sequence method_name, sequence args = {})` : Invokes an instance method on a Java object handle.
- `public function get_field(object target_or_class, sequence field_name)` : Retrieves the value of a public field from a class or object.
- `public function set_field(object target_or_class, sequence field_name, object val)` : Sets the value of a public field on a class or object.
- `public function to_string(atom obj_handle)` : Calls toString() on a Java object handle and returns the Euphoria string.
- `public function create_listener(object interface_class_or_name, integer routine_id)` : Creates a dynamic proxy implementing a Java listener interface (e.g. ActionListener, WindowListener, Runnable) which routes method calls to a Euphoria routine.

`std/dll.e` (Namespace: `dll`)

- **Purpose:** C Foreign Function Interface (FFI): dynamic linking to native shared libraries (.so / .dll) and C calling conventions.
- **Public Functions & Procedures (6):**
- `public function open_dll(sequence file_name)` : Opens a *Windows* dynamic link library (`.dll`) file, or a *Unix* shared library (`.so`) file.
- `public function define_c_var(atom lib, sequence variable_name)` : Gets the address of a symbol in a shared library or in RAM.
- `public function define_c_proc(object lib, object routine_name, sequence arg_types)` : Defines the characteristics of either a C function, or a machine-code routine that you wish to call as a procedure from your Euphoria program.
- `public function define_c_func(object lib, object routine_name, sequence arg_types, atom return_type)` : Defines the characteristics of either a C function, or a machine-code routine that returns a value.
- `public function call_back(object id)` : Gets a machine address for an Euphoria procedure.
- `public function sizeof(integer x)` : --** -- @nodoc@.

`std/machine.e` (Namespace: `machine`)

- **Purpose:** Low-level machine and memory access: direct memory manipulation (`peek` , `poke`), block allocation, and low-level system calls.
- **Public Functions & Procedures (24):**
- `public procedure poke8(atom address, object x)` : --** -- @nodoc@.
- `public procedure poke_pointer(atom address, object x)` : --** -- @nodoc@.
- `public function peek8s(object address)` : --** -- @nodoc@.
- `public function peek8u(object address)` : --** -- @nodoc@.
- `public function peek_pointer(object x)` : --** -- @nodoc@.
- `public procedure poke_long(atom address, object x)` : --** -- @nodoc@.
- `public function peek_longs(object x)` : --** -- @nodoc.
- `public function peek_longu(object x)` : --** -- @nodoc@.

- `public function allocate( memory:positive_int n, types:boolean cleanup = 0)` : This does the same as `allocate_data` but allows the DATA_EXECUTE defined word to cause it to return executable memory.
- `public function allocate_data( memory:positive_int n, types:boolean cleanup = 0)` : Allocate a contiguous block of data memory.
- `public function allocate_pointer_array(sequence pointers, types:boolean cleanup = 0)` : Allocate a NULL terminated pointer array.
- `public procedure free_pointer_array(atom pointers_array)` : Free a NULL terminated pointers array.
- `public function allocate_string_pointer_array(object string_list, types:boolean cleanup = 0)` : Allocate a C-style null-terminated array of strings in memory.
- `public function allocate_wstring(sequence s, types:boolean cleanup = 0 )` : Create a C-style null-terminated wchar_t string in memory.
- `public function peek_wstring(atom addr)` : Returns a unicode (utf16) string that are stored at machine address a.
- `public function poke_string(atom buffaddr, integer buffsize, sequence s)` : Stores a C-style null-terminated ANSI string in memory.
- `public function poke_wstring(atom buffaddr, integer buffsize, sequence s)` : Stores a C-style null-terminated Double-Byte string in memory.
- `public function is_DEP_supported()` : @nodoc@.
- `public function is_using_DEP()` : @nodoc@.
- `public procedure DEP_on(integer value)` : @nodoc@.
- `public function allocate_code( object data, memconst:valid_wordsize wordsize = 1 )` : Allocates and copies data into executable memory.
- `public function allocate_string(sequence s, types:boolean cleanup = 0 )` : Allocate a C-style null-terminated string in memory.
- `public function allocate_protect( object data, memconst:valid_wordsize wordsize = 1, valid_memory_protection_constant protection )` : Allocates and copies data into memory and gives it protection using `Standard Library Memory Protection Constants` or `Microsoft Windows Memory Protection Constants` . The user may only pass in one of these constants. If you only wish to execute a sequence as machine code use `allocate_code` . If you only want to read and write data into memory use `allocate` . See [[http://msdn.microsoft.com/en-us/library/aa366786(VS.85).aspx "MSDN: Microsoft's Memory Protection Constants"]].
- `public procedure free(object addr)` : Frees up a previously allocated block of memory.

`std/memory.e` (Namespace: `memory`)

- **Purpose:** Low-level native memory management: heap memory allocation (`allocate`), deallocation (`free`), memory protection, and pointer operations.
- **Public Functions & Procedures (8):**
- `export procedure deallocate(atom addr)` : @nodoc@ Internal use of the library only. free() calls this. It works with only atoms and in the SAFE implementation is different. @nodoc@.
- `public procedure register_block(atom block_addr, atom block_len, integer protection )` : @nodoc@.
- `public procedure unregister_block(atom block_addr)` : @nodoc@.
- `public function safe_address(atom start, integer len, positive_int action)` : @nodoc@.
- `public procedure check_all_blocks()` : @nodoc@.
- `export function prepare_block( atom addr, integer a, integer protection )` : @nodoc@.
- `export function dep_works()` : @nodoc@ Returns 1 if the DEP executing data only memory would cause an exception @nodoc@.
- `public procedure free_code( atom addr, integer size, valid_wordsize wordsize = 1 )` : @nodoc@.

`std/memconst.e` (Namespace: `memconst`)

- **Purpose:** Constants for word sizes, machine data types, and memory protection access modes (read, write, execute).
- **Public Functions & Procedures (3):**
- `export function test_read( valid_memory_protection_constant protection )` : @nodoc@.
- `export function test_write( valid_memory_protection_constant protection )` : @nodoc@.
- `export function test_exec( valid_memory_protection_constant protection )` : @nodoc@.

`std/safe.e` (Namespace: `safe`)

- **Purpose:** Safe debug variant of `memory.e` and `machine.e` : automatic detection of buffer overruns, wild pointers, and invalid memory dereferences.
- **Public Functions & Procedures (12):**
- `public function safe_address(machine_addr start, natural len, positive_int action )` : @nodoc@.
- `public procedure show_block(sequence block_info)` : @nodoc@.
- `public procedure check_all_blocks()` : @nodoc@.
- `public procedure register_block(machine_addr block_addr, positive_int block_len, valid_memory_protection_constant memory_protection = PAGE_READ_WRITE )` : @nodoc@.
- `public procedure unregister_block(machine_addr block_addr)` : @nodoc@.
- `export function prepare_block(int_addr iaddr, positive_int n, natural protection)` : @nodoc@.
- `export procedure deallocate(atom a)` : @nodoc@.

- `export function dep_works()` : @nodoc@.
- `public procedure free_code( atom addr, integer size, valid_wordsize wordsize = 1 )` : @nodoc@.
- `public function info()` : @nodoc@.
- `public function memory_used()` : @nodoc@.
- `public function allocations()` : @nodoc@.

`std/eumem.e` (Namespace: `eumem`)

- **Purpose:** Pseudo-memory management emulating heap pointers and pass-by-reference semantics within Euphoria RAM.
- **Public Functions & Procedures (3):**
- `export function malloc(object mem_struct_p = 1, integer cleanup_p = 1)` : Allocates a block of (pseudo) memory.
- `export procedure free(atom mem_p)` : Deallocates a block of (pseudo) memory.
- `export function valid(object mem_p, object mem_struct_p = 1)` : Validates a block of (pseudo) memory.

2.8.10 Modern Concurrency, Threads & Multitasking

`std/thread.e` (Namespace: `thread`)

- **Purpose:** High-scalability concurrency powered by Java 25 Virtual Threads: `spawn`, async/await futures (`async`, `await`, `await_all`), and sleep management.
- **Public Functions & Procedures (9):**
- `public function spawn(integer routine_id, sequence args = {})` : Start an independent background task on a lightweight Java 25 Virtual Thread. Returns the unique thread ID (> 0).
- `public function async(integer routine_id, sequence args = {})` : Run a function asynchronously in the background and return a Job / Future handle. The return value of the function can later be retrieved via `await`.
- `public function await(atom job_id, atom timeout = -1)` : Wait for an asynchronous job to finish and return its result value. If `timeout` is negative (-1), waits indefinitely. If `timeout` is positive, returns "TIMEOUT" if not finished within that time.
- `public function is_done(atom job_id)` : Check whether an asynchronous job has finished without blocking. Returns 1 if completed, 0 if still running.
- `public function await_all(sequence job_ids, atom timeout = -1)` : Wait for a list of asynchronous jobs to complete and return a sequence of their results.
- `public function join(atom thread_id, atom timeout = 0)` : Wait for an independent thread to terminate. Returns 1 if the thread has finished, 0 if timeout expired.
- `public procedure sleep(atom seconds)` : Pause the current Virtual Thread non-blockingly for `seconds`. Other virtual threads continue running with zero performance degradation.
- `public function current()` : Return the integer ID of the currently executing thread.
- `public procedure cancel(atom job_or_thread_id)` : Cancel an active async job or interrupt a running virtual thread.

`std/channel.e` (Namespace: `channel`)

- **Purpose:** Go-style channels (CSP): thread-safe, blocking or non-blocking message queues for lock-free communication between concurrent threads.
- **Public Functions & Procedures (9):**
- `public function new(integer capacity = 0)` : Create a new Channel. *`capacity = 0` creates an unbuffered rendezvous channel (synchronous handoff).* `capacity > 0` creates a buffered channel with the given queue size.
- `public function send(atom ch, object val, atom timeout = -1)` : Send a value into the channel. Blocks until space is available or timeout expires.
- `public function receive(atom ch, atom timeout = -1)` : Receive a message from the channel. Blocks until a message is available or timeout expires. Returns a 2-element sequence: `{status, value}` *`{1, val}` : Success* `{0, 0}` : Timeout * `{-1, 0}` : Channel is closed.
- `public function read(atom ch, object default_val = 0, atom timeout = -1)` : Convenience wrapper: receives and returns the message value directly. Returns `default_val` if closed or timed out.
- `public function try_send(atom ch, object val)` : Non-blocking send. Returns 1 if sent immediately, 0 if queue full, -1 if closed.
- `public function try_receive(atom ch)` : Non-blocking receive. Returns `{status, value}` immediately.
- `public procedure close(atom ch)` : Close a channel. Pending items can still be drained; subsequent sends will fail.
- `public function size(atom ch)` : Return the current number of buffered items in the channel.
- `public function is_closed(atom ch)` : Return 1 if the channel has been closed, 0 if open.

`std/task.e` (Namespace: `task`)

- **Purpose:** Classic cooperative multitasking: task creation, time slicing, scheduling priorities, and yielding controls.
- **Public Functions & Procedures (2):**
- `public procedure task_delay(atom delaytime)` : Suspends a task for a short period, allowing other tasks to run in the meantime.

- `public function create_thread(integer rid, sequence args = {})` : Start an independent Task running on a Java 25 Virtual Thread. Runs concurrently across CPU cores without blocking other tasks on I/O.

2.8.11 Error Handling, Testing & Utilities

`std/error.e` (Namespace: `error`)

- **Purpose:** Centralized error handling: controlled abnormal termination (`crash`), crash message log files, and custom crash callback hooks.
- **Public Functions & Procedures (5):**
- `public procedure crash(sequence fmt, object data={})` : Crashes the running program and displays a formatted error message.
- `public procedure crash_message(sequence msg)` : Specifies a final message to be displayed to your user, in the event that Euphoria has to shut down your program due to an error.
- `public procedure crash_file(sequence file_path)` : Specifies a file path name in place of `"ex.err"` where you want any diagnostic information to be written.
- `public procedure warning_file(object file_path)` : Specifies a file path where to output warnings.
- `public procedure crash_routine(integer func)` : Specifies a function to be called when an error takes place at run time.

`std/unittest.e` (Namespace: `unittest`)

- **Purpose:** Built-in unit testing framework: test assertions (`test_equal` , `test_true` , `test_false`), failure reporting, and summary statistics.
- **Public Functions & Procedures (12):**
- `public procedure set_test_verbosity(atom verbosity)` : Set the amount of information that is returned about passed and failed tests.
- `public procedure set_wait_on_summary(integer to_wait)` : Requests the test report to pause before exiting.
- `public procedure set_accumulate_summary(integer accumulate)` : Requests the test report to save run stats in `"unittest.dat"` before exiting.
- `public function set_test_abort(integer abort_test)` : Sets the behavior on test failure, and return previous value.
- `public procedure test_report()` : Outputs the test report.
- `public procedure test_equal(sequence name, object expected, object outcome)` : Records whether a test passes by comparing two values.
- `public procedure test_not_equal(sequence name, object a, object b)` : Records whether a test passes by comparing two values.
- `public procedure test_true(sequence name, object outcome)` : Records whether a test passes.
- `public procedure assert(object name, object outcome)` : Records whether a test passes. If it fails, the program also fails.
- `public procedure test_false(sequence name, object outcome)` : Records whether a test passes by comparing two values.
- `public procedure test_fail(sequence name)` : Records that a test failed.
- `public procedure test_pass(sequence name)` : Records that a test passed.

`std/utils.e` (Namespace: `utils`)

- **Purpose:** General language utilities: inline if function (`iif`), conditional expressions, and multi-variable assignment (`assign`).
- **Public Functions & Procedures (3):**
- `public function iif(atom test, object ifTrue, object ifFalse)` : Used to embed an 'if' test inside an expression. `iif` stands for inline if or immediate if.
- `public function assign(integer i1, object val1, integer i2 = 0, object val2 = 0, integer i3 = 0, object val3 = 0, integer i4 = 0, object val4 = 0, integer i5 = 0, object val5 = 0, integer i6 = 0, object val6 = 0, integer i7 = 0, object val7 = 0, integer i8 = 0, object val8 = 0, integer i9 = 0, object val9 = 0, integer i10 = 0, object val10 = 0)` : Assigns multiple target variables in a single compact call.
- `public function iff(atom test, object ifTrue, object ifFalse)` : @nodoc@.

2.8.12 Windows Platform-Specific Modules

`std/win32/msgbox.e` (Namespace: `msgbox`)

- **Purpose:** Windows-specific dialog popups: displays native Win32 message boxes with configurable buttons and system alert icons.
- **Public Functions & Procedures (1):**
- `public function message_box(sequence text, sequence title, object style)` : Displays a window with a title, message, buttons and an icon, usually known as a message box.

`std/win32/sounds.e` (Namespace: `sound`)

- **Purpose:** Windows system audio alerts: plays platform notification sounds and acoustic warnings (`MessageBeep`).
- **Public Functions & Procedures (1):**

- `public procedure sound( atom sound_type = SND_DEFAULT )` : Makes a sound.

`std/win32/w32dllconst.ew` (Namespace: `w32dll`)

- **Purpose:** Win32 API data type definitions and constants (`HWND` , `HANDLE` , `DWORD` , `WPARAM` , `LPARAM`) for native DLL binding.
- **Notes & Key Constants:** Defines Win32 API type constants for shared library calls (`C_HANDLE` , `C_HWND` , `C_WPARAM` , `C_LPARAM` , `C_HRESULT` , etc.). Contains constants only.

2.8.13 Euphoria Internals, Debugger & Syntax Tokenizer

`std/csv.e` (Namespace: `csv`)

- **Purpose:** RFC 4180 compliant CSV Engine: Parse and serialize tabular data with customizable delimiters (comma, semicolon, tab), automatic quote escaping (`""`), newline preservation, and UTF-8 encoding.
- **Public Functions & Procedures (5):**
- `public function format_field(object val, integer delimiter = ',', integer quote_all = 0)` : Formats an individual value conforming to RFC 4180 with auto-quoting.
- `public function to_csv(sequence headers, sequence rows, integer delimiter = ',', integer quote_all = 0)` : Converts headers and rows to a complete CSV string.
- `public function parse(sequence csv_text, integer delimiter = ',')` : Parses CSV text into row and column sequences (`{ {col1, col2, ...}, ... }`).
- `public function write_file(sequence filepath, sequence headers, sequence rows, integer delimiter = ',', integer quote_all = 0)` : Writes tabular data to a CSV file.
- `public function read_file(sequence filepath, integer delimiter = ',', integer has_header = 1)` : Reads a CSV file and returns `{headers, rows}` .

`std/excel.e` (Namespace: `excel`)

- **Purpose:** Native Microsoft Excel (.xlsx) OpenXML Engine: Generates and reads modern `.xlsx` workbooks directly using Java 25 standard runtime zip & xml engine—zero external dependencies or POI needed.
- **Public Functions & Procedures (3):**
- `public function write_file(sequence filepath, sequence headers, sequence rows, sequence sheet_name = "Tabelle1")` : Creates a valid `.xlsx` spreadsheet with styled headers and typed numbers.
- `public function export_table(sequence filepath, sequence headers, sequence rows, sequence sheet_name = "Export")` : Convenient export helper for SQL queries and UI grids.
- `public function read_file(sequence filepath)` : Reads an `.xlsx` workbook and extracts table rows and cells (`{1, rows}`).

`std/pdf_report.e` (Namespace: `pdf_report`)

- **Purpose:** Native Business PDF Engine: Generates standard PDF 1.4 reports, invoices, summaries, and data tables with corporate headers, metadata cards, zebra striping, and automatic multi-page layout.
- **Public Functions & Procedures (2):**
- `public function create_report(sequence filepath, sequence title, sequence metadata, sequence headers, sequence rows, sequence options = {})` : Generates a professional multi-page PDF business report.
- `public function export_table(sequence filepath, sequence title, sequence headers, sequence rows)` : Quick PDF export for tabular data.

`std/auth.e` (Namespace: `auth`)

- **Purpose:** Enterprise Security & Authentication Module: PBKDF2 password hashing (65,536 iterations with salt), constant-time verification, JSON Web Tokens (JWT HS256), and cryptographic random tokens.
- **Public Functions & Procedures (5):**
- `public function hash_password(sequence password, sequence salt = "")` : Computes a PBKDF2-HMAC-SHA256 password hash.
- `public function verify_password(sequence password, sequence hash_str)` : Verifies plaintext password against stored hash in constant time (protecting against timing attacks).
- `public function random_token(integer byte_count = 32)` : Generates a cryptographically secure random token in hex format.
- `public function create_jwt(sequence payload, sequence secret_key, integer expiry_secs = 3600)` : Creates a signed JWT with HS256 signature and expiration (`exp`).
- `public function verify_jwt(sequence token, sequence secret_key)` : Cryptographically verifies JWT signature and expiration, returning `{1, payload_json}` .

`std/session.e` (Namespace: `session`)

- **Purpose:** HTTP Session Management for intranet and web applications: In-memory session store, automatic `Set-Cookie` header generation, and session lifetime tracking.
- **Public Functions & Procedures (7):**
- `public function extract_sid(sequence cookie_header, sequence cookie_name = "EUI_SESSION")` : Extracts session ID from incoming `Cookie` header.
- `public function start(sequence cookie_str = "", sequence cookie_name = "EUI_SESSION")` : Resumes existing or creates new session (`{sid, is_new}`).
- `public procedure set(sequence sid, sequence key, object val)` : Stores a key-value attribute in the session.
- `public function get(sequence sid, sequence key, object default_val = {})` : Retrieves an attribute value from the session.
- `public function has(sequence sid, sequence key)` : Checks if a key exists in the session.
- `public procedure destroy(sequence sid)` : Terminates and cleans up an active session.
- `public function cookie_header(sequence sid, integer max_age = 86400, sequence cookie_name = "EUI_SESSION", sequence path = "/", integer http_only = 1)` : Generates `Set-Cookie` header string.

`std/cron.e` (Namespace: `cron`)

- **Purpose:** Built-in Task Scheduler: Executes recurring background tasks (interval timers or daily wall-clock schedules) on lightweight Java 25 Virtual Threads.
- **Public Functions & Procedures (6):**
- `public function schedule_interval(sequence name, atom interval_seconds, integer routine_id, sequence args = {})` : Registers interval task.
- `public function schedule_daily(sequence name, integer hour, integer minute, integer routine_id, sequence args = {})` : Registers daily task at HH:MM.
- `public procedure start()` : Starts background scheduler thread.
- `public procedure stop()` : Stops background scheduler.
- `public function list_jobs()` : Returns list of registered jobs.
- `public function trigger_now(sequence name)` : Manually triggers a scheduled job immediately.

`std/sse.e` (Namespace: `sse`)

- **Purpose:** Server-Sent Events (SSE) Streaming Engine: Formats W3C-compliant `text/event-stream` message frames for browser live-push without WebSocket complexity.
- **Public Functions & Procedures (2):**
- `public function event(sequence event_name, sequence data, sequence id = "", integer retry_ms = 0)` : Formats an SSE message frame.
- `public function headers()` : Returns standard HTTP headers for SSE streaming connection.

`euphoria/debug/debug.e` (Namespace: `debug`)

- **Purpose:** Interactive VM debugger: call stack inspection, breakpoint placement, single-step execution (step over/into), and symbol queries.
- **Public Functions & Procedures (23):**
- `public function call_stack()` : Returns information about the call stack of the code currently running.
- `public procedure initialize_debugger( atom init_ptr )` : Initializes an external debugger.
- `public procedure set_debug_rid( DEBUG_ROUTINE rtn, integer rid )` : Used to initialize the external debuggers handlers.
- `public function read_object( atom sym )` : Reads a Euphoria object directly from a symbol memory address.
- `public procedure trace_off()` : Disables execution trace for subsequent statements.
- `public procedure disable_trace()` : Permanently disables interactive tracing for current session.
- `public procedure step_over()` : Steps over the next instruction without entering subroutines.
- `public procedure abort_program()` : Terminates program execution gracefully from debugger.
- `public function get_current_line()` : Retrieves currently executing line number.
- `public function symbol_lookup( sequence name, integer line = get_current_line(), atom pc = get_pc() )` : Finds symbol by name in local or global symbol table.
- `public function get_pc()` : Returns the VM program counter.
- `public function is_novalue( atom sym_ptr )` : Checks if a symbol is uninitialized (NOVALUE state).
- `public function debugger_call_stack()` : Returns the active call stack frames.
- `public function break_routine( atom routine_sym, integer enable )` : Sets a breakpoint at entry point of specified routine ID.
- `public function get_name( atom sym )` : Returns identifier name of a symbol.
- `public function get_source( integer line )` : Retrieves source text for a file and line number.
- `public function get_file_no( integer line )` : Returns file index where symbol was declared.

- `public function get_file_name( integer file_no )` : Returns source file path of a registered file index.
- `public function get_file_line( integer line )` : Returns line number of symbol declaration.
- `public function get_next( atom sym )` : Retrieves next symbol in linked symbol table chain.
- `public function is_variable( atom sym_ptr )` : Checks whether symbol is a variable (not constant or routine).
- `public function get_parameter_syms( atom rtn_sym )` : Returns list of symbol pointers for routine formal parameters.
- `public function get_symbol_table()` : Returns root pointer of the VM symbol table.

`euphoria/info.e` (Namespace: `info`)

- **Purpose:** Compiler and interpreter runtime metadata: VM version numbers, patch release, target platform name, and machine bit-width.
- **Public Functions & Procedures (17):**
- `public function platform_name()` : Get the platform name.
- `public function arch_bits()` : Get the native architecture word size.
- `public function version()` : Get the version, as an integer, of the host Euphoria.
- `public function version_major()` : Get the major version of the host Euphoria.
- `public function version_minor()` : Get the minor version of the hosting Euphoria.
- `public function version_patch()` : Get the patch version of the hosting Euphoria.
- `public function version_node(integer full = 0, integer n = 12)` : Get the source code node id of the hosting Euphoria.
- `public function version_revision()` : Get the source code revision of the hosting Euphoria.
- `public function version_date(integer full = 0)` : Get the compilation date of the hosting Euphoria.
- `public function version_type()` : Get the type version of the hosting Euphoria.
- `public function version_string(integer full = 0)` : Get a normal version string.
- `public function version_string_short()` : Get a short version string.
- `public function version_string_long(integer full = 0)` : Get a long version string.
- `public function euphoria_copyright()` : Get the copyright statement for Euphoria.
- `public function pcre_copyright()` : Get the copyright statement for PCRE.
- `public function all_copyrights()` : Get all copyrights associated with this version of Euphoria.
- `public function start_time()` : Euphoria start time. This time represents the time Euphoria itself started. This time is recorded before any of the users code is opened, parsed or executed. It can provide accurate timing information as to how long it takes for your application to go from start time to usable time.

`euphoria/keywords.e` (Namespace: `keywords`)

- **Purpose:** Constant tables of all reserved OpenEuphoria keywords and built-in routine names recognized by the parser.
- **Notes & Key Constants:** Defines sequence constants `keywords` (all reserved language keywords like `if`, `while`, `function`) and `builtins` (all built-in routines).

`euphoria/stddebug.e` (Namespace: `stddebug`)

- **Purpose:** Comprehensive debug aggregator: includes all standard library modules to make all routines available during interactive debugging.
- **Notes & Key Constants:** Aggregator include file that imports all standard library modules so all symbols are accessible during an interactive debugging session.

`euphoria/symstruct.e` (Namespace: `symstruct`)

- **Purpose:** Low-level memory offsets and structural constants of the Euphoria virtual machine symbol table.
- **Notes & Key Constants:** Defines memory offset constants (`ST_OBJ`, `ST_NEXT`, `ST_MODE`, `ST_SCOPE`, `ST_NAME`, etc.) for internal symbol table structures.

`euphoria/syncolor.e` (Namespace: `syncolor`)

- **Purpose:** Syntax highlighting engine: lexical coloring and HTML/ANSI markup generation for Euphoria source code.
- **Public Functions & Procedures (6):**
- `public procedure set_colors(sequence pColorList)` : ****.
- `public procedure init_class()` : Initializes syntax token colors and style classes.
- `public function new()` : Create a new colorizer state.
- `public procedure reset(atom state = g_state)` : Resets syntax colorizer state to defaults.
- `public procedure keep_newlines(integer val = 1, atom state = g_state)` : Configures whether newlines are preserved in highlighted output.
- `public function SyntaxColor(sequence pline, atom state=g_state, multiline_token multi = 0)` : Parse Euphoria code into tokens of like colors.

`euphoria/tokenize.e` (Namespace: `tokenize`)

- **Purpose:** Full-featured Euphoria source code lexer and tokenizer: breaks source text into tokens with line, column, and type information.
- **Public Functions & Procedures (15):**
- `public function error_string(integer err)` : Get an error message string for a given error code.
- `public function new()` : Create a new tokenizer state.
- `public procedure reset(atom state = g_state)` : Reset the state to begin parsing a new file.
- `public procedure keep_builtins(integer val = 1, atom state = g_state)` : Specify whether to identify builtins specially or not default is FALSE.
- `public procedure keep_keywords(integer val = 1, atom state = g_state)` : Specify whether to identify keywords specially or not default is TRUE.
- `public procedure keep_whitespace(integer val = 1, atom state = g_state)` : Return white space (other than newlines) as tokens. default is FALSE.
- `public procedure keep_newlines(integer val = 1, atom state = g_state)` : Return new lines as tokens. default is FALSE.
- `public procedure keep_comments(integer val = 1, atom state = g_state)` : Return comments as tokens default is FALSE.
- `public procedure return_literal_string(integer val = 1, atom state = g_state)` : When returning string tokens, we have the option to process them and return their value, or to return the literal text that made up the original string. Right now, this option only affects the processing of hex strings. default is FALSE - process the string and return its value.
- `public procedure string_strip_quotes(integer val = 1, atom state = g_state)` : When returning string tokens, we have the option to strip the quotes. default is TRUE.
- `public procedure string_numbers(integer val = 1, atom state = g_state)` : Return TDATA for all T_NUMBER tokens in "string" format. Defaults: *T_NUMBER tokens return atoms* T_CHAR tokens return single integer chars *T_EOF tokens return undefined data* Other tokens return strings.
- `public function last_multiline_token()` : If the line did not end in the middle of a multiline construct, or the value for the respective token. This is meant to facilitate proper tokenizing of individual lines of code.
- `public function tokenize_string(sequence code, atom state = g_state, integer stop_on_error = TRUE, multiline_token multi = 0)` : Tokenize euphoria source code.
- `public function tokenize_file(sequence fname, atom state = g_state, integer mode = io:BINARY_MODE )` : Tokenize euphoria source code.
- `public procedure show_tokens(integer fh, sequence tokens)` : Print token names and data for each token in `tokens` to the file handle `fh`.

2.8.14 Artificial Intelligence, MCP, LLM & RAG

`std/mcp.e` (Namespace: `mcp`)

- **Purpose:** Model Context Protocol (MCP, specification 2024-11-05): Exposes Euphorix programs as MCP servers to modern AI assistants (Claude Desktop, Antigravity, Cursor) or controls external tools as an autonomous MCP client over `stdio` (subprocesses) or HTTP with Server-Sent Events (SSE).
- **Public Functions & Procedures (10):**
- `public function server_create(sequence name, sequence version, sequence capabilities = {})` : Creates a new MCP server instance with name, version, and declared capabilities.
- `public function server_register_tool(sequence srv, sequence name, sequence description, sequence parameters, integer routine_id)` : Registers an executable tool with JSON Schema parameters and handler callback.
- `public procedure server_run_stdio(sequence srv)` : Runs the server in `stdio` mode (standard input/output) for integration with AI desktop clients and CLI agents.
- `public procedure server_run_http(sequence srv, integer port = 8085)` : Runs the MCP server as a lightweight HTTP/SSE service on the specified port.
- `public function client_stdio(sequence command_line)` : Spawns an MCP server subprocess over `stdio` and returns `{1, client}`.
- `public function client_http(sequence base_url)` : Connects an MCP client to a remote MCP server via HTTP/SSE.
- `public function client_init(sequence cli, sequence client_name = "Euphorix-Client", sequence client_version = "1.0.0")` : Performs protocol handshake (`initialize` and `notifications/initialized`).
- `public function client_list_tools(sequence cli)` : Retrieves the list of available tools from the server (`{1, tools_list, updated_cli}`).
- `public function client_call_tool(sequence cli, sequence name, sequence arguments = {})` : Executes a registered tool on the server and returns the text result (`{1, result_text, raw_json, updated_cli}`).
- `public procedure client_close(sequence cli)` : Closes the connection to the MCP server and cleanly terminates subprocess streams.

`std/llm.e` (Namespace: `llm`)

- **Purpose:** Universal client for Large Language Models (LLMs) and semantic vector embeddings. Supports 100% offline operation (deterministic mock engine), local Ollama (`/api/chat` , `/api/embeddings`), OpenAI (`gpt-4o-mini` , `/chat/completions`), and Anthropic Claude (`/v1/messages`).
- **Public Functions & Procedures (5):**

- `public function client(integer provider, sequence options = {})` : Initializes an LLM client for `PROVIDER_MOCK`, `PROVIDER_OLLAMA`, `PROVIDER_OPENAI`, or `PROVIDER_ANTHROPIC`.
- `public function complete(sequence cli, sequence prompt, sequence options = {})` : Sends a prompt to the model and returns the generated text (`{1, text, raw}`).
- `public function chat(sequence cli, sequence messages, sequence options = {})` : Conducts a multi-turn chat conversation (`{role, content}`) with customizable `temperature` and `max_tokens`.
- `public function embed(sequence cli, sequence text_str, sequence options = {})` : Computes a normalized vector (64-dim TF-IDF feature vector in mock mode or high-dimensional dense embeddings via Ollama/OpenAI).
- `public function cosine_similarity(sequence vec_a, sequence vec_b)` : Computes the cosine similarity between two vectors from 0.0 (orthogonal) to 1.0 (identical).

`std/rag.e` (Namespace: `rag`)

- **Purpose:** Retrieval-Augmented Generation (RAG) engine: Hybrid search over documents and enterprise handbooks using SQLite Full-Text Search (FTS5 / BM25) and Cosine vector similarity, augmenting LLM prompts with relevant context chunks and producing verifiable citations.
- **Public Functions & Procedures (10):**
- `public function create_kb(sequence name, sequence llm_client, sequence db_path = ":memory:")` : Creates a new RAG knowledge base with SQLite FTS5 tables and vector storage.
- `public procedure close_kb(sequence kb)` : Closes the knowledge base and releases all SQLite resources.
- `public function chunk_text(sequence text_str, integer target_size = 500, integer overlap = 100)` : Chunks text intelligently across paragraph and sentence boundaries with configurable overlap.
- `public function add_document(sequence kb, sequence doc_id, sequence title, sequence content, sequence metadata = {})` : Chunks a document, generates embeddings, and indexes it into SQLite FTS5.
- `public function add_text(sequence kb, sequence doc_id, sequence title, sequence chunk_text, integer chunk_index, sequence metadata = {})` : Inserts a single pre-processed text chunk into the knowledge base.
- `public function retrieve(sequence kb, sequence query, integer top_k = 3, sequence mode = "hybrid")` : Executes hybrid retrieval (BM25 + Cosine similarity via Reciprocal Rank Fusion) and returns top chunks.
- `public function ask(sequence kb, sequence query, sequence options = {})` : Executes the full RAG pipeline (retrieval, prompt augmentation, LLM generation, and source attribution).
- `public function answer(sequence rag_result)` : Extracts the generated text from an `ask()` result.
- `public function sources(sequence rag_result)` : Extracts the list of cited sources with title, chunk index, and relevance score.
- `public function format_sources(sequence rag_result)` : Formats citations into a clean, human-readable list.

---

3. Java Interoperability & JAR Integration (`std/java.e`)

The `std/java.e` module provides bidirectional access to the Java 25 ecosystem and external JAR files.

3.1 Loading External JAR Files (`java:load_jar`)

```
include std/java.e as java

integer ok = java:load_jar("lib/my_custom_library.jar")
if not ok then
    puts(2, "Error: Could not load JAR file!\n")
end if
```

3.2 Class Resolution & Object Instantiation

- **`java:find_class(sequence class_name) -> atom`**

Resolves class handle (`Class<?>`) by fully-qualified name:

```
atom cls_calc = java:find_class("com.example.Calculator")
```

- **`java:new_object(object class_or_name, sequence args = {}) -> atom`**

Instantiates an object using reflection and constructor resolution:

```
atom sb = java:new_object("java.lang.StringBuilder", {"Initial text: "})
atom list = java:new_object("java.util.ArrayList", {})
```

3.3 Method Invocation (Static & Instance)

- **`java:call_method(atom handle, sequence method_name, sequence args = {}) -> object`**

Calls an instance method:

```
java:call_method(sb, "append", {123})
java:call_method(sb, "append", {" EUR"})
sequence text = java:to_string(sb)   -- "Initial text: 123 EUR"
```

- **`java:call_static(object class_or_name, sequence method_name, sequence args = {}) -> object`**

Calls a static method:

```
atom now = java:call_static("java.lang.System", "currentTimeMillis", {})
atom root = java:call_static("java.lang.Math", "sqrt", {144.0})   -- 12.0
atom max_val = java:call_static("java.lang.Math", "max", {10, 25}) -- 25
```

3.4 Reading and Writing Fields

- **`java:get_field(atom obj_or_class, sequence field_name) -> object`**
- **`java:set_field(atom obj_or_class, sequence field_name, object value)`**

```
atom max_int = java:get_field("java.lang.Integer", "MAX_VALUE")
atom pi = java:get_field("java.lang.Math", "PI")
```

3.5 Automatic Type Matching & Conversions

The interop bridge scores available overloads and translates types automatically:

Euphoria Type	Java Types
`integer`	`int`, `Integer`, `long`, `Long`, `short`, `byte`, `boolean` (`!= 0`)
`atom` (Float)	`double`, `Double`, `float`, `Float`
`sequence` (String)	`java.lang.String`, `java.lang.CharSequence` (UTF-8)
`sequence` (Numbers)	`byte[]`, `int[]`, `long[]`
`sequence` (Objects)	`java.util.List<?>`, `java.lang.Object[]`
Handle (`atom > 0`)	Java instance reference
`0`	`null`

3.6 Event Listeners & Callbacks (`java:create_listener`)

Using **`java:create_listener(sequence interface_name, integer routine_id)`**, the runtime creates a dynamic `java.lang.reflect.Proxy` directing Java events back to an OpenEuphoria routine:

```
procedure on_button_click(atom sender)
    puts(1, "Button clicked!\n")
end procedure

atom listener = java:create_listener("java.awt.event.ActionListener", routine_id("on_button_click"))
java:call_method(btn, "addActionListener", {listener})
```

Identity preservation: The `sender` parameter in callbacks matches the exact atom handle of the triggering widget (`sender = btn`).

3.7 Subprocess Streaming Pipes with Virtual Threads

Subprocesses execute asynchronously via virtual threads:

```
atom pipe = java:pipe_open("uname -a", java:PIPE_READ)
sequence line = java:pipe_read_line(pipe)
puts(1, "Kernel: " & line & "\n")
java:pipe_close(pipe)
```

---

4. Graphical User Interfaces with Swing & AWT (`std/gui.e`)

The `std/gui.e` module provides a declarative API for developing desktop GUI applications under Linux.

4.1 Linux Desktop Integration & Theme Selection (`gui:set_theme`)

GUI initialization (`gui:init_gui()`) defaults to the native **GTK+ Look and Feel** (`com.sun.java.swing.plaf.gtk.GTKLookAndFeel`):

- Buttons, inputs, checkboxes, and scrollbars match the user's desktop environment (GNOME, Ubuntu, XFCE).
- Use `gui:set_theme(sequence theme_name)` to switch designs:
- `"gtk"`: Native Linux desktop theme (default)
- `"flatlaf"` or `"flatlaf-light"`: Modern flat light design
- `"flatlaf-dark"` or `"dark"`: Modern dark mode
- `"nimbus"`: Clean vector UI
- `"system"`: Host system default

```
include std/gui.e as gui

-- Switch to modern dark mode before creating widgets:
gui:set_theme("flatlaf-dark")

atom win = gui:window("Dark Mode Application", 600, 400)
```

4.2 Typography & Subpixel LCD Anti-Aliasing

Enabled through JVM parameters `-Dawt.useSystemAAFontSettings=lcd` and `-Dswing.aatext=true`:

- **Default Font**: `Ubuntu` (or `SansSerif`) at 13pt.
- **Customization**: `gui:set_font(widget, "Ubuntu", 16, 1)` configures family, size, and bold style.

4.3 Window Management (`window`, `show`, `wait_window`)

```
atom win = gui:window("My Application", 600, 400, gui:CLOSE_ON_EXIT)
gui:show(win)
gui:wait_window(win)
```

Close actions:

- `gui:CLOSE_ON_EXIT` (Default): Terminates application when window is closed.
- `gui:DISPOSE_ON_CLOSE`: Closes window and releases graphical resources.
- `gui:HIDE_ON_CLOSE`: Hides window without disposal.

4.4 Layout Managers with Gap Control (`hgap` / `vgap`)

- `gui:BORDER_LAYOUT`: Divides container into 5 zones (`NORTH`, `SOUTH`, `EAST`, `WEST`, `CENTER`).
- `gui:FLOW_LAYOUT`: Horizontal centered arrangement.
- `gui:GRID_LAYOUT`: Tabular grid with rows, columns, and spacing.

```
-- 5-row, 2-column form grid with 14px horizontal and 12px vertical spacing:
atom pnl = gui:panel(gui:GRID_LAYOUT, 5, 2, 14, 12)
```

4.5 Widget Component Reference

Function	Description
`gui:text_label(sequence text, integer align = ALIGN_LEFT) -> atom`	Label (`ALIGN_LEFT` , `ALIGN_CENTER` , `ALIGN_RIGHT`).
`gui:text_field(sequence text = "", integer columns = 20) -> atom`	Single-line text input.
`gui:text_area(sequence text = "", integer rows = 5, integer cols = 20) -> atom`	Multi-line text area.
`gui:scroll_pane(atom client_widget) -> atom`	Scrolling wrapper.
`gui:button(sequence text, integer on_click_rid = 0) -> atom`	Pushbutton with click callback.
`gui:check_box(sequence text, integer checked = 0) -> atom`	Checkbox.
`gui:radio_button(sequence text, integer selected = 0) -> atom`	Radio button.
`gui:combo_box(sequence items = {}) -> atom`	Dropdown selection box.
`gui:progress_bar(integer min_val = 0, integer max_val = 100) -> atom`	Progress bar with text display.
`gui:slider(integer min_val = 0, integer max_val = 100, integer init_val = 50) -> atom`	Slider control.
`gui:panel(integer layout_type = FLOW_LAYOUT, ...)`	Container panel.

4.6 Properties, Styling & Padding

- `gui:set_text(widget, text)` / `gui:get_text(widget) -> sequence`
- `gui:set_value(widget, val)` / `gui:get_value(widget) -> integer`
- `gui:is_selected(widget) -> integer` / `gui:set_selected(widget, selected)`
- `gui:set_padding(widget, top, left, bottom, right)`
- `gui:set_font(widget, name = "Ubuntu", size = 13, is_bold = 0)`
- `gui:set_enabled(widget, enabled)`

4.7 Standard Dialogs (Message, Input, Confirm, FileDialog)

- `gui:msgbox(title, message, icon_type = MSG_INFO)` : Dialog (`MSG_INFO` , `MSG_WARN` , `MSG_ERROR` , `MSG_QUESTION`).
- `gui:input_box(title, prompt, default_val = "") -> sequence` : Text input dialog.
- `gui:confirm_box(title, message) -> integer` : Returns `1` for Yes, `0` for No.
- `gui:file_dialog(title = "Select File", is_save = 0, start_dir = ".") -> sequence` : File dialog returning path or `""` .

4.8 Complete Demo Application (`demo/gui_demo.ex`)

```
include std/gui.e as gui
include std/text.e

atom win = gui:window("OpenEuphoria 4.1 GUI Demo (Java 25 Swing & GTK)", 620, 460)
gui:set_layout(win, gui:BORDER_LAYOUT)
gui:set_padding(win, 12, 14, 12, 14)

-- Header
atom pnl_top = gui:panel(gui:GRID_LAYOUT, 2, 1, 0, 6)
atom lbl_title = gui:text_label("OpenEuphoria GUI: Native Linux Desktop", gui:ALIGN_CENTER)
gui:set_font(lbl_title, "Ubuntu", 17, 1)
atom lbl_sub = gui:text_label("100% UTF-8 Support for Umlauts (ä, ö, ü, ß) & Modern Fonts", gui:ALIGN_CENTER)
gui:set_font(lbl_sub, "Ubuntu", 12, 0)
gui:add(pnl_top, lbl_title)
gui:add(pnl_top, lbl_sub)
gui:add(win, pnl_top, gui:NORTH)

-- Form panel
atom pnl_center = gui:panel(gui:GRID_LAYOUT, 5, 2, 14, 12)
atom lbl_name = gui:text_label("User Name:")
atom txt_name = gui:text_field("Euphoria Developer", 20)
atom lbl_lang = gui:text_label("Target Platform:")
atom cmb_lang = gui:combo_box({"Linux 64-Bit (Max Performance)", "Ubuntu / Debian", "Arch / Fedora"})
atom lbl_mode = gui:text_label("Preferences:")
atom cb_expert = gui:check_box("Enable automatic validation", 1)
atom lbl_pbar = gui:text_label("Progress:")
atom pbar = gui:progress_bar(0, 100)
gui:set_value(pbar, 65)
atom lbl_status_title = gui:text_label("Status:")
atom lbl_status = gui:text_label("Ready for actions")
gui:set_font(lbl_status, "Ubuntu", 12, 1)

gui:add(pnl_center, lbl_name)
gui:add(pnl_center, txt_name)
gui:add(pnl_center, lbl_lang)
gui:add(pnl_center, cmb_lang)
gui:add(pnl_center, lbl_mode)
gui:add(pnl_center, cb_expert)
gui:add(pnl_center, lbl_pbar)
gui:add(pnl_center, pbar)
gui:add(pnl_center, lbl_status_title)
gui:add(pnl_center, lbl_status)
gui:add(win, pnl_center, gui:CENTER)

-- Callbacks
integer count = 0
procedure on_submit(atom sender)
    count += 1
    sequence name = gui:get_text(txt_name)
    gui:set_text(lbl_status, sprintf("Action #%d executed for '%s'", {count, name}))
    gui:msgbox("Confirmation",
        sprintf("Hello %s!\n\nAction processed in Swing event thread!\nUmlauts test: ä ö ü Ä Ö Ü ß", {name}),
        gui:MSG_INFO)
end procedure

procedure on_file_pick(atom sender)
    sequence path = gui:file_dialog("Select File", 0, ".")
    if length(path) > 0 then
        gui:set_text(lbl_status, "Selected: " & path)
    end if
end procedure

procedure on_quit(atom sender)
    gui:close_window(win)
end procedure

-- Buttons
atom pnl_bottom = gui:panel(gui:FLOW_LAYOUT)
atom btn_submit = gui:button("Execute Action", routine_id("on_submit"))
atom btn_file   = gui:button("Select File...", routine_id("on_file_pick"))
atom btn_quit   = gui:button("Close",          routine_id("on_quit"))
gui:add(pnl_bottom, btn_submit)
gui:add(pnl_bottom, btn_file)
gui:add(pnl_bottom, btn_quit)
gui:add(win, pnl_bottom, gui:SOUTH)
```

```
gui:show(win)
gui:wait_window(win)
```

5. UTF-8 String Processing (`std/utf8.e` & VM Engine)

5.1 Comprehensive UTF-8 Integration in Java 25 Interpreter

1. **Source String Literals**:

Source files read as UTF-8. Non-ASCII characters (`c > 127` , e.g. `ä` , `ö` , `ü` , `ß` , `€`) encode as UTF-8 multi-byte sequences in the constant pool (`"ü"` $\rightarrow$ `{195, 188}`).

1. **Robust Decoding in `Value.toUtf8String()`** :

When passed to Java (Swing, JDBC, HTTP, JSON), byte sequences decode losslessly. Malformed bytes fall back to ISO-8859-1, preventing replacement character corruption (`\uFFFD`).

5.2 Exact Unicode Operations (`std/utf8.e`)

- `utf8:len(s)` : Total Unicode codepoints.
- `utf8:slice(s, start_cp, end_cp)` : Slices by character position rather than byte index.
- `utf8:upper(s)` / `utf8:lower(s)` : Full Unicode case mapping.
- `utf8:to_codepoints(s)` / `utf8:from_codepoints(cps)` : Translates to and from 32-bit codepoint integers.

---

6. JSON & RFC 9535 JSONPath (`std/json.e`)

6.1 JSON Parsing & Serialization (`std/json.e`)

Powered by **Jackson Databind**:

- High-speed parsing and serialization (`json:parse` , `json:stringify` , `json:pretty`).
- Seamless interoperability with database query results (`sqlite:query_assoc` , `pgsql:query_assoc`).

6.2 RFC 9535 JSONPath Engine with Path Queries

- **RFC 9535 JSONPath**: Evaluates complex path expressions with filters:

```
sequence r = json:path_query(doc, "$.store.book[?(@.price < 10)].title")
```

7. JSON-Schema Draft 2020-12 (`std/json_schema.e`)

7.1 Declarative Schema Validation (Draft 2020-12)

Validates structures against **Draft 2020-12**:

- `json_schema:validate(data, schema) -> sequence`: Returns `{valid, error_list}`.
- `json_schema:is_valid(data, schema) -> integer`: Boolean validation (`1` or `0`).

8. Document & Tabular Formats: XML, CSV, Excel & PDF (`std/xml.e`, `std/csv.e`, `std/excel.e`, `std/pdf_report.e`)

Euphorix includes robust built-in modules for creating, processing, and exchanging structured documents, spreadsheets, and business reports completely autonomously without external libraries.

8.1 XML DOM, W3C XPath 1.0 & XSD Validation (`std/xml.e`)

Powered by JDK XML modules (`javax.xml.parsers`, `javax.xml.xpath`, `javax.xml.validation`):

- **DOM**: `xml:parse(xml_str)`, `xml:stringify(node)`, `xml:element()`, `xml:tag()`, `xml:text()`, `xml:attr()`.
- **W3C XPath 1.0**: `xml:xpath(doc, "//catalog/book[@id='B1']/title")` or scalar aggregations `xml:xpath(doc, "sum(//price)")`.
- **XSD Schema Validation**: `xml:validate_xsd(doc, xsd_str)`.

8.2 RFC 4180 High-Performance CSV Engine (`std/csv.e`)

The `std/csv.e` module ensures strict compliance with RFC 4180 while maximizing I/O performance:

- `csv:write_file(sequence filepath, sequence headers, sequence rows, integer delimiter = ',', integer quote_all = 0)`: Writes tabular data to a CSV file.
- `csv:read_file(sequence filepath, integer delimiter = ',', integer has_header = 1)`: Reads a CSV file and returns `{headers, rows}`.
- `csv:to_csv(sequence headers, sequence rows, integer delimiter = ',', integer quote_all = 0)`: Converts in-memory sequences into a standard CSV string.
- `csv:parse(sequence csv_text, integer delimiter = ',')`: Parses CSV content into nested sequences of rows and fields.
- `csv:format_field(object val, integer delimiter = ',', integer quote_all = 0)`: Formats an individual field value with automatic quote escaping (`"` to `""`).

Supports configurable delimiters (comma `,`, semicolon `;`, tab `\t` for TSV, pipe `|`) and multi-line quoted values.

8.3 Native Microsoft Excel OpenXML Engine (`std/excel.e`)

The `std/excel.e` module implements a native OpenXML engine (OOXML / `.xlsx`) that writes and reads modern workbook files directly on the Java 25 VM runtime. No external software or third-party dependencies (such as Apache POI) are required. The generated spreadsheets open cleanly in Microsoft Excel (2007–2024 / 365), LibreOffice Calc, Apple Numbers, and Google Sheets.

- `excel:write_file(sequence filepath, sequence headers, sequence rows, sequence sheet_name = "Sheet1")`: Writes a complete, formatted `.xlsx` workbook.
- Automatically distinguishes integers, floating-point numbers, and UTF-8 strings.
- Automatically formats header rows with corporate blue styling (`#365F91`), white bold typography, and centered alignment.
- Assembles the complete Open Packaging Convention (OPC) package (`[Content_Types].xml`, `_rels/.rels`, `xl/workbook.xml`, `xl/styles.xml`, `xl/worksheets/sheet1.xml`).
- `excel:read_file(sequence filepath)`: Reads an existing `.xlsx` file, extracts shared strings, and returns `{status, rows}` where each row contains cell values.
- `excel:export_table(filepath, headers, rows, sheet_name)`: Convenience function for database queries and reports.

```
include std/excel.e as excel

sequence headers = {"CustomerID", "Company", "City", "Revenue"}
sequence rows = {
    {1001, "Mueller Engineering", "Munich", 84500.50},
    {1002, "Schmidt IT Consulting", "Berlin", 42300.00}
}
excel:write_file("customers.xlsx", headers, rows, "CustomerList")
```

8.4 Standard PDF 1.4 Business Report Generator (`std/pdf_report.e`)

`std/pdf_report.e` generates publication-ready PDF business reports complying with the standard PDF 1.4 specification directly from Euphoria sequences:

- `pdf_report:create_report(filepath, title, metadata, headers, rows, options = {})`:
- **Header Banner:** Polished color title bar displaying document title and generation timestamp.
- **Metadata Card:** Key-value summary cards (e.g. Account, Period, Status).
- **Data Table:** Dynamic column width calculation, dark headers, and alternating zebra striping.

- **Number Formatting:** Right-aligned numerical columns for financial accuracy.
- **Pagination:** Automatic footer with page numbering ("Page X of Y").
- `pdf_report:export_table(filepath, title, headers, rows)` : Quick tabular export without metadata cards.

A complete runnable example demonstrating CSV, Excel, and PDF reporting is available in examples/09_excel_und_csv_demo.ex.

---

9. SQL Databases: SQLite 3 (`std/sqlite.e`) & PostgreSQL (`std/pgsql.e`)

9.1 Embedded SQLite 3 Database (`std/sqlite.e`)

Embedded SQLite engine accessed via JDBC supporting file databases and `:memory:` mode.

9.2 Enterprise PostgreSQL Database (`std/pgsql.e`)

Enterprise-grade PostgreSQL connectivity with connection pooling, transactions, and SSL.

Both modules share an **identical unified API** via JDBC:

Function	SQLite 3 (`std/sqlite.e`)	PostgreSQL (`std/pgsql.e`)
Connect	`sqlite:open("data.db")`	`pgsql:open("postgresql://user:pass@host:5432/db")`
Close	`sqlite:close(db)`	`pgsql:close(db)`
DDL/DML	`sqlite:exec(db, sql, params)`	`pgsql:exec(db, sql, params)`
SELECT (Rows)	`sqlite:query(db, sql, params)`	`pgsql:query(db, sql, params)`
SELECT (JSON Objects)	`sqlite:query_assoc(db, sql, params)`	`pgsql:query_assoc(db, sql, params)`
Scalar Value	`sqlite:query_value(db, sql, params)`	`pgsql:query_value(db, sql, params)`
Transactions	`sqlite:begin`, `commit`, `rollback`	`pgsql:begin`, `commit`, `rollback`

10. Web Servers, Microservices, Security & Realtime (`std/http.e`, `std/auth.e`, `std/session.e`, `std/sse.e`)

Euphorix delivers a complete web service and microservice stack featuring an asynchronous client, embedded high-concurrency HTTP server, enterprise cryptographic tools, session management, and live event streaming.

10.1 HTTP & HTTPS Client (`std/http.e`)

Built on the modern Java 25 `HttpClient`:

- `http:get(url, headers = {})`, `http:post(url, body, headers = {})`, `http:post_json(url, json_data, headers = {})`, `http:put`, `http:delete_req`, `http:request`.
- Automatic TLS certificate validation, timeouts, custom headers, and transparent HTTP/2 multiplexing.

10.2 Embedded Web Server & REST Routing (`std/http.e`)

Lightweight, high-density HTTP server utilizing Java 25 Virtual Threads—each connection runs concurrently without thread pool exhaustion:

- `http:server_create(port)`: Creates a server instance on the specified port.
- `http:route(srv, method, path, handler_rid)`: Registers route handlers (`GET`, `POST`, `PUT`, `DELETE`, or `*`).
- `http:response_json(status, body)`: Convenience helper for sending formatted JSON responses.
- `http:server_background(srv)`: Runs the server asynchronously in the background.

10.3 Enterprise Security: PBKDF2 & JWT HS256 (`std/auth.e`)

For secure web APIs and account credentials, `std/auth.e` implements industry-standard cryptography:

- **Secure Password Hashing (PBKDF2)**:
- 65,536 iterations of HMAC-SHA256 with a 16-byte random salt (`SecureRandom`).
- `auth:hash_password(password)`: Generates formatted hashes (`pbkdf2:sha256:65536:<salt>:<hash>`).
- `auth:verify_password(password, hash_str)`: Verifies passwords in constant time (`MessageDigest.isEqual`), mitigating side-channel timing attacks.
- **JSON Web Tokens (JWT)**:
- RFC 7519 compliant tokens signed using HMAC-SHA256 (HS256).
- `auth:create_jwt(payload_json, secret_key, expiry_secs = 3600)`: Issues signed JWT tokens with automatic `exp` expiration claim.
- `auth:verify_jwt(token, secret_key)`: Validates cryptographic signature and token expiry. Returns `{1, payload_json}` or `{0, "error"}`.
- **Cryptographic Random Tokens**:
- `auth:random_token(byte_count = 32)`: Generates high-entropy hexadecimal tokens for CSRF mitigation and API authentication keys.

```
include std/auth.e as auth

sequence token = auth:create_jwt(`{"user":"admin","role":"root"}`, "secret", 3600)
sequence res = auth:verify_jwt(token, "secret")
```

10.4 HTTP Session & Cookie Management (`std/session.e`)

The `std/session.e` module manages stateful server sessions with configurable TTL (Time-To-Live):

- `session:start(cookie_header, cookie_name = "EUI_SESSION")`: Recovers an existing session ID or initializes a new session (`{sid, is_new}`).
- `session:set(sid, key, val)` & `session:get(sid, key, default_val)`: Stores and retrieves arbitrary Euphoria values.
- `session:destroy(sid)`: Terminates the session and frees memory.
- `session:cookie_header(sid, max_age, cookie_name, path, http_only)`: Produces standardized `Set-Cookie` response headers enforced with `HttpOnly`, `SameSite=Lax`, and `Path=/`.

A complete code example is available in examples/10_auth_jwt_und_session_demo.ex.

10.5 Realtime Server-Sent Events (SSE) Push (`std/sse.e`)

Server-Sent Events allow unidirectional streaming of telemetry, live updates, or real-time metrics directly to browser clients:

- `sse:event(event_name, data, id = "")`: Formats W3C-standard SSE messages (`event: ...\ndata: ...\nid: ...\n\n`).
- `sse:message(data)`: Plain data message frame.

- `sse:comment(text)` : Comment frame for keep-alive heartbeats to prevent proxy timeouts.
- `sse:headers()` : Streaming HTTP headers (`text/event-stream` , `no-cache` , `keep-alive`).

---

11. Dual-Stack Networking & Sockets (`std/socket.e` & `std/net/dns.e`)

11.1 Dual-Stack IPv4 & IPv6 Sockets (`std/socket.e`)

NIO socket channels supporting IPv4 and IPv6:

- `sockets:tcp_connect(host, port, timeout_ms)`
- `sockets:tcp_listen(port, bind_addr, backlog)`
- `sockets:tcp_accept(srv_sock, timeout_ms)`
- `sockets:send_all(sock, data)`
- `sockets:recv_line(sock)`

11.2 DNS Name Resolution & URLs (`std/net/dns.e`, `std/net/url.e`)

Host name resolution and RFC 3986 compliant URL parsing.

---

12. Demos, Examples & Execution

12.1 Overview of Practical Demos

Example applications reside in `demo/` and `examples/`:

12.2 Running Demonstration Programs

```
# 1. Run interactive Desktop GUI (Swing / FlatLaf)
./euphorix demo/gui_demo.ex

# 2. REST API Microservice with SQLite & Schema validation
./euphorix examples/05_rest_api_microservice.ex

# 3. Database integration tests (SQLite & PostgreSQL)
./euphorix examples/04_sqlite_datenbank.ex
./euphorix examples/06_postgresql_datenbank.ex

# 4. XML, XPath & XSD validation
./euphorix examples/03_xml_xpath_xsd.ex

# 5. SAP UI5 Web Components Cockpit
./euphorix examples/07_sapui5_webcomponents_demo.ex

# 6. Self-Contained Web-UI Suite (VS-UI) & DataGrid
./euphorix examples/08_vsui_cockpit_demo.ex
./euphorix demo/vsui_demo.ex

# 7. Excel (.xlsx), CSV & PDF Business Reporting
./euphorix examples/09_excel_und_csv_demo.ex

# 8. Enterprise Security: Auth, PBKDF2, JWT & HTTP Sessions
./euphorix examples/10_auth_jwt_und_session_demo.ex

# 9. VSUI: Form Validation, Drag & Drop Upload & SVG Charts
./euphorix examples/11_vsui_charts_und_upload_demo.ex

# 10. Background Task Scheduler (Cron) & Realtime SSE
./euphorix examples/12_cron_scheduler_demo.ex

# 11. Systemd Service Unit Generator for Linux Daemons
./euphorix tools/make_service.ex examples/05_rest_api_microservice.ex euphorix-api "REST API" volkix

# 12. Run full test suites (Java 25 Runner)
./tools/run_all_tests ./euphorix tests/t_gui.e tests/t_java_interop.e tests/t_concurrency.ex
```

13. Modern Concurrency: Virtual Threads, Async/Await & Channels (`std/thread.e`, `std/channel.e`, `std/task.e`, `std/cron.e`)

Java 25 Virtual Threads (Project Loom) eliminate the historical single-core bottleneck of OpenEuphoria. Blocking network, database, or disk I/O operations will never freeze the application. Three unified approaches cover all use cases:

13.1 Approach 1: Classic Tasks with Virtual Threads (`task:create_thread`)

The `std/task.e` module maintains 100% backward compatibility with classic cooperative multitasking, while offering `create_thread` to run tasks on preemptive, multi-core Java 25 Virtual Threads:

```
include std/task.e
include std/thread.e

procedure worker(sequence name)
    for i = 1 to 5 do
        thread:sleep(0.1) -- Does NOT block the main program!
        printf(1, "Task %s: Step %d\n", {name, i})
    end for
end procedure

-- Spawn as independent Virtual Threads on multi-core CPU:
atom t1 = task:create_thread(routine_id("worker"), {"A"})
atom t2 = task:create_thread(routine_id("worker"), {"B"})
```

13.2 Approach 2: Channels / Go-Style Concurrency (`std/channel.e`)

Thread-safe message pipes (CSP - Communicating Sequential Processes) enable lock-free data transfer without deadlocks or race conditions:

```
include std/thread.e
include std/channel.e

atom ch = channel:new(10) -- Buffered channel with capacity 10

procedure producer(atom out_ch)
    for i = 1 to 5 do
        channel:send(out_ch, sprintf("Packet #%d", {i}))
    end for
    channel:close(out_ch)
end procedure

thread:spawn(routine_id("producer"), {ch})

-- Consumer drains channel until closed:
while 1 do
    sequence msg = channel:receive(ch)
    if msg[1] != channel:OK then exit end if
    printf(1, "Received: %s\n", {msg[2]})
end while
```

13.3 Approach 3: Async / Await Futures (`std/thread.e`)

The most developer-friendly concurrency pattern: launch asynchronous computations in the background and retrieve results synchronously when needed:

```
include std/thread.e

function compute_heavy_task(atom x)
    thread:sleep(0.05)
    return x * x
end function

-- Run concurrently in the background:
atom job1 = thread:async(routine_id("compute_heavy_task"), {7})
atom job2 = thread:async(routine_id("compute_heavy_task"), {12})

-- Main thread continues execution (e.g., rendering UI or handling events):
if not thread:is_done(job1) then
    puts(1, "Job 1 is still processing in background...\n")
end if

-- Wait for and retrieve results:
atom res1 = thread:await(job1)
atom res2 = thread:await(job2)

-- Or await multiple jobs in parallel:
-- sequence all_results = thread:await_all({job1, job2})
```

13.4 Approach 4: Background Tasks & Cron Scheduler (`std/cron.e`)

With `std/cron.e`, recurring background jobs run inside the Euphorix process without requiring external operating system crontabs or background scripts. Tasks run on isolated Java 25 Virtual Threads and will never stall web requests or the main thread:

- `cron:schedule_interval(sequence name, atom interval_seconds, integer routine_id, sequence args = {})` : Executes a Euphoria procedure repeatedly every N seconds.
- `cron:schedule_daily(sequence name, integer hour, integer minute, integer routine_id, sequence args = {})` : Schedules a task at a specific time daily (e.g. daily backup at 02:30).
- `cron:start()` & `cron:stop()` : Starts or stops the scheduler thread cleanly.

```
include std/cron.e as cron
include std/thread.e as thread

procedure cleanup_temp()
    puts(1, "Temporary files deleted.\n")
end procedure

-- Run every 60 seconds
integer job_id = cron:schedule_interval("cleanup", 60, routine_id("cleanup_temp"))
cron:start()
```

A complete runnable example is provided in examples/12_cron_scheduler_demo.ex.

---

14. Web Applications with SAP UI5 Web Components 2.29 (`std/ui5.e`)

The `std/ui5.e` module brings standardized **SAP UI5 Web Components (targeting v2.29)** directly to the Euphorix ecosystem:

14.1 Architecture & Fiori Horizon Design

- **Zero Node.js / Zero Build Step**: Developers write pure OpenEuphoria syntax. No `package.json`, no Webpack/Vite or Angular/React needed.
- **SAP Fiori Themes**: Full support for `THEME_HORIZON` (*Morning Horizon*), `THEME_HORIZON_DARK` (*Evening Horizon*), and `THEME_FIORI_3`.
- **Standard HTML5 Web Components**: Standard elements including `<ui5-shellbar>`, `<ui5-card>`, `<ui5-table>`, `<ui5-button>`, `<ui5-input>`, `<ui5-dialog>`, `<ui5-toast>`, `<ui5-badge>`, `<ui5-datepicker>`, `<ui5-switch>`, etc.
- **Offline Fallback**: Embedded CSS styling ensures clean typography and layout rendering even without an active internet connection.

14.2 Component Suite & Declaration

```
include std/ui5.e as ui5

-- ShellBar (header with avatar, search, notifications)
sequence sb = ui5:shellbar("Enterprise Cloud", "ERP Cockpit", {{"profile", "VS"}})

-- Fiori Card
sequence crd = ui5:card("Open Tasks", "<p>3 Urgent Activities</p>", {
    {"subtitle", "Status: Active"},
    {"icon", "order-status"}
})

-- Interactive Data Table
sequence tbl = ui5:table(
    {{"ID", "80px"}, {"Title", ""}, {"Status", "120px"}},
    {
        {"#1", "Server Maintenance", ui5:badge("Done", 8)},
        {"#2", "DB Backup", ui5:badge("Urgent", 2)}
    }
)

-- Form Inputs & Buttons
sequence inp = ui5:input("task_title", "", {{"placeholder", "New task..."}, {"icon", "edit"}})
sequence btn = ui5:button("btn_add", "Add Task", {{"design", ui5:DESIGN_EMPHASIZED}, {"icon", "add"}, {"action", "add_item"}}
```

14.3 Reactive Server Bridge & Live DOM Updates (`/__ui5_action`)

User clicks and form changes are automatically forwarded to server-side `routine_id` handlers. The server responds with targeted DOM patches without a page reload:

```
function on_add_item(sequence ctx)
    sequence title = ui5:val(ctx, "task_title")
    if length(title) = 0 then
        return ui5:updates({
            ui5:set_value_state("task_title", ui5:STATE_ERROR, "Title is required!"),
            ui5:toast_show("app_toast", "Validation error!")
        })
    end if

    -- Persist in SQLite / PostgreSQL...
    sqlite:exec(db, "INSERT INTO items (name) VALUES (?);", {title})

    -- Dispatch reactive DOM updates back to browser:
    return ui5:updates({
        ui5:patch("table_container", render_table()),
        ui5:set_val("task_title", ""),
        ui5:toast_show("app_toast", "Item saved successfully!")
    })
end function
```

14.4 Complete Application Example (`examples/07_sapui5_webcomponents_demo.ex`)

A complete runnable example of an Enterprise Cockpit with SQLite integration is located in examples/07_sapui5_webcomponents_demo.ex. Run it directly via:

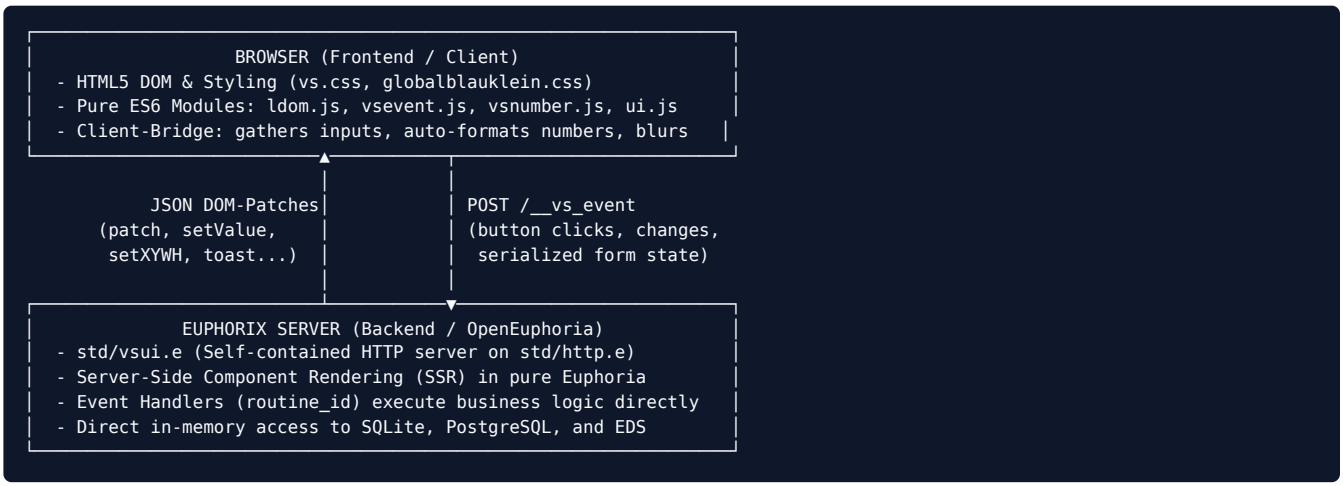
```
./euphorix examples/07_sapui5_webcomponents_demo.ex
```

15. Self-Contained High-Density Web-UI Suite (`std/vsui.e`)

As a lightweight, self-contained alternative to SAP UI5 Web Components, Euphorix integrates an enterprise-ready web framework directly inside the project repository under `html/` .

15.1 Architecture & Philosophy: The Fusion of HTMX and Vaadin

`std/vsui.e` provides an enterprise web development paradigm that unifies the best aspects of two modern, acclaimed architectures:



What It Shares with HTMX:

- **Hypermedia / Server-Driven UI**: The backend serves real HTML markup and maintains authority over state and business rules.
- **Zero SPA Build Overhead**: Eliminates Node.js, npm, Webpack, Vite, React, Vue, and bloated bundle files. The browser loads pure HTML5 and native ES6 modules.
- **Flicker-Free Partial DOM Updates**: Form submissions and button clicks do not trigger full page reloads (`F5`). Instead, targeted JSON patches (`vsui:updates`) surgically swap only affected DOM nodes (`vsui:patch` , `vsui:set_val`).

What It Shares with Vaadin:

- **Backend-Centric Component Model**: The entire user interface is constructed in the developer's core language (pure OpenEuphoria instead of Java): `vsui:button()` , `vsui:grid()` , `vsui:tabs()` , `vsui:splitter()` , `vsui:numinput()` .
- **Zero REST API / DTO Glue Code**: Developers no longer write manual REST endpoints, serializable DTOs, JSON parsing pipelines, and frontend fetch services just to connect a button to a database action. Events invoke backend handlers (`routine_id`) directly.
- **Direct Database Proximity**: Event handlers execute on the server right alongside transactional databases (SQLite, PostgreSQL, EDS) and CSP channels, executing updates atomically and updating the screen instantly.

The Euphorix Enterprise Edge:

1. **High-Density Business Ergonomics**: Unlike consumer frameworks (Tailwind, Bootstrap) with wide whitespace, globalblauklein.css and vs.css provide compact, monospace-friendly layouts tailored for high-density ERP, CRM, inventory, and accounting workstations.
2. **Commercial Accounting Precision**: Localized number formatting (vsnumber.js & `vsnuminput.js`) with configurable decimal places (`nk`), German thousand separators (`50.000,00 €`), right-aligned monospace styling, and automatic blur-formatting.
3. **Pixel-Exact Positioning**: Native coordinate positioning (`setXYWH` , `opt_xywh(x, y, w, h)`) and automatic window/parent centering (`centerWinX` , `centerWinY` via ldom.js).
4. **100% Self-Contained & Offline-Ready**: All assets reside locally in `html/` . No external CDNs, no telemetry, and zero internet connectivity required.

15.2 The Complete 22-Component Suite (`html/lib2/components/`)

`std/vsui.e` provides constructors for every single component in the `lib2` component library:

Component	Euphorix Function	Description / Feature
Button	`vsui:button(id, text, opts)`	Native `.vs-button` with click-event forwarding & action dispatch
Input	`vsui:input(id, val, opts)`	Standard text input field `.vs-input`
Number Input	`vsui:numinput(id, val, nk, opts)`	Localized number input (`vs-number`) with `nk` decimal places and `vsnumber.js` formatting
Date Input	`vsui:dateinput(id, val, opts)`	HTML5 date selector `.vs-date` with centered text
Time Input	`vsui:timeinput(id, val, opts)`	HTML5 time selector `.vs-time`
Checkbox	`vsui:check(id, label, checked, opts)`	Checkbox with text label `.vs-check`
Textarea	`vsui:textarea(id, val, rows, opts)`	Multi-line text field `.vs-input`
Dropdown	`vsui:combo(id, items, selected, opts)`	Select dropdown `.vs-combo`
Label	`vsui:ui_label(text, for_id, opts)`	Field label `.vs-label`
Fieldset	`vsui:group(legend, content, opts)`	Grouping frame `.vs-group` with title
Panel	`vsui:panel(id, title, content, opts)`	Card/container `.vs-panel` with blue header
Collapse	`vsui:collapse(id, sections, open_first, opts)`	Collapsible sections (`<details>` / `<summary>`)
DataGrid	`vsui:grid(id, cols, rows, opts)`	Table component with **sticky header**, **client-side live search**, and **automatic mobile card-view**
Tabs	`vsui:tabs(id, tabs_list, opts)`	Tab navigation with dynamic switching and horizontal mobile swipe
Modal	`vsui:modal(id, title, content, footer, opts)`	Centered popup dialog with backdrop (fullscreen on mobile)
Splitter	`vsui:splitter(id, left, right, pct, opts)`	2-column container with customizable split ratio
Blocker	`vsui:blocker(id, zindex, opts)`	Full-page overlay blocker element (`div.blocker`) with dynamic z-index
Treeview	`vsui:treeview(id, nodes, opts)`	Hierarchical tree structure (`ul.vs-treeview`) with collapsible branch nodes (▼)
Context Menu	`vsui:with_contextmenu(html, items)`	Custom right-click context menu with action triggers and dividers
Tooltip	`vsui:with_tooltip(html, text)`	Floating tooltip hovering centered above target elements
File Upload	`vsui:upload(id, target_url, opts)`	Drag & Drop dropzone with progress tracking and file selection
SVG Charts	`vsui:chart(id, type, lbls, data, opts)`	Server-rendered responsive vector graphics (`bar` , `line` , `pie` , `doughnut`)

15.3 Integrated Asset Server (`/lib2/` , `/css/` , `/img/*`)

The integrated server in `std/vsui.e` automatically serves static assets directly from the repository's `html/` directory:

- `/css/globalblauklein.css` , `/css/vs.css` , and `/css/vs-mobile.css`
- `/lib2/ldom.js` , `/lib2/vsevent.js` , `/lib2/ui.js` , `/lib2/vsgrid.js`
- `/img/` *(logos, icons) and* `/app/`

15.4 Reactive Event Bridge with `vsevent` & DOM Patches

Frontend interactions (`button_click` , `changevalue`) collect form values and send them via POST to `/__vs_event` . The bridge intelligently invokes component-native `.getValue()` functions, guaranteeing properly typed numbers and standardized dates.

The Euphorix server returns a list of reactive DOM patches (`vsui:updates({...})`):

Patch Function	Parameters	Browser Effect
`vsui:patch(id, html)`	`id, html`	Replaces `innerHTML` of target element
`vsui:append_html(id, html)`	`id, html`	Appends HTML to target element (`beforeend`)
`vsui:set_val(id, val)`	`id, val`	Sets input value and invokes `.setValue(val)`
`vsui:grid_set_data(id, data)`	`id, data`	**New:** Smoothly updates DataGrid rows via JSON without HTML rebuild
`vsui:set_focus(id)`	`id`	**New:** Sets keyboard focus directly into field
`vsui:set_enabled(id, bool)`	`id, 0/1`	**New:** Enables or disables inputs and buttons reactively
`vsui:set_visible(id, bool)`	`id, 0/1`	**New:** Toggles element visibility (`display: none` / `''`)
`vsui:set_css(id, prop, val)`	`id, prop, val`	**New:** Dynamic inline styling (e.g. red border on error)
`vsui:set_class(id, cls, act)`	`id, cls, act`	**New:** Adds, removes, or toggles CSS classes
`vsui:clear_form(container_id)`	`container_id`	**New:** Clears all form fields within target container
`vsui:toast_show(txt, type, ms)`	`txt, type, ms`	Displays floating toast notification
`vsui:modal_open(id)` / `close`	`id`	Opens or closes modal dialog popup
`vsui:redirect(url)`	`url`	Redirects browser to specified URL

```
function on_save_customer(sequence ctx)
    sequence name = vsui:val(ctx, "txt_name")
    sequence cust_id = vsui:val(ctx, "txt_cust_id")

    -- Store in SQLite...
    sqlite:exec(db, "INSERT INTO customers (cust_id, name) VALUES (?, ?);", {cust_id, name})

    -- Query updated records
    sequence fresh_customers = sqlite:query_all(db, "SELECT cust_id, name FROM customers ORDER BY cust_id DESC;")

    return vsui:updates({
        vsui:grid_set_data("grid_customers", fresh_customers),
        vsui:clear_form("form_customer"),
        vsui:set_focus("txt_cust_id"),
        vsui:toast_show("Customer created successfully!", vsui:TOAST_SUCCESS)
    })
end function
```

15.5 Pixel-Exact Positioning & Window Centering (`ldom.setXYWH` , `opt_xywh`)

Unlike conventional web frameworks that force developers to write verbose custom CSS rules for desktop dialog placement, `std/vsui.e` integrates native pixel-exact coordinate positioning directly through the ldom.js coordinate system:

- **In Euphoria Component Declarations (`opt_xywh`)**:

Every component constructor accepts `vsui:opt_xywh(x, y, w, h)` :

```
  -- Position a button at X=100, Y=250 with width 180px and height 36px:
  sequence btn = vsui:button("btn_save", "Speichern", {vsui:opt_xywh(100, 250, 180, 36)})

  -- Place a DataGrid into a fixed workspace viewport:
  sequence grid = vsui:grid("grd_kunden", cols, rows, {vsui:opt_xywh(20, 80, 900, 500)})
```

- **Dynamic Server-Side Reactive Patches**:
- `vsui:set_xywh(id, x, y, w, h)` : Dynamically repositions and resizes any DOM element via `ldom: setXYWH` .
- `vsui:center_win(id, axis = "both")` : Horizontally and/or vertically centers an element within the browser viewport (`centerWinX` , `centerWinY`).
- `vsui:center_parent(id, axis = "both")` : Centers an element relative to its enclosing parent container (`centerX` , `centerY`).

15.6 Commercial Accounting Precision & German Number Formatting (`vsnumber.js`)

Standard HTML5 `<input type="number">` strictly enforces international dot-decimals and lacks thousand-separators, making it unsuitable for European ERP and accounting software. `std/vsui.e` bridges this gap with `lib2` 's dedicated vsnumber.js engine:

- **Configurable Decimal Places (`nk`)**: `vsui:numinput(id, value, nk, options)` sets exact decimal precision (`nk = 2` for EUR currency, `0` for integers, `4` for exchange rates).

- **Automatic Blur Formatting**: Users can type `1234,5` or `1234.5`; upon leaving the field (`blur`), the browser automatically formats it to `1.234,50` via `vsnumber.fz(val, nk)`.
- **High-Density Accounting Styling**: Formatted fields receive the class `.vs-number` from `globalblauklein.css`, ensuring right-aligned monospace numerals.
- **Bi-Directional Conversion in Euphoria**:
- `vsui:format_num_de(val, nk)`: Formats numeric atoms into localized strings (e.g., `50000.5` -> `"50.000,50"`).
- `vsui:parse_num_de(str)`: Robustly parses both German (`"1.234,56"`) and international (`"1234.56"`) strings into raw Euphoria atoms.
- `vsui:val_num(ctx, id)`: Helper that retrieves and converts submitted form inputs into numeric atoms in a single call.

15.7 Mobile Responsiveness & Touch Ergonomics (`vs-mobile.css`)

Via the automatically included stylesheet `/css/vs-mobile.css`, the entire `vsui` UI suite seamlessly adapts to mobile viewports (`<= 768px`) without affecting the desktop experience:

- **iOS Safari Auto-Zoom Prevention**: All input fields (`.vs-input`, `.vs-combo`, `.vs-spreadsheet-search`) enforce `font-size: 16px;` on mobile devices, preventing iOS Safari from automatically zooming into the page on focus.
- **Fitts's Law Touch Targets**: Buttons and inputs provide a minimum height of 42px with generous touch padding for reliable thumb navigation.
- **DataGrid Mobile Card-View**: On screen widths `<= 768px`, CSS transforms the tabular `vsgrid` into a sleek list of individual record cards, displaying column names via `data-label` attributes.
- **Horizontal Tab Swiping**: `.vs-tabs` enables smooth momentum horizontal swiping without vertical stacking or breaking tabs.
- **Splitters & Modals**: `.vs-splitter-container` collapses to vertical layout, while modals adapt as full-screen views with fluid touch scrolling.

15.8 Form Validation & Error Tooltips (`vsui:field_error`)

For robust user input validation, `std/vsui.e` provides a direct reactive error-feedback bridge between server and browser. When server-side validation detects an error, the corresponding input field can be highlighted in red with an explanatory tooltip:

- `vsui:field_error(id, error_msg)`: Produces a reactive DOM patch that adds the `.vs-input-error` class to the input and displays an inline error tip `<div class="vs-field-error-tip">` below it.
- `vsui:field_clear_error(id)`: Removes the error styling and removes the tooltip.
- `vsui:validate_required(ctx, field_id, field_label)`: Helper function to verify non-empty inputs. Returns `{1, ""}` if valid, or `{0, "The field '...' is required."}` on validation failure.
- `vsui:validate_number(str, min, max)`: Verifies numeric syntax (supporting comma/dot notation) and checks range bounds.

```
function on_submit_form(sequence ctx)
    sequence email = vsui:val(ctx, "txt_email")
    if length(email) = 0 then
        return vsui:updates({
            vsui:field_error("txt_email", "Please provide a valid e-mail address."),
            vsui:set_focus("txt_email")
        })
    end if

    return vsui:updates({
        vsui:field_clear_error("txt_email"),
        vsui:toast_show("Input validated successfully!", vsui:TOAST_SUCCESS)
    })
end function
```

15.9 Drag & Drop File Upload Dropzone (`vsui:upload`)

With `vsui:upload`, developers can embed a ready-to-use file upload dropzone supporting drag-and-drop, native file browser selection, and visual progress tracking:

- `vsui:upload(id, target_url, options)`: Generates the dropzone markup `.vs-upload-dropzone` with an upload icon, instructional text, file preview area, and upload progress bar.
- **Options:**
- `"prompt"`: Custom prompt text (e.g. `"Drag and drop Excel or CSV file here or click to browse"`).
- `"accept"`: Accepted file types or extensions (e.g. `".xlsx,.csv,.pdf"`).

```
sequence upload_box = vsui:upload("my_dropzone", "/api/upload", {
    {"prompt", "Drop business reports (XLSX, CSV) here or browse"},
    {"accept", ".xlsx,.csv"}
})
```

15.10 Native Responsive SVG Charts (`vsui:chart`)

Euphorix includes a fully self-contained SVG chart engine. Charts are rendered directly on the server as lightweight, responsive vector graphics (`viewBox="0 0 600 300"`) without requiring any third-party JavaScript libraries (no Chart.js, D3, or Canvas needed):

- `vsui:chart(id, chart_type, labels, datasets, options)` : Generates an SVG chart (`"bar"` , `"line"` , `"pie"` , `"doughnut"`).
- `vsui:chart_set_data(id, chart_type, labels, datasets)` : Generates a live DOM patch to update an existing chart via SSE or event bus without reloading the web page.

15.11 Complete Application Examples & Demos (`demo/vsui_demo.ex` , `examples/08_vsui_cockpit_demo.ex` , `examples/11_vsui_charts_und_upload_demo.ex`)

Euphorix provides three tailored demo and test programs covering 100% of the components and features in Chapter 15:

1. **Interactive All-in-One Flagship Cockpit**: demo/vsui_demo.ex

- **100% Component Coverage**: Brings together all Chapter 15 components into a single interactive enterprise application on port 8089 (DataGrid with sticky header & search, splitter layout, tab navigation, input fields for text, dates, times, numbers, and textareas, server-side form validation with error tooltips, reactive live SVG charts updated via button, and drag & drop file upload).
- **Launch and open in browser**:

```
./euphorix demo/vsui_demo.ex
# Open in browser: http://127.0.0.1:8089/
```

1. **Automated End-to-End Test & Headless Verification**: examples/08_vsui_cockpit_demo.ex

- Executes a headless test verifying HTTP GET on static CSS/JS assets, grid rendering, synthetic event dispatching to `/__vs_event` , reactive DOM patches, and SQLite database integrity:

```
./euphorix examples/08_vsui_cockpit_demo.ex
```

1. **Focused SVG Charts & Upload Dashboard**: examples/11_vsui_charts_und_upload_demo.ex

- Concise code example specifically showcasing server-rendered SVG bar and doughnut charts, drag & drop file upload, and email validation:

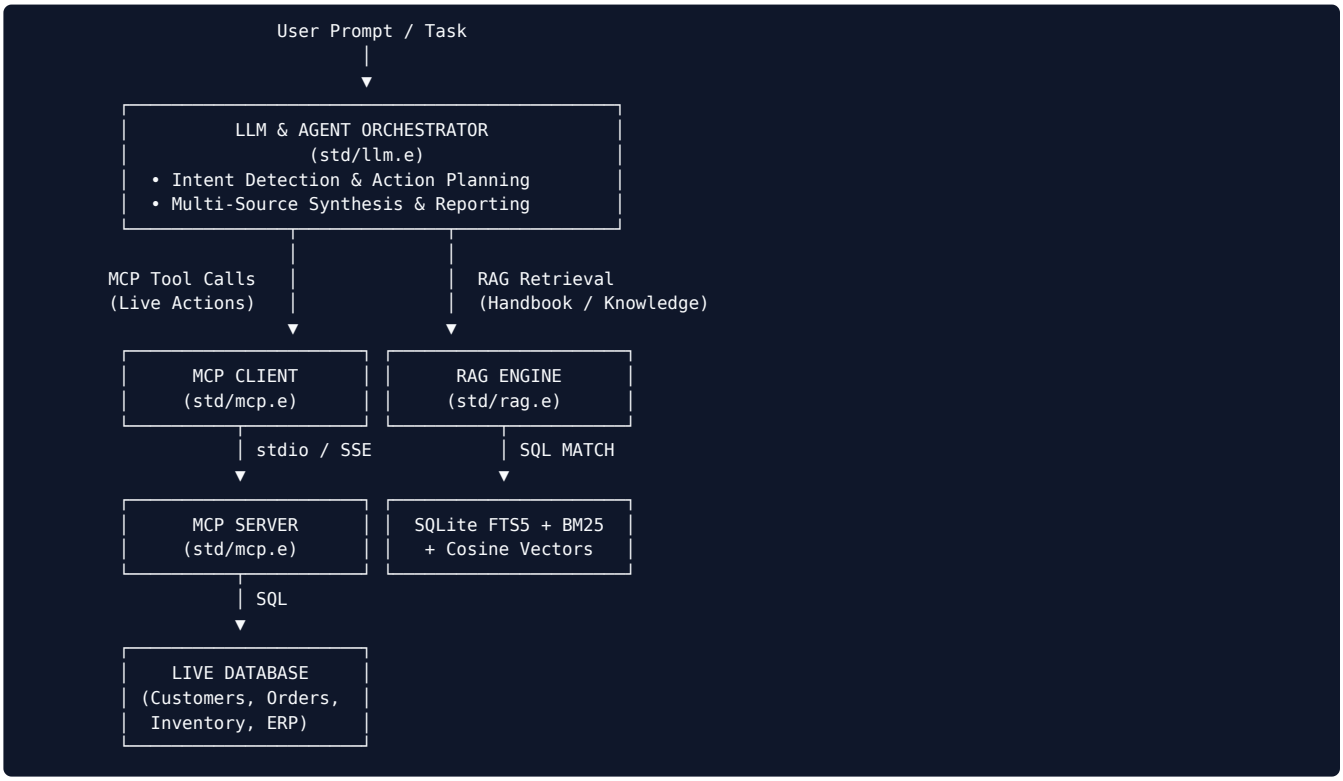
```
./euphorix examples/11_vsui_charts_und_upload_demo.ex --run
# Open in browser: http://127.0.0.1:8092/
```

16. Artificial Intelligence: Autonomous Agents, Model Context Protocol (MCP), RAG & LLMs (`std/mcp.e`, `std/rag.e`, `std/llm.e`)

Euphorix 1.0 integrates native enterprise interfaces for modern artificial intelligence and autonomous agents. Through three coordinated modules, any Euphorix application can be exposed as an **MCP server** for AI assistants (such as Claude Desktop, Antigravity, or Cursor), operated as an **autonomous agent with its own MCP client**, or connected to enterprise knowledge bases using **RAG (Retrieval-Augmented Generation)**.

16.1 Architecture: The Interaction between LLM, MCP, and RAG

A modern AI agent is built upon three core pillars:



1. **LLM (`std/llm.e`) – The Brain:**

Interprets user intentions, decides which external tools to invoke, and synthesizes live database facts with policy texts into an authoritative answer.

1. **MCP Server & Client (`std/mcp.e`) – The Hands:**

Executes controlled read and write operations on live production systems (e.g., search customer, check inventory, insert order into SQLite).

1. **RAG Engine (`std/rag.e`) – The Long-Term Memory:**

Searches unstructured documents (guidelines, warranty policies, manuals) via hybrid full-text (BM25) and vector similarity, providing verifiable citations.

16.2 When LLM + MCP Suffice Alone (and When RAG Is Needed)

In practice, there is often a misconception that every AI architecture requires a RAG pipeline. **For relational database, ERP, and CRM applications, LLM + MCP alone are entirely sufficient!**

Criterion	LLM + MCP (Direct Tool Access)	RAG (Document Vector Search)
Primary Data Source	Structured relational data (tables, columns, IDs, foreign keys).	Unstructured text (PDFs, handbooks, contracts, emails, support tickets).
Precision & Calculations	**100% mathematically exact**: SQL queries calculate sums, averages, and counts directly on the database engine.	**Probabilistic**: The LLM must infer and count numbers from text chunks (risk of hallucination).
Freshness	**Real-Time (Zero-Latency)**: Every tool invocation reads the exact live state of the database.	**Delayed**: Documents must be chunked, embedded, and indexed in advance.
Actions & Write Operations	**Active & Executable**: Can create records (`INSERT`), update statuses (`UPDATE`), or send emails.	**Read-Only**: Strictly limited to passive information retrieval.
Recommended Usage	**90% of business ERP/CRM workflows**: Order status, customer lookup, inventory queries.	**Complement for unstructured texts**: Policy lookups, troubleshooting guides, contract terms.

> [!TIP]

> **RAG as an MCP Tool:** You do not have to choose between the two. When you need to search document text, simply implement RAG as an additional MCP tool (`tool: search_handbook(topic)`). To the LLM, RAG is simply another specialized tool among many!

16.3 Model Context Protocol (MCP) – Server & Client (`std/mcp.e`)

The Model Context Protocol (MCP, official specification 2024-11-05) is the modern standard for connecting AI models safely to local databases and tools.

16.3.1 Creating a Custom MCP Server for Databases & Tools

An Euphorix MCP server declares tools with JSON Schema parameters and registers handler routines:

```
include std/mcp.e as mcp
include std/sqlite.e as sqlite

-- 1. Create server instance
sequence srv = mcp:server_create("Euphorix-CRM-Server", "1.0.0")

-- 2. Define tool handler
function handle_customer_search(sequence args)
    sequence term = ""
    for i = 1 to length(args) do
        if equal(args[i][1], "search_term") then term = args[i][2] end if
    end for

    sequence sql = sprintf("SELECT kdnr, name, city, status FROM customers WHERE name LIKE '%%%s%%'", {term})
    sequence res = sqlite:query(db, sql)
    return {1, "Found customers: " & sprint(res), {}}
end function

-- 3. Register tool with schema
srv = mcp:server_register_tool(srv,
    "customer_search",
    "Searches the SQLite ERP customer database by name or city",
    {
        {"search_term", "string", "Company name or search keyword", 1}
    },
    routine_id("handle_customer_search")
)

-- 4. Run server in desired transport mode
-- For Claude Desktop / Subprocess (stdio):
mcp:server_run_stdio(srv)

-- For network & web access via HTTP/SSE:
-- mcp:server_run_http(srv, 8085)
```

16.3.2 The MCP Client: Dynamically Listing and Calling Tools

An Euphorix application can act as an autonomous MCP client to drive any MCP server:

```
include std/mcp.e as mcp

-- 1. Launch MCP server subprocess over stdio
sequence res = mcp:client_stdio("./euphorix examples/12_mcp_server_demo.ex --stdio")
sequence client = res[2]

-- 2. Initialize handshake
sequence init = mcp:client_init(client, "Euphorix-Agent", "1.0.0")
client = init[3]

-- 3. Query available tools
sequence tools_res = mcp:client_list_tools(client)
client = tools_res[3]

-- 4. Invoke tool
sequence call_res = mcp:client_call_tool(client, "customer_search", {{"search_term", "Schulze"}})
client = call_res[4]
puts(1, "Server response:
" & call_res[2] & "
")

-- 5. Close client connection
mcp:client_close(client)
```

16.4 Universal LLM Client (`std/llm.e`)

The `std/llm.e` module provides a unified, provider-agnostic interface for chat completions and vector embeddings.

16.4.1 Connecting to Local Ollama, OpenAI & Claude

```
include std/llm.e as llm

-- 1. 100% Offline Mode (Zero external daemons or internet required)
sequence cli_mock = llm:client(llm:PROVIDER_MOCK)

-- 2. Local Ollama (e.g., Llama 3, Mistral, Qwen)
sequence cli_ollama = llm:client(llm:PROVIDER_OLLAMA, {
    {"model", "llama3:latest"},
    {"base_url", "http://localhost:11434"}
})

-- 3. OpenAI Cloud API
sequence cli_openai = llm:client(llm:PROVIDER_OPENAI, {
    {"model", "gpt-4o-mini"},
    {"api_key", "sk-..."}
})

-- 4. Anthropic Claude API
sequence cli_claude = llm:client(llm:PROVIDER_ANTHROPIC, {
    {"model", "claude-3-5-sonnet-20241022"},
    {"api_key", "sk-ant-..."}
})
```

16.4.2 Generating Text & Multi-Turn Chat

```
-- Simple text completion
sequence reply = llm:complete(cli_ollama, "Summarize the key advantages of SQLite in 2 sentences.")
puts(1, reply[2] & "
")

-- Structured chat with roles
sequence messages = {
    {"system", "You are an enterprise ERP assistant."},
    {"user", "What payment terms apply to Tier-A customers?"}
}
sequence chat_res = llm:chat(cli_ollama, messages, {{"temperature", 0.3}})
puts(1, chat_res[2] & "
")
```

16.4.3 Generating Vector Embeddings & Similarity Calculation

```
-- 64-dimensional normalized feature embedding vector
sequence v1 = llm:embed(cli_mock, "Customer pays invoice with cash discount")
sequence v2 = llm:embed(cli_mock, "Payment received with early payment discount deduction")

-- Cosine similarity (1.0 = identical, 0.0 = orthogonal)
atom sim = llm:cosine_similarity(v1, v2)
printf(1, "Semantic similarity: %.2f%%
", {sim * 100.0})
```

16.5 Retrieval-Augmented Generation with SQLite FTS5 & Vectors (`std/rag.e`)

The RAG engine couples SQLite Full-Text Search (FTS5 with BM25 ranking) and semantic vector embeddings into a **hybrid retrieval system**.

16.5.1 Creating a Knowledge Base & Indexing Documents

```
include std/rag.e as rag
include std/llm.e as llm

-- Create knowledge base in file or in-memory
sequence kb = rag:create_kb("corporate_policies", llm_cli, ":memory:")

-- Index documents with automatic smart chunking (paragraph and sentence boundaries)
integer chunks = rag:add_document(kb, "DOC-01", "Sales Policy 2026",
    "Customer Tier A: Receives 15% special discount. Payment term 30 days net or 3% discount within 10 days.

" &
    "Customer Tier B: Receives 5% discount above 25,000 € turnover. Payment term 14 days net.")
```

16.5.2 Hybrid Retrieval & Grounded Answers (`rag:ask`)

```
-- Execute hybrid search and generate fact-grounded response with citations
sequence result = rag:ask(kb, "What discount rules apply to Tier-A customers?", {{"top_k", 2}})

-- Print generated response
puts(1, "Answer:
" & rag:answer(result) & "

")

-- Format source citations
puts(1, "Sources cited:
" & rag:format_sources(result) & "
")
```

16.6 Complete Autonomous AI Agent (`examples/15_autonomous_agent_mcp_rag_demo.ex`)

The demonstration program examples/15_autonomous_agent_mcp_rag_demo.ex combines all components into a battle-tested end-to-end workflow:

1. **Agent spawns MCP server** as a subprocess (`stdio`) and binds live ERP database tools.
2. **Agent indexes policy handbooks** into SQLite FTS5 via RAG.
3. **Task 1 (Read & Report):** Inspects customer *Schulze Maschinenbau* (reads active manufacturing order from SQLite via MCP and combines it with payment terms from RAG handbook into a consolidated briefing).
4. **Task 2 (Write & Rules):** Executes an MCP write action creating new customer *Quantum Logistics GmbH* in the SQLite database while concurrently looking up freight rules for Tier-B customers from the handbook.
5. **Clean Shutdown** of all subprocesses and database handles.

Run with:

```
# 100% Offline (deterministic Euphorix mock synthesis)
./euphorix examples/15_autonomous_agent_mcp_rag_demo.ex

# Using local Ollama
./euphorix examples/15_autonomous_agent_mcp_rag_demo.ex --ollama

# Using OpenAI Cloud API
./euphorix examples/15_autonomous_agent_mcp_rag_demo.ex --openai
```