Euphorix 1.0 (Java 25 VM für OpenEuphoria-Syntax) – Entwicklerhandbuch & API-Referenz

> 🇬🇧 **English version available:** MANUAL.md | 📄 **Übersicht:** README.md | ⚖️ **Lizenz:** GNU GPLv3

> [!NOTE]

> **Hinweis zur OpenEuphoria-Syntax: Euphorix** ist eine eigenständige, von Grund auf neu entwickelte Hochleistungs-VM in Java 25. Sie interpretiert die etablierte, intuitive **OpenEuphoria 4.x-Syntax** und Standardbibliothek in Reinform, erweitert diese jedoch um modernste Java-25-Fähigkeiten (Virtual Threads, Go-Style Channels, Async/Await Futures, Swing/FlatLaf Desktop-GUI, JDBC für SQLite & PostgreSQL, UTF-8 und JSON/XML).

Dieses Handbuch ist die vollständige Dokumentation und Referenz für **Euphorix** (`./euphorix` bzw. `euphorix-standalone` und `euphorix.jar`). Kompatibilitäts-Aliase wie `./eui-java` und `./eui-standalone` bleiben vollumfänglich erhalten.

Die Laufzeitumgebung kombiniert die schlanke, intuitive Syntax von OpenEuphoria mit den modernsten Fähigkeiten der Java-Plattform (OpenJDK 25):

- **Virtuelle Threads (Project Loom)** für hochskalierbare Server, asynchrone Pipelines und Subprozesse – komplett **ohne fehleranfälliges POSIX `fork()`** .
- **Nahtlose Java-Interoperabilität (`std/java.e`)**: Laden externer JAR-Dateien zur Laufzeit, Instanziierung von Java-Klassen, Aufruf statischer und dynamischer Methoden, Feldzugriffe sowie dynamische Proxy-Listener für Callbacks.
- **Native GUI-Entwicklung mit Java Swing & AWT (`std/gui.e`)**: Modernes Linux-Desktop-Design via **GTK+ Look & Feel**, gestochen scharfes **Subpixel-LCD-Antialiasing**, native Ubuntu-Typografie und deklarative Widget-Erstellung.
- **Vollständige UTF-8-Unterstützung**: Konsistente Verarbeitung deutscher Umlaute (`ä` , `ö` , `ü` , `ß`) und weltweiter Unicode-Zeichen durchgehend in Lexer, Compiler, Bytecode-VM, Swing-GUI und I/O.
- **Integrierte Enterprise-Module**: JSON & RFC 9535 JSONPath, JSON-Schema (Draft 2020-12), XML DOM mit W3C XPath 1.0 & XSD, SQLite 3 & PostgreSQL (JDBC), HTTP/HTTPS Client & Server sowie Dual-Stack IPv4/IPv6 Sockets.
- **C-FFI & nativer Rohspeicher**: Direkte Einbindung nativer Linux-Bibliotheken (`.so`) über das JDK 25 Foreign Function & Memory (FFM) API (`java.lang.foreign.*`).

Inhaltsverzeichnis

1. Architektur, Installation, Build & Betrieb (Java 25 `euphorix.jar` auf Linux 64-Bit)

- 1.1 Projektstruktur & Fat-JAR Konzept
- 1.2 Virtual Threads Architektur (Subprozesse & Nebenläufigkeit ohne `fork()`)
- 1.3 Native C-Einbindung (.so) via JDK 25 FFM API
- 1.4 Voraussetzungen & Maven-Build
- 1.5 Externe Java-Bibliotheken verwalten & einbinden via pom.xml
- 1.6 Autarkes Single-Binary mit integrierter Minimal-JRE (`./euphorix-standalone`)
- 1.7 Ahead-of-Time Bytecode-Kompilierung (`.exb`), Standalone-Binding & Dedizierte Runtime (`runtime.jar` / `euphorix-rt`)
- 1.8 Plattformunabhängigkeit & Windows-Unterstützung (`euphorix.cmd` / `euphorix.bat`)
- 1.9 Entwickler-Workflow mit Hot-Reload File-Watcher (`--watch` / `-w`)
- 1.10 Linux-Dauerbetrieb & Systemd-Dienste (`tools/make_service.ex`)
- 1.11 Reverse-Proxy-Betrieb (lighttpd, Nginx, Caddy)
- 1.12 IDE-Integration & Language Server Protocol (LSP) für Apache NetBeans & VS Code

1. Vollständige Sprach-, Syntax- & Standard-Referenz

- 2.1 Datentypen & Typsystem
- 2.2 Sichtbarkeiten, Namespaces & Includes
- 2.3 Operatoren, Indizierung & Slicing
- 2.4 Kontrollstrukturen & Schleifen
- 2.5 Prozeduren, Funktionen & Callbacks
- 2.6 Bedingte Kompilierung (`ifdef`)
- 2.7 Alle fest eingebauten Built-in Routinen
- 2.8 Vollständige Referenz aller 77 Include-Dateien & Standardmodule (`std/.e` & `euphoria/ .e`)
- 2.8.1 Datenstrukturen, Maps & Algorithmen
- 2.8.2 I/O, Dateisystem, Konsole & Betriebssystem

- 2.8.3 Textverarbeitung, Unicode & Lokalisierung
- 2.8.4 Mathematik, Statistik & Zufall
- 2.8.5 Datum, Uhrzeit & Kalender
- 2.8.6 Netzwerk, Internet, DNS & URLs
- 2.8.7 Datenformate, XML, JSON & Datenbanken
- 2.8.8 Desktop-GUI & Web-Frontend Frameworks
- 2.8.9 Java-Interoperabilität, C-FFI & Maschinenspeicher
- 2.8.10 Moderne Nebenläufigkeit, Threads & Multitasking
- 2.8.11 Fehlerbehandlung, Testing & Hilfsprogramme
- 2.8.12 Windows-Plattformspezifische Module
- 2.8.13 Euphoria Interna, Debugger & Syntax-Tokenizer
- 2.8.14 Künstliche Intelligenz, MCP, LLM & RAG

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

- 3.1 Laden externer JAR-Dateien (`java:load_jar`)
- 3.2 Klassenauflösung & Objekterzeugung
- 3.3 Methodenaufrufe (statisch & dynamisch)
- 3.4 Lesen und Schreiben von Feldern
- 3.5 Automatisches Typ-Matching & Konvertierungen
- 3.6 Ereignis-Listener & Callbacks (`java:create_listener`)
- 3.7 Subprozess-Pipes mit Virtual Threads

1. Grafische Benutzeroberflächen mit Swing & AWT (`std/gui.e`)

- 4.1 Linux-Desktop-Integration & Theme-Auswahl (`gui:set_theme`)
- 4.2 Typografie & Subpixel-Antialiasing
- 4.3 Fensterverwaltung (`window` , `show` , `wait_window`)
- 4.4 Layout-Manager mit Abstandssteuerung (`hgap` / `vgap`)
- 4.5 Widget-Komponenten-Referenz
- 4.6 Eigenschaften, Styling & Polsterung
- 4.7 Standard-Dialoge (Message, Input, Confirm, FileDialog)
- 4.8 Vollständiges Demo-Programm (`demo/gui_demo.ex`)

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

- 5.1 Vollständige UTF-8-Integration im Java 25 Interpreter
- 5.2 Zeichengenaue Unicode-Operationen (`std/utf8.e`)

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

- 6.1 JSON-Parsing & Serialisierung (`std/json.e`)
- 6.2 RFC 9535 JSONPath Engine mit Pfadabfragen

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

- 7.1 Deklarative Schema-Validierung nach Draft 2020-12

1. Dokumenten- & Tabellenformate: 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-Validierung (`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-Datenbanken: SQLite 3 (`std/sqlite.e`) & PostgreSQL (`std/pgsql.e`)

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

1. Web-Server, Microservices, Sicherheit & Realtime (`std/http.e` , `std/auth.e` , `std/session.e` , `std/sse.e`)

- 10.1 HTTP- & HTTPS-Client (`std/http.e`)
- 10.2 Integrierter Web-Server & REST-Routing (`std/http.e`)
- 10.3 Enterprise-Sicherheit: 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 Netzwerk & Sockets (`std/socket.e` & `std/net/dns.e`)

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

1. Demos, Beispiele & Ausführung

- 12.1 Praxisbeispiele & Demos im Überblick
- 12.2 Ausführen der Demonstrationsprogramme

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

- 13.1 Weg 1: Klassische Tasks mit Virtual Threads (`task:create_thread`)
- 13.2 Weg 2: Channels / Go-Style Concurrency (`std/channel.e`)
- 13.3 Weg 3: Async / Await Futures (`std/thread.e`)
- 13.4 Weg 4: Hintergrundaufgaben & Task-Scheduler (`std/cron.e`)

1. Web-Anwendungen mit SAP UI5 Web Components 2.29 (`std/ui5.e`)

- 14.1 Architektur & Fiori Horizon Design
- 14.2 Komponenten-Übersicht & Web-Standards
- 14.3 Reaktive Euphorix-Bridge & Live DOM-Updates
- 14.4 Vollständiges Anwendungsbeispiel (`examples/07_sapui5_webcomponents_demo.ex`)

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

- 15.1 Architektur & Philosophie: Die Symbiose aus HTMX und Vaadin
- 15.2 Vollständige 22-Komponenten-Suite (`html/lib2/components/`)
- 15.3 Integrierter Asset-Server (`/lib2/` , `/css/` , `/img/*`)
- 15.4 Reaktive Event-Bridge mit `vsevent` & DOM-Patches
- 15.5 Pixelgenaue Positionierung & Fensterzentrierung (`ldom.setXYWH` , `opt_xywh`)
- 15.6 Kaufmännische Präzision & Deutsche Zahlenformatierung (`vsnumber.js`)
- 15.7 Mobile Responsiveness & Touch-Ergonomie (`vs-mobile.css`)
- 15.8 Formularvalidierung & Fehlertooltips (`vsui:field_error`)
- 15.9 Drag & Drop Datei-Upload Dropzone (`vsui:upload`)
- 15.10 Native Responsive SVG-Charts (`vsui:chart`)
- 15.11 Vollständige Anwendungsbeispiele & Demos (`demo/vsui_demo.ex` , `examples/08_vsui_cockpit_demo.ex` , `examples/11_vsui_charts_und_upload_demo.ex`)

1. Künstliche Intelligenz: Autonome Agenten, Model Context Protocol (MCP), RAG & LLMs (`std/mcp.e` , `std/rag.e` , `std/llm.e`)

- 16.1 Architektur: Das Zusammenspiel von LLM, MCP und RAG
- 16.2 Wann LLM + MCP allein genügen (und wann RAG benötigt wird)
- 16.3 Model Context Protocol (MCP) – Server & Client (`std/mcp.e`)
- 16.4 Universeller LLM-Client (`std/llm.e`)
- 16.5 Retrieval-Augmented Generation mit SQLite FTS5 & Vektoren (`std/rag.e`)
- 16.6 Vollständiger Autonomer KI-Agent (`examples/15_autonomous_agent_mcp_rag_demo.ex`)

---

1. Architektur, Installation, Build & Betrieb (Java 25 `euphorix.jar` auf Linux 64-Bit)

1.1 Projektstruktur & Fat-JAR Konzept

Der Interpreter ist als **autarkes Fat-JAR (`euphorix.jar`)** konzipiert. Beim Bauen mit Maven werden alle Abhängigkeiten (JDBC-Treiber, Jackson JSON, FlatLaf GUI, eingebettete Standardbibliothek) in ein einziges Archiv gebündelt:

```
prj/intpret/
├── euphorix (eui-java, eux) <- Ausführbares Shell-Startskript mit optimierten JVM-Flags
├── euphorix.jar (eui.jar)   <- Autarkes Java-25-Fat-JAR (19,4 MB inkl. aller Abhängigkeiten)
├── euphorix-standalone      <- Autarkes 53-MB-Single-Binary (inkl. minimaler JRE)
├── euphorix.cmd / .bat      <- Windows-Startskripte
├── pom.xml                  <- Maven Projektdatei für JDK 25 & Maven-Shade-Plugin
├── eu_include/              <- Standardbibliothek (std/*.e, euphoria/*.e)
│   ├── std/
│   │   ├── gui.e            <- Swing & AWT Desktop GUI API
│   │   ├── java.e           <- Java Interoperabilität, JAR-Loader & Virtual Thread Pipes
│   │   ├── utf8.e           <- Unicode & UTF-8 Textverarbeitung
│   │   ├── 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 usw.
├── java/src/main/java/eu/   <- Java 25 Quellcode
│   ├── Main.java            <- CLI-Einstiegspunkt, UTF-8 & AWT-Property-Setup
│   ├── Lexer.java           <- Lexer mit nativer UTF-8-Multi-Byte-Unterstützung
│   ├── Compiler.java        <- Single-Pass Bytecode-Compiler & Constant Folder
│   ├── Bytecode.java        <- 3-Adress-Register-Instruktionssatz (OpCodes)
│   ├── VM.java              <- Register-VM mit Virtual Threads & Engine-Dispatch
│   ├── Value.java           <- 48-Bit Int, IEEE-754 Atom & Copy-on-Write Sequenzen
│   ├── JavaInterop.java     <- Reflection, JAR-ClassLoader, Dynamic Proxy & Pipes
│   └── DataEngines.java     <- JSON, XPath & Schema-Engines
└── demo/
    └── gui_demo.ex          <- Interaktive GTK+ Swing-GUI Demonstration
```

Das Startskript `./euphorix`

Das Startskript (`./euphorix` bzw. Aliase `./eui-java` , `./eux`) sorgt für optimale Linux-64-Bit-JVM-Einstellungen:

```
#!/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 Architektur (Subprozesse & Nebenläufigkeit ohne `fork()`)

In C-Programmen erzeugt `fork()` eine vollständige Kopie des Adressraums des Prozesses. Unter modernen Java-Laufzeitumgebungen ist `fork()` veraltet, birgt Deadlock-Risiken bei Multithreading und erzeugt unnötigen Overhead.

Die Java 25 Edition nutzt stattdessen **Virtuelle Threads (Project Loom)**:

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

Jeder externe Subprozess wird asynchron über `java.lang.ProcessBuilder` gestartet. Zwei leichtgewichtige Virtual Threads lesen kontinuierlich `stdout` und `stderr` in lock-freie `LinkedBlockingQueue` -Puffer ein. Es gibt keine Blockaden des Euphoria-Hauptthreads und keine Deadlocks bei vollen OS-Pipe-Puffern.

1. **HTTP- & HTTPS-Server**:

Anfragen an `http:server_create` werden auf einem virtuellen Thread-Pool verarbeitet. Tausende gleichzeitige Verbindungen benötigen nur minimale Systemressourcen.

1. **Reentrante Callbacks**:

Ereignisse aus GUI- und Hintergrund-Threads können über die Proxy-Engine sicher in den Euphoria-Interpreter zurückrufen (`VM.invokeRoutine`).

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

Die klassischen OpenEuphoria C-FFI-Funktionen (`open_dll` , `define_c_func` , `define_c_proc` , `c_func` , `c_proc` , `allocate` , `free` , `peek` , *`poke`*) arbeiten über das moderne **JDK 25 Foreign Function & Memory (FFM) API**:

- Shared Libraries (`.so`) werden über `SymbolLookup.libraryLookup(Path, Arena)` geöffnet.
- Funktionsaufrufe werden über `Linker.nativeLinker().downcallHandle()` mit exakten nativen ABI-Typen (`ValueLayout.JAVA_INT` , `JAVA_LONG` , `ADDRESS` , `JAVA_DOUBLE`) durchgeführt.
- Speicherreservierung erfolgt über native `MemorySegment.allocateNative()` .

Damit können beliebige Linux-Systembibliotheken (z. B. `libc.so.6` , `libm.so.6` , `libcrypto.so`) ohne C-Compiler direkt aus Euphoria aufgerufen werden.

1.4 Voraussetzungen & Maven-Build

- **Betriebssystem**: Linux 64-Bit (`x86_64`)
- **Java SDK**: OpenJDK 25 (unter `/home/volkix/prg/jdk25` oder via `JAVA_HOME`)
- **Build-Tool**: Apache Maven (`mvn`)

```
# Bauen des Fat-JARs euphorix.jar
JAVA_HOME=/home/volkix/prg/jdk25 mvn clean package -DskipTests

# Ausführen eines Euphoria-Programms
./euphorix demo/gui_demo.ex

# Ausführen der Unit-Test-Suite (Java 25 Runner)
./tools/run_all_tests ./euphorix tests/t_gui.e tests/t_java_interop.e tests/t_concurrency.ex
```

1.5 Externe Java-Bibliotheken verwalten & einbinden via `pom.xml`

Das Verwalten von Abhängigkeiten über die zentrale `pom.xml` ist der eleganteste und stabilste Weg, das gesamte Java-Ökosystem in OpenEuphoria nutzbar zu machen.

Funktionsweise des Fat-JAR Dependency-Managements:

1. **Zentrale Definition**: Jede in `pom.xml` eingetragene `<dependency>` wird von Maven automatisch aus Maven Central heruntergeladen – inklusive aller transitiven Abhängigkeiten.
2. **Automatisches Bundling (Shading)**: Das `maven-shade-plugin` entpackt alle Klassen der Abhängigkeiten und verschmilzt sie direkt in die Datei `euphorix.jar` .
3. **Sofortige Verfügbarkeit in Euphoria**: Es müssen keine Classpaths, Umgebungsvariablen oder separate `.jar` -Dateien auf Zielsystemen verwaltet werden. Jede Klasse aus den Dependencies steht in OpenEuphoria sofort über `java:new_object()` und `java:call_static()` bereit.

Beispiel: Eine neue Java-Bibliothek hinzufügen (3 Schritte)

Wenn Sie beispielsweise Diagramme mit **JFreeChart**, PDF-Erzeugung mit **iText/OpenPDF** oder Barcodes mit **ZXing** in Euphoria nutzen möchten:

1. **In `pom.xml` eintragen**:

```
<dependencies>
    <!-- Beispiel: JFreeChart für Datenvisualisierung -->
    <dependency>
        <groupId>org.jfree</groupId>
        <artifactId>jfreechart</artifactId>
        <version>1.5.4</version>
    </dependency>
</dependencies>
```

1. **Fat-JAR neu paketieren**:

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

1. **Direkt in Euphoria verwenden**:

```
include std/java.e as java

-- Die Klassen der neuen Library sind sofort verfügbar:
atom chart = java:call_static("org.jfree.chart.ChartFactory", "createBarChart", {
    "Umsatz 2026", "Kategorie", "EUR", dataset
})
```

Aktuell in `pom.xml` vorkonfigurierte Enterprise-Dependencies:

- **SQLite JDBC** (`org.xerial:sqlite-jdbc`): Vollständige native SQLite-Unterstützung.
- **PostgreSQL JDBC** (`org.postgresql:postgresql`): Direkte PostgreSQL-Server-Verbindungen.
- **Jackson Databind** (`com.fasterxml.jackson.core:jackson-databind`): High-Speed JSON- und RFC 9535 JSONPath-Engine.
- **FlatLaf Modern Look & Feel** (`com.formdev:flatlaf`): Moderner Desktop-Skin (Light, Dark, HiDPI).
- **SLF4J-NOP** (`org.slf4j:slf4j-nop`): Lautloses Logging für saubere CLI-Ausgaben.

1.6 Autarkes Single-Binary mit integrierter Minimal-JRE (`./euphorix-standalone`)

Sie können den gesamten Interpreter inklusive einer **auf ca. 60 MB abgespeckten Java-25-Runtime** in eine **einzige, eigenständige Binärdatei** packen:

```
# Erzeugt ./euphorix-standalone (ca. 53 MB xz-komprimiert)
make standalone
# oder direkt:
./tools/build_standalone.sh ./euphorix-standalone
```

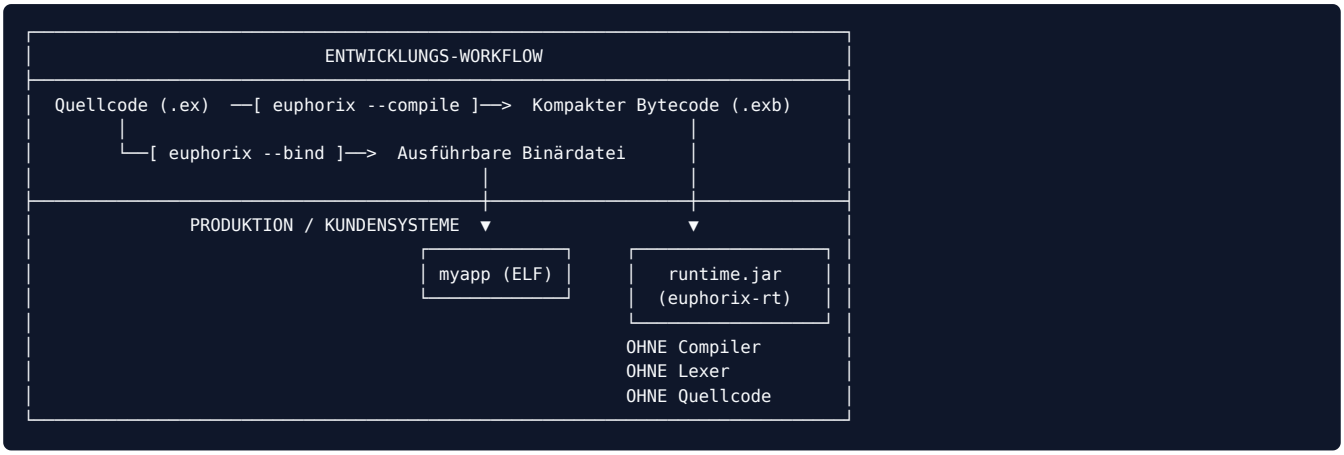
Wie das Single-Binary funktioniert:

1. **`jlink` (Java Linker)** analysiert die Module und extrahiert exakt die benötigten JVM-Komponenten (`java.base` , `java.desktop` , `java.sql` , `java.net.http` , `java.xml` etc.). Header, Manuals, Debug-Symbole und ungenutzte Module werden entfernt (von ~400 MB auf 60 MB).
2. Das Skript `tools/build_standalone.sh` verpackt die Mini-JRE und `euphorix.jar` in ein komprimiertes Payload-Archiv mit einem intelligenten Selbstextraktions-Header.
3. **Ausführung**:

- Die erzeugte Datei `euphorix-standalone` (bzw. Alias `eui-standalone`) kann auf jeden beliebigen Linux-64-Bit-Rechner kopiert werden (z. B. nach `/usr/local/bin/euphorix`).
- Auf der Zielmaschine **muss kein Java und kein JDK installiert sein**!
- Beim ersten Start entpackt sich das Bundle automatisch nach `~/.cache/euphorix/java25-runtime/<hash>/` . Alle nachfolgenden Aufrufe starten sofort aus dem Cache.
- **Vorteil gegenüber GraalVM**: Volle Unterstützung für dynamisches Nachladen von JARs (`java:load_jar`), Swing & GTK+ Desktop-GUIs, Virtual Threads und JIT-Optimierung ohne Closed-World-Einschränkungen!

1.7 Ahead-of-Time Bytecode-Kompilierung (`.exb`), Standalone-Binding & Dedizierte Runtime (`runtime.jar` / `euphorix-rt`)

Für Unternehmen und Entwickler, die ihren **Euphoria-Quellcode (`.ex` , `.e`) schützen und nicht im Klartext weitergeben** möchten, oder maximale Startperformance benötigen, bietet Euphorix ein vollständiges Ahead-of-Time (AOT) Bytecode- und Runtime-Ökosystem:



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

Kompiliert eine `.ex` -Quelldatei inklusive aller aufgelösten `include` -Bibliotheken in eine eigenständige, binäre Bytecode-Datei (`.exb`):

```
# Kompiliert app.ex zu app.exb:
./euphorix --compile app.ex -o app.exb
# oder kurz:
./euphorix -b app.ex -o app.exb
```

- **Dateiformat:** Binäres, komprimiertes GZIP-Format mit Magic Header `EXB1` (`0x45584231`). Enthält alle serialisierten Konstanten, Register-Instruktionen, Routinen-Signaturen und Metadaten.
- **Schutz des geistigen Eigentums:** Auf Kundensystemen wird keinerlei `.ex` - oder `.e` -Quellcode mehr benötigt.

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

Zusätzlich zum vollständigen Interpreter-JAR (`euphorix.jar`) erzeugt der Build-Prozess (`make runtime` bzw. `make java`) eine eigens minimierte Laufzeitumgebung:

```
# Ausführung einer .exb-Datei mit der schlanken Runtime:
java -jar runtime.jar app.exb [parameter...]

# Oder direkt über das CLI-Script:
./euphorix-rt app.exb [parameter...]
# bzw. symlink:
./runtime app.exb
```

- **Komplett Compiler- und Quellcode-frei:** `runtime.jar` schließt `Compiler.class` , `Lexer .class` , `Token.class` , `TokenType .class` und den gesamten Quellcode-Baum `/eu_include/**` vollständig aus.
- **Sicherheit:** Endanwender können auf dem Zielsystem keine unautorisierten Skripte kompilieren oder verändern.
- **Klare Fehlerdiagnose:** Wird versehentlich eine Quellcode-Datei (`.ex`) an die Runtime übergeben, weist sie diese mit einer klaren Meldung ab.

3. Startup-Benchmark-Vergleich (`--time` / `-t`)

Die Vorkompilierung eliminiert das Einlesen von Quelltexten, Tokenizing, AST-Generierung und Include-Resolution vollständig:

Modus	Befehl	Startup-Zeit (Load)	VM-Ausführung	Gesamtlaufzeit
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**

> 🚀 **Ergebnis:** Das Laden des Bytecodes ist **7x bis 10x schneller**; die Gesamtausführungszeit sinkt um über **50%**.

4. Selbst-ausführende Binärdateien (`--bind`)

Erzeugt eine direkt ausführbare Linux-Binärdatei mit eingebettetem Bytecode-Payload:

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

Die Binärdatei enthält einen schlanken POSIX-Boot-Stub, der automatisch nach `euphorix-rt` , `runtime.jar` oder `euphorix` im Pfad oder Cache sucht und den angehängten Bytecode direkt im Arbeitsspeicher ausführt.

5. Autarkes Standalone-Bundle mit Mini-JRE (`tools/build_standalone.sh --app`)

Das "Königsmodell" für maximale Unabhängigkeit:

```
# Bündelt Anwendung, Mini-JRE (60 MB), runtime.jar und statische Web-Assets (html/):
./tools/build_standalone.sh --app mein_projekt.ex -o mein_projekt-standalone
```

- Das fertige Executable benötigt **weder Java noch eine installierte Runtime**.
- Es enthält **keinen Quellcode**, sondern nur den vorkompilierten Bytecode (`app.exb`), die `runtime.jar` und alle statischen VS-UI Webkomponenten.

1.8 Plattformunabhängigkeit & Windows-Unterstützung (`euphorix.cmd` / `euphorix.bat`)

Dank des reinen Java-25-Bytecode-Kerns läuft `euphorix.jar` ohne jegliche Modifikation direkt unter **Microsoft Windows** (10, 11, Server) und **macOS**:

- **Dynamische OS-Erkennung**: Beim Kompilieren werden Symbole wie `WINDOWS` , `WIN32` bzw. `LINUX` , `UNIX` dynamisch über `System.getProperty("os.name")` gesetzt. `platform()` liefert unter Windows automatisch `2` (`WIN32`).
- **Pfade & Umgebungsvariablen**: `EUINC` trennt Pfade plattformgerecht über `File.pathSeparator` (Semikolon `;` unter Windows, Doppelpunkt `:` unter Linux).
- **Shell & Subprozesse**: Shell-Aufrufe (`system()` , `JavaInterop.pipeExec`) verwenden unter Windows automatisch `cmd.exe /c` statt `/bin/sh -c` .
- **Windows-Starter**: Zur bequemen Ausführung liegen im Projektverzeichnis euphorix.cmd und euphorix.bat bereit:

```
  euphorix.cmd mein_skript.ex
```

1.9 Entwickler-Workflow mit Hot-Reload File-Watcher (`--watch` / `-w`)

Beim Entwickeln von Web-APIs, Servern oder Desktop-UIs entfällt das manuelle Neustarten nach jeder Codeänderung. Mit dem Parameter `--watch` (oder kurz `-w`) überwacht Euphorix das Skriptverzeichnis und alle Projektdateien kontinuierlich auf Änderungen:

```
# Startet den File-Watcher für Ihr Programm
./euphorix --watch mein_server.ex

# Kurzform:
./euphorix -w mein_server.ex
```

- **Funktionsweise:** Nutzt den Java 25 `WatchService`, um Dateisystem-Ereignisse (Erstellen, Ändern, Löschen von `.ex`- und `.e`-Dateien) im gesamten Verzeichnisbaum in Echtzeit zu erfassen.
- **Intelligentes Debouncing:** Schnelle aufeinanderfolgende Änderungen (z. B. beim Speichern mehrerer Dateien im Editor) werden mit einem 200ms Debounce-Intervall zusammengefasst, um Mehrfachstarts zu verhindern.
- **Automatischer Neustart:** Sobald eine Änderung erkannt wird, beendet Euphorix den laufenden Kindprozess geordnet und startet das Skript mit aktuellem Quellcode neu.

1.10 Linux-Dauerbetrieb & Systemd-Dienste (`tools/make_service.ex`)

Für den Produktivbetrieb unter Linux generiert das integrierte Werkzeug `tools/make_service.ex` vorkonfigurierte, gehärtete `systemd`-Service-Dateien:

```
# Verwendung:
./euphorix tools/make_service.ex <skript.ex> [service-name] [beschreibung] [benutzer]

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

Das Tool generiert eine vollständige Service-Konfigurationsdatei (z. B. `euphorix-api.service`) mit folgenden Produkteigenschaften:

- **Automatischer Wiederanlauf (`Restart=always` , `RestartSec=5`)**: Bei unerwarteten Fehlern oder Serverneustarts wird der Dienst automatisch neu gestartet.
- **Dedizierter Benutzer & Arbeitsverzeichnis**: Läuft sicher unter dem angegebenen unprivilegierten Benutzerkonto.
- **Journald-Integration**: Standardausgabe und Fehlerausgaben werden direkt an `systemd-journald` weitergeleitet.
- **Ressourcengrenzen & Sicherheit**: Konfigurierte `LimitNOFILE=65536` für hochperformante Socket- und Datenbankverbindungen.

Das Skript gibt direkt die fertigen Einrichtungsbefehle aus:

```
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-Betrieb (lighttpd, Nginx, Caddy)

Für den Produktiveinsatz von Webanwendungen und Dashboards (std/vsui.e) empfiehlt sich der Betrieb des Euphorix-Dienstes hinter einem leichtgewichtigen Reverse-Proxy.

> [!TIP]

> **Vollständig autark:** Bei Verwendung des 52 MB Single-Binaries (`tools/build_standalone.sh --app app.ex`) sind der integrierte Webserver, alle 22 VS-UI-Webkomponenten, CSS- und JS-Dateien **fest im Binary eingebettet**. Es müssen auf dem Server keine separaten Dateien oder Ordner abgelegt werden!

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

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

$HTTP["host"] == "cockpit.meine-firma.de" {
    # Dynamische Requests und alle Web-Assets an Euphorix weiterleiten:
    proxy.server = ( "" => (
        ( "host" => "127.0.0.1", "port" => 8089 )
    ))
    proxy.header = (
        "map-host-request" => ( "-" => "$0" )
    )
}
```

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

```
server {
    listen 80;
    server_name cockpit.meine-firma.de;

    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; # Wichtig für reaktive VS-UI-Events und Server-Sent Events (SSE)
    }
}
```

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

Für komfortables, fehlerfreies Entwickeln mit OpenEuphoria in modernen Entwicklungsumgebungen (IDEs) bringt Euphorix einen vollwertigen **Language Server** gemäß der offiziellen **LSP 3.17-Spezifikation** mit:

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

Der Sprachserver kommuniziert über Standard-I/O (JSON-RPC 2.0) direkt mit Ihrer bevorzugten IDE.

Unterstützte Fähigkeiten (LSP Capabilities)

1. **Echtzeit-Diagnose (Diagnostics & Syntax-Checking):**

- Bei jeder Code-Änderung im Editor führt der Compiler eine In-Memory-Synthese und lexikalische Validierung durch.
- Syntaxfehler werden sofort mit exakter Zeilennummer, Spalte und deutscher Fehlermeldung rot im Editor unterstrichen.
- Sobald der Fehler behoben ist, verschwinden die Warnhinweise automatisch.

1. **Intelligente Code-Vervollständigung (IntelliSense & Snippets):**

- **Keywords & Snippets:** Tippen von `if`, `for`, `while`, `function`, `procedure`, `switch` etc. schlägt fertige Codegerüste mit Tabstopps vor.
- **Eingebaute Standardfunktionen:** Alle Built-ins (`puts`, `printf`, `length`, `append`, `match`, etc.) mit Parameterhinweisen.
- **Modul-Namespaces:** Durch Eingabe des Doppelpunkts (`vsui:`, `map:`, `json:`, `sequence:`, `io:`, `math:`, `sqlite:`, `http:`) werden direkt alle exportierten Modulroutinen samt Signaturen aufgelistet.
- **Lokale Symbole:** Alle im Dokument definierten Funktionen, Prozeduren, Typen und Konstanten stehen sofort zur Autovervollständigung bereit.

1. **Hover-Dokumentation (Mauszeiger-Hilfe):**

- Zeigen Sie mit der Maus auf Schlüsselwörter, Built-ins oder Standardroutinen (`vsui:numinput`, `map:put`, etc.), um formatierte Markdown-Kurzhilfen mit Parametern und Typbeschreibungen zu sehen.

1. **Gehe zu Definition (`F12` / `Strg + Klick`):**

- Direkter Sprung an die Deklarationsstelle von Prozeduren, Funktionen und Typen im Code.

1. **Dokument-Gliederung (Outline / Document Symbols):**

- Hierarchische Übersicht aller im aktuellen Quelltext deklarierten Prozeduren, Funktionen und Datentypen in der Symbolleiste der IDE.

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

Apache NetBeans bringt native Unterstützung für generische Language Server (`org-netbeans-modules-lsp-client.jar`) mit.

Automatisches Setup (Empfohlen)

Führen Sie im Projektverzeichnis einfach das mitgelieferte Einrichtungsskript aus:

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

Das Skript trägt den Euphorix-LSP-Server automatisch in Ihre NetBeans-Konfiguration ein (`~/.netbeans/<version>/config/Preferences/org/netbeans/modules/lsp/client.properties`).

Manuelle Konfiguration in der NetBeans-Benutzeroberfläche

Alternativ können Sie den Sprachserver direkt im NetBeans-Menü einrichten:

1. Öffnen Sie **Tools** -> **Options** -> **Editor** -> Reiter **Language Servers**.
2. Klicken Sie auf **Add...** und tragen Sie folgende Werte ein:

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

1. Bestätigen Sie mit **Apply** und **OK**.

Beim Öffnen einer beliebigen Euphoria-Datei (`.ex` , `.e`) startet NetBeans automatisch den Sprachserver im Hintergrund.

Visual Studio Code (VS Code) Integration

Für VS Code steht die Erweiterung im Verzeichnis editors/vscode/ bereit.

Option A: Installation des VS Code Plugins

Kopieren Sie den Ordner direkt in das VS Code Erweiterungsverzeichnis:

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

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

Nach einem Neustart von VS Code wird die Euphoria-Sprachunterstützung mitsamt TextMate-Syntax-Highlighting und LSP-Features automatisch aktiviert.

Option B: Über Generic LSP Client Extension

Falls Sie das Plugin **Generic LSP Client** (oder **glspc**) in VS Code installiert haben, ergänzen Sie einfach Ihre `settings.json` :

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

---

2. Vollständige Sprach-, Syntax- & Standard-Referenz

2.1 Datentypen & Typsystem

OpenEuphoria besitzt ein klares, hierarchisches Typsystem aus vier Grundtypen:

```
       object (jeder Wert: Zahl oder Sequenz)
      /      \
   atom      sequence (Liste beliebiger Länge und Tiefe, z. B. {1, "Text", {2, 3}})
    |
integer (48-Bit-Ganzzahl ohne Nachkommastellen)
```

- **`integer`**: 48-Bit-Ganzzahlen im Wertebereich von `-140737488355328` bis `+140737488355327`.
- **`atom`**: Jede Zahl – ganze Zahlen oder 64-Bit IEEE-754 Gleitkommazahlen (`double`).
- **`sequence`**: Beliebig schachtelbare, **1-basierte** Listen (`{}`). Strings sind Sequenzen von Zeichencodes (in UTF-8-Bytes).

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

Euphorix unterstützt drei komfortable Arten von String-Literalen:

1. **Klassische Strings (`"..."`)**: Einzeilige Zeichenketten mit Escape-Sequenzen (`\n`, `\t`, `\"`, `\\`).
2. **Backtick-Strings (`` `...` ``)**: Mehrzeilige Raw-Strings. Doppelte Anführungszeichen (`"`) müssen **nicht** maskiert werden. Perfekt für JSON, SQL und HTML:

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

1. **Triple-Quote-Strings (`"""..."""`)**: Mehrzeilige Textblöcke (bekannt aus Python/Kotlin). Auch hier sind Zeilenumbrüche und Quotes frei verwendbar:

```
sequence query = """
SELECT kdnr, name, umsatz
FROM kunden
WHERE status = 'A-Kunde'
"""
```

- **`object`**: Universeller Typ, der sowohl `atom` als auch `sequence` aufnehmen kann.

Variablen, Konstanten & Aufzählungen

```
integer anzahl = 42
atom preis = 19.95
sequence name = "Euphoria"
constant MWST = 0.19, WAEHRUNG = "EUR"
enum STATUS_BEREIT = 1, STATUS_AKTIV, STATUS_FERTIG
```

Eigene Typen definieren (`type ... end type`)

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

prozent rabatt = 15   -- Wird bei Zuweisung zur Laufzeit geprüft
```

2.2 Sichtbarkeiten, Namespaces & Includes

Schlüsselwort	Bedeutung
(ohne)	Nur innerhalb der aktuellen Datei sichtbar (`local`).
`export`	Sichtbar für Dateien, die diese Datei direkt einbinden.
`public`	Sichtbar für Einbinder (und wird weitergereicht).
`global`	Im gesamten Programm global verfügbar.
`override`	Überschreibt gezielt eine eingebaute oder globale Routine.

```
-- In bibliothek.e:
namespace bib
public function verdopple(atom x)
    return x * 2
end function

-- Im Hauptprogramm:
include bibliothek.e as b
? b:verdopple(21)   -- Ausgabe: 42
```

2.3 Operatoren, Indizierung & Slicing

Indizierung und Slicing (1-basiert)

- `s[1]` : Erstes Element
- `s[$]` : Letztes Element (`$` entspricht `length(s)`)
- `s[$-1]` : Vorletztes Element
- `s[2..4]` : Teilsequenz von Position 2 bis 4
- `s[2..$-1]` : Teilsequenz ohne erstes und letztes Element

Sequenz-Verkettung & Elementweise Operationen

- `&` : Hängt Sequenzen oder Elemente flach aneinander (`"Hallo " & "Welt"`).
- Alle Rechenoperatoren (`+` , `-` , `*` , `/`) arbeiten automatisch elementweise:

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

- **Gleichheitsprüfung**: Der Operator `=` vergleicht elementweise. Für den Vergleich zweier Sequenzen als Ganzes dient **`equal(a, b)`** !

2.4 Kontrollstrukturen & Schleifen

```
-- 1. Verzweigung
if bedingung then
    aktion1()
elsif andere_bedingung then
    aktion2()
else
    aktion3()
end if

-- 2. Switch-Anweisung
switch befehl do
    case "start", "run" then
        start()
        break
    case "stop" then
        stop()
        break
    case else
        unbekannt()
end switch

-- 3. Zählschleife
for i = 1 to 10 by 2 do
    ? i
end for

-- 4. Kopfgesteuerte Schleife
while laeuft do
    verarbeite()
end while

-- 5. Fußgesteuerte Schleife
loop do
    schritt()
    until fertig
```

2.5 Prozeduren, Funktionen & Callbacks

- **`procedure`**: Führt Aktionen aus, gibt keinen Wert zurück.
- **`function`**: Gibt immer einen Wert über `return <wert>` zurück.
- **Default-Parameter**: Parameter können Standardwerte haben (`function f(integer a, integer b = 10)`).
- **Mehrfachzuweisung**:

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

- **Routinen-IDs & Callbacks**:

`integer rid = routine_id("meine_funktion")`

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

2.6 Bedingte Kompilierung (**`ifdef`**)

```
ifdef LINUX then
    constant OS_NAME = "Linux"
elsifdef WINDOWS then
    constant OS_NAME = "Windows"
end ifdef
```

Vordefinierte Symbole in Euphorix (`./euphorix`): `EU4` , `EU4_0` , `EU4_1` , `EU4_1_0` , `UNIX` , `LINUX` , `BITS64` , `LONG64` , `X86_64` , `EUI` , `EUPHORIX` , `JAVA` , `JAVA25` , `CONSOLE` .

2.7 Alle fest eingebauten Built-in Routinen

Typprüfungen & Vergleiche

- `integer(x)` : `1` wenn Ganzzahl, sonst `0` .
- `atom(x)` : `1` wenn Zahl (`integer` oder `double`), sonst `0` .
- `sequence(x)` : `1` wenn Liste/String, sonst `0` .
- `object(x)` : `1` wenn Variable initialisiert ist.
- `equal(a, b)` : Prüft beliebige Objekte auf Gleichheit (`1` oder `0`).
- `compare(a, b)` : Lexikographischer Vergleich (`-1` , `0` , `+1`).

Sequenz- & Stringoperationen

- `length(s)` : Anzahl der Elemente / Bytes in `s` .
- `append(s, x)` : Hängt `x` als Element an das Ende an.
- `prepend(s, x)` : Fügt `x` am Anfang ein.
- `repeat(item, count)` : Erzeugt Sequenz aus `count` Wiederholungen von `item` .
- `find(needle, haystack, start=1)` : Findet Element in Sequenz (gibt 1-basierten Index oder `0`).
- `match(sub, haystack, start=1)` : Sucht Teilsequenz in Sequenz (gibt Startposition oder `0`).
- `insert(s, item, index)` / `splice(s, items, index)` : Fügt Element oder Teilsequenz ein.
- `remove(s, start, stop=start)` : Entfernt Elemente im Bereich.
- `replace(s, rep, start, stop=start)` : Ersetzt Bereich durch `rep` .

Konsole & Datei-I/O

- `puts(fn, text)` : Textausgabe auf Datei/Kanal (`1` = STDOUT, `2` = STDERR).
- `printf(fn, fmt, values)` : Formatierte Ausgabe (`%d` , `%f` , `%s` , `%x`).
- `sprintf(fmt, values)` : Formatiert String ohne Ausgabe.
- `gets(fn)` : Liest Zeile aus Datei (oder `-1` bei EOF).
- `getc(fn)` : Liest einzelnes Byte (oder `-1` bei EOF).
- `open(path, mode)` : Öffnet Datei (`"r"` , `"w"` , `"a"` , `"rb"` , `"wb"`).
- `close(fn)` : Schließt Datei.
- `seek(fn, pos)` / `where(fn)` : Dateizeiger positionieren und abfragen.

Mathematik

- `floor(x)` , `remainder(a, b)` , `sqrt(x)` , `power(base, exp)` , `log(x)` , `sin(x)` , `cos(x)` , `tan(x)` , `arctan(x)` .

2.8 Vollständige Referenz aller 77 Include-Dateien & Standardmodule (`std/.e` & `euphoria/ .e`)

Dieser Abschnitt dokumentiert jede einzelne Include-Datei der Standardbibliothek von Euphorix, ihren Zweck sowie ausnahmslos alle enthaltenen öffentlichen Prozeduren (`public procedure`) und Funktionen (`public function`).

Modul-Übersicht nach Kategorien:

- 2.8.1 Datenstrukturen, Maps & Algorithmen (10 Module)
- 2.8.2 I/O, Dateisystem, Konsole & Betriebssystem (8 Module)
- 2.8.3 Textverarbeitung, Unicode & Lokalisierung (10 Module)
- 2.8.4 Mathematik, Statistik & Zufall (5 Module)
- 2.8.5 Datum, Uhrzeit & Kalender (1 Module)
- 2.8.6 Netzwerk, Internet, DNS & URLs (5 Module)
- 2.8.7 Datenformate, XML, JSON & Datenbanken (6 Module)
- 2.8.8 Desktop-GUI & Web-Frontend Frameworks (6 Module)
- 2.8.9 Java-Interoperabilität, C-FFI & Maschinenspeicher (7 Module)
- 2.8.10 Moderne Nebenläufigkeit, Threads & Multitasking (3 Module)
- 2.8.11 Fehlerbehandlung, Testing & Hilfsprogramme (3 Module)
- 2.8.12 Windows-Plattformspezifische Module (3 Module)
- 2.8.13 Euphoria Interna, Debugger & Syntax-Tokenizer (7 Module)

2.8.1 Datenstrukturen, Maps & Algorithmen

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

- **Zweck:** Umfassende Manipulation von Sequenzen: mehrdimensionales Slicing, Transformationen, Filterung, Flattening, Kombinationen und Spaltenoperationen.
- **Öffentliche Funktionen & Prozeduren (39):**
- `public function binop_ok(object a, object b)` : Prüft, ob zwei Operanden gemeinsam für eine elementweise binäre Sequenzoperation zulässig sind.
- `public function fetch(sequence source, sequence indexes)` : Liest ein beliebig tief in einer verschachtelten Sequenz liegendes Element anhand eines Indexpfads aus.
- `public function store(sequence target, sequence indexes, object x)` : Schreibt einen Wert an eine beliebig tief verschachtelte Position in einer Sequenz.
- `public function valid_index(sequence st, object x)` : Prüft, ob der angegebene Index innerhalb der gültigen Grenzen der Sequenz liegt.
- `public function rotate(sequence source, integer shift, integer start=1, integer stop=length(source))` : Rotiert einen Teilbereich oder alle Elemente einer Sequenz zyklisch um eine Anzahl an Positionen.
- `public function columnize(sequence source, object cols = {}, object defval = 0)` : Wandelt eine Liste von Zeilensequenzen in eine Liste von Spaltenvektoren um.
- `public function apply(sequence source, integer rid, object userdata = {})` : Wendet eine benutzerdefinierte Callback-Funktion elementweise auf eine Sequenz an.
- `public function mapping(object source_arg, sequence from_set, sequence to_set, integer one_level = 0)` : Ersetzt Elemente einer Sequenz basierend auf zwei Abbildungstabellen (`from_set` -> `to_set`).
- `public function reverse(object target, integer pFrom = 1, integer pTo = 0)` : Kehrt die Reihenfolge der Elemente in einer Sequenz oder einem Teilbereich um.
- `public function shuffle(object seq)` : Mischt die Elemente einer Sequenz in zufälliger Reihenfolge (Fisher-Yates-Algorithmus).
- `public function series(object start, object increment, integer count = 2, integer op = '+')` : Erzeugt eine arithmetische oder geometrische Zahlenreihe als Sequenz.
- `public function repeat_pattern(object pattern, integer count)` : Erzeugt eine periodisch wiederkehrende Sequenz aus einem Muster und einer Wiederholungsanzahl.
- `public function pad_head(object target, integer size, object ch=' ')` : Füllt den Anfang einer Sequenz mit einem Füllelement auf eine Mindestlänge auf.
- `public function pad_tail(object target, integer size, object ch=' ')` : Füllt das Ende einer Sequenz mit einem Füllelement auf eine Mindestlänge auf.
- `public function add_item(object needle, sequence haystack, integer pOrder = 1)` : Fügt ein Element hinzu, falls es noch nicht in der Sequenz enthalten ist.
- `public function remove_item(object needle, sequence haystack)` : Entfernt das erste Vorkommen eines bestimmten Elements aus der Sequenz.
- `public function mid(sequence source, atom start, atom len)` : Liefert einen Teilbereich einer Sequenz, definiert durch Startindex und Länge.
- `public function slice(sequence source, atom start = 1, atom stop = 0)` : Schneidet einen Teilbereich einer Sequenz von Start- bis Endposition aus.
- `public function vslice(sequence source, atom colno, object error_control=0)` : Führt einen vertikalen Spaltenschnitt über eine verschachtelte 2D-Tabelle/Matrix durch.
- `public function patch(sequence target, sequence source, integer start, object filler = ' ')` : Überschreibt einen Teilbereich einer Sequenz mit neuen Elementen (mit optionalem Auffüllen).
- `public function remove_all(object needle, sequence haystack)` : Entfernt alle Vorkommen eines bestimmten Elements aus einer Sequenz.
- `public function retain_all(object needles, sequence haystack)` : Behält nur die in der Menge angegebenen Elemente und entfernt alle übrigen.
- `public function filter(sequence source, object rid, object userdata = {}, object rangetype = "")` : Filtert eine Sequenz anhand einer benutzerdefinierten Prädikatsfunktion.
- `public function extract(sequence source, sequence indexes)` : Extrahiert eine Teilmenge von Elementen anhand einer Sequenz von Ziel-Indizes.
- `public function project(sequence source, sequence coords)` : Projiziert ausgewählte Spalteneinträge aus Untersequenzen in eine neue Liste.
- `public function split( sequence st, object delim=' ', integer no_empty = 0, integer limit=0)` : Zerlegt eine Sequenz an einem Trennzeichen in eine Liste von Teilsequenzen.
- `public function split_any(sequence source, object delim=", \t|", integer limit=0, integer no_empty=0)` : Zerlegt eine Sequenz an beliebigen aus einer Menge von Trennzeichen.
- `public function join(sequence items, object delim=" ")` : Verbindet eine Liste von Sequenzen mit einem Trennzeichen zu einer Gesamtsequenz.
- `public function breakup(sequence source, object size, integer style = BK_LEN)` : Unterteilt eine Sequenz in Blöcke einer festen Maximallänge.

- `public function flatten(sequence pSource, object pDelim={})` : Hebt alle Verschachtelungsebenen auf und liefert eine flache 1D-Sequenz aller Atome.
- `public function pivot(object data_p, object pivot_p = 0)` : Teilt eine Sequenz anhand eines Pivot-Wertes in drei Teilsequenzen (kleiner, gleich, größer).
- `public function build_list( sequence source, object transformer, integer singleton = 1, object user_data = {})` : List-Comprehension: Baut eine neue Liste basierend auf Transformationen einer Quelldatei auf.
- `public function transform( sequence source_data, object transformer_rids)` : Transformiert eine Sequenz elementweise unter Berücksichtigung des aktuellen Index.
- `public function transmute(sequence source_data, sequence current_items, sequence new_items, integer start=1, integer limit = length(source_data))` : Führt komplexe strukturelle Transformationen auf verschachtelten Daten durch.
- `public function sim_index(sequence A, sequence B)` : Berechnet den Ähnlichkeitsindex zwischen zwei Sequenzen.
- `public function remove_subseq( sequence source_list, object alt_value = SEQ_NOALT)` : Entfernt eine bestimmte Teilsequenz aus einer größeren Sequenz.
- `public function remove_dups(sequence source_data, integer proc_option = RD_PRESORTED)` : Entfernt alle doppelten Elemente und behält nur eindeutige Werte bei.
- `public function combine(sequence source_data, integer proc_option = COMBINE_SORTED)` : Kombiniert zwei Sequenzen elementweise unter Anwendung einer binären Operation.
- `public function minsize(object source_data, integer min_size = floor(length(source_data) * 1.5), object new_data = 0)` : Stellt sicher, dass eine Sequenz mindestens die vorgegebene Mindestlänge besitzt.

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

- **Zweck:** Hochperformante Schlüssel-Wert-Speicher (Hash-Maps) mit dynamischem Rehashing, Iteratoren, Filterung und Transformationsfunktionen.
- **Öffentliche Funktionen & Prozeduren (26):**
- `public function calc_hash(object key_p, integer max_hash_p)` : Berechnet den Hash-Index für einen Schlüssel basierend auf der Map-Größe.
- `public function threshold(integer new_value_p = 0)` : Setzt oder liest den Füllgrad-Schwellenwert für das automatische Rehashing der Map.
- `public function type_of(map the_map_p)` : Ermittelt den internen Speichertyp der Map.
- `public procedure rehash( map the_map_p, integer requested_size_p = 0 )` : Reorganisiert und vergrößert die interne Hash-Tabelle der Map.
- `public function new( integer initial_size_p = DEFAULT_SIZE )` : Erstellt eine neue leere Hash-Map mit optionaler Anfangskapazität.
- `public function new_extra(object the_map_p, integer initial_size_p = 8)` : Erstellt eine Hash-Map mit benutzerdefinierten Schwellenwerten und Wachstumsfaktor.
- `public function compare(map map_1_p, map map_2_p, integer scope_p = 'd')` : Vergleicht zwei Hash-Maps auf identische Schlüssel und Werte.
- `public function has( map the_map_p, object key )` : Prüft, ob der angegebene Schlüssel in der Map existiert (1 = ja, 0 = nein).
- `public function get( map the_map_p, object key, object default = 0 )` : Liest den Wert zu einem Schlüssel aus der Map (oder liefert den angegebenen Standardwert).
- `public function nested_get( map the_map_p, sequence the_keys_p, object default_value_p = 0)` : Liest einen Wert aus hierarchisch verschachtelten Maps entlang eines Schlüsselpfads.
- `public procedure put( map the_map_p, object key, object val, object op = PUT, object deprecated = 0 )` : Fügt ein Schlüssel-Wert-Paar in die Map ein oder aktualisiert den bestehenden Eintrag.
- `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)` : Schreibt einen Wert in hierarchisch verschachtelte Maps entlang eines Schlüsselpfads.
- `public procedure remove( map the_map_p, object key )` : Entfernt einen Schlüssel und seinen zugehörigen Wert aus der Map.
- `public procedure clear( map the_map_p )` : Leert die Hash-Map und entfernt alle gespeicherten Schlüssel-Wert-Paare.
- `public function size( map the_map_p )` : Liefert die Anzahl der in der Map gespeicherten Schlüssel-Wert-Paare.
- `public function statistics(map the_map_p)` : Liefert Leistungskennzahlen (Kollisionen, Belegung, Füllgrad) der Hash-Map.
- `public function keys( map the_map_p, integer sorted_result = 0 )` : Gibt alle Schlüssel der Map als geordnete oder ungeordnete Sequenz zurück.
- `public function values( map the_map, object keys=0, object default_values=0 )` : Gibt alle in der Map gespeicherten Werte als Sequenz zurück.
- `public function pairs( map the_map, integer sorted_result = 0 )` : Gibt alle Einträge als Sequenz von 2-Element-Sequenzen `{key, value}` zurück.
- `public procedure optimize(map the_map_p, integer deprecated_max_p = 0, atom deprecated_grow_p = 0)` : Optimiert die interne Eimer-Verteilung der Hash-Map.
- `public function load_map(object input_file_name)` : Lädt eine serialisierte Map aus einer Datei (Alias für `load`).
- `public function save_map(map the_map_, object file_name_p, integer type_ = SM_TEXT)` : Speichert eine Map in eine Datei (Alias für `save`).
- `public function copy(map source_map, object dest_map=0, integer put_operation = PUT)` : Erstellt eine flache oder tiefe Kopie der angegebenen Hash-Map.
- `public function new_from_kvpairs(sequence kv_pairs)` : Erstellt eine Map direkt aus einer Liste von `{key, val}` Paaren.
- `public function new_from_string(sequence kv_string)` : Erstellt eine Map aus einem String mit Schlüssel-Wert-Zeilen.

- `public function for_each(map source_map, integer user_rid, object user_data = 0, integer in_sorted_order = 0, integer signal_boundary = 0)` : Führt eine Callback-Routine für jedes Schlüssel-Wert-Paar der Map aus.

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

- **Zweck:** Generische LIFO- und FIFO-Stack-Datenstrukturen mit Push-, Pop-, Peek-, Rotations- und Kapazitätsoperationen.
- **Öffentliche Funktionen & Prozeduren (14):**
- `public function new(integer typ = FILO)` : Erstellt einen neuen Stack (LIFO oder FIFO) mit optionaler Kapazität.
- `public function is_empty(stack sk)` : Prüft, ob der Stack leer ist (1 = leer, 0 = enthält Elemente).
- `public function size(stack sk)` : Gibt die aktuelle Anzahl der auf dem Stack gespeicherten Elemente zurück.
- `public function at(stack sk, integer idx = 1)` : Liest das Element an der angegebenen Stack-Position ohne es zu entfernen.
- `public procedure push(stack sk, object value)` : Legt ein neues Element oben auf den Stack.
- `public function top(stack sk)` : Liest das oberste Element des Stacks ohne es zu entfernen.
- `public function last(stack sk)` : Liest das unterste Element des Stacks ohne es zu entfernen.
- `public function pop(stack sk, integer idx = 1)` : Entfernt das oberste Element vom Stack und gibt es zurück.
- `public function peek_top(stack sk, integer idx = 1)` : Liest das oberste Element des Stacks sicher ab.
- `public function peek_end(stack sk, integer idx = 1)` : Liest das unterste Element des Stacks sicher ab.
- `public procedure swap(stack sk)` : Vertauscht die beiden obersten Elemente des Stacks.
- `public procedure dup(stack sk)` : Dupliziert das oberste Element des Stacks.
- `public procedure set(stack sk, object val, integer idx = 1)` : Überschreibt das Element an einer bestimmten Position auf dem Stack.
- `public procedure clear(stack sk)` : Entleert den gesamten Inhalt des Stacks.

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

- **Zweck:** Hochoptimierte Sortieralgorithmen (Merge Sort, Quicksort, Insertion Sort) mit aufsteigender/absteigender Ordnung und benutzerdefinierten Vergleichsroutinen.
- **Öffentliche Funktionen & Prozeduren (5):**
- `public function sort(sequence x, integer order = ASCENDING)` : Sortiert die Elemente einer Sequenz in aufsteigender Reihenfolge.
- `public function custom_sort(integer custom_compare, sequence x, object data = {}, integer order = NORMAL_ORDER)` : Sortiert eine Sequenz anhand einer benutzerdefinierten Vergleichsfunktion (Routine-ID).
- `public function sort_columns(sequence x, sequence column_list)` : Sortiert eine mehrdimensionale 2D-Tabelle nach einer Liste vorgegebener Spaltenindizes.
- `public function merge(sequence a, sequence b, integer compfunc = -1, object userdata = "")` : Führt zwei bereits sortierte Sequenzen zu einer gemeinsamen sortierten Sequenz zusammen.
- `public function insertion_sort(sequence s, object e = "", integer compfunc = -1, object userdata = "")` : Sortiert kleinere Sequenzen effizient mittels Insertion-Sort.

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

- **Zweck:** Umfassende Such- und Abgleichsfunktionen in Sequenzen und Strings (find_all, match_all, begins, ends, binary_search, Ersetzungen).
- **Öffentliche Funktionen & Prozeduren (18):**
- `public function find_any(object needles, sequence haystack, integer start=1)` : Findet das erste Vorkommen eines beliebigen Elements aus einer Menge in der Sequenz.
- `public function match_any(sequence needles, sequence haystack, integer start=1)` : Sucht nach dem ersten Vorkommen beliebiger Teilsequenzen in der Zielsequenz.
- `public function find_each(sequence needles, sequence haystack, integer start=1)` : Findet alle Indizes eines Elements in der Sequenz und liefert sie als Liste.
- `public function find_all(object needle, sequence haystack, integer start=1)` : Liefert alle 1-basierten Indizes, an denen ein gesuchtes Element in der Sequenz auftritt.
- `public function find_all_but( object needle, sequence haystack, integer start = 1 )` : Liefert alle Indizes von Elementen, die *nicht* mit dem Suchwert übereinstimmen.
- `public function find_nested(object needle, sequence haystack, integer flags=0, integer rtn_id=types:NO_ROUTINE_ID)` : Sucht rekursiv nach einem Element in verschachtelten Sequenzen und liefert den Indexpfad.
- `public function rfind(object needle, sequence haystack, integer start=length(haystack))` : Sucht rückwärts vom Ende der Sequenz nach einem Element.
- `public function find_replace(object needle, sequence haystack, object replacement, integer max=0)` : Ersetzt Vorkommen eines Elements in einer Sequenz durch einen neuen Wert.
- `public function match_replace(object needle, sequence haystack, object replacement, integer max=0)` : Ersetzt alle Vorkommen einer Teilsequenz in einem String/Sequenz durch eine Ersatzsequenz.
- `public function binary_search(object needle, sequence haystack, integer start_point = 1, integer end_point = 0)` : Führt eine schnelle binäre Suche O(log n) auf einer vorsortierten Sequenz durch.

- `public function match_all(sequence needle, sequence haystack, integer start=1)` : Findet alle Startpositionen einer Teilsequenz in einer Zielsequenz.
- `public function rmatch(sequence needle, sequence haystack, integer start=length(haystack))` : Sucht rückwärts vom Ende der Sequenz nach einer Teilsequenz.
- `public function begins(object sub_text, sequence full_text)` : Prüft, ob eine Sequenz oder ein String mit einer bestimmten Teilsequenz beginnt (Präfix).
- `public function ends(object sub_text, sequence full_text)` : Prüft, ob eine Sequenz oder ein String mit einer bestimmten Teilsequenz endet (Suffix).
- `public function is_in_range(object item, sequence range_limits, sequence boundries = "[]")` : Prüft, ob ein Wert innerhalb eines geschlossenen Intervalls `[min, max]` liegt.
- `public function is_in_list(object item, sequence list)` : Prüft, ob ein Element in einer vorgegebenen Elementliste enthalten ist.
- `public function lookup(object find_item, sequence source_list, sequence target_list, object def_value = 0)` : Sucht einen Schlüssel in einer 2-Spalten-Tabelle und liefert den zugehörigen Wert (Lookup).
- `public function vlookup(object find_item, sequence grid_data, integer source_col, integer target_col, object def_value = 0)` : Führt einen vertikalen Tabellenabgleich (VLOOKUP) über mehrere Spalten durch.

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

- **Zweck:** Binäre Serialisierung und Deserialisierung beliebiger, komplex verschachtelter Euphoria-Objektstrukturen zur persistenten Speicherung oder Socket-Übertragung.
- **Öffentliche Funktionen & Prozeduren (4):**
- `public function deserialize(object sdata, integer pos = 1)` : Rekonstruiert ein Euphoria-Objekt aus einem zuvor serialisierten Byte-Stream.
- `public function serialize(object x)` : Serialisiert ein beliebiges Euphoria-Objekt in einen kompakten Byte-Stream.
- `public function dump(sequence data, sequence filename)` : Serialisiert ein Euphoria-Objekt direkt in einen offenen Dateikanal.
- `public function load(sequence filename)` : Liest und deserialisiert ein Euphoria-Objekt direkt aus einer Datei.

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

- **Zweck:** Definition von Hash-Algorithmen und Enumerationskonstanten für die fest eingebaute Hashing-Engine `hash(source, algo)`.
- **Hinweis & Wichtige Konstanten:** Definiert die Hash-Algorithmen `HSIEH30`, `HSIEH32`, `ADLER32`, `FLETCHER32`, `MD5` und `SHA256`. Die Ausführung erfolgt über die fest integrierte Built-in Funktion `hash(object source, atom algo)`.

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

- **Zweck:** Erweiterte Datentypprüfungen, Zeichensatz-Validierungen (ASCII, alphanumerisch, Interpunktion, Whitespace) und boolesche Typen.
- **Öffentliche Funktionen & Prozeduren (4):**
- `public function char_test(object test_data, sequence char_set)` : Prüft ein Zeichen oder eine Zeichenkette gegen einen Zeichensatz (z. B. ASCII, Alpha).
- `public procedure set_default_charsets()` : Setzt die Standard-Zeichensatzdefinitionen auf die Werkseinstellungen zurück.
- `public function get_charsets()` : Liefert die aktuell konfigurierten Zeichensatztabellen.
- `public procedure set_charsets(sequence charset_list)` : Konfiguriert benutzerdefinierte Zeichensätze für Typüberprüfungen.

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

- **Zweck:** Bitweise Flag-Manipulation, Bitmasken-Auswertung und lesbare Textkonvertierung von Optionsflags.
- **Öffentliche Funktionen & Prozeduren (2):**
- `public function which_bit( object theValue )` : Ermittelt die 1-basierte Bitposition eines einzelnen gesetzten Bits in einer Ganzzahl.
- `public function flags_to_string(object flag_bits, sequence flag_names, integer expand_flags = 0 )` : Konvertiert eine Bitmaske anhand einer Tabelle in eine lesbare Liste von Flagnamen.

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

- **Zweck:** Primzahlgenerierung nach dem Sieb des Eratosthenes, Primzahlprüfungen und Ermittlung der nächsten Primzahl.
- **Öffentliche Funktionen & Prozeduren (3):**
- `public function calc_primes(integer approx_limit, atom time_limit_p = 10)` : Berechnet alle Primzahlen bis zu einer Obergrenze mittels Sieb des Eratosthenes.
- `public function next_prime(integer n, object fail_signal_p = -1, atom time_out_p = 1)` : Ermittelt die nächstgrößere Primzahl nach einem vorgegebenen Startwert.
- `public function prime_list(integer top_prime_p = 0)` : Gibt eine vorberechnete Liste bekannter Primzahlen zurück.

2.8.2 I/O, Dateisystem, Konsole & Betriebssystem

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

- **Zweck:** Erweiterte Datei- und Kanal-E/A: vollständiges Lesen/Schreiben von Dateien (`read_file` , `write_file`), zeilenweises Einlesen, Binärpuffer und Dateisperren.
- **Öffentliche Funktionen & Prozeduren (19):**
- `public function get_bytes(integer fn, integer n)` : Liest eine exakte Anzahl von Bytes aus einem offenen Dateikanal.
- `public function get_integer32(integer fh)` : Liest eine 32-Bit-Ganzzahl im Binärformat aus einer Datei.
- `public function get_integer16(integer fh)` : Liest eine 16-Bit-Ganzzahl im Binärformat aus einer Datei.
- `public procedure put_integer32(integer fh, atom val)` : Schreibt eine 32-Bit-Ganzzahl binär in eine Datei.
- `public procedure put_integer16(integer fh, atom val)` : Schreibt eine 16-Bit-Ganzzahl binär in eine Datei.
- `public function get_dstring(integer fh, integer delim = 0)` : Liest einen längengepräfixen dynamischen String aus einer Binärdatei.
- `public function seek(file_number fn, file_position pos)` : Setzt den Dateizeiger auf eine Byte-Position in der Datei.
- `public function where(file_number fn)` : Liefert die aktuelle Byte-Position des Dateizeigers.
- `public procedure flush(file_number fn)` : Leert die E/A-Puffer des Dateikanals und erzwingt das Schreiben auf die Festplatte.
- `public function lock_file(file_number fn, lock_type t, byte_range r={})` : Setzt eine Lese- oder Schreibsperre (Lock) auf einen Bereich einer Datei.
- `public procedure unlock_file(file_number fn, byte_range r={})` : Gibt eine zuvor gesetzte Dateisperre wieder frei.
- `public function read_lines(object file)` : Liest eine Textdatei zeilenweise in eine Sequenz von Strings ein.
- `public function process_lines(object file, integer proc, object user_data = 0)` : Verarbeitet alle Zeilen einer Datei sequentiell über eine Callback-Routine.
- `public function write_lines(object file, sequence lines)` : Schreibt eine Sequenz von Textzeilen in eine Datei (mit Zeilenumbrüchen).
- `public function append_lines(sequence file, sequence lines)` : Hängt eine Sequenz von Zeilen an das Ende einer bestehenden Textdatei an.
- `public function read_file(object file, integer as_text = BINARY_MODE)` : Liest den gesamten Inhalt einer Datei als Sequenz in den Speicher.
- `public function write_file(object file, sequence data, integer as_text = BINARY_MODE)` : Schreibt eine Sequenz von Daten vollständig in eine Datei (überschreibt bestehende Inhalte).
- `public procedure writef(object fm, object data={}, object fn = 1, object data_not_string = 0)` : Gibt formatierte Textdaten auf einem Kanal aus (modernes `printf`).
- `public procedure writefln(object fm, object data={}, object fn = 1, object data_not_string = 0)` : Gibt formatierte Textdaten mit anschließendem Zeilenumbruch aus.

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

- **Zweck:** Plattformunabhängige Dateisystemoperationen: Verzeichnisse auslesen (`dir`), rekursive Traversierung (`walk_dir`), Erstellen, Kopieren, Verschieben und Pfadanalyse.
- **Öffentliche Funktionen & Prozeduren (37):**
- `public function dir(sequence name)` : Liest Datei- und Verzeichnisinformationen (Name, Attribute, Größe, Datum) für einen Pfad aus.
- `public function current_dir()` : Gibt den absoluten Pfad des aktuellen Arbeitsverzeichnisses zurück.
- `public function chdir(sequence newdir)` : Wechselt das aktuelle Arbeitsverzeichnis auf den angegebenen Pfad.
- `public function walk_dir(sequence path_name, object your_function, integer scan_subdirs = types:FALSE, object dir_source = types:NO_ROUTINE_ID)` : Traversiert ein Verzeichnis rekursiv und führt für jeden Treffer eine Callback-Routine aus.
- `public function create_directory(sequence name, integer mode=448, integer mkparent = 1)` : Erstellt ein neues Verzeichnis im Dateisystem (optional inklusive Elternverzeichnisse).
- `public function create_file(sequence name)` : Erstellt eine neue leere Datei auf dem Dateisystem.
- `public function delete_file(sequence name)` : Löscht eine Datei vom Dateisystem.
- `public function curdir(integer drive_id = 0)` : Liefert das aktuelle Arbeitsverzeichnis (Alias für `current_dir`).
- `public function init_curdir()` : Initialisiert und synchronisiert den Verzeichniscache des Dateisystems.
- `public function clear_directory(sequence path, integer recurse = 1)` : Löscht alle Dateien und Unterverzeichnisse innerhalb eines Verzeichnisses.
- `public function remove_directory(sequence dir_name, integer force=0)` : Entfernt ein leeres oder rekursives Verzeichnis vom Dateisystem.
- `public function pathinfo(sequence path, integer std_slash = 0)` : Zerlegt einen Dateipfad in Verzeichnis, Dateinamen, Basisnamen und Dateiendung.
- `public function dirname(sequence path, integer pcd = 0)` : Gibt den Verzeichnisteil eines vollständigen Pfades zurück.
- `public function pathname(sequence path)` : Ermittelt den bereinigten Verzeichnispfad ohne Dateinamen.
- `public function filename(sequence path)` : Extrahiert den Dateinamen inklusive Endung aus einem Pfad.
- `public function filebase(sequence path)` : Extrahiert den Basisnamen einer Datei (ohne Pfad und ohne Dateiendung).
- `public function fileext(sequence path)` : Gibt die Dateiendung (Extension) eines Dateinamens zurück.
- `public function driveid(sequence path)` : Extrahiert den Laufwerksbuchstaben unter Windows oder ein leeres Präfix unter Linux.

- `public function defaultext( sequence path, sequence defext)` : Hängt eine Standard-Dateiendung an, falls der Pfad noch keine besitzt.
- `public function absolute_path(sequence filename)` : Löst einen relativen Pfad zu einem vollständigen absoluten Dateipfad auf.
- `public function canonical_path(sequence path_in, integer directory_given = 0, case_flagset_type case_flags = AS_IS)` : Löst alle symbolischen Links und relativen Segmente (`.` und `..`) zu einem kanonischen Pfad auf.
- `public function abbreviate_path(sequence orig_path, sequence base_paths = {})` : Kürzt einen absoluten Pfad relativ zum aktuellen Verzeichnis ab.
- `public function split_path(sequence fname)` : Zerlegt einen Dateipfad in seine einzelnen Verzeichniskomponenten.
- `public function join_path(sequence path_elements)` : Verbindet mehrere Verzeichnissegmente mit dem plattformspezifischen Trennzeichen (`/` oder `\`).
- `public function file_type(sequence filename)` : Ermittelt den Typ eines Eintrags (reguläre Datei, Verzeichnis, Socket, Pipe).
- `public function file_exists(object name)` : Prüft, ob eine Datei oder ein Verzeichnis auf dem Dateisystem existiert (1 = ja, 0 = nein).
- `public function file_timestamp(sequence fname)` : Liefert das letzte Änderungsdatum einer Datei als datetime-Sequenz.
- `public function copy_file(sequence src, sequence dest, integer overwrite = 0)` : Kopiert eine Datei von einer Quelle zu einem Zielpfad.
- `public function move_file(sequence src, sequence dest, integer overwrite = 0)` : Verschiebt eine Datei an einen neuen Speicherort.
- `public function rename_file(sequence old_name, sequence new_name, integer overwrite = 0)` : Benennt eine Datei oder ein Verzeichnis um.
- `public function file_length(sequence filename)` : Ermittelt die Größe einer Datei in Bytes anhand ihres Pfadnamens.
- `public function locate_file(sequence filename, sequence search_list = {}, sequence subdir = {})` : Sucht nach einer Datei entlang einer Liste von Suchpfaden.
- `public function disk_size(object disk_path)` : Gibt die Gesamtgröße des Datenträgers für den angegebenen Pfad zurück.
- `public function disk_metrics(object disk_path)` : Liefert Speicherplatzstatistiken (Gesamtkapazität und verfügbarer Speicher) für ein Dateisystem.
- `public function dir_size(sequence dir_path, integer count_all = 0)` : Berechnet die rekursive Gesamtgröße aller Dateien in einem Verzeichnis.
- `public function temp_file(sequence temp_location = "", sequence temp_prefix = "", sequence temp_extn = "_T_", integer reserve_temp = 0)` : Erstellt einen eindeutigen temporären Dateinamen im System-Temp-Verzeichnis.
- `public function checksum(sequence filename, integer size = 4, integer usename = 0, integer return_text = 0)` : Berechnet eine Prüfsumme über den gesamten Dateiinhalt.

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

- **Zweck:** Betriebssystem-Interaktion: Umgebungsvariablen (`setenv` , `getenv`), Prozess-ID (`get_pid`), Plattformerkennung und Befehlsausführung.
- **Öffentliche Funktionen & Prozeduren (7):**
- `public function instance()` : Prüft, ob bereits eine andere Instanz des Programms auf dem System aktiv ist.
- `public function get_pid()` : Gibt die Prozess-ID (PID) des aktuellen Euphorix-Prozesses zurück.
- `public function uname()` : Ermittelt Betriebssysteminformationen (Systemname, Release, Architektur).
- `public function is_win_nt()` : Prüft, ob das Betriebssystem eine moderne Windows-NT-Plattform ist.
- `public function setenv(sequence name, sequence val, integer overwrite=1)` : Setzt oder aktualisiert eine Umgebungsvariable im Prozesskontext.
- `public function unsetenv(sequence env)` : Löscht eine Umgebungsvariable aus dem Prozesskontext.
- `public procedure sleep(atom t)` : Pausiert die Ausführung des aktuellen Threads für eine angegebene Anzahl an Sekunden.

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

- **Zweck:** Deklaratives Parsing von Befehlszeilenparametern und Schaltern mit automatischer Validierung und formatierter Hilfe-Ausgabe (`show_help`).
- **Öffentliche Funktionen & Prozeduren (4):**
- `public procedure show_help(sequence opts, object add_help_rid=-1, sequence cmds = command_line(), object parse_options = {})` : Gibt eine formatierte Befehlszeilen-Hilfeseite basierend auf den Optionen aus.
- `public function cmd_parse(sequence opts, object parse_options = {}, sequence cmds = command_line())` : Parst die Argumente der Kommandozeile anhand einer deklarativen Optionsdefinition.
- `public function build_commandline(sequence cmds)` : Erstellt einen korrekt quotierten Befehlszeilenstring aus einer Liste von Argumenten.
- `public function parse_commandline(sequence cmdline)` : Zerlegt einen Befehlszeilenstring unter Berücksichtigung von Anführungszeichen in Argumente.

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

- **Zweck:** Terminal- und Konsolensteuerung: Bildschirmdimensionen, ANSI/Konsolenfarben, Cursorpositionierung, nicht-blockierende Tastenabfrage und Bildschirmpuffer.
- **Öffentliche Funktionen & Prozeduren (20):**
- `public function has_console()` : Prüft, ob der Prozess mit einem Konsolenfenster/Terminal verknüpft ist.
- `public function key_codes(object codes = 0)` : Ermittelt oder setzt die intern von Euphoria verwendeten Tastaturcodes.
- `public function set_keycodes( object kcfile )` : Ändert die standardmäßig von der Tastatur zurückgegebenen Tastencodes.

- `public procedure allow_break( types:boolean b)` : Aktiviert oder unterbindet den Programmabbruch durch Strg+C / Strg+Untbr.
- `public function check_break()` : Gibt die Anzahl der seit dem letzten Aufruf gedrückten Strg+C-Tasten zurück.
- `public function wait_key()` : Wartet blockierend auf den Tastendruck des Benutzers und liefert den Tastencode.
- `public procedure any_key(sequence prompt="Press Any Key to continue...", integer con = 1)` : Gibt einen Aufforderungstext aus und wartet auf einen beliebigen Tastendruck.
- `public procedure maybe_any_key(sequence prompt="Press Any Key to continue...", integer con = 1)` : Wartet auf einen Tastendruck, jedoch nur wenn das Programm in einem eigenen Konsolenfenster läuft.
- `public function prompt_number(sequence prompt, sequence range={})` : Fragt den Benutzer nach einer Zahleneingabe innerhalb eines optionalen Bereichs.
- `public function prompt_string(sequence prompt)` : Fordert den Benutzer zur Eingabe einer Textzeile auf.
- `public function get_screen_char(positive_atom line, positive_atom column, integer fgbg = 0)` : Liest das Zeichen und Farbattribut an einer bestimmten Bildschirmkoordinate.
- `public procedure put_screen_char(positive_atom line, positive_atom column, sequence char_attr)` : Gibt eine Zeichensequenz mit Farbattributen an einer Bildschirmposition aus.
- `public function attr_to_colors(integer attr_code)` : Zerlegt einen Farbattribut-Code in Vordergrund- und Hintergrundfarbe.
- `public function colors_to_attr(object fgbg, integer bg = 0)` : Kombiniert Vordergrund- und Hintergrundfarbe zu einem Attribut-Byte.
- `public procedure display_text_image(text_point xy, sequence text)` : Zeigt eine zweidimensionale Matrix aus Text und Attributen auf dem Bildschirm an.
- `public function save_text_image(text_point top_left, text_point bottom_right)` : Kopiert einen rechteckigen Textbereich des Bildschirmspeichers in eine Sequenz.
- `public function text_rows(positive_int rows)` : Setzt die Anzahl der Textzeilen auf dem Konsolenbildschirm.
- `public procedure cursor(integer style)` : Konfiguriert das Erscheinungsbild des Textcursors (kein Cursor, Unterstrich, Block).
- `public procedure free_console()` : Gibt das mit dem Programm verknüpfte Terminal- oder Konsolenfenster frei.
- `public procedure display( object data_in, object args = 1, integer finalnl = -918_273_645)` : Gibt strukturierte Daten formatiert an der aktuellen Cursorposition auf der Konsole aus.

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

- **Zweck:** Subprozess-Pipes und Stream-Umleitungen (STDIN, STDOUT, STDERR) mit Java 25 Virtual Threads ohne POSIX `fork()`.
- **Öffentliche Funktionen & Prozeduren (7):**
- `public function create()` : Erstellt eine neue Pipe zur Interprozesskommunikation.
- `public function exec(sequence cmd, sequence pipe = {})` : Startet einen Subprozess mit verbundenen STDIN-, STDOUT- und STDERR-Pipes.
- `public function read(atom fd, integer bytes)` : Liest Daten aus einer Pipe ohne Blockieren.
- `public function write(atom fd, sequence str)` : Schreibt Daten in eine Pipe.
- `public function close(atom fd)` : Schließt einen Pipe-Kanal.
- `public procedure kill(process p, atom signal = 15)` : Sendet ein Abbruchsignal an einen über Pipes verwalteten Prozess.
- `public function error_no()` : Liefert den letzten Betriebssystem-Fehlercode einer Pipe-Operation.

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

- **Zweck:** Parsen von Euphoria-Objektliteralen, Zahlen und Datenstrukturen aus Strings oder Dateikanälen (`value` , `get`).
- **Öffentliche Funktionen & Prozeduren (3):**
- `public function get(integer file, integer offset=0, integer answer=GET_SHORT_ANSWER)` : Liest ein wohlgeformtes Euphoria-Objektliteral aus einer Datei.
- `public function value(sequence st, integer start_point=1, integer answer=GET_SHORT_ANSWER)` : Parst eine Textdarstellung eines Euphoria-Objekts aus einem String.
- `public function defaulted_value(object st, object def, integer start_point=1)` : Parst ein Objekt aus einem String und liefert bei Syntaxfehlern einen Vorgabewert.

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

- **Zweck:** Strukturierte, hierarchisch eingerückte Textausgabe und String-Formatierung beliebig komplex verschachtelter Sequenzen.
- **Öffentliche Funktionen & Prozeduren (2):**
- `public procedure pretty_print(integer fn, object x, sequence options = PRETTY_DEFAULT )` : Gibt eine Sequenz hierarchisch formatiert mit Einrückungen auf einem Kanal aus.
- `public function pretty_sprint(object x, sequence options = PRETTY_DEFAULT )` : Formatiert eine Sequenz hierarchisch in einen lesbaren String.

2.8.3 Textverarbeitung, Unicode & Lokalisierung

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

- **Zweck:** Erweiterte String- und Textmanipulation: Whitespace-Trimming (`trim`), Auffüllen (`pad_head`, `pad_tail`), Groß-/Kleinschreibung und Formatierung.
- **Öffentliche Funktionen & Prozeduren (15):**
- `public function sprint(object x)` : Formatiert beliebige Euphoria-Objekte in ihre String-Darstellung.
- `public function trim_head(sequence source, object what=" \t\r\n", integer ret_index = 0)` : Entfernt führende Whitespace-Zeichen am Anfang eines Strings.
- `public function trim_tail(sequence source, object what=" \t\r\n", integer ret_index = 0)` : Entfernt nachfolgende Whitespace-Zeichen am Ende eines Strings.
- `public function trim(sequence source, object what=" \t\r\n", integer ret_index = 0)` : Entfernt führende und nachfolgende Leerzeichen/Tabs/Zeilenumbrüche aus einem String.
- `public procedure set_encoding_properties(sequence en = "", sequence lc = "", sequence uc = "")` : Konfiguriert die Zeichenkodierungsattribute für String-Operationen.
- `public function get_encoding_properties( )` : Liest die aktuellen Zeichenkodierungseinstellungen aus.
- `public function lower(object x)` : Wandelt alle Großbuchstaben in einem String in Kleinbuchstaben um.
- `public function upper(object x)` : Wandelt alle Kleinbuchstaben in einem String in Großbuchstaben um.
- `public function proper(sequence x)` : Formatiert Text in Groß-/Kleinschreibung nach Titelschriftart (Capitalize).
- `public function keyvalues(sequence source, object pair_delim = ";,", object kv_delim = ":=", object quotes = "\"‘ "`, object whitespace = " \t\n\r", integer haskeys = 1) `: Zerlegt eine Sequenz von` key=value`-Zeilen in eine Liste von Wertepaaren.
- `public function escape(sequence s, sequence what="\"")` : Maskiert Steuerzeichen in einem String für Literale.
- `public function quote( sequence text_in, object quote_pair = {"\"", "\""}, integer esc = -1, t_text sp = "" )` : Setzt einen String in Anführungszeichen und maskiert Sonderzeichen.
- `public function dequote(sequence text_in, object quote_pairs = {{"\"", "\""}}, integer esc = -1)` : Entfernt umschließende Anführungszeichen und demaskiert Escape-Sequenzen.
- `public function format(sequence format_pattern, object arg_list = {})` : Formatiert Werte anhand von Platzhalter-Vorlagen (Template-String).
- `public function wrap(sequence content, integer width = 78, sequence wrap_with = "\n", sequence wrap_at = " \t")` : Führt einen automatischen Zeilenumbruch (Word Wrap) für Fließtext durch.

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

- **Zweck:** Native Java 25 UTF-8 Unicode-Engine: Codepoint-Validierung, Unicode-Zeichenkettenlänge, zeichenweises Slicing und Zeichensatzkonvertierungen.
- **Öffentliche Funktionen & Prozeduren (16):**
- `public function valid(sequence s)` : Prüft, ob eine Sequenz von Bytes eine valide UTF-8 kodierte Zeichenfolge darstellt.
- `public function len(sequence s)` : Gibt die tatsächliche Anzahl der Unicode-Codepoints (Zeichen) in einem UTF-8-String zurück.
- `public function char_at(sequence s, integer idx)` : Liefert den Unicode-Codepoint am 1-basierten Zeichenindex.
- `public function slice(sequence s, integer start_cp, integer end_cp)` : Schneidet einen Teilbereich eines UTF-8-Strings basierend auf Codepoint-Grenzen aus.
- `public function to_codepoints(sequence s)` : Konvertiert einen UTF-8 Byte-String in eine Sequenz von 32-Bit Unicode-Codepoints.
- `public function from_codepoints(object cp_seq)` : Erzeugt einen validen UTF-8 kodierten Byte-String aus einer Liste von Unicode-Codepoints.
- `public function upper(sequence s)` : Wandelt UTF-8 Strings unter voller Beachtung internationaler Unicode-Regeln in Großbuchstaben um.
- `public function lower(sequence s)` : Wandelt UTF-8 Strings unter voller Beachtung internationaler Unicode-Regeln in Kleinbuchstaben um.
- `public function find(sequence needle, sequence haystack, integer start_cp = 1)` : Findet das erste Vorkommen eines Zeichens in einem UTF-8 String.
- `public function byte_length(sequence s)` : Gibt die Rohgröße eines UTF-8 Strings in Bytes zurück.
- `public function strip_bom(sequence s)` : Entfernt das UTF-8 Byte Order Mark (BOM), falls am Anfang vorhanden.
- `public function getc_utf8(integer fn)` : Liest das nächste vollständige UTF-8-Zeichen als String aus einer Datei.
- `public function getc_codepoint(integer fn)` : Liest das nächste Unicode-Zeichen als 32-Bit-Ganzzahl aus einer Datei.
- `public function read_file(sequence path, integer remove_bom = 1)` : Liest eine UTF-8 Textdatei vollständig unter automatischer BOM-Bereinigung.
- `public function read_lines(sequence path, integer remove_bom = 1)` : Liest eine UTF-8 Textdatei zeilenweise als Unicode-Strings ein.
- `public function write_file(sequence path, sequence content, integer add_bom = 0)` : Schreibt eine Unicode-Zeichenkette als valides UTF-8 in eine Datei.

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

- **Zweck:** Reguläre Ausdrücke (Perl- und Java-kompatibel): Kompilierung, Pattern-Matching, Suchen, Ersetzen und Token-Splitting.
- **Öffentliche Funktionen & Prozeduren (17):**

- `public function option_spec_to_string(option_spec o)` : Formatiert Regex-Kompilierungsoptionen als lesbaren String.
- `public function error_to_string(integer i)` : Übersetzt einen internen Regex-Fehlercode in eine verständliche Fehlermeldung.
- `public function new(string pattern, option_spec options=DEFAULT)` : Kompiliert einen regulären Ausdruck mit optionalen Flags (z. B. Case-Insensitive).
- `public function error_message(object re)` : Liefert die textuelle Beschreibung des letzten Regex-Kompilierungsfehlers.
- `public function escape(string s)` : Maskiert alle regulären Metazeichen in einem Suchstring für Literalsuchen.
- `public function get_ovector_size(regex ex, integer maxsize=0)` : Gibt die interne Puffergröße für Treffergruppen zurück.
- `public function find(regex re, string haystack, integer from=1, option_spec options=DEFAULT, integer size = get_ovector_size(re, 30))` : Findet das erste Vorkommen eines Regex-Musters in einer Zeichenkette.
- `public function find_all(regex re, string haystack, integer from=1, option_spec options=DEFAULT, integer size = get_ovector_size(re, 30))` : Findet alle nicht-überlappenden Treffer eines Musters in einem String.
- `public function has_match(regex re, string haystack, integer from=1, option_spec options=DEFAULT)` : Prüft schnell, ob ein regulärer Ausdruck im Zieltext mindestens einen Treffer erzielt.
- `public function is_match(regex re, string haystack, integer from=1, option_spec options=DEFAULT)` : Prüft, ob der reguläre Ausdruck den gesamten Zieltext exakt matcht.
- `public function matches(regex re, string haystack, integer from=1, option_spec options=DEFAULT)` : Extrahiert alle Capture-Gruppen des ersten Treffers als Sequenz.
- `public function all_matches(regex re, string haystack, integer from=1, option_spec options=DEFAULT)` : Extrahiert alle Capture-Gruppen aller Treffer im Zielstring.
- `public function split(regex re, string text, integer from=1, option_spec options=DEFAULT)` : Zerlegt einen String an Treffern des regulären Ausdrucks in Teilstrings.
- `public function split_limit(regex re, string text, integer limit=0, integer from=1, option_spec options=DEFAULT)` : Zerlegt einen String anhand eines Musters bis zu einer maximalen Anzahl an Teilstrings.
- `public function find_replace(regex ex, string text, sequence replacement, integer from=1, option_spec options=DEFAULT)` : Ersetzt alle Vorkommen eines Musters durch einen Ersatzstring.
- `public function find_replace_limit(regex ex, string text, sequence replacement, integer limit, integer from=1, option_spec options=DEFAULT)` : Ersetzt maximal N Vorkommen eines Regex-Musters.
- `public function find_replace_callback(regex ex, string text, integer rid, integer limit=0, integer from=1, option_spec options=DEFAULT)` : Ersetzt Treffer dynamisch durch den Rückgabewert einer benutzerdefinierten Callback-Funktion.

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

- **Zweck:** Einfacher Wildcard-Musterabgleich (`*` und `?`) für Dateinamen, Pfade und Filterstrings.
- **Öffentliche Funktionen & Prozeduren (1):**
- `public function is_match(sequence pattern, sequence string)` : Prüft, ob eine Zeichenkette einem Wildcard-Muster mit `*` und `?` entspricht.

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

- **Zweck:** RFC 4648-konforme Base64-Kodierung und -Dekodierung für Binärdaten, Strings und Dokumente mit optionalem Zeilenumbruch.
- **Öffentliche Funktionen & Prozeduren (2):**
- `public function encode(sequence in, integer wrap_column = 0)` : Kodiert eine Sequenz von Bytes in einen RFC 4648 Base64-String (optional mit Zeilenumbruch).
- `public function decode(sequence in)` : Dekodiert einen Base64-String zurück in die ursprünglichen Binärdaten oder Text.

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

- **Zweck:** Konvertierung zwischen Datentypen: Integers zu Bytes/Bits, IEEE 754 Floats (32/64/80 Bit), Hexadezimalstrings und Zahlenformate.
- **Öffentliche Funktionen & Prozeduren (15):**
- `public function int_to_bytes(atom x, integer size = 4 )` : Zerlegt eine Ganzzahl in eine Sequenz von 4 oder 8 Bytes (Little-Endian).
- `public function bytes_to_int(sequence s)` : Setzt eine Byte-Sequenz zu einer vorzeichenbehafteten Ganzzahl zusammen.
- `public function int_to_bits(atom x, integer nbits = 32)` : Wandelt eine Ganzzahl in eine Sequenz binärer Bits (0 und 1) um.
- `public function bits_to_int(sequence bits)` : Wandelt eine Bitsequenz zurück in eine Ganzzahl.
- `public function atom_to_float64(atom a)` : Konvertiert eine Zahl in eine 8-Byte IEEE 754 Double-Precision Sequenz.
- `public function atom_to_float80(atom a)` : Konvertiert eine Zahl in eine 10-Byte Extended-Precision Sequenz.
- `public function float80_to_atom( sequence bytes )` : Liest ein Atom aus einer 10-Byte Fließkommadarstellung.
- `public function atom_to_float32(atom a)` : Konvertiert eine Zahl in eine 4-Byte IEEE 754 Single-Precision Sequenz.
- `public function float64_to_atom(sequence_8 ieee64)` : Liest ein Atom aus einer 8-Byte IEEE 754 Sequenz.
- `public function float32_to_atom(sequence_4 ieee32)` : Liest ein Atom aus einer 4-Byte IEEE 754 Sequenz.
- `public function hex_text(sequence text)` : Konvertiert eine Zahl in eine formatierte Hexadezimal-Zeichenkette.
- `public function set_decimal_mark(integer new_mark)` : Konfiguriert das Dezimaltrennzeichen (Punkt oder Komma).
- `public function to_number( sequence text_in, integer return_bad_pos = 0)` : Parst einen String in ein Atom oder Integer.
- `public function to_integer(object data_in, integer def_value = 0)` : Konvertiert ein Objekt sicher in einen Integer-Wert.

- `public function to_string(object data_in, integer string_quote = 0, integer embed_string_quote = '"')` : Wandelt beliebige Datenstrukturen in eine formatierte Stringdarstellung um.

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

- **Zweck:** Internationalisierung (i18n) und Lokalisierung: Laden von Übersetzungsdateien, Sprachauswahl und lokalisierte String-Routinen.
- **Öffentliche Funktionen & Prozeduren (13):**
- `public procedure set_lang_path(object pp)` : Setzt den Verzeichnispfad zu den Lokalisierungsdateien.
- `public function get_lang_path()` : Gibt den Pfad zu den Lokalisierungsverzeichnissen zurück.
- `public function lang_load(sequence filename)` : Lädt eine Sprachdatei für eine bestimmte Sprache in den Speicher.
- `public procedure set_def_lang( object langmap )` : Setzt die Standard-Sprachkennung der Anwendung.
- `public function get_def_lang( )` : Gibt die aktuell eingestellte Standardsprache zurück.
- `public function translate(sequence word, object langmap = 0, object defval="", integer mode = 0)` : Übersetzt einen Textstring in die aktive Sprache.
- `public function trsprintf(sequence fmt, sequence data, object langmap = 0)` : Formatiert einen übersetzten Text mit Platzhaltern.
- `public function set(sequence new_locale)` : Setzt eine Locale-Konfigurationsoption.
- `public function get()` : Liest eine Locale-Konfigurationsoption aus.
- `public function money(object amount)` : Formatiert einen Geldbetrag mit Währungssymbol nach Locale-Regeln.
- `public function number(object num)` : Formatiert eine Zahl nach den landesspezifischen Tausender- und Dezimaltrennzeichen.
- `public function datetime(sequence fmt, datetime:datetime dtm)` : Formatiert ein Datum nach den länderspezifischen Konventionen des Locales.
- `public function get_text(integer MsgNum, sequence LocalQuals = {}, sequence DBBase = "teksto")` : Alias für `get_message` – liest übersetzten Text aus.

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

- **Zweck:** Kanonische Lokalisierungsbezeichner, Sprach- und Ländercodes sowie Konvertierung zwischen Unix- und Windows-Locale-Formaten.
- **Öffentliche Funktionen & Prozeduren (3):**
- `public function canonical(sequence new_locale)` : Konvertiert einen plattformspezifischen Locale-String in das standardisierte kanonische Format.
- `public function decanonical(sequence new_locale)` : Wandelt ein kanonisches Locale in das plattformspezifische Format um.
- `public function canon2win(sequence new_locale)` : Konvertiert ein kanonisches Locale in den entsprechenden Windows-Namen.

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

- **Zweck:** Windows Locale Identifier (LCID) Zuordnungstabellen und Abfragen für länderspezifische Windows-Einstellungen.
- **Öffentliche Funktionen & Prozeduren (1):**
- `public function get_lcid(sequence name)` : Ermittelt die numerische Windows LCID für einen ISO-Sprachcode.

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

- **Zweck:** Wissenschaftliche Fließkommanotation: Konvertierung von Exponentialstrings (`1.23e-04`) in Atome und umgekehrt.
- **Öffentliche Funktionen & Prozeduren (4):**
- `public function bits_to_bytes( sequence bits )` : Konvertiert eine Bit-Sequenz in Byte-Werte.
- `public function bytes_to_bits( sequence bytes )` : Zerlegt Bytes in einzelne Bits.
- `public function scientific_to_float( sequence s, floating_point fp = NATIVE )` : Parst einen wissenschaftlich notierten String in eine IEEE-Float-Sequenz.
- `public function scientific_to_atom( sequence s, floating_point fp = NATIVE )` : Konvertiert einen Exponentialstring (`1.23e-4`) in ein numerisches Atom.

2.8.4 Mathematik, Statistik & Zufall

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

- **Zweck:** Umfassende mathematische Funktionen: Trigonometrie, Hyperbelfunktionen, Logarithmen, Wurzeln, Rundungen, Modulo, GCD und LCM.
- **Öffentliche Funktionen & Prozeduren (38):**
- `public function abs(object a)` : Berechnet den absoluten Betrag (Betragswert) einer Zahl oder Sequenz.
- `public function sign(object a)` : Liefert das Vorzeichen einer Zahl (-1 bei negativ, 0 bei null, +1 bei positiv).
- `public function larger_of(object objA, object objB)` : Liefert den größeren von zwei Werten.
- `public function smaller_of(object objA, object objB)` : Liefert den kleineren von zwei Werten.
- `public function max(object a)` : Ermittelt den Maximalwert aus allen Elementen einer Sequenz.
- `public function min(object a)` : Ermittelt den Minimalwert aus allen Elementen einer Sequenz.

- `public function ensure_in_range(object item, sequence range_limits)` : Begrenzt einen Wert auf das geschlossene Intervall `[min, max]` (Clamp).
- `public function ensure_in_list(object item, sequence list, integer default=1)` : Stellt sicher, dass ein Wert in einer erlaubten Liste vorkommt.
- `public function mod(object x, object y)` : Berechnet den Divisionsrest nach mathematischer Definition (Modulo).
- `public function trunc(object x)` : Schneidet die Nachkommastellen einer Zahl ab (Rundung Richtung Null).
- `public function frac(object x)` : Liefert den Nachkommateil einer Fließkommazahl.
- `public function intdiv(object a, object b)` : Führt eine ganzzahlige Division (Integer-Division) durch.
- `public function ceil(object a)` : Rundet eine Zahl auf die nächsthöhere Ganzzahl auf.
- `public function round(object a, object precision=1)` : Rundet eine Fließkommazahl kaufmännisch auf die angegebene Anzahl an Dezimalstellen.
- `public function arccos(trig_range x)` : Alias für `acos` – berechnet den Arkuskosinus.
- `public function arcsin(trig_range x)` : Alias für `asin` – berechnet den Arkussinus.
- `public function atan2(atom y, atom x)` : Berechnet den vier-quadranten Arkustangens von y und x.
- `public function rad2deg (object x)` : Rechnet Bogenmaß (Radiant) in Gradmaße um.
- `public function deg2rad (object x)` : Rechnet Gradmaße in Bogenmaß (Radiant) um.
- `public function log10(object x1)` : Berechnet den dekadischen Logarithmus (Basis 10).
- `public function exp(atom x)` : Berechnet die Exponentialfunktion e^x.
- `public function fib(integer i)` : Berechnet die n-te Fibonacci-Zahl.
- `public function cosh(object a)` : Berechnet den Kosinus Hyperbolicus.
- `public function sinh(object a)` : Berechnet den Sinus Hyperbolicus.
- `public function tanh(object a)` : Berechnet den Tangens Hyperbolicus.
- `public function arcsinh(object a)` : Alias für `asinh` – berechnet den Areal-Sinus Hyperbolicus.
- `public function arccosh(not_below_1 a)` : Alias für `acosh` – berechnet den Areal-Kosinus Hyperbolicus.
- `public function arctanh(abs_below_1 a)` : Alias für `atanh` – berechnet den Areal-Tangens Hyperbolicus.
- `public function sum(object a)` : Berechnet die Summe aller Elemente in einer Sequenz.
- `public function product(object a)` : Berechnet das Produkt aller Elemente in einer Sequenz.
- `public function or_all (object a)` : Führt eine bitweise ODER-Verknüpfung über alle Elemente einer Sequenz aus.
- `public function shift_bits(object source_number, integer shift_distance)` : Verschiebt die Bits einer Zahl um eine Anzahl an Positionen nach links oder rechts.
- `public function rotate_bits(object source_number, integer shift_distance)` : Rotiert die Bits einer Zahl zyklisch.
- `public function gcd(atom p, atom q)` : Berechnet den größten gemeinsamen Teiler (ggT) zweier Zahlen.
- `public function approx(object p, object q, atom epsilon = 0.005)` : Prüft zwei Zahlen auf approximative Gleichheit innerhalb einer Toleranz epsilon.
- `public function powof2(object p)` : Prüft, ob eine Ganzzahl eine Zweierpotenz (2^k) ist.
- `public function is_even(integer test_integer)` : Prüft, ob eine Ganzzahl gerade ist (1 = ja, 0 = ungerade).
- `public function is_even_obj(object test_object)` : Prüft für alle Elemente eines Objekts/Sequenz, ob sie gerade Ganzzahlen sind.

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

- **Zweck:** Mathematische Naturkonstanten und Grenzen für wissenschaftliche Berechnungen.
- **Hinweis & Wichtige Konstanten:** Definiert Standardkonstanten: `PI`, `HALFPI`, `QUARTPI`, `TWOPI`, `E`, `SQRT2`, `SQRT3`, `LN2`, `LN10`, `DEG_TO_RAD`, `RAD_TO_DEG`. Enthält keine Routinen, sondern ausschließlich typisierte Konstanten.

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

- **Zweck:** Statistische Auswertungsroutinen: Mittelwert, Median, Modus, Varianz, Standardabweichung, Minimum/Maximum und Perzentile.
- **Öffentliche Funktionen & Prozeduren (20):**
- `public function small(sequence data_set, integer ordinal_idx)` : Ermittelt das k-t-kleinste Element einer Zahlenfolge.
- `public function largest(object data_set)` : Ermittelt die k größten Werte einer Sequenz.
- `public function smallest(object data_set)` : Ermittelt die k kleinsten Werte einer Sequenz.
- `public function range(object data_set)` : Berechnet die Spannweite (Differenz zwischen Maximum und Minimum).
- `public function stdev(sequence data_set, object subseq_opt = ST_ALLNUM, integer population_type = ST_SAMPLE)` : Berechnet die Stichproben-Standardabweichung einer Messreihe.
- `public function avedev(sequence data_set, object subseq_opt = ST_ALLNUM, integer population_type = ST_SAMPLE)` : Berechnet die mittlere absolute Abweichung vom Mittelwert.
- `public function sum(object data_set, object subseq_opt = ST_ALLNUM)` : Berechnet die Summe aller Elemente einer statistischen Reihe.
- `public function count(object data_set, object subseq_opt = ST_ALLNUM)` : Zählt die Anzahl gültiger numerischer Werte in einer Sequenz.
- `public function average(object data_set, object subseq_opt = ST_ALLNUM)` : Berechnet das arithmetische Mittel aller Zahlen.
- `public function geomean(object data_set, object subseq_opt = ST_ALLNUM)` : Berechnet das geometrische Mittel einer Stichprobe.

- `public function harmean(sequence data_set, object subseq_opt = ST_ALLNUM)` : Berechnet das harmonische Mittel einer Stichprobe.
- `public function movavg(object data_set, object period_delta)` : Berechnet den gleitenden Durchschnitt über ein Zeitfenster.
- `public function emovavg(object data_set, atom smoothing_factor)` : Berechnet den exponentiell geglätteten gleitenden Durchschnitt.
- `public function median(object data_set, object subseq_opt = ST_ALLNUM)` : Ermittelt den Median (Zentralwert) einer sortierten Datenreihe.
- `public function raw_frequency(object data_set, object subseq_opt = ST_ALLNUM)` : Erstellt eine Häufigkeitstabelle der in der Sequenz vorkommenden Werte.
- `public function mode(sequence data_set, object subseq_opt = ST_ALLNUM)` : Ermittelt den Modalwert (häufigsten Wert) einer Datenreihe.
- `public function central_moment(sequence data_set, object datum, integer order_mag = 1, object subseq_opt = ST_ALLNUM)` : Berechnet das k-te zentrale Moment einer Verteilung.
- `public function sum_central_moments(object data_set, integer order_mag = 1, object subseq_opt = ST_ALLNUM)` : Berechnet die Summe zentraler Momente.
- `public function skewness(object data_set, object subseq_opt = ST_ALLNUM)` : Berechnet die Schiefe einer statistischen Verteilung.
- `public function kurtosis(object data_set, object subseq_opt = ST_ALLNUM)` : Berechnet die Wölbung (Kurtosis) einer Verteilung.

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

- **Zweck:** Zufallszahlengeneratoren: Pseudozufall, kryptographisch sichere Zufallsfolgen, Bereichsauswahl und Array-Permutationen.
- **Öffentliche Funktionen & Prozeduren (8):**
- `public function rand_range(atom lo, atom hi)` : Erzeugt eine gleichverteilte Zufalls-Ganzzahl im geschlossenen Bereich `[min, max]`.
- `public function rnd()` : Erzeugt eine gleichverteilte Zufallszahl als Atom zwischen 0.0 und 1.0.
- `public function rnd_1()` : Erzeugt eine Zufalls-Gleitkommazahl.
- `public procedure set_rand(object seed)` : Initialisiert den Zufallszahlengenerator mit einem festen Seed-Wert.
- `public function get_rand()` : Liest den aktuellen internen Seed-Zustand des Zufallsgenerators aus.
- `public function chance(atom my_limit, atom top_limit = 100)` : Liefert mit der Wahrscheinlichkeit `p` den Wert 1, andernfalls 0.
- `public function roll(object desired, integer sides = 6)` : Simuliert das Werfen mehrerer Würfel (z. B. `3d6`).
- `public function sample(sequence population, integer sample_size, integer sampling_method = 0)` : Zieht eine zufällige Stichprobe von Elementen aus einer Sequenz.

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

- **Zweck:** Fließkomma-Rechenumgebung (Floating-Point Environment): Abfrage und Steuerung von Fließkomma-Ausnahmeflags (Division durch Null, Überlauf).
- **Öffentliche Funktionen & Prozeduren (3):**
- `public function clear(object e_p)` : Löscht alle anstehenden Fließkomma-Ausnahmeflags.
- `public function raise(object e_p)` : Löst manuell eine Fließkomma-Ausnahme (FE_DIVBYZERO, FE_OVERFLOW, FE_INVALID, etc.) aus.
- `public function test(object e_p)` : Prüft, ob bestimmte Fließkomma-Ausnahmeflags im aktuellen Rechenkontext gesetzt sind.

2.8.5 Datum, Uhrzeit & Kalender

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

- **Zweck:** Umfassendes Datums- und Zeitmodul: Zeitstempel (`now`), ISO 8601-Formatierung, Datumsarithmetik (`add`, `subtract`), Differenzen und Zeitzonen.
- **Öffentliche Funktionen & Prozeduren (17):**
- `public function from_date(sequence src)` : Konvertiert eine von der Built-in-Funktion `date()` zurückgegebene Sequenz in ein datetime-Objekt.
- `public function now()` : Erzeugt ein neues datetime-Objekt, initialisiert mit aktuellem lokalem Datum und Uhrzeit.
- `public function now_gmt()` : Erzeugt ein neues datetime-Objekt in koordinierter Weltzeit (GMT/UTC).
- `public function new(integer year=0, integer month=0, integer day=0, integer hour=0, integer minute=0, atom second=0)` : Erstellt ein neues datetime-Objekt aus den angegebenen Jahres-, Monats-, Tages- und Zeitwerten.
- `public function new_time(integer hour, integer minute, atom second)` : Erstellt ein Zeitobjekt mit den angegebenen Stunden, Minuten und Sekunden.
- `public function weeks_day(datetime dt)` : Liefert den Wochentag (1 = Sonntag bis 7 = Samstag) für das gegebene Datum.
- `public function years_day(datetime dt)` : Liefert den Tag des Jahres (1 bis 366).
- `public function is_leap_year(datetime dt)` : Prüft, ob das Jahr des Datums ein Schaltjahr ist (1 = ja, 0 = nein).
- `public function days_in_month(datetime dt)` : Gibt die Anzahl der Tage des Monats unter Berücksichtigung von Schaltjahren zurück.
- `public function days_in_year(datetime dt)` : Gibt die Gesamtanzahl der Tage im Jahr (365 oder 366) zurück.
- `public function to_unix(datetime dt)` : Konvertiert ein datetime-Objekt in einen Unix-Timestamp (Sekunden seit 1970).
- `public function from_unix(atom unix)` : Konvertiert einen Unix-Timestamp in ein reguläres datetime-Objekt.
- `public function format(datetime d, sequence pattern = "%Y-%m-%d %H:%M:%S")` : Formatiert ein Datum gemäß dem angegebenen Formatmuster (z. B. `%Y-%m-%d %H:%M:%S`).

- `public function parse(sequence val, sequence fmt="%Y-%m-%d %H:%M:%S", integer yylower = -80)` : Parst einen formatierten Datumsstring anhand einer Mustervorlage in ein datetime-Objekt.
- `public function add(datetime dt, object qty, integer interval)` : Addiert eine Zeitspanne (Tage, Stunden, Minuten, Sekunden) zu einem Datum hinzu.
- `public function subtract(datetime dt, atom qty, integer interval)` : Subtrahiert eine Zeitspanne von einem Datum.
- `public function diff(datetime dt1, datetime dt2)` : Berechnet die Zeitdifferenz zwischen zwei Zeitpunkten in Sekunden.

2.8.6 Netzwerk, Internet, DNS & URLs

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

- **Zweck:** Dual-Stack IPv4/IPv6 Socket-API: Erstellung von TCP/UDP Server- und Client-Sockets, non-blocking I/O, Multiplexing (`select`) und Timeouts.
- **Öffentliche Funktionen & Prozeduren (18):**
- `public function error_code()` : Liefert den letzten socket-spezifischen Fehlercode.
- `public function create(integer family, integer sock_type, integer protocol)` : Erstellt einen neuen Socket (AF_INET/AF_INET6, SOCK_STREAM/SOCK_DGRAM).
- `public function close(socket sock)` : Schließt einen offenen Socket.
- `public function shutdown(socket sock, atom method=SD_BOTH)` : Beendet selektiv das Senden, Empfangen oder beides auf einem Socket.
- `public function select(object sockets_read, object sockets_write, object sockets_err, integer timeout=0, integer timeout_micro=0)` : Überwacht mehrere Sockets auf Lese-, Schreib- oder Fehlerbereitschaft mit Timeout.
- `public function send(socket sock, sequence data, atom flags=0)` : Sendet Daten über eine bestehende Stream-Socketverbindung.
- `public function receive(socket sock, atom flags=0)` : Liest Daten aus einem verbundenen Socket mit Timeout.
- `public function get_option(socket sock, integer level, integer optname)` : Liest Socket-Optionen (z. B. SO_REUSEADDR, SO_TIMEOUT) aus.
- `public function set_option(socket sock, integer level, integer optname, object val)` : Konfiguriert Socket-Optionen am Socket-Handle.
- `public function connect(socket sock, sequence address, integer port=-1)` : Stellt eine ausgehende Verbindung zu einem Host und Port her.
- `public function bind(socket sock, sequence address, integer port=-1)` : Bindet einen Server-Socket an eine lokale IP-Adresse und einen Port.
- `public function listen(socket sock, integer backlog)` : Versetzt einen Socket in den passiven Abhörmodus für eingehende Clientverbindungen.
- `public function accept(socket sock)` : Nimmt eine eingehende Client-Verbindung an und liefert den Client-Socket zurück.
- `public function send_to(socket sock, sequence data, sequence address, integer port=-1, atom flags=0)` : Sendet ein UDP-Datagramm an eine angegebene Zieladresse und Port.
- `public function receive_from(socket sock, atom flags=0)` : Empfängt ein UDP-Datagramm inklusive Absenderadresse.
- `public function service_by_name(sequence name, object protocol=0)` : Ermittelt die Portnummer zu einem Dienstnamen (z. B. `http` -> 80).
- `public function service_by_port(integer port, object protocol=0)` : Ermittelt den Dienstnamen zu einer Portnummer.
- `public function info(integer Type)` : Liefert Status- und Verbindungsinformationen zu einem Socket.

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

- **Zweck:** Hochmoderner HTTP/HTTPS Client & Server mit Virtual Threads: REST-Anfragen (GET, POST, PUT, DELETE), SSL/TLS, Header, Query-Params und Multithreading.
- **Öffentliche Funktionen & Prozeduren (20):**
- `public function request(sequence method, sequence url, sequence body = "", sequence headers = {}, integer timeout_ms = 10000, integer verify_ssl = 1)` : Führt einen allgemeinen HTTP-Request mit frei wählbarer Methode, Headern und Body aus.
- `public function get(sequence url, sequence headers = {}, integer verify_ssl = 1)` : Sendet einen HTTP-GET Request an eine URL und liefert Statuscode, Header und Body zurück.
- `public function post(sequence url, sequence body, sequence headers = {}, integer verify_ssl = 1)` : Sendet einen HTTP-POST Request mit Request-Body an eine URL.
- `public function post_json(sequence url, sequence json_body, sequence headers = {}, integer verify_ssl = 1)` : Sendet ein JSON-Objekt mit passendem `Content-Type: application/json` Header per POST.
- `public function put(sequence url, sequence body, sequence headers = {}, integer verify_ssl = 1)` : Führt einen HTTP-PUT Request zur Ressourcenaktualisierung aus.
- `public function delete_req(sequence url, sequence headers = {}, integer verify_ssl = 1)` : Führt einen HTTP-DELETE Request zum Löschen einer Ressource aus.
- `public function header_get(sequence headers, sequence name, sequence default_val = "")` : Liest den Wert eines bestimmten Headers aus einer Response-Header-Sequenz.
- `public function server_create(sequence host = "127.0.0.1", integer port = 8080)` : Erstellt einen neuen HTTP-Server auf einem angegebenen TCP-Port.
- `public function server_create_tls(sequence host, integer port, sequence cert_pem, sequence key_pem)` : Erstellt einen verschlüsselten HTTPS-Server mit SSL/TLS Zertifikaten.

- `public procedure route(integer srv_id, sequence method, sequence path, integer handler_rid)` : Registriert eine URL-Route und Methode mit einer zugehörigen Handler-Funktion.
- `public procedure clear_routes(integer srv_id = 0)` : Entfernt alle registrierten Routen vom HTTP-Server.
- `public function response(integer status, sequence body, sequence content_type = "text/plain; charset=utf-8", sequence extra_headers = {})` : Erzeugt eine standardisierte HTTP-Antwort mit Statuscode, Content-Type und Body.
- `public function response_json(integer status, sequence json_body, sequence extra_headers = {})` : Erzeugt eine HTTP-Antwort mit serialisiertem JSON-Inhalt und korrektem MIME-Type.
- `public function response_html(integer status, sequence html_body, sequence extra_headers = {})` : Erzeugt eine HTML-Antwort mit `text/html; charset=utf-8` Header.
- `public function accept_request(integer srv_id, integer timeout_ms = 5000)` : Nimmt eine eingehende HTTP-Anfrage auf dem Server-Socket entgegen.
- `public function reply(integer conn_id, sequence resp)` : Sendet eine HTTP-Antwort zurück an den anfragenden Client.
- `public function handle_one(integer srv_id, integer timeout_ms = 5000)` : Bearbeitet eine einzelne Client-Anfrage auf einem Worker-Thread.
- `public procedure serve_forever(integer srv_id)` : Startet die Hauptschleife des HTTP-Servers und bedient Anfragen blockierend.
- `public function server_background(integer srv_id, integer max_requests = 0)` : Startet den HTTP-Server als asynchronen Virtual-Thread-Hintergrunddienst.
- `public function server_close(integer srv_id, integer bg_pid = 0)` : Fährt den HTTP-Server kontrolliert herunter und schließt den Port.

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

- **Zweck:** DNS-Namensauflösung: Vorwärts- und Rückwärtsauflösung von Hostnamen und IP-Adressen (IPv4 und IPv6).
- **Öffentliche Funktionen & Prozeduren (2):**
- `public function host_by_name(sequence name)` : Löst einen Hostnamen (Domain) in seine zugehörigen IPv4/IPv6-Adressen auf.
- `public function host_by_addr(sequence address)` : Führt einen Reverse-DNS-Lookup durch, um den Hostnamen zu einer IP zu ermitteln.

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

- **Zweck:** URL-Verarbeitung nach RFC 3986: Zerlegung in Schema, Host, Port, Pfad, Query-Parameter und URL-Encoding/-Decoding.
- **Öffentliche Funktionen & Prozeduren (4):**
- `public function parse_querystring(object query_string)` : Parst einen URL-Query-String (`a=1&b=2`) in eine Key-Value-Map/Sequenz.
- `public function parse(sequence url, integer querystring_also=0)` : Zerlegt eine URL nach RFC 3986 in Protokoll, Benutzer, Host, Port, Pfad und Query.
- `public function encode(sequence what, sequence spacecode="+")` : Enkodiert Sonderzeichen in einer URL gemäß URL-Percent-Encoding.
- `public function decode(sequence what)` : Dekodiert URL-Percent-Sequenzen (`%20` -> Leerzeichen) zurück in Klartext.

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

- **Zweck:** Gemeinsame Netzwerk-Hilfsfunktionen: Validierung von IP-Adressen, Socket-Hilfskonstanten und Adressformate.
- **Öffentliche Funktionen & Prozeduren (3):**
- `public function is_inetaddr(sequence address)` : Prüft, ob ein String eine wohlgeformte IPv4- oder IPv6-Adresse ist.
- `public function parse_ip_address(sequence address, integer port = -1)` : Zerlegt eine textuelle IP-Adresse in ihre numerischen Komponenten.
- `public function parse_url(sequence url)` : Allgemeines Parsing und Validierung von Netzwerk-Adressen.

2.8.7 Datenformate, XML, JSON & Datenbanken

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

- **Zweck:** Hochgeschwindigkeits-JSON-Parser, Builder und RFC 9535 JSONPath-Engine auf Basis von Jackson und Java 25.
- **Öffentliche Funktionen & Prozeduren (22):**
- `public function null()` : Erzeugt einen typisierten JSON-Null-Wert (`JSON_NULL`).
- `public function bool(integer b)` : Erzeugt einen typisierten JSON-Boolean-Wert (`JSON_TRUE` oder `JSON_FALSE`).
- `public function str(sequence s)` : Erzeugt einen expliziten typisierten JSON-String.
- `public function array(sequence items = {})` : Erzeugt ein typisiertes JSON-Array aus einer Sequenz von Elementen.
- `public function object_new(sequence pairs = {})` : Erzeugt ein neues JSON-Objekt aus Schlüssel-Wert-Paaren.
- `public function type_of(object v)` : Ermittelt den JSON-Knotentyp eines Objekts (`JSON_OBJECT` , `JSON_ARRAY` , etc.).
- `public function is_null(object v)` : Prüft, ob ein Objekt ein JSON-Null-Wert ist.
- `public function is_bool(object v)` : Prüft, ob ein Objekt ein JSON-Boolean ist.
- `public function is_object(object v)` : Prüft, ob ein Objekt ein JSON-Objekt (Map) ist.
- `public function is_array(object v)` : Prüft, ob ein Objekt ein JSON-Array ist.
- `public function unwrap(object v)` : Konvertiert ein internes JSON-Knotenobjekt rekursiv in native Euphoria-Datenstrukturen.

- `public function parse(sequence json_str)` : Parst einen JSON-String in ein natives Euphoria-Objekt (Maps und Sequenzen).
- `public function decode(sequence json_str)` : Dekodiert einen JSON-String (Alias für `parse`).
- `public function stringify(object val, integer indent = 0)` : Serialisiert ein Euphoria-Objekt in einen kompakten oder eingerückten JSON-String.
- `public function encode(object val, integer indent = 0)` : Serialisiert ein Euphoria-Objekt in einen JSON-String (Alias für `stringify`).
- `public function get_node(object doc, object path, object default_val = 0)` : Liefert das rohe interne JSON-Knotenobjekt an einem Pfad.
- `public function get(object doc, object path, object default_val = 0)` : Liest einen verschachtelten Wert über einen Schlüssel- oder Indexpfad aus dem JSON-Dokument.
- `public function has(object doc, sequence path)` : Prüft, ob ein Schlüssel in einem JSON-Objekt existiert.
- `public function set(sequence obj_doc, sequence key, object val)` : Setzt oder aktualisiert einen Wert in einem JSON-Objekt entlang eines Pfades.
- `public function keys(object obj_doc)` : Liefert alle Schlüssel eines JSON-Objekts als Sequenz.
- `public function size(object doc)` : Liefert die Anzahl der Einträge in einem JSON-Objekt oder -Array.
- `public function jsonpath(object doc_or_str, sequence expr)` : Führt eine JSONPath-Abfrage auf dem Dokument aus (Alias für `json_path`).

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

- **Zweck:** Schema-Validierung von JSON-Dokumenten nach der Spezifikation JSON Schema Draft 2020-12 mit detaillierten Fehlerberichten.
- **Öffentliche Funktionen & Prozeduren (3):**
- `public function validate(object instance_or_str, object schema_or_str)` : Validiert ein JSON-Dokument gegen ein JSON-Schema (Draft 2020-12).
- `public function is_valid(object instance_or_str, object schema_or_str)` : Prüft, ob ein JSON-Dokument ein Schema erfüllt (1 = gültig, 0 = ungültig).
- `public function format_errors(sequence validation_res)` : Formatiert alle Schema-Validierungsfehler als lesbare Fehlerliste.

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

- **Zweck:** W3C-konformer XML DOM-Parser, programmatischer Element-Builder, W3C XPath 1.0 Abfrage-Engine und XSD-Schema-Validierung.
- **Öffentliche Funktionen & Prozeduren (20):**
- `public function parse(sequence xml_str)` : Parst einen XML-String in einen hierarchischen W3C XML-DOM-Knotenbaum.
- `public function decode(sequence xml_str)` : Dekodiert einen XML-String in einen DOM-Baum (Alias für `parse`).
- `public function element(sequence tag_name, sequence attrs = {}, sequence children = {}, sequence text_val = "")` : Erstellt programmatisch einen neuen XML-Elementknoten mit Tag, Attributen und Kindern.
- `public function stringify(sequence node, integer indent = 0)` : Serialisiert einen DOM-Knotenbaum in wohlgeformten XML-Text (mit Einrückung).
- `public function to_string(sequence node, integer indent = 0)` : Konvertiert einen DOM-Knoten in einen XML-String (Alias für `stringify`).
- `public function is_element(object node)` : Prüft, ob ein Knoten ein XML-Element darstellt.
- `public function tag(sequence node)` : Gibt den Elementnamen (Tag-Namen) eines XML-Knotens zurück.
- `public function text(sequence node)` : Liest den aggregierten Textinhalt eines XML-Elements aus.
- `public function attrs(sequence node)` : Liefert alle Attribute eines XML-Knotens als Sequenz von Key-Value-Paaren.
- `public function children(sequence node)` : Gibt die Liste aller Kindknoten eines XML-Elements zurück.
- `public function attr(sequence node, sequence name, sequence default_val = "")` : Liest den Wert eines bestimmten Attributs eines XML-Knotens aus.
- `public function set_attr(sequence node, sequence name, sequence val)` : Setzt oder aktualisiert ein Attribut an einem XML-Elementknoten.
- `public function add_child(sequence node, sequence child)` : Fügt einen Kindknoten an das Ende der Kindknotenliste eines XML-Elements an.
- `public function find_all(sequence node, sequence tag_or_path)` : Findet alle direkten Kindelemente mit dem angegebenen Tag-Namen.
- `public function find_first(sequence node, sequence tag_or_path, object default_val = 0)` : Findet das erste Kindelement mit dem angegebenen Tag-Namen.
- `public function xpath(object node_or_xml, sequence xpath_expr)` : Wertet einen W3C XPath 1.0 Ausdruck auf dem DOM-Dokument aus und liefert Treffer.
- `public function validate_xsd(object node_or_xml, sequence xsd_str)` : Validiert ein XML-Dokument gegen eine XSD-Schemadatei und liefert Fehlerdetails.
- `public function is_valid_xsd(object node_or_xml, sequence xsd_str)` : Prüft, ob ein XML-Dokument valide bezüglich eines XSD-Schemas ist (1 = ja, 0 = nein).
- `public function validate_relaxng(object node_or_xml, sequence rng_str)` : Validiert ein XML-Dokument gegen ein RelaxNG-Schema.
- `public function is_valid_relaxng(object node_or_xml, sequence rng_str)` : Prüft, ob ein XML-Dokument gegen ein RelaxNG-Schema valide ist.

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

- **Zweck:** Integrierter SQLite 3 Datenbank-Treiber via JDBC: ACID-Transaktionen, parametrisierte Prepared Statements, Tabellen-Queries und DDL.
- **Öffentliche Funktionen & Prozeduren (15):**

- `public function is_null(object val)` : Prüft, ob ein von SQLite zurückgegebener Feldwert ein SQL-NULL darstellt.
- `public function open(sequence path = ":memory:")` : Öffnet eine SQLite 3 Datenbankdatei (oder `:memory:` für eine RAM-Datenbank).
- `public function close(integer db_id)` : Schließt die SQLite-Datenbankverbindung und gibt alle Ressourcen frei.
- `public function exec(integer db_id, sequence sql, sequence params = {})` : Führt ein parametrisiertes SQL-Kommando (INSERT, UPDATE, DELETE, DDL) aus.
- `public function query_raw(integer db_id, sequence sql, sequence params = {})` : Führt eine SQL-Abfrage aus und liefert Spaltennamen und Rohdaten als Sequenz.
- `public function query(integer db_id, sequence sql, sequence params = {})` : Führt eine SELECT-Abfrage mit Parametern aus und liefert alle Zeilen als Liste.
- `public function query_one(integer db_id, sequence sql, sequence params = {}, object default_val = {})` : Liefert die erste Ergebniszeile einer SQL-Abfrage als Sequenz.
- `public function query_value(integer db_id, sequence sql, sequence params = {}, object default_val = 0)` : Liefert den einzelnen Skalarwert der ersten Spalte der ersten Zeile.
- `public function query_assoc(integer db_id, sequence sql, sequence params = {})` : Liefert Abfrageergebnisse als Sequenz von Key-Value-Maps (Spaltenname -> Wert).
- `public function last_insert_id(integer db_id)` : Gibt die automatisch generierte Zeilen-ID (`ROWID`) des letzten INSERT-Befehls zurück.
- `public function changes(integer db_id)` : Gibt die Anzahl der durch das letzte SQL-Statement betroffenen Zeilen zurück.
- `public function errmsg(integer db_id = 0)` : Liefert die letzte Fehlermeldung der SQLite-Engine.
- `public function begin(integer db_id)` : Startet eine neue Transaktion (`BEGIN TRANSACTION`).
- `public function commit(integer db_id)` : Bestätigt die laufende Transaktion in der SQLite-Datenbank (`COMMIT`).
- `public function rollback(integer db_id)` : Setzt die laufende Transaktion in der SQLite-Datenbank zurück (`ROLLBACK`).

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

- **Zweck:** Enterprise PostgreSQL Datenbank-Client via JDBC: Connection-Pooling, Prepared Statements, Transaktionsverwaltung und Datentypkonvertierung.
- **Öffentliche Funktionen & Prozeduren (19):**
- `public function is_available()` : Prüft, ob der PostgreSQL-JDBC-Treiber im Classpath verfügbar ist.
- `public function is_null(object val)` : Prüft, ob ein Spaltenwert SQL-NULL darstellt.
- `public function open(sequence conninfo = "")` : Stellt eine JDBC-Verbindung zu einem PostgreSQL-Server her.
- `public function connect(sequence conninfo = "")` : Verbindet zu einer PostgreSQL-Datenbank mit Verbindungsparametern.
- `public function close(integer db_id)` : Schließt die PostgreSQL-Verbindung und gibt JDBC-Ressourcen frei.
- `public function exec(integer db_id, sequence sql, sequence params = {})` : Führt ein parametrisiertes SQL-Kommando auf der PostgreSQL-Datenbank aus.
- `public function query_raw(integer db_id, sequence sql, sequence params = {})` : Führt eine Rohdatenabfrage auf PostgreSQL aus.
- `public function query(integer db_id, sequence sql, sequence params = {})` : Führt eine parametrisierte SELECT-Abfrage aus und liefert alle Zeilen als Sequenz.
- `public function query_one(integer db_id, sequence sql, sequence params = {}, object default_val = {})` : Liefert die erste Zeile eines Abfrageergebnisses.
- `public function query_value(integer db_id, sequence sql, sequence params = {}, object default_val = 0)` : Liefert den ersten Skalarwert der ersten Zeile.
- `public function query_assoc(integer db_id, sequence sql, sequence params = {})` : Führt eine Abfrage aus und liefert Zeilen als Maps mit Spaltennamen als Schlüssel.
- `public function changes(integer db_id)` : Liefert die Anzahl der geänderten Zeilen des letzten Kommandos.
- `public function server_version(integer db_id)` : Liest die Versionszeichenkette des PostgreSQL-Datenbankservers aus.
- `public function errmsg(integer db_id = 0)` : Gibt die letzte Fehlermeldung des PostgreSQL-Treibers zurück.
- `public function escape_literal(integer db_id, sequence val)` : Maskiert ein String-Literal für die sichere SQL-Einbettung.
- `public function escape_identifier(integer db_id, sequence ident)` : Maskiert einen SQL-Bezeichner (Tabellen-/Spaltenname).
- `public function begin(integer db_id)` : Startet eine Transaktion auf der PostgreSQL-Verbindung.
- `public function commit(integer db_id)` : Bestätigt die Transaktion auf der PostgreSQL-Verbindung (`COMMIT`).
- `public function rollback(integer db_id)` : Verwirft die Transaktion auf der PostgreSQL-Verbindung (`ROLLBACK`).

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

- **Zweck:** Euphoria Database System (EDS): Leichtgewichtige, autarke NoSQL-Hierarchie-Datenbank mit Tabellen, Schlüsseln und Record-Einträgen.
- **Öffentliche Funktionen & Prozeduren (30):**
- `public function db_get_errors(integer clearing = 1)` : Liefert den Fehlerstatus der letzten EDS-Datenbankoperation.
- `public procedure db_dump(object file_id, integer low_level_too = 0)` : Gibt eine Textübersicht aller Tabellen und Records der aktuellen EDS-Datenbank aus.
- `public procedure check_free_list()` : Überprüft die interne Freiliste gelöschter Blöcke auf Konsistenz.

- `public function db_connect(sequence dbalias, sequence path="", sequence dboptions = {})` : Verbindet eine bestehende EDS-Datenbank oder öffnet sie im Cache.
- `public function db_create(sequence path, integer lock_method = DB_LOCK_NO, integer init_tables = DEF_INIT_TABLES, integer init_free = DEF_INIT_FREE )` : Erstellt eine neue leere EDS-Datenbankdatei auf der Festplatte.
- `public function db_open(sequence path, integer lock_method = DB_LOCK_NO)` : Öffnet eine bestehende EDS-Datenbankdatei für Lese- oder Schreibzugriffe.
- `public function db_select(sequence path, integer lock_method = -1)` : Wählt eine geöffnete EDS-Datenbank als aktive Arbeitsdatenbank aus.
- `public procedure db_close()` : Schließt die angegebene oder aktuell ausgewählte EDS-Datenbankdatei.
- `public function db_select_table(sequence name)` : Wählt eine Tabelle innerhalb der aktiven Datenbank als aktuelle Arbeitstabelle aus.
- `public function db_current_table()` : Gibt den Namen der aktuell ausgewählten Tabelle zurück.
- `public function db_create_table(sequence name, integer init_records = DEF_INIT_RECORDS)` : Erstellt eine neue Tabelle in der aktuellen Datenbank.
- `public procedure db_delete_table(sequence name)` : Löscht eine Tabelle und alle darin enthaltenen Datensätze.
- `public procedure db_clear_table(sequence name, integer init_records = DEF_INIT_RECORDS)` : Löscht alle Datensätze einer Tabelle, behält die Tabelle jedoch bei.
- `public procedure db_rename_table(sequence name, sequence new_name)` : Benennt eine Tabelle in der aktuellen Datenbank um.
- `public function db_table_list()` : Gibt eine Liste aller in der aktiven Datenbank existierenden Tabellennamen zurück.
- `public function db_find_key(object key, object table_name=current_table_name)` : Sucht einen Datensatz anhand seines eindeutigen Schlüssels und gibt die Record-Nummer zurück.
- `public function db_insert(object key, object data, object table_name=current_table_name)` : Fügt einen neuen Datensatz mit Schlüssel und Nutzdaten in die aktuelle Tabelle ein.
- `public procedure db_delete_record(integer key_location, object table_name=current_table_name)` : Löscht einen Datensatz anhand seiner Record-Nummer.
- `public procedure db_replace_data(integer key_location, object data, object table_name=current_table_name)` : Aktualisiert die Nutzdaten eines bestehenden Datensatzes.
- `public function db_table_size(object table_name=current_table_name)` : Gibt die Anzahl der in der aktuellen Tabelle gespeicherten Datensätze zurück.
- `public function db_record_data(integer key_location, object table_name=current_table_name)` : Liest die Nutzdaten eines Datensatzes anhand seiner Record-Nummer.
- `public function db_fetch_record(object key, object table_name=current_table_name)` : Liest Schlüssel und Nutzdaten eines Datensatzes in einem Aufruf aus.
- `public function db_record_key(integer key_location, object table_name=current_table_name)` : Liest den Schlüssel eines Datensatzes anhand seiner Record-Nummer.
- `public function db_compress()` : Komprimiert die Datenbankdatei und gibt ungenutzten Speicherplatz frei.
- `public function db_current ()` : Liefert das Handle der aktuell selektierten Arbeitsdatenbank.
- `public procedure db_cache_clear()` : Leert den internen Record-Cache der Datenbank.
- `public function db_set_caching(atom new_setting)` : Aktiviert oder deaktiviert das Caching für EDS-Lesezugriffe.
- `public procedure db_replace_recid(integer recid, object data)` : Ersetzt Daten über die interne Datensatz-ID.
- `public function db_record_recid(integer recid)` : Liest Daten anhand der internen Record-ID aus.
- `public function db_get_recid(object key, object table_name=current_table_name)` : Ermittelt die interne ID eines Datensatzes.

2.8.8 Desktop-GUI & Web-Frontend Frameworks

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

- **Zweck:** Native Java Swing & FlatLaf Desktop-GUI Bibliothek: GTK+/Nord/Dark Themes, Subpixel-LCD Antialiasing, Widgets, Dialoge und Event-Listener.
- **Öffentliche Funktionen & Prozeduren (36):**
- `public procedure init_gui()` : Initialisiert das Swing-Subsystem und setzt Optionen für Grafikbeschleunigung.
- `public procedure set_theme(sequence theme_name)` : Wählt das globale Erscheinungsbild (`THEME_GTK` , `THEME_FLAT_LIGHT` , `THEME_FLAT_DARK` , `THEME_NORD` , `THEME_SOLARIZED`).
- `public function window(sequence title = "Euphoria", integer width = 600, integer height = 400, integer on_close = CLOSE_ON_EXIT)` : Erstellt ein neues Hauptfenster (JFrame) mit Titel und optionaler Größe.
- `public procedure show(atom widget)` : Macht ein Fenster oder Widget auf dem Bildschirm sichtbar.
- `public procedure hide(atom widget)` : Blendet ein Fenster oder Widget aus.
- `public procedure close_window(atom widget)` : Schließt und zerstört ein Fenster und gibt dessen Ressourcen frei.
- `public procedure pack(atom win)` : Passt die Fenstergröße automatisch an die bevorzugte Größe der enthaltenen Widgets an.
- `public function button(sequence text, integer on_click_rid = -1)` : Erstellt einen Druckknopf (JButton) mit Text und optionalem Klick-Handler.

- `public procedure on_click(atom widget, integer on_click_rid)` : Verknüpft eine Routine-ID als Klick-Ereignis-Handler mit einem Button oder Widget.
- `public function text_label(sequence text = "", integer align = ALIGN_LEFT)` : Erstellt ein Text-Anzeigelabel (JLabel).
- `public function create_label(sequence text = "", integer align = ALIGN_LEFT)` : Erstellt ein Textlabel (JLabel) mit anpassbarem Text und Ausrichtung.
- `public function text_field(sequence initial_text = "", integer columns = 20)` : Erstellt ein einzeiliges Texteingabefeld (JTextField).
- `public function text_area(sequence initial_text = "", integer rows = 10, integer cols = 30)` : Erstellt ein mehrzeiliges Textbearbeitungsfeld (JTextArea).
- `public function scroll_pane(atom view_widget)` : Umschließt ein Widget mit Bildlaufleisten (JScrollPane).
- `public function check_box(sequence text, integer checked = 0, integer on_change_rid = -1)` : Erstellt eine Kontrollbox (JCheckBox) mit Label.
- `public function radio_button(sequence text, integer selected = 0)` : Erstellt einen Optionsschalter (JRadioButton).
- `public function combo_box(sequence items = {})` : Erstellt eine Auswahlliste (JComboBox / Dropdown).
- `public function panel(integer layout_type = FLOW_LAYOUT, integer rows = 0, integer cols = 0, integer hgap = 0, integer vgap = 0)` : Erstellt einen Container-Bereich (JPanel) für Layouts.
- `public function progress_bar(integer min_val = 0, integer max_val = 100)` : Erstellt einen Fortschrittsbalken (JProgressBar).
- `public function slider(integer min_val = 0, integer max_val = 100, integer init_val = 50)` : Erstellt einen stufenlosen Schieberegler (JSlider).
- `public procedure add(atom container, atom child, object constraint = 0)` : Fügt ein Unter-Widget zu einem Eltern-Container oder Fenster hinzu.
- `public procedure set_layout(atom container, integer layout_type, integer rows = 0, integer cols = 0, integer hgap = 0, integer vgap = 0)` : Setzt den Layout-Manager eines Containers (Flow, Border, Grid, Box).
- `public procedure set_text(atom widget, sequence text)` : Setzt den Textinhalt eines Eingabefeldes, Buttons oder Labels.
- `public function get_text(atom widget)` : Liest den Textinhalt eines Eingabefeldes oder Labels aus.
- `public procedure set_value(atom widget, integer val)` : Setzt den numerischen Wert eines Reglers oder Fortschrittsbalkens.
- `public function get_value(atom widget)` : Liest den numerischen Wert eines Schiebereglers aus.
- `public procedure set_padding(atom widget, integer top, integer left, integer bottom, integer right)` : Setzt Innenabstände (Padding/Insets) für ein Widget.
- `public procedure set_font(atom widget, sequence name = "Ubuntu", integer size = 13, integer is_bold = 0)` : Konfiguriert Schriftart, Schriftschnitt und Schriftgröße eines Widgets.
- `public procedure set_enabled(atom widget, integer enabled)` : Aktiviert oder deaktiviert die Benutzerinteraktion für ein Widget.
- `public function is_selected(atom widget)` : Prüft, ob eine Checkbox oder ein Radiobutton aktiviert ist.
- `public procedure set_selected(atom widget, integer sel)` : Aktiviert oder deaktiviert eine Checkbox programmgesteuert.
- `public procedure msgbox(sequence title, sequence message, integer msg_type = MSG_INFO)` : Zeigt einen standardisierten modalen Hinweisdialog an.
- `public function input_box(sequence title, sequence message)` : Öffnet einen Dialog zur schnellen Texteingabe durch den Benutzer.
- `public function confirm_box(sequence title, sequence message)` : Öffnet einen Bestätigungsdialog mit Ja/Nein-Auswahl.
- `public function file_dialog(sequence title = "Datei auswählen", integer is_save = 0, sequence cur_dir = "")` : Öffnet den nativen Datei-Auswahldialog zum Öffnen oder Speichern von Dateien.
- `public procedure wait_window(atom win)` : Hält den aufrufenden Thread an, bis das angegebene Fenster geschlossen wird.

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

- **Zweck:** Enterprise Web-Frontend Framework auf Basis von SAP UI5 Web Components 2.29: Fiori Horizon Design, reaktive DOM-Patches und integrierter Webserver.
- **Öffentliche Funktionen & Prozeduren (65):**
- `public function page(sequence title, object content, sequence options = {})` : Erstellt eine vollständige HTML5-Seite mit Fiori Horizon Theme, ShellBar und reaktiver Event-Bridge.
- `public function title(sequence text, sequence level = TITLE_H2, sequence options = {})` : Erstellt ein Titel-Element (`ui5-title`) mit Überschriften-Ebene H1 bis H6.
- `public function ui_label(sequence text, sequence for_id = "", integer required = 0, sequence options = {})` : Erstellt ein Standard-Beschriftungsfeld (`ui5-label`).
- `public function form_label(sequence text, sequence for_id = "", integer required = 0, sequence options = {})` : Erstellt ein Formular-Label (`ui5-label`) mit Pflichtfeld-Sternchen (`required`).
- `public function link(sequence text, sequence href = "#", sequence design = LINK_DEFAULT, sequence options = {})` : Erstellt einen Hyperlink (`ui5-link`) mit Design-Optionen (Default, Subtle, Emphasized).
- `public function text_block(sequence text, sequence options = {})` : Erstellt einen typografischen Textblock oder Absatz.
- `public function shellbar(sequence primary_title, sequence secondary_title = "", sequence options = {})` : Erstellt eine Enterprise-Kopfzeile (`ui5-shellbar`) mit Logo, Profil-Avatar und Menüaktionen.
- `public function breadcrumbs(sequence items, sequence options = {})` : Erstellt einen Brotkrümel-Navigationspfad (`ui5-breadcrumbs`).

- `public function bar(object start_content, object middle_content = {}, object end_content = {}, sequence design = BAR_HEADER, sequence options = {})` : Erstellt eine Aktionsleiste (`ui5-bar`) für Kopf- oder Fußzeilen (Header, Subheader, Footer).
- `public function card(sequence title, object content, sequence options = {})` : Erstellt eine Fiori-Karte (`ui5-card`) mit Kopfzeile, Status-Icon und Inhaltsbereich.
- `public function panel(sequence header_text, object content, integer collapsed = 0, sequence options = {})` : Erstellt einen einklappbaren Container (`ui5-panel`) mit Überschrift.
- `public function tab_container(sequence tabs, sequence options = {})` : Erstellt eine Reiter-Navigation (`ui5-tabcontainer`) mit Reitern und Inhaltsseiten.
- `public function table(sequence columns, sequence rows, sequence options = {})` : Erstellt eine responsive Datentabelle (`ui5-table`) mit Spaltenüberschriften und Zeilen.
- `public function table_from_query(sequence headers, sequence query_rows, sequence options = {})` : Generiert eine fertige Fiori-Tabelle direkt aus einem SQL-Abfrageergebnis.
- `public function list(sequence items, sequence options = {})` : Erstellt eine Auswahlliste (`ui5-list`) mit Listeneinträgen (`ui5-li`).
- `public function tree(sequence items, sequence options = {})` : Erstellt eine hierarchische Baumstruktur (`ui5-tree`) für Verzeichnisse oder Taxonomien.
- `public function button(sequence id, sequence text, sequence options = {})` : Erstellt einen Button (`ui5-button`) mit Design (Default, Emphasized, Positive, Negative, Attention).
- `public function toggle_button(sequence id, sequence text, integer pressed = 0, sequence options = {})` : Erstellt einen Umschalt-Button (`ui5-toggle-button`), der gedrückt bleiben kann.
- `public function menu_button(sequence id, sequence text, sequence options = {})` : Erstellt einen Button mit anhängendem Dropdown-Menü (`ui5-menu-button`).
- `public function segmented_button(sequence id, sequence items, sequence options = {})` : Erstellt eine Gruppe zusammenhängender Segment-Schaltflächen (`ui5-segmented-button`).
- `public function input(sequence id, sequence value = "", sequence options = {})` : Erstellt ein einzeiliges Text-Eingabefeld (`ui5-input`) mit Platzhalter und Typ (Text, Email, Tel, Number).
- `public function step_input(sequence id, atom value = 0, atom min_val = 0, atom max_val = 100, atom step_val = 1, sequence options = {})` : Erstellt ein numerisches Schritt-Eingabefeld (`ui5-step-input`) mit Plus-/Minus-Tasten.
- `public function slider(sequence id, atom value = 0, atom min_val = 0, atom max_val = 100, atom step_val = 1, sequence options = {})` : Erstellt einen Schieberegler (`ui5-slider`) für Zahlenwerte.
- `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 = {})` : Erstellt einen Bereichs-Schieberegler (`ui5-range-slider`) mit zwei Ziehpunkten.
- `public function rating_indicator(sequence id, atom value = 3, integer max_val = 5, sequence options = {})` : Erstellt eine Sterne-Bewertungskomponente (`ui5-rating-indicator`).
- `public function textarea(sequence id, sequence value = "", sequence options = {})` : Erstellt ein mehrzeiliges Texteingabefeld (`ui5-textarea`).
- `public function checkbox(sequence id, sequence text, integer checked = 0, sequence options = {})` : Erstellt eine Kontrollbox (`ui5-checkbox`) mit Status und Label.
- `public function radio_button(sequence id, sequence name, sequence text, sequence value = "", integer checked = 0, sequence options = {})` : Erstellt einen Optionsschalter (`ui5-radio-button`) innerhalb einer benannten Gruppe.
- `public function switch_btn(sequence id, sequence text_on = "Ein", sequence text_off = "Aus", integer checked = 0, sequence options = {})` : Erstellt einen grafischen Kippschalter (`ui5-switch`) mit Ein/Aus-Zustand.
- `public function select(sequence id, sequence options_list, sequence options = {})` : Erstellt eine Auswahlliste (`ui5-select`) mit Optionen.
- `public function combobox(sequence id, sequence value = "", sequence items = {}, sequence options = {})` : Erstellt eine filterbare Auswahlliste (`ui5-combobox`).
- `public function multi_combobox(sequence id, sequence items = {}, sequence options = {})` : Erstellt ein Mehrfachauswahlfeld (`ui5-multi-combobox`) mit Token-Anzeige.
- `public function datepicker(sequence id, sequence value = "", sequence options = {})` : Erstellt ein Datumsauswahlfeld (`ui5-datepicker`) mit Kalender-Popup.
- `public function time_picker(sequence id, sequence value = "", sequence options = {})` : Erstellt ein Zeitauswahlfeld (`ui5-time-picker`) für Stunden und Minuten.
- `public function datetime_picker(sequence id, sequence value = "", sequence options = {})` : Erstellt ein kombiniertes Datums- und Zeitauswahlfeld (`ui5-datetime-picker`).
- `public function daterange_picker(sequence id, sequence value = "", sequence options = {})` : Erstellt ein Zeitraum-Auswahlfeld (`ui5-daterange-picker`) für Start- und Enddatum.
- `public function file_uploader(sequence id, sequence placeholder = "Datei auswählen...", sequence options = {})` : Erstellt eine Datei-Upload-Schaltfläche (`ui5-file-uploader`).
- `public function color_picker(sequence id, sequence color = "#0070f2", sequence options = {})` : Erstellt eine grafische Farbpalette (`ui5-color-picker`).
- `public function dialog(sequence id, sequence header_text, object content, object footer_buttons = {}, sequence options = {})` : Erstellt einen modalen Fiori-Dialog (`ui5-dialog`) mit Kopfzeile und Fußzeile.

- `public function popover(sequence id, sequence header_text, object content, object footer_buttons = {}, sequence options = {})` : Erstellt ein schwebendes Info-Overlay (`ui5-popover`) an einem Bezugselement.
- `public function toast(sequence id, sequence text = "", integer duration = 3000)` : Erstellt eine temporäre Toast-Benachrichtigungskomponente (`ui5-toast`).
- `public function badge(sequence text, integer color_scheme = 1, sequence options = {})` : Erstellt eine farbige Statusplakette (`ui5-badge`) mit Zahl oder Text.
- `public function tag(sequence text, integer color_scheme = 1, sequence options = {})` : Erstellt ein interaktives Tag-Label (`ui5-tag`).
- `public function icon(sequence name, sequence options = {})` : Bindet ein SAP-Icon (`ui5-icon`) aus der Fiori-Icon-Sammlung ein.
- `public function avatar(sequence initials_or_src, sequence size = AVATAR_M, sequence shape = AVATAR_CIRCLE, sequence options = {})` : Erstellt ein Benutzerprofilbild oder Initien-Avatar (`ui5-avatar`).
- `public function busy_indicator(integer active = 1, sequence size = BUSY_MEDIUM, sequence text = "", object content = {}, sequence options = {})` : Erstellt eine Lade- und Aktivitätsanzeige (`ui5-busy-indicator`).
- `public function progress_indicator(integer value_pct, sequence display_value = "", sequence state = STATE_NONE, sequence options = {})` : Erstellt einen prozentualen Fortschrittsbalken (`ui5-progress-indicator`).
- `public function message_strip(sequence text, sequence design = "Information", sequence options = {})` : Erstellt eine farbige Hinweisbox (`ui5-message-strip`) für Erfolge, Warnungen oder Fehler.
- `public function timeline(sequence items, sequence options = {})` : Erstellt eine chronologische Zeitleiste (`ui5-timeline`) für Aktivitäten und Historien.
- `public function illustrated_message(sequence name = "NoData", sequence title_text = "Keine Daten vorhanden", sequence subtitle_text = "", sequence options = {})` : Erstellt eine leere Statusseite mit Vektorgrafik und Erläuterung (`ui5-illustrated-message`).
- `public function patch(sequence id, sequence html)` : Erzeugt eine DOM-Patch-Anweisung zur gezielten Inhaltsaktualisierung von Element `id` .
- `public function append_html(sequence id, sequence html)` : Erzeugt eine DOM-Patch-Anweisung zum Anhängen von HTML an Element `id` .
- `public function set_val(sequence id, sequence value)` : Erzeugt eine Patch-Anweisung zum Setzen der `value` -Eigenschaft von Feld `id` .
- `public function set_value_state(sequence id, sequence state, sequence message = "")` : Setzt den visuellen Validierungsstatus (`None` , `Error` , `Warning` , `Success` , `Information`).
- `public function toast_show(sequence id, sequence text = "")` : Blendet eine Benachrichtigungsmeldung (`ui5-toast`) mit konfigurierbarer Dauer ein.
- `public function open_dialog(sequence id)` : Öffnet einen modalen UI5-Dialog über dessen ID.
- `public function close_dialog(sequence id)` : Schließt einen geöffneten modalen UI5-Dialog.
- `public function redirect(sequence url)` : Sendet einen Client-Redirect-Befehl zur Navigation an eine andere URL.
- `public function updates(sequence patches_list)` : Bündelt eine Sequenz von Patch-Anweisungen als strukturierte Event-Antwort.
- `public function val(sequence ctx, sequence key, sequence default_val = "")` : Liest den vom Client übermittelten Formularwert des Parameters `id` aus dem Kontext aus.
- `public function app(sequence title, integer port = 8080, sequence theme = THEME_HORIZON, sequence version = DEFAULT_VERSION, sequence cdn = DEFAULT_CDN_BASE)` : Erzeugt das vollständige Anwendungsgerüst für eine Single-Page-Application (SPA).
- `public procedure page_route(integer app_id, sequence path, integer handler_rid)` : Registriert eine URL-Route (`path`) mit einer zuständigen Render-Funktion.
- `public procedure action(integer app_id, sequence action_name, integer handler_rid)` : Registriert einen Event-Handler für UI-Interaktionen (`button_click` , `change` , etc.).
- `public function init_server(integer app_id)` : Initialisiert den integrierten HTTP-Server mit Port, Seitentitel und Head-Optionen.
- `public procedure run(integer app_id)` : Startet die reaktive UI5-Webanwendung auf dem konfigurierten Port.

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

- **Zweck:** Autarkes High-Density ERP Web-Framework mit lokalen Assets (`html/`): Sticky-Header DataGrid, vsevent-Bridge, reaktive DOM-Patches, SVG-Charts, Upload und mobile Responsiveness.
- **Öffentliche Funktionen & Prozeduren (47):**
- `public function page(sequence title, object content, sequence options = {})` : Erstellt das vollständige HTML-Grundgerüst mit VS-UI CSS (`globalblauklein.css` , `vs.css`), `vs-mobile.css` und ES6-Modulen.
- `public function button(sequence id, sequence text, sequence options = {})` : Erstellt einen High-Density Button (`.vs-button`) mit Klick-Event-Weiterleitung.
- `public function input(sequence id, sequence value = "", sequence options = {})` : Erstellt ein kompaktes Eingabefeld (`.vs-input`).
- `public function numinput(sequence id, atom value = 0, sequence options = {})` : Erstellt ein numerisches Eingabefeld mit Schrittweite und Min/Max.
- `public function dateinput(sequence id, sequence value = "", sequence options = {})` : Erstellt ein HTML5-Datumsfeld (`type="date"`).
- `public function timeinput(sequence id, sequence value = "", sequence options = {})` : Erstellt ein HTML5-Zeitfeld (`type="time"`).
- `public function check(sequence id, sequence label_text, integer checked = 0, sequence options = {})` : Erstellt eine Checkbox mit Beschriftung (`.vs-check`).
- `public function textarea(sequence id, sequence value = "", integer rows = 4, sequence options = {})` : Erstellt ein mehrzeiliges kompaktes Textfeld (`.vs-input`).

- `public function combo(sequence id, sequence options_list, sequence selected_val = "", sequence options = {})` : Erstellt ein Dropdown-Auswahlfeld (`.vs-combo`).
- `public function ui_label(sequence text, sequence for_id = "", sequence options = {})` : Erstellt ein Formular-Label (`.vs-label`).
- `public function group(sequence legend_text, object content, sequence options = {})` : Erstellt eine Fieldset-Gruppe (`.vs-group`) mit blauem Rahmen und Titel.
- `public function panel(sequence id, sequence title, object content, sequence options = {})` : Erstellt einen High-Density Container (`.vs-panel`) mit Header und Body.
- `public function collapse(sequence id, sequence sections, integer open_first = 1, sequence options = {})` : Erstellt klappbare Abschnitte (`<details>` / `<summary>`).
- `public function grid(sequence id, sequence columns, sequence rows, sequence options = {})` : Erstellt die High-Density Tabelle `vsgrid` mit Sticky-Header, Client-Live-Suche und automatischem Mobile-Card-View.
- `public function tabs(sequence id, sequence tabs_list, sequence options = {})` : Erstellt eine Reiter-Navigation mit dynamischer Inhaltsumschaltung und horizontalem Mobile-Swipe.
- `public function modal(sequence id, sequence title, object content, object footer_buttons = {}, sequence options = {})` : Erstellt ein modales Dialogfenster mit Abdunklungs-Overlay (mobil Vollbild).
- `public function splitter(sequence id, object left_content, object right_content, integer split_pct = 50, sequence options = {})` : Erstellt ein 2-Spalten-Layout mit anpassbarer Breitenaufteilung (mobil vertikal gestapelt).
- `public function upload(sequence id, sequence target_url = "/api/upload", sequence options = {})` : Erstellt eine moderne Drag & Drop Datei-Upload Dropzone mit Ladebalken.
- `public function chart(sequence id, sequence chart_type, sequence labels, sequence datasets, sequence options = {})` : Erzeugt ein responsives Server-gerendertes SVG-Vektordiagramm (`"bar"` , `"line"` , `"pie"` , `"doughnut"`).
- `public function chart_set_data(sequence id, sequence chart_type, sequence labels, sequence datasets)` : Erzeugt eine Live-Patch-Anweisung zur dynamischen Diagrammaktualisierung ohne Reload.
- `public function field_error(sequence id, sequence error_msg)` : Markiert ein Formularfeld rot und blendet einen Fehlertooltip darunter ein.
- `public function field_clear_error(sequence id)` : Entfernt Fehlermarkierung und Tooltip von einem Formularfeld.
- `public function validate_required(sequence ctx, sequence field_id, sequence field_label = "")` : Prüft, ob ein Pflichtfeld im Request-Kontext ausgefüllt ist.
- `public function validate_email(sequence email_str)` : Prüft ein Feld auf gültiges E-Mail-Format (`user@domain.tld`).
- `public function validate_number(sequence num_str, atom min_val = -1e9, atom max_val = 1e9)` : Validiert numerische Formulareingaben innerhalb von Min/Max-Grenzen.
- `public function patch(sequence id, sequence html)` : Erzeugt eine DOM-Patch-Anweisung für ein Element über seine ID.
- `public function append_html(sequence id, sequence html)` : Erzeugt eine DOM-Patch-Anweisung zum Anhängen von HTML an ein Target-Element.
- `public function set_val(sequence id, sequence value)` : Erzeugt eine Patch-Anweisung zum Aktualisieren eines Formularwerts im Frontend (nutzt `setValue()` des Elements).
- `public function grid_set_data(sequence id, sequence data)` : Aktualisiert die Datenzeilen eines Grids oder einer Tabelle per JSON-Patch ohne HTML-Neuaufbau.
- `public function set_focus(sequence id)` : Setzt den Tastatur-Fokus auf das angegebene Formularfeld.
- `public function set_enabled(sequence id, integer is_enabled)` : Aktiviert oder sperrt ein Eingabefeld bzw. einen Button reaktiv.
- `public function set_visible(sequence id, integer is_visible)` : Blendet ein DOM-Element reaktiv ein oder aus (`display: none` vs. `''`).
- `public function set_css(sequence id, sequence prop, sequence val)` : Setzt eine CSS-Style-Eigenschaft direkt an einem DOM-Element.
- `public function set_class(sequence id, sequence class_name, sequence action = "add")` : Fügt eine CSS-Klasse hinzu, entfernt sie oder schaltet sie um ("add", "remove", "toggle").
- `public function clear_form(sequence container_id = "")` : Leert alle Formularfelder (Inputs, Checks, Combos, Textareas) eines Containers.
- `public function toast_show(sequence text, sequence toast_type = TOAST_INFO, integer duration = 3000)` : Zeigt eine schwebende Erfolgs- oder Fehlermeldung im High-Density Theme an.
- `public function modal_open(sequence id)` : Öffnet ein modales Dialogfenster.
- `public function modal_close(sequence id)` : Schließt das aktive modale Fenster.
- `public function redirect(sequence url)` : Leitet den Webbrowser auf eine neue URL um.
- `public function updates(sequence patches_list)` : Serialisiert eine Liste von DOM-Patch-Anweisungen für die `vsevent` -Bridge.
- `public function val(sequence ctx, sequence key, sequence default_val = "")` : Extrahiert den gesendeten Formularwert eines Feldes aus dem Anfragekontext (nutzt `getValue()`).
- `public function app(sequence title, integer port = DEFAULT_PORT, sequence static_dir = DEFAULT_STATIC_DIR)` : Erstellt ein vollständiges ERP-Portal mit Navigation und Hauptbereich.
- `public procedure page_route(integer app_id, sequence path, integer handler_rid)` : Verknüpft einen relativen URL-Pfad mit einer Seiten-Generatorfunktion.
- `public procedure on_event(integer app_id, sequence event_name, integer handler_rid)` : Registriert einen Event-Handler für ein benutzerdefiniertes Client-Ereignis.
- `public procedure on_click(integer app_id, sequence button_id, integer handler_rid)` : Registriert einen Klick-Handler für einen Button anhand seiner ID.

- `public function init_server(integer app_id)` : Initialisiert den VS-UI Webserver mit lokalem Asset-Pfad und Titelleiste.
- `public procedure run(integer app_id)` : Startet den autarken High-Density ERP-Webserver.

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

- **Zweck:** Plattformübergreifende Konsolen- und Textgrafikroutinen: Vorder-/Hintergrundfarben, Cursorpositionierung und Zeilenumbruch.
- **Öffentliche Funktionen & Prozeduren (7):**
- `public function get_position()` : Gibt die aktuelle Cursorposition (Zeile und Spalte) auf der Textkonsole zurück.
- `public procedure text_color(color c)` : Setzt die Vordergrundfarbe für nachfolgende Konsolenausgaben.
- `public procedure bk_color(color c)` : Setzt die Hintergrundfarbe für nachfolgende Textausgaben.
- `public function console_colors(sequence colorset = {})` : Liest die aktuellen Vorder- und Hintergrundfarben der Konsole aus.
- `public procedure wrap(object on = 1)` : Aktiviert oder deaktiviert den automatischen Zeilenumbruch der Textkonsole.
- `public procedure scroll(integer amount, console:positive_int top_line, console:positive_int bottom_line)` : Scrollt einen rechteckigen Bildschirmbereich um eine Zeilenanzahl nach oben oder unten.
- `public function graphics_mode(object m = -1)` : Wählt einen Bildschirmgrafik- oder Textmodus aus.

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

- **Zweck:** Videomodi-Konstanten, Farbpaletten-Definitionen und Bildschirmkonfigurationstabellen.
- **Öffentliche Funktionen & Prozeduren (1):**
- `public function video_config()` : Liefert eine Konfigurationssequenz über Bildschirmauflösung, Farben und Zeilenanzahl.

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

- **Zweck:** Grafik- und Bitmapverarbeitung: Lesen und Schreiben von unkomprimierten Bitmap-Dateien (.BMP) und Farbpaletten-Zuordnung.
- **Öffentliche Funktionen & Prozeduren (2):**
- `public function read_bitmap(sequence file_name)` : Liest eine unkomprimierte Windows-Bitmapdatei (.BMP) ein und liefert Palette und Pixelmatrix.
- `public function save_bitmap(two_seq palette_n_image, sequence file_name)` : Speichert eine 2D-Pixelmatrix als Windows-Bitmapdatei (.BMP) auf die Festplatte.

2.8.9 Java-Interoperabilität, C-FFI & Maschinenspeicher

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

- **Zweck:** Java 25 Interoperabilitäts-Bridge: Dynamisches Laden externer JARs (`load_jar`), Instanziierung von Klassen, Methodenaufrufe und Callback-Proxies.
- **Öffentliche Funktionen & Prozeduren (9):**
- `public function load_jar(sequence jar_path)` : Lädt eine externe JAR-Datei zur Laufzeit dynamisch in den Classpath der VM.
- `public function find_class(sequence class_name)` : Sucht eine Java-Klasse über ihren vollqualifizierten Klassennamen.
- `public function new_object(object class_or_name, sequence args = {})` : Alias für `create_object` – instanziiert eine Java-Klasse.
- `public function call_static(object class_or_name, sequence method_name, sequence args = {})` : Ruft eine statische Klassenmethode einer Java-Klasse direkt auf.
- `public function call_method(atom obj_handle, sequence method_name, sequence args = {})` : Ruft eine dynamische Instanzmethode auf einem Java-Objekt auf.
- `public function get_field(object target_or_class, sequence field_name)` : Liest den Wert eines öffentlichen Instanzfeldes eines Java-Objekts.
- `public function set_field(object target_or_class, sequence field_name, object val)` : Setzt den Wert eines öffentlichen Instanzfeldes eines Java-Objekts.
- `public function to_string(atom obj_handle)` : Ruft die Java `toString()` -Methode auf einem Java-Objekt auf.
- `public function create_listener(object interface_class_or_name, integer routine_id)` : Erstellt einen dynamischen Java-Proxy-Listener, der Aufrufe an eine Euphoria Routine-ID weiterleitet.

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

- **Zweck:** C Foreign Function Interface (FFI): Laden nativer Linux Shared Libraries (.so) und Windows DLLs mit C-Aufrufkonventionen.
- **Öffentliche Funktionen & Prozeduren (6):**
- `public function open_dll(sequence file_name)` : Lädt eine native Shared Library (`.so` unter Linux, `.dll` unter Windows) dynamisch.
- `public function define_c_var(atom lib, sequence variable_name)` : Deklariert eine exportierte globale Variable einer nativen C-Bibliothek.
- `public function define_c_proc(object lib, object routine_name, sequence arg_types)` : Deklariert eine C-Prozedur (ohne Rückgabewert) aus einer geladenen Bibliothek.
- `public function define_c_func(object lib, object routine_name, sequence arg_types, atom return_type)` : Deklariert eine C-Funktion (mit Rückgabewert) aus einer geladenen Bibliothek.

- `public function call_back(object id)` : Erstellt eine C-kompatible Funktionszeiger-Adresse für eine Euphoria-Callback-Routine.
- `public function sizeof(integer x)` : Ermittelt die Speichergröße eines C-Datentyps in Bytes.

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

- **Zweck:** Maschinennaher Speicher- und Systemzugriff: Direkte Speicheradressen (`peek` , `poke`), Speicherblöcke und Systemroutinen.
- **Öffentliche Funktionen & Prozeduren (24):**
- `public procedure poke8(atom address, object x)` : Schreibt eine 64-Bit-Ganzzahl an eine Speicheradresse.
- `public procedure poke_pointer(atom address, object x)` : Schreibt einen maschinenbreiten Zeiger an eine Speicheradresse.
- `public function peek8s(object address)` : Liest eine vorzeichenbehaftete 64-Bit-Ganzzahl von einer Speicheradresse.
- `public function peek8u(object address)` : Liest eine vorzeichenlose 64-Bit-Ganzzahl von einer Speicheradresse.
- `public function peek_pointer(object x)` : Liest einen maschinenbreiten Zeiger (32- oder 64-Bit) von einer Speicheradresse.
- `public procedure poke_long(atom address, object x)` : Schreibt eine maschinenbreite Ganzzahl in den Speicher.
- `public function peek_longs(object x)` : Liest eine vorzeichenbehaftete maschinenbreite Ganzzahl aus dem Speicher.
- `public function peek_longu(object x)` : Liest eine vorzeichenlose maschinenbreite Ganzzahl aus dem Speicher.
- `public function allocate( memory:positive_int n, types:boolean cleanup = 0)` : Reserviert einen zusammenhängenden Block nativer Bytes im Speicher.
- `public function allocate_data( memory:positive_int n, types:boolean cleanup = 0)` : Reserviert Datenspeicher mit Lese-/Schreibzugriff.
- `public function allocate_pointer_array(sequence pointers, types:boolean cleanup = 0)` : Reserviert ein Zeiger-Array im Speicher.
- `public procedure free_pointer_array(atom pointers_array)` : Gibt ein zuvor reserviertes Zeiger-Array frei.
- `public function allocate_string_pointer_array(object string_list, types:boolean cleanup = 0)` : Erstellt ein Array nativer C-Strings im Speicher.
- `public function allocate_wstring(sequence s, types:boolean cleanup = 0 )` : Reserviert Speicher für einen UTF-16 Wide-String (Windows).
- `public function peek_wstring(atom addr)` : Liest einen nullterminierten Wide-String aus dem Speicher.
- `public function poke_string(atom buffaddr, integer buffsize, sequence s)` : Schreibt einen String als nullterminierte C-Zeichenkette in den Speicher.
- `public function poke_wstring(atom buffaddr, integer buffsize, sequence s)` : Schreibt einen UTF-16 Wide-String in den Speicher.
- `public function is_DEP_supported()` : Prüft, ob Hardware-DEP (Data Execution Prevention) unterstützt wird.
- `public function is_using_DEP()` : Prüft, ob DEP im laufenden Prozess aktiv ist.
- `public procedure DEP_on(integer value)` : Aktiviert DEP für den aktuellen Prozess.
- `public function allocate_code( object data, memconst:valid_wordsize wordsize = 1 )` : Reserviert ausführbaren Speicher für Maschinencode.
- `public function allocate_string(sequence s, types:boolean cleanup = 0 )` : Reserviert Speicher für eine Zeichenkette und kopiert den String nullterminiert hinein.
- `public function allocate_protect( object data, memconst:valid_wordsize wordsize = 1, valid_memory_protection_constant protection )` : Reserviert Speicher mit spezifischen Schutzeinstellungen (Read, Write, Execute).
- `public procedure free(object addr)` : Gibt zuvor reservierten Speicher frei.

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

- **Zweck:** Natives Speichermanagement: Zuweisung von Rohspeicherblöcken (`allocate`), Freigabe (`free`), Speicherschutz und Zeigerarithmetik.
- **Öffentliche Funktionen & Prozeduren (8):**
- `export procedure deallocate(atom addr)` : Gibt Speicherblock frei (Alias für `free`).
- `public procedure register_block(atom block_addr, atom block_len, integer protection )` : Registriert einen externen Speicherblock im Speicherüberwachungssystem.
- `public procedure unregister_block(atom block_addr)` : Hebt die Registrierung eines überwachten Speicherblocks auf.
- `public function safe_address(atom start, integer len, positive_int action)` : Validiert die Zugriffsrechte einer Speicheradresse.
- `public procedure check_all_blocks()` : Validiert die Integrität aller verwalteten Speicherblöcke.
- `export function prepare_block( atom addr, integer a, integer protection )` : Initialisiert Puffer-Schutzzonen für einen Speicherbereich.
- `export function dep_works()` : Prüft die Verfügbarkeit von Data Execution Prevention (DEP).
- `public procedure free_code( atom addr, integer size, valid_wordsize wordsize = 1 )` : Gibt ausführbaren Maschinencode-Speicher sicher frei.

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

- **Zweck:** Konstanten für Wortgrößen, Datentypen und Speicherzugriffsmodi (Lesen, Schreiben, Ausführen).
- **Öffentliche Funktionen & Prozeduren (3):**
- `export function test_read( valid_memory_protection_constant protection )` : Prüft, ob eine Schutzmaske Leserechte erlaubt.
- `export function test_write( valid_memory_protection_constant protection )` : Prüft, ob eine Schutzmaske Schreibrechte erlaubt.
- `export function test_exec( valid_memory_protection_constant protection )` : Prüft, ob eine Schutzmaske Ausführungsrechte erlaubt.

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

- **Zweck:** Sichere Debug-Version von `memory.e` und `machine.e`: Automatische Erkennung von Pufferüberläufen, Wild-Pointern und illegalen Zugriffen.
- **Öffentliche Funktionen & Prozeduren (12):**
- `public function safe_address(machine_addr start, natural len, positive_int action )`: Prüft, ob eine Speicheradresse für den angegebenen Lese-/Schreib-/Ausführungsmodus sicher ist.
- `public procedure show_block(sequence block_info)`: Gibt Debug-Informationen über einen überwachten Speicherblock aus.
- `public procedure check_all_blocks()`: Überprüft alle registrierten Speicherblöcke auf Pufferüberläufe und Grenzverletzungen.
- `public procedure register_block(machine_addr block_addr, positive_int block_len, valid_memory_protection_constant memory_protection = PAGE_READ_WRITE )`: Registriert einen externen Speicherblock in der Safe-Mode-Verwaltung.
- `public procedure unregister_block(machine_addr block_addr)`: Entfernt einen Speicherblock aus der Safe-Mode-Überwachung.
- `export function prepare_block(int_addr iaddr, positive_int n, natural protection)`: Initialisiert Schutzzonen (Redzones) um einen Speicherblock.
- `export procedure deallocate(atom a)`: Gibt Speicherblock im sicheren Modus frei.
- `export function dep_works()`: Prüft, ob Data Execution Prevention (DEP) auf der Plattform aktiv ist.
- `public procedure free_code( atom addr, integer size, valid_wordsize wordsize = 1 )`: Gibt ausführbaren Speicherblock sicher frei.
- `public function info()`: Gibt statistische Auswertungen über reservierten Speicher im Safe-Modus zurück.
- `public function memory_used()`: Gibt die Gesamtmenge an aktuell reserviertem Speicher zurück.
- `public function allocations()`: Gibt die Anzahl der aktiven Speicherallokationen zurück.

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

- **Zweck:** Pseudo-Speicherverwaltung zur Emulation von Zeigern und Pass-by-Reference in reinem Euphoria-RAM-Heap.
- **Öffentliche Funktionen & Prozeduren (3):**
- `export function malloc(object mem_struct_p = 1, integer cleanup_p = 1)`: Reserviert einen Eintrag im simulierten Euphoria-Pseudo-RAM-Heap.
- `export procedure free(atom mem_p)`: Gibt einen Pseudo-Speichereintrag im RAM wieder frei.
- `export function valid(object mem_p, object mem_struct_p = 1)`: Prüft, ob eine Pseudo-Speicheradresse gültig und allokiert ist.

2.8.10 Moderne Nebenläufigkeit, Threads & Multitasking

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

- **Zweck:** Hochskalierbare Nebenläufigkeit mit Java 25 Virtual Threads (Project Loom): `spawn`, Async/Await Futures (`async`, `await`, `await_all`) und Thread-Schlaf.
- **Öffentliche Funktionen & Prozeduren (9):**
- `public function spawn(integer routine_id, sequence args = {})`: Startet eine unabhängige Prozedur asynchron auf einem leichtgewichtigen Java 25 Virtual Thread.
- `public function async(integer routine_id, sequence args = {})`: Startet eine Funktion asynchron im Hintergrund und liefert ein Job-/Future-Handle zurück.
- `public function await(atom job_id, atom timeout = -1)`: Wartet blockierend auf die Fertigstellung eines Jobs und liefert dessen Rückgabewert.
- `public function is_done(atom job_id)`: Prüft ohne Blockieren, ob ein asynchroner Hintergrund-Job bereits abgeschlossen ist.
- `public function await_all(sequence job_ids, atom timeout = -1)`: Wartet auf eine Liste mehrerer Jobs und sammelt alle Ergebnisse parallel ein.
- `public function join(atom thread_id, atom timeout = 0)`: Wartet auf das Beenden eines mit `spawn` gestarteten Virtual Threads.
- `public procedure sleep(atom seconds)`: Pausiert den aktuellen Virtual Thread für eine Dauer in Sekunden ohne Systemressourcen zu blockieren.
- `public function current()`: Gibt eine eindeutige Thread-ID des aktuell ausführenden Virtual Threads zurück.
- `public procedure cancel(atom job_or_thread_id)`: Bricht einen laufenden asynchronen Hintergrund-Job kontrolliert ab.

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

- **Zweck:** Go-Style Channels (CSP): Thread-sichere, blockierende oder nicht-blockierende Nachrichtenkanäle zur lock-freien Datenübergabe zwischen Threads.
- **Öffentliche Funktionen & Prozeduren (9):**
- `public function new(integer capacity = 0)`: Erstellt einen neuen CSP-Kanal mit optionaler Pufferkapazität (0 = ungebuffert/rendezvous).
- `public function send(atom ch, object val, atom timeout = -1)`: Sendet ein Datenpaket in den Kanal (blockiert bei vollem Kanal bis zum Abholen).
- `public function receive(atom ch, atom timeout = -1)`: Liest ein Datenpaket aus dem Kanal (blockiert bei leerem Kanal bis neue Daten eintreffen).
- `public function read(atom ch, object default_val = 0, atom timeout = -1)`: Liest Daten aus dem Kanal oder liefert einen Standardwert bei Timeout/Schließung.

- `public function try_send(atom ch, object val)` : Versucht sofort ein Paket zu senden ohne zu blockieren (1 = Erfolg, 0 = voll).
- `public function try_receive(atom ch)` : Versucht sofort ein Datenpaket zu lesen ohne den Thread zu blockieren.
- `public procedure close(atom ch)` : Schließt den Kanal und signalisiert allen Empfängern das Ende der Datenübertragung.
- `public function size(atom ch)` : Liefert die Anzahl der aktuell ungesehenen Datenpakete im Kanalpuffer.
- `public function is_closed(atom ch)` : Prüft, ob der Kanal geschlossen wurde (1 = geschlossen, 0 = offen).

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

- **Zweck:** Klassisches kooperatives Multitasking: Task-Erstellung, Zeitschlitz-Vergabe, Priorisierung und Scheduler-Yielding.
- **Öffentliche Funktionen & Prozeduren (2):**
- `public procedure task_delay(atom delaytime)` : Verzögert die Ausführung eines kooperativen Tasks um eine Mindestzeit.
- `public function create_thread(integer rid, sequence args = {})` : Erstellt einen kooperativen Multitasking-Thread innerhalb des Euphoria-Schedulers.

2.8.11 Fehlerbehandlung, Testing & Hilfsprogramme

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

- **Zweck:** Zentrale Fehler- und Ausnahmebehandlung: Programmatischer Absturz (`crash`), konfigurierbare Crash-Dateien und benutzerdefinierte Crash-Handler.
- **Öffentliche Funktionen & Prozeduren (5):**
- `public procedure crash(sequence fmt, object data={})` : Beendet das Programm sofort mit einer benutzerdefinierten Fehlermeldung und Stack-Trace.
- `public procedure crash_message(sequence msg)` : Konfiguriert eine zusätzliche informative Meldung, die im Fall eines Absturzes ausgegeben wird.
- `public procedure crash_file(sequence file_path)` : Bestimmt den Dateipfad, in den der Absturzbericht (`ex.err`) geschrieben wird.
- `public procedure warning_file(object file_path)` : Bestimmt die Zieldatei für Compiler- und Laufzeitwarnungen.
- `public procedure crash_routine(integer func)` : Registriert eine Aufräumroutine, die vor dem Programmende bei einem Absturz aufgerufen wird.

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

- **Zweck:** Integriertes Test-Framework: Test-Assertions (`test_equal` , `test_true` , `test_false`), Testsuiten-Reporting und Zusammenfassungen.
- **Öffentliche Funktionen & Prozeduren (12):**
- `public procedure set_test_verbosity(atom verbosity)` : Setzt den Detaillierungsgrad der Testausgaben auf der Konsole.
- `public procedure set_wait_on_summary(integer to_wait)` : Steuert, ob nach Testabschluss auf einen Tastendruck gewartet werden soll.
- `public procedure set_accumulate_summary(integer accumulate)` : Sammelt Testergebnisse über mehrere Testsuiten hinweg.
- `public function set_test_abort(integer abort_test)` : Konfiguriert, ob die Testsuite beim ersten aufgetretenen Fehler sofort abbricht.
- `public procedure test_report()` : Gibt die zusammenfassende Statistik (Erfolgreich, Fehlgeschlagen) aller Tests aus.
- `public procedure test_equal(sequence name, object expected, object outcome)` : Prüft, ob der tatsächliche Wert exakt mit dem erwarteten Wert übereinstimmt.
- `public procedure test_not_equal(sequence name, object a, object b)` : Prüft, ob zwei Werte ungleich sind.
- `public procedure test_true(sequence name, object outcome)` : Prüft, ob ein Ausdruck wahr (ungleich 0) ist.
- `public procedure assert(object name, object outcome)` : Prüft eine Bedingung und wirft einen Testfehler bei Nichterfüllung.
- `public procedure test_false(sequence name, object outcome)` : Prüft, ob ein Ausdruck falsch (gleich 0) ist.
- `public procedure test_fail(sequence name)` : Erfasst manuell einen fehlgeschlagenen Testfall mit Meldung.
- `public procedure test_pass(sequence name)` : Erfasst manuell einen erfolgreich bestandenen Testfall.

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

- **Zweck:** Allgemeine Sprachhilfen: Inline-If-Funktion (`iif`), bedingte Ausdrücke und Multi-Zuweisung (`assign`).
- **Öffentliche Funktionen & Prozeduren (3):**
- `public function iif(atom test, object ifTrue, object ifFalse)` : Inline-If (Ternärer Operator): Liefert `ifTrue` wenn `test` wahr ist, sonst `ifFalse` .
- `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)` : Weist mehreren Zielvariablen in einem kompakten Aufruf simultan Werte zu.
- `public function iff(atom test, object ifTrue, object ifFalse)` : Bedingte Funktionsauswertung (Alias für `iif`).

2.8.12 Windows-Plattformspezifische Module

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

- **Zweck:** Windows-spezifische Dialogboxen: Anzeige nativer Win32 MessageBox-Dialoge mit OK/Cancel/Yes/No-Buttons und Icons.
- **Öffentliche Funktionen & Prozeduren (1):**
- `public function message_box(sequence text, sequence title, object style)` : Zeigt eine native Windows-MessageBox mit Titel, Text und Schaltflächenstil an.

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

- **Zweck:** Windows-Systemtöne: Abspielen von System-Audiosignalen und akustischen Warnungen (`MessageBeep`).
- **Öffentliche Funktionen & Prozeduren (1):**
- `public procedure sound( atom sound_type = SND_DEFAULT )` : Spielt einen Windows-Systemklang (`MessageBeep`) ab.

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

- **Zweck:** Win32-spezifische Datentyp- und API-Konstanten (`HWND` , `HANDLE` , `DWORD` , `WPARAM` , `LPARAM`) für DLL-Aufrufe.
- **Hinweis & Wichtige Konstanten:** Definiert Win32-Typkonstanten für Shared-Library-Aufrufe (`C_HANDLE` , `C_HWND` , `C_WPARAM` , `C_LPARAM` , `C_HRESULT` , etc.). Enthält keine Funktionen.

2.8.13 Euphoria Interna, Debugger & Syntax-Tokenizer

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

- **Zweck:** RFC 4180-konforme CSV-Engine: Parsing und Serialisierung von Tabellendaten mit Trennzeichen-Konfiguration (Komma, Semikolon, Tabulator), Maskierung von Quotes (`""`), Zeilenumbrüchen und UTF-8-Encoding.
- **Öffentliche Funktionen & Prozeduren (5):**
- `public function format_field(object val, integer delimiter = ',', integer quote_all = 0)` : Formatiert einen einzelnen Wert nach RFC 4180 mit automatischer Anführungszeichen-Maskierung.
- `public function to_csv(sequence headers, sequence rows, integer delimiter = ',', integer quote_all = 0)` : Konvertiert Header- und Datenzeilen in einen vollständigen CSV-String.
- `public function parse(sequence csv_text, integer delimiter = ',')` : Zerlegt einen CSV-Text in Zeilen und Spalten (`{ {col1, col2, ...}, ... }`).
- `public function write_file(sequence filepath, sequence headers, sequence rows, integer delimiter = ',', integer quote_all = 0)` : Schreibt tabellarische Daten direkt in eine CSV-Datei.
- `public function read_file(sequence filepath, integer delimiter = ',', integer has_header = 1)` : Liest eine CSV-Datei ein und liefert `{headers, rows}` .

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

- **Zweck:** Native Microsoft Excel (.xlsx) OpenXML-Engine: Erzeugt und liest moderne `.xlsx` -Arbeitsmappen direkt über die Java 25 Runtime – ohne externe Abhängigkeiten oder Apache POI. Kompatibel mit Excel, LibreOffice Calc und Google Sheets.
- **Öffentliche Funktionen & Prozeduren (3):**
- `public function write_file(sequence filepath, sequence headers, sequence rows, sequence sheet_name = "Tabelle1")` : Erzeugt eine vollwertige `.xlsx` -Datei mit formatierten Kopfzeilen, Zahlenspalten und Datumsfeldern.
- `public function export_table(sequence filepath, sequence headers, sequence rows, sequence sheet_name = "Export")` : Komfortable Export-Funktion für Datenbankabfragen und UI-Tabellen.
- `public function read_file(sequence filepath)` : Liest eine `.xlsx` -Datei ein und extrahiert Tabellenzeilen und Zellwerte (`{1, rows}` bei Erfolg).

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

- **Zweck:** Native Business-PDF-Engine: Generiert Standard-PDF-1.4-Geschäftsberichte, Monatsstatistiken, Rechnungen und Tabellenauszüge mit Firmenbannern, Metadatenkarten, Zebrastreifen und automatischer Mehrseiten-Paginierung.
- **Öffentliche Funktionen & Prozeduren (2):**
- `public function create_report(sequence filepath, sequence title, sequence metadata, sequence headers, sequence rows, sequence options = {})` : Erstellt einen formatierten PDF-Geschäftsbericht mit Titelbanner, Metadaten, Tabellenlayout und Seitennummerierung ("Seite X von Y").
- `public function export_table(sequence filepath, sequence title, sequence headers, sequence rows)` : Schneller Export einer Datentabelle als PDF-Dokument.

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

- **Zweck:** Enterprise-Sicherheits- & Authentifizierungs-Modul: PBKDF2-Passwort-Hashing mit 65.536 Iterationen und Salt, Timing-sichere Hash-Validierung, JSON Web Tokens (JWT HS256) und kryptografische Zufallstoken.
- **Öffentliche Funktionen & Prozeduren (5):**

- `public function hash_password(sequence password, sequence salt = "")` : Erzeugt einen sicheren PBKDF2-HMAC-SHA256-Passworthash mit individuellem Salt.
- `public function verify_password(sequence password, sequence hash_str)` : Prüft ein Klartext-Passwort gegen einen PBKDF2-Hash in konstanter Zeit (Schutz vor Timing-Angriffen).
- `public function random_token(integer byte_count = 32)` : Generiert einen kryptografisch sicheren Zufallstoken in Hexadezimaldarstellung.
- `public function create_jwt(sequence payload, sequence secret_key, integer expiry_secs = 3600)` : Erstellt ein signiertes JSON Web Token (JWT) mit HMAC-SHA256 und Ablaufzeit (`exp`).
- `public function verify_jwt(sequence token, sequence secret_key)` : Validiert die Signatur und Ablaufzeit eines JWT-Tokens und liefert die Payload zurück (`{1, payload_json}`).

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

- **Zweck:** HTTP-Session-Management für Intranet- und Webanwendungen: Server-seitige Attribut-Verwaltung, automatische Cookie-Header-Erstellung (`Set-Cookie`) und Session-Lebenszeitüberwachung.
- **Öffentliche Funktionen & Prozeduren (7):**
- `public function extract_sid(sequence cookie_header, sequence cookie_name = "EUI_SESSION")` : Liest die Session-ID aus einem HTTP-`Cookie`-Header aus.
- `public function start(sequence cookie_str = "", sequence cookie_name = "EUI_SESSION")` : Startet eine neue Session oder setzt eine bestehende fort (`{sid, is_new}`).
- `public procedure set(sequence sid, sequence key, object val)` : Speichert ein Schlüssel-Wert-Attribut in der Session.
- `public function get(sequence sid, sequence key, object default_val = {})` : Liest ein Attribut aus der Session aus.
- `public function has(sequence sid, sequence key)` : Prüft, ob ein Attribut in der Session existiert (1 = ja, 0 = nein).
- `public procedure destroy(sequence sid)` : Beendet und löscht eine aktive Session.
- `public function cookie_header(sequence sid, integer max_age = 86400, sequence cookie_name = "EUI_SESSION", sequence path = "/", integer http_only = 1)` : Erzeugt den HTTP-`Set-Cookie`-Header mit `HttpOnly` , `Path` und `SameSite=Lax` .

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

- **Zweck:** Integrierter Task-Scheduler: Führt wiederkehrende Hintergrundaufgaben (Intervall-Timer oder tägliche Uhrzeiten wie 02:00 Uhr) auf leichtgewichtigen Java 25 Virtual Threads aus.
- **Öffentliche Funktionen & Prozeduren (6):**
- `public function schedule_interval(sequence name, atom interval_seconds, integer routine_id, sequence args = {})` : Registriert eine Aufgabe, die alle N Sekunden ausgeführt wird.
- `public function schedule_daily(sequence name, integer hour, integer minute, integer routine_id, sequence args = {})` : Registriert eine tägliche Aufgabe zu einer festen Uhrzeit (HH:MM).
- `public procedure start()` : Startet den Hintergrund-Scheduler-Thread.
- `public procedure stop()` : Hält den Hintergrund-Scheduler an.
- `public function list_jobs()` : Gibt eine Liste aller registrierten Jobs zurück.
- `public function trigger_now(sequence name)` : Führt einen registrierten Job sofort einmalig aus.

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

- **Zweck:** Server-Sent Events (SSE) Streaming-Engine: Erzeugt W3C-konforme `text/event-stream`-Nachrichten für Push-Benachrichtigungen an Web-Browser ohne WebSocket-Overhead.
- **Öffentliche Funktionen & Prozeduren (2):**
- `public function event(sequence event_name, sequence data, sequence id = "", integer retry_ms = 0)` : Formatiert eine SSE-Nachricht mit Event-Name, Datenblock (auch mehrzeilig) und optionaler ID.
- `public function headers()` : Liefert die Standard-HTTP-Header für eine persistente SSE-Streamingverbindung.

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

- **Zweck:** Interaktiver VM-Debugger: Call-Stack-Inspektion, Breakpoints setzen, Einzelschritt-Ausführung (Step Over/Into) und Symbolabfragen.
- **Öffentliche Funktionen & Prozeduren (23):**
- `public function call_stack()` : Gibt den aktuellen Aufrufstapel als Sequenz von Frame-Informationen zurück.
- `public procedure initialize_debugger( atom init_ptr )` : Initialisiert die internen Strukturen und Zeiger des Debuggers.
- `public procedure set_debug_rid( DEBUG_ROUTINE rtn, integer rid )` : Registriert eine Callback-Routine für Debugging-Ereignisse.
- `public function read_object( atom sym )` : Liest ein Euphoria-Objekt direkt aus der Speicheradresse eines Symbols.
- `public procedure trace_off()` : Deaktiviert den Trace-Modus für nachfolgende Anweisungen.
- `public procedure disable_trace()` : Schaltet den interaktiven Debugger dauerhaft für die aktuelle Session aus.
- `public procedure step_over()` : Führt die nächste Anweisung aus, ohne in Unterroutinen zu verzweigen.
- `public procedure abort_program()` : Beendet die Programmausführung kontrolliert aus dem Debugger.
- `public function get_current_line()` : Ermittelt die aktuell ausgeführte Quellcodezeile.

- `public function symbol_lookup( sequence name, integer line = get_current_line(), atom pc = get_pc() )` : Sucht ein Symbol anhand seines Namens in der Symboltabelle.
- `public function get_pc()` : Gibt den aktuellen Programmzähler (PC) der virtuellen Maschine zurück.
- `public function is_novalue( atom sym_ptr )` : Prüft, ob ein Symbol uninitialisiert ist (NOVALUE-Zustand).
- `public function debugger_call_stack()` : Liefert den formatierten Call-Stack für das Debugger-Interface.
- `public function break_routine( atom routine_sym, integer enable )` : Setzt einen automatischen Breakpoint auf den Einsprungspunkt einer Routine-ID.
- `public function get_name( atom sym )` : Ermittelt den Bezeichnernamen eines Symbols aus der VM-Tabelle.
- `public function get_source( integer line )` : Holt den Quelltext einer bestimmten Quellcodezeile einer Datei.
- `public function get_file_no( integer line )` : Liefert die interne Dateinummer, in der ein Symbol definiert wurde.
- `public function get_file_name( integer file_no )` : Liefert den Dateipfad einer registrierten internen Dateinummer.
- `public function get_file_line( integer line )` : Gibt die Deklarationszeile eines Symbols zurück.
- `public function get_next( atom sym )` : Ermittelt das nachfolgende Symbol in der verketteten Symbolliste.
- `public function is_variable( atom sym_ptr )` : Prüft, ob das Symbol eine Variable (keine Konstante/Routine) ist.
- `public function get_parameter_syms( atom rtn_sym )` : Gibt die Liste der Symbol-Pointer der formalen Parameter einer Routine zurück.
- `public function get_symbol_table()` : Liefert den Wurzelzeiger auf die vollständige Symboltabelle der VM.

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

- **Zweck:** Compiler- und Interpreter-Laufzeitinformationen: VM-Versionsnummer, Build-Nummer, Plattformbezeichnung und Wortbreite.
- **Öffentliche Funktionen & Prozeduren (17):**
- `public function platform_name()` : Gibt den Namen der aktuellen Plattform (z. B. 'Linux', 'Windows') zurück.
- `public function arch_bits()` : Liefert die Bitbreite der virtuellen Maschine (64 unter Linux 64-Bit).
- `public function version()` : Gibt die Versionsnummer der Euphorix-Laufzeitumgebung als Sequenz zurück.
- `public function version_major()` : Liefert die Hauptversionsnummer (z. B. 1 oder 4).
- `public function version_minor()` : Liefert die Nebenversionsnummer (z. B. 0 oder 1).
- `public function version_patch()` : Liefert die Patch-Level-Nummer.
- `public function version_node(integer full = 0, integer n = 12)` : Liefert die Build-Knoten-Information.
- `public function version_revision()` : Gibt den Git-Commit-Hash oder Revisionsstring des Builds zurück.
- `public function version_date(integer full = 0)` : Gibt das Build-Datum der VM zurück.
- `public function version_type()` : Liefert den Releasetyp ('release', 'beta', 'development').
- `public function version_string(integer full = 0)` : Liefert die formatierte Versionszeichenkette.
- `public function version_string_short()` : Gibt die Kurzform der Versionsnummer zurück.
- `public function version_string_long(integer full = 0)` : Liefert die ausführliche Versionsangabe inklusive Build- und JVM-Infos.
- `public function euphoria_copyright()` : Gibt die Copyright-Information von OpenEuphoria zurück.
- `public function pcre_copyright()` : Gibt die Copyright-Information der Regex-Bibliothek zurück.
- `public function all_copyrights()` : Gibt eine formatierte Zusammenfassung aller Lizenzen und Urheberrechte zurück.
- `public function start_time()` : Gibt den Systemzeitstempel des Programmstarts zurück.

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

- **Zweck:** Tabellen aller reservierten OpenEuphoria-Schlüsselwörter und integrierten Built-in-Routinennamen.
- **Hinweis & Wichtige Konstanten:** Definiert die Sequenz-Konstanten `keywords` (alle reservierten Schlüsselwörter wie `if`, `while`, `function`) und `builtins` (alle Built-in-Routinen).

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

- **Zweck:** Umfassender Debug-Aggregator: Bindet alle Standardmodule ein, um beim Debuggen Zugriff auf alle Bibliotheken zu gewährleisten.
- **Hinweis & Wichtige Konstanten:** Sammel-Include, das automatisch alle Standardbibliotheken importiert, damit während einer interaktiven Debugger-Sitzung alle Symbole erreichbar sind.

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

- **Zweck:** Interne Speicheroffsets und Strukturen der Euphoria-Symboltabelle der virtuellen Maschine.
- **Hinweis & Wichtige Konstanten:** Definiert Speicher-Offset-Konstanten (`ST_OBJ`, `ST_NEXT`, `ST_MODE`, `ST_SCOPE`, `ST_NAME`, etc.) für die interne Symboltabellenstruktur.

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

- **Zweck:** Syntax-Highlighting Engine: Lexikalische Farbgebung und HTML-/ANSI-Einfärbung von Euphoria-Quellcode.
- **Öffentliche Funktionen & Prozeduren (6):**
- `public procedure set_colors(sequence pColorList)` : Setzt die Farbpalette für die Quellcode-Syntaxhervorhebung.
- `public procedure init_class()` : Initialisiert die Syntaxfarben und Stilklassen für den Colorizer.

- `public function new()` : Erstellt eine neue Instanz des Syntax-Colorizers.
- `public procedure reset(atom state = g_state)` : Setzt den Zustand des Syntax-Colorizers auf Standardwerte zurück.
- `public procedure keep_newlines(integer val = 1, atom state = g_state)` : Steuert, ob Zeilenumbrüche beim Einfärben des Quellcodes erhalten bleiben.
- `public function SyntaxColor(sequence pline, atom state=g_state, multiline_token multi = 0)` : Färbt einen Euphoria-Quellcodetext gemäß konfiguriertem Schema ein.

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

- **Zweck:** Vollwertiger Euphoria-Quellcode-Lexer / Tokenizer: Zerlegt Euphoria-Code in Tokens mit Zeilen-, Spalten- und Typinformationen.
- **Öffentliche Funktionen & Prozeduren (15):**
- `public function error_string(integer err)` : Liefert die Textbeschreibung eines Tokenizer-Fehlers.
- `public function new()` : Erstellt einen neuen Lexer/Tokenizer-Zustand.
- `public procedure reset(atom state = g_state)` : Setzt den Tokenizer-Status auf Ausgangswerte zurück.
- `public procedure keep_builtins(integer val = 1, atom state = g_state)` : Steuert, ob Built-in-Routinennamen als eigenständige Tokens erkannt werden.
- `public procedure keep_keywords(integer val = 1, atom state = g_state)` : Steuert die Erkennung von reservierten Sprachschlüsselwörtern.
- `public procedure keep_whitespace(integer val = 1, atom state = g_state)` : Bestimmt, ob Leerzeichen und Tabs als Tokens erhalten bleiben.
- `public procedure keep_newlines(integer val = 1, atom state = g_state)` : Bestimmt, ob Zeilenumbrüche als Tokens erhalten bleiben.
- `public procedure keep_comments(integer val = 1, atom state = g_state)` : Bestimmt, ob Quellcode-Kommentare als Tokens ausgegeben werden.
- `public procedure return_literal_string(integer val = 1, atom state = g_state)` : Steuert die Rückgabe von String-Literalen mit oder ohne Anführungszeichen.
- `public procedure string_strip_quotes(integer val = 1, atom state = g_state)` : Entfernt umschließende Quotes aus String-Tokens.
- `public procedure string_numbers(integer val = 1, atom state = g_state)` : Konvertiert numerische Tokens automatisch in Atome.
- `public function last_multiline_token()` : Liefert Statusinformationen zu mehrzeiligen Tokens.
- `public function tokenize_string(sequence code, atom state = g_state, integer stop_on_error = TRUE, multiline_token multi = 0)` : Zerlegt einen Quellcode-String in eine Liste typisierter Tokens.
- `public function tokenize_file(sequence fname, atom state = g_state, integer mode = io:BINARY_MODE )` : Liest eine Quellcodedatei ein und zerlegt sie vollständig in Tokens.
- `public procedure show_tokens(integer fh, sequence tokens)` : Gibt eine formatierte Übersicht aller erkannten Tokens auf der Konsole aus.

2.8.14 Künstliche Intelligenz, MCP, LLM & RAG

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

- **Zweck:** Model Context Protocol (MCP, Spezifikation 2024-11-05): Verbindet Euphorix-Programme als MCP-Server mit modernen KI-Assistenten (Claude Desktop, Antigravity, Cursor) oder steuert externe Werkzeuge als autonomer MCP-Client über `stdio` (Subprozesse) oder HTTP mit Server-Sent Events (SSE).
- **Öffentliche Funktionen & Prozeduren (10):**
- `public function server_create(sequence name, sequence version, sequence capabilities = {})` : Erstellt eine neue MCP-Server-Instanz mit Namen, Version und deklarierten Fähigkeiten.
- `public function server_register_tool(sequence srv, sequence name, sequence description, sequence parameters, integer routine_id)` : Registriert ein ausführbares Werkzeug inklusive Parameter-JSON-Schema und ausführender Handler-Routine.
- `public procedure server_run_stdio(sequence srv)` : Startet den Server im `stdio`-Modus (Standard Input/Output) für die Einbindung in KI-Desktop-Clients und CLI-Agenten.
- `public procedure server_run_http(sequence srv, integer port = 8085)` : Startet den MCP-Server als leichtgewichtigen HTTP/SSE-Dienst auf dem angegebenen Port.
- `public function client_stdio(sequence command_line)` : Startet einen MCP-Server als Subprozess über `stdio` und liefert das Client-Handle `{1, client}`.
- `public function client_http(sequence base_url)` : Verbindet einen MCP-Client über HTTP/SSE mit einem entfernten MCP-Server.
- `public function client_init(sequence cli, sequence client_name = "Euphorix-Client", sequence client_version = "1.0.0")` : Führt den Protokoll-Handshake (`initialize` und `notifications/initialized`) mit dem Server durch.
- `public function client_list_tools(sequence cli)` : Ruft die Liste aller vom Server bereitgestellten Werkzeuge ab (`{1, tools_list, updated_cli}`).
- `public function client_call_tool(sequence cli, sequence name, sequence arguments = {})` : Führt ein registriertes Werkzeug auf dem Server aus und liefert das textuelle Ergebnis (`{1, result_text, raw_json, updated_cli}`).
- `public procedure client_close(sequence cli)` : Beendet die Verbindung zum MCP-Server und schließt alle Subprozess-Streams sauber.

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

- **Zweck:** Universeller Client für Large Language Models (LLMs) und semantische Vektoreinbettungen. Unterstützt 100% Offline-Betrieb (deterministische Mock-Engine), lokales Ollama (`/api/chat` , `/api/embeddings`), OpenAI (`gpt-4o-mini` , `/chat/completions`) und Anthropic Claude (`/v1/messages`).
- **Öffentliche Funktionen & Prozeduren (5):**
- `public function client(integer provider, sequence options = {})` : Initialisiert einen LLM-Client für `PROVIDER_MOCK` , `PROVIDER_OLLAMA` , `PROVIDER_OPENAI` oder `PROVIDER_ANTHROPIC` .
- `public function complete(sequence cli, sequence prompt, sequence options = {})` : Sendet einen Text-Prompt an das Sprachmodell und liefert die generierte Antwort (`{1, text, raw}`).
- `public function chat(sequence cli, sequence messages, sequence options = {})` : Führt eine mehrstufige Chat-Konversation (`{role, content}`) mit konfigurierbarer `temperature` und `max_tokens` aus.
- `public function embed(sequence cli, sequence text_str, sequence options = {})` : Berechnet einen normalisierten Vektor (64-dim TF-IDF Feature-Vektor im Mock-Modus oder hochdimensionale dichte Embeddings via Ollama/OpenAI).
- `public function cosine_similarity(sequence vec_a, sequence vec_b)` : Berechnet die Kosinus-Ähnlichkeit zwischen zwei Vektoren im Bereich von 0.0 (orthogonal/unähnlich) bis 1.0 (identisch).

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

- **Zweck:** Retrieval-Augmented Generation (RAG) Engine: Durchsucht Dokumente und Fachhandbücher hybrid über SQLite Full-Text Search (FTS5 / BM25) und Cosine-Vektorähnlichkeit, augmentiert LLM-Prompts mit relevanten Kontextblöcken und generiert quellenbelegte Antworten.
- **Öffentliche Funktionen & Prozeduren (10):**
- `public function create_kb(sequence name, sequence llm_client, sequence db_path = ":memory:")` : Erstellt eine neue RAG-Wissensdatenbank mit SQLite FTS5-Tabellen und Vektorspeicher.
- `public procedure close_kb(sequence kb)` : Schließt die Wissensdatenbank und gibt alle SQLite-Ressourcen frei.
- `public function chunk_text(sequence text_str, integer target_size = 500, integer overlap = 100)` : Zerlegt Fließtexte intelligent entlang von Absatz- und Satzgrenzen in überlappende Abschnitte.
- `public function add_document(sequence kb, sequence doc_id, sequence title, sequence content, sequence metadata = {})` : Zerlegt ein Dokument automatisch in Abschnitte, generiert Embeddings und indiziert es in SQLite FTS5.
- `public function add_text(sequence kb, sequence doc_id, sequence title, sequence chunk_text, integer chunk_index, sequence metadata = {})` : Fügt einen einzelnen vorverarbeiteten Textabschnitt in die Wissensbasis ein.
- `public function retrieve(sequence kb, sequence query, integer top_k = 3, sequence mode = "hybrid")` : Führt eine hybride Recherche (BM25 + Cosine Similarity via Reciprocal Rank Fusion) durch und liefert die relevantesten Abschnitte.
- `public function ask(sequence kb, sequence query, sequence options = {})` : Führt die komplette RAG-Pipeline aus (Retrieval, Prompt-Augmentation, LLM-Generierung und Quellenzuordnung).
- `public function answer(sequence rag_result)` : Extrahiert den reinen Antworttext aus einem `ask()` -Ergebnis.
- `public function sources(sequence rag_result)` : Extrahiert die Liste aller zitierten Quellen mit Titel, Abschnittsnummer und Relevanzbewertung.
- `public function format_sources(sequence rag_result)` : Formatiert die Quellennachweise in eine übersichtliche, lesbare Zitierleiste.

---

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

Das Modul `std/java.e` ermöglicht den vollständigen, bidirektionalen Zugriff auf das Java 25-Ökosystem und beliebige externe JAR-Archive direkt aus OpenEuphoria.

3.1 Laden externer JAR-Dateien (`java:load_jar`)

Sie können beliebige `.jar` -Bibliotheken zur Laufzeit dynamisch in den Classpath laden:

```
include std/java.e as java

integer ok = java:load_jar("lib/meine_bibliothek.jar")
if not ok then
    puts(2, "Fehler: JAR konnte nicht geladen werden!\n")
end if
```

3.2 Klassenauflösung & Objekterzeugung

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

Ermittelt das Klassen-Handle (`Class<?>`) anhand des vollqualifizierten Namens:

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

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

Ruft den passenden Konstruktor auf und instanziiert das Java-Objekt. Gibt ein numerisches Handle (`> 0`) zurück:

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

3.3 Methodenaufrufe (statisch & dynamisch)

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

Ruft eine Instanzmethode auf:

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

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

Ruft eine statische Klassenmethode auf:

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

3.4 Lesen und Schreiben von Feldern

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

Liest den Wert eines Feldes (Instanz- oder statisches Feld):

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

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

Schreibt einen neuen Wert in ein zugängliches Feld:

```
java:set_field(mein_obj, "konfiguration", "Produktion")
```

3.5 Automatisches Typ-Matching & Konvertierungen

Der Interop-Layer konvertiert Datentypen bidirektional und wählt automatisch die am besten passende Java-Methode anhand eines Scoring-Verfahrens aus:

Euphoria-Typ	Java-Typen (automatische Konvertierung)
`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` (Zahlen)	`byte[]`, `int[]`, `long[]`
`sequence` (Objekte)	`java.util.List<?>`, `java.lang.Object[]`
Java-Handle (`atom > 0`)	Referenz auf registrierte Java-Instanz
`0`	`null`

3.6 Ereignis-Listener & Callbacks (`java:create_listener`)

Über **`java:create_listener(sequence interface_name, integer routine_id)`** erzeugt die Laufzeitumgebung einen dynamischen `java.lang.reflect.Proxy`. Wenn Swing oder ein Java-Framework eine Event-Methode aufruft, wird der Aufruf automatisch an die entsprechende Euphoria-Routine weitergeleitet:

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

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

Die Handles bleiben identisch: Ruft ein Callback auf `sender` auf, entspricht der Wert exakt dem Handle, das bei der Erzeugung des Widgets vergeben wurde (`sender = btn`).

3.7 Subprozess-Pipes mit Virtual Threads

Für das Ausführen externer Linux-Befehle mit voller Streaming-Kontrolle stehen Virtual-Thread-Pipes bereit:

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

---

4. Grafische Benutzeroberflächen mit Swing & AWT (`std/gui.e`)

Das Modul `std/gui.e` bietet eine declarative, intuitive High-Level-API zur Entwicklung grafischer Desktop-Anwendungen für Linux.

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

Beim Initialisieren der GUI (`gui:init_gui()`) aktiviert das Modul standardmäßig das native **GTK+ Look and Feel** (`com.sun.java.swing.plaf.gtk.GTKLookAndFeel`):

- Die Anwendung übernimmt Farben, Schaltflächen, Eingabefeld-Designs, Fokus-Rahmen und Bildlaufleisten des aktiven Linux-Desktop-Themes (GNOME, Ubuntu, XFCE).
- Mit der Prozedur **`gui:set_theme(sequence theme_name)`** lässt sich das Erscheinungsbild der gesamten Benutzeroberfläche jederzeit anpassen:
- `"gtk"` : Natives Linux-Desktop-Theme (Standard)
- `"flatlaf"` oder `"flatlaf-light"` : Moderner, flacher Hell-Modus (via `com.formdev.flatlaf`)
- `"flatlaf-dark"` oder `"dark"` : Eleganter Dunkelmodus für moderne Desktop-Tools
- `"nimbus"` : Moderner Vektor-Look von Swing
- `"system"` : Standard-Systemdesign des Hostsystems

```
include std/gui.e as gui

-- Vor dem Erzeugen der Widgets auf Dark-Mode umschalten:
gui:set_theme("flatlaf-dark")

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

4.2 Typografie & Subpixel-Antialiasing

Durch die JVM-Parameter `-Dawt.useSystemAAFontSettings=lcd` und `-Dswing.aatext=true` sowie die Marlin-2D-Engine rendert die GUI Texte mit gestochen scharfem LCD-Subpixel-Antialiasing:

- **Standardschriftart**: `Ubuntu` (bzw. `SansSerif`) in Größe 13pt.
- **Individuelle Anpassung**: Mit `gui:set_font(widget, "Ubuntu", 16, 1)` lassen sich Schriftfamilie, Größe und Fettung (`is_bold = 1`) jederzeit festlegen.

4.3 Fensterverwaltung (`window` , `show` , `wait_window`)

```
-- Fenster anlegen (Titel, Breite, Höhe, Schließaktion)
atom win = gui:window("Meine Anwendung", 600, 400, gui:CLOSE_ON_EXIT)

-- Fenster anzeigen
gui:show(win)

-- Hauptschleife: Blockiert den Thread, bis das Fenster geschlossen wird
gui:wait_window(win)
```

Konstanten für Schließaktionen:

- `gui:CLOSE_ON_EXIT` (Standard): Beendet die Anwendung beim Klick auf das Schließen-Kreuz.
- `gui:DISPOSE_ON_CLOSE` : Schließt das Fenster und gibt Grafikressourcen frei.
- `gui:HIDE_ON_CLOSE` : Versteckt das Fenster nur.

4.4 Layout-Manager mit Abstandssteuerung (`hgap` / `vgap`)

- **`gui:BORDER_LAYOUT`** : Unterteilt den Container in 5 Regionen (`gui:NORTH` , `gui:SOUTH` , `gui:EAST` , `gui:WEST` , `gui:CENTER`).
- **`gui:FLOW_LAYOUT`** : Ordnet Elemente zentriert in einer fließenden Reihe an.
- **`gui:GRID_LAYOUT`** : Exaktes Tabellenraster mit Spalten, Zeilen und Abständen.

```
-- Erzeugt ein 5-Zeilen, 2-Spalten Formular-Panel mit 14px H-Gap und 12px V-Gap:
atom pnl = gui:panel(gui:GRID_LAYOUT, 5, 2, 14, 12)
```

4.5 Widget-Komponenten-Referenz

Funktion	Beschreibung
`gui:text_label(sequence text, integer align = ALIGN_LEFT) -> atom`	Erzeugt ein Textlabel (`ALIGN_LEFT` , `ALIGN_CENTER` , `ALIGN_RIGHT`).
`gui:text_field(sequence text = "", integer columns = 20) -> atom`	Einzeiliges Texteingabefeld.
`gui:text_area(sequence text = "", integer rows = 5, integer cols = 20) -> atom`	Mehrzeiliges Textfeld.
`gui:scroll_pane(atom client_widget) -> atom`	Scroll-Container für Textbereiche oder Listen.
`gui:button(sequence text, integer on_click_rid = 0) -> atom`	Druckknopf mit optionalem Klick-Callback.
`gui:check_box(sequence text, integer checked = 0) -> atom`	Kontrollkästchen (Checkbox).
`gui:radio_button(sequence text, integer selected = 0) -> atom`	Optionsfeld (Radio-Button).
`gui:combo_box(sequence items = {}) -> atom`	Dropdown-Auswahlbox.
`gui:progress_bar(integer min_val = 0, integer max_val = 100) -> atom`	Fortschrittsbalken mit Textanzeige.
`gui:slider(integer min_val = 0, integer max_val = 100, integer init_val = 50) -> atom`	Schieberegler.
`gui:panel(integer layout_type = FLOW_LAYOUT, integer rows = 0, integer cols = 0, integer hgap = 0, integer vgap = 0) -> atom`	Panel-Container zur Strukturierung.

4.6 Eigenschaften, Styling & Polsterung

- **`gui:set_text(atom widget, sequence text)`** / **`gui:get_text(atom widget) -> sequence`** : Text setzen oder lesen (z. B. aus Eingabefeldern oder Labels).
- **`gui:set_value(atom widget, integer val)`** / **`gui:get_value(atom widget) -> integer`** : Wert von Fortschrittsbalken oder Schiebereglern.
- **`gui:is_selected(atom widget) -> integer`** / **`gui:set_selected(atom widget, integer selected)`** : Status von Checkboxen und Radio-Buttons abfragen oder setzen.
- **`gui:set_padding(atom widget, integer top, integer left, integer bottom, integer right)`** : Innenabstand (Padding in Pixeln).
- **`gui:set_font(atom widget, sequence name = "Ubuntu", integer size = 13, integer is_bold = 0)`** : Schriftart anpassen.
- **`gui:set_enabled(atom widget, integer enabled)`** : Widget aktivieren oder ausgrauen.

4.7 Standard-Dialoge (Message, Input, Confirm, FileDialog)

- **`gui:msgbox(sequence title, sequence message, integer icon_type = MSG_INFO)`** :

Zeigt einen Dialog an (`MSG_INFO` , `MSG_WARN` , `MSG_ERROR` , `MSG_QUESTION`).

- **`gui:input_box(sequence title, sequence prompt, sequence default_val = "") -> sequence`** :

Fordert eine Texteingabe an.

- **`gui:confirm_box(sequence title, sequence message) -> integer`** :

Ja/Nein-Bestätigungsdialog (gibt `1` für Ja, `0` für Nein zurück).

- **`gui:file_dialog(sequence title = "Datei auswählen", integer is_save = 0, sequence start_dir = ".") -> sequence`** :

Nativer Datei-Öffnen- oder -Speichern-Dialog. Gibt den ausgewählten Pfad zurück oder `""` bei Abbruch.

4.8 Vollständiges Demo-Programm (`demo/gui_demo.ex`)

Das folgende vollständige Programm demonstriert ein sauberes, zentriertes zweispaltiges Formular mit Callbacks und voller UTF-8-Unterstützung:

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

-- 1. Fenster anlegen (620x460 Pixel)
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)

-- 2. Header
atom pnl_top = gui:panel(gui:GRID_LAYOUT, 2, 1, 0, 6)
atom lbl_title = gui:text_label("OpenEuphoria GUI: Native Linux-Oberfläche", gui:ALIGN_CENTER)
gui:set_font(lbl_title, "Ubuntu", 17, 1)
atom lbl_sub = gui:text_label("100% UTF-8 Unterstützung für Umlaute (ä, ö, ü, ß) & moderne Schriftarten", 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)

-- 3. Formularbereich (5 Zeilen, 2 Spalten)
atom pnl_center = gui:panel(gui:GRID_LAYOUT, 5, 2, 14, 12)
atom lbl_name = gui:text_label("Benutzername:")
atom txt_name = gui:text_field("Euphoria Entwickler", 20)

atom lbl_lang = gui:text_label("Bevorzugte Plattform:")
atom cmb_lang = gui:combo_box({"Linux 64-Bit (Höchste Leistung)", "Ubuntu / Debian (Zuverlässig)", "Arch / Fedora (Aktuell)"})

atom lbl_mode = gui:text_label("Einstellungen:")
atom cb_expert = gui:check_box("Automatische Überprüfung aktivieren", 1)

atom lbl_pbar = gui:text_label("Fortschritt:")
atom pbar = gui:progress_bar(0, 100)
gui:set_value(pbar, 65)

atom lbl_status_title = gui:text_label("Aktueller Status:")
atom lbl_status = gui:text_label("Bereit für Eingaben & Aktionen")
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)

-- 4. Callbacks
integer count = 0
procedure on_submit(atom sender)
    count += 1
    sequence name = gui:get_text(txt_name)
    gui:set_text(lbl_status, sprintf("Aktion #%d von '%s' ausgeführt", {count, name}))
    gui:msgbox("Bestätigung",
        sprintf("Hallo %s!\n\nAktion erfolgreich im Swing Event-Thread verarbeitet!\nUmlaute-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("Datei auswählen", 0, ".")
    if length(path) > 0 then
        gui:set_text(lbl_status, "Ausgewählt: " & path)
    end if
end procedure

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

-- 5. Buttons (Süd)
atom pnl_bottom = gui:panel(gui:FLOW_LAYOUT)
atom btn_submit = gui:button("Aktion ausführen", routine_id("on_submit"))
atom btn_file   = gui:button("Datei auswählen...", routine_id("on_file_pick"))
atom btn_quit   = gui:button("Schließen",          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)
```

```
-- 6. Anzeigen & Warten
gui:show(win)
gui:wait_window(win)
```

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

5.1 Vollständige UTF-8-Integration im Java 25 Interpreter

1. **String-Literale im Quellcode**:

Der Lexer liest Quellcodedateien im UTF-8-Format ein. Zeichen `> 127` (z. B. `ä`, `ö`, `ü`, `ß`, `€`) werden als echte UTF-8-Multi-Byte-Sequenzen in die Bytecode-Konstantentabelle eingetragen (`"ü"` $\rightarrow$ `{195, 188}`).

1. **Robuste Konvertierung in `Value.toUtf8String()`**:

Beim Übertragen an Java (Swing-GUI, JDBC, HTTP, JSON) decodiert die VM die Bytesequenzen verlustfrei. Sollten unvollständige Einzelbytes vorliegen, greift ein automatischer Fallback auf ISO-8859-1, sodass niemals fehlerhafte Zeichen (`\uFFFD`) im GUI oder in der Ausgabe entstehen.

5.2 Zeichengenaue Unicode-Operationen (`std/utf8.e`)

- `utf8:len(s)`: Anzahl der tatsächlichen Unicode-Zeichen (Codepoints).
- `utf8:slice(s, start_cp, end_cp)`: Schneidet nach Zeichenposition statt Byte-Index.
- `utf8:upper(s)` / `utf8:lower(s)`: Echte Unicode-Groß-/Kleinschreibung für alle Alphabete und Umlaute.
- `utf8:to_codepoints(s)` / `utf8:from_codepoints(cps)`: Umwandlung in 32-Bit-Unicode-Zahlen (z. B. `'€'` $\rightarrow$ `8364`).

---

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

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

Das Modul `std/json.e` nutzt intern **Jackson Databind**:

- Schnelles Parsen und Pretty-Printing (`json:parse`, `json:stringify`, `json:pretty`).
- Direkte Kompatibilität mit Datenbank-Ergebnissen (`sqlite:query_assoc`, `pgsql:query_assoc`).

6.2 RFC 9535 JSONPath Engine mit Pfadabfragen

- **RFC 9535 JSONPath**: Direkte Auswertung komplexer Pfadausdrücke mit Filtern:

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

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

7.1 Deklarative Schema-Validierung nach Draft 2020-12

Validiert JSON-Datenstrukturen gegen den aktuellen Standard **Draft 2020-12**:

- `json_schema:validate(data, schema) -> sequence`: Gibt `{valid, error_list}` zurück.
- `json_schema:is_valid(data, schema) -> integer`: Schnelle boolesche Prüfung (`1` oder `0`).

---

8. Dokumenten- & Tabellenformate: XML, CSV, Excel & PDF (`std/xml.e` , `std/csv.e` , `std/excel.e` , `std/pdf_report.e`)

Euphorix bietet umfassende Standardmodule für die Erstellung, Verarbeitung und den Austausch strukturierter Dokumente, Geschäftsberichte und Tabellenkalkulationen, vollständig autark und ohne externe Fremdbibliotheken.

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

Basiert auf den JDK-eigenen XML-Engines (`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, "//katalog/buch[@id='B1']/titel")` oder Skalarfunktionen `xml:xpath(doc, "sum(//preis)")` .
- **XSD Schema-Validierung**: `xml:validate_xsd(doc, xsd_str)` .

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

Das Modul `std/csv.e` garantiert strikte Konformität mit dem RFC 4180 Standard bei maximaler Verarbeitungsgeschwindigkeit:

- `csv:write_file(sequence filepath, sequence headers, sequence rows, integer delimiter = ',', integer quote_all = 0)` : Schreibt tabellarische Daten in eine CSV-Datei.
- `csv:read_file(sequence filepath, integer delimiter = ',', integer has_header = 1)` : Liest eine CSV-Datei ein und trennt Header von Datenzeilen (`{headers, rows}`).
- `csv:to_csv(sequence headers, sequence rows, integer delimiter = ',', integer quote_all = 0)` : Konvertiert Daten in einen CSV-formatierten String im Speicher.
- `csv:parse(sequence csv_text, integer delimiter = ',')` : Zerlegt einen CSV-String in Zeilen und Spalten.
- `csv:format_field(object val, integer delimiter = ',', integer quote_all = 0)` : Formatiert einen einzelnen Zellwert mit automatischer Anführungszeichen-Maskierung (`"` wird zu `""`).

Unterstützt konfigurierbare Trennzeichen (Komma `,` , Semikolon `;` , Tabulator `\t` für TSV, Pipe `|`) sowie mehrzeilige Textfelder mit eingebetteten Zeilenumbrüchen (`CRLF`).

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

Das Modul `std/excel.e` implementiert eine native OpenXML-Engine (OOXML / `.xlsx`), die Arbeitsmappen direkt über die Java 25 VM erzeugt und einliest. Es ist keine Installation von Microsoft Office oder Apache POI erforderlich. Die generierten Dateien öffnen fehlerfrei in Microsoft Excel (2007–2024 / 365), LibreOffice Calc, Google Sheets und Apple Numbers.

- `excel:write_file(sequence filepath, sequence headers, sequence rows, sequence sheet_name = "Tabelle1")` : Erzeugt eine echte formatierte Excel-Arbeitsmappe.
- Erkennt automatisch Ganzzahlen, Fließkommazahlen und Text.
- Formatiert Kopfzeilen automatisch mit blauem Füllhintergrund (`#365F91`), weißem Fettdruck und Zentrierung.
- Erstellt die vollständige OpenXML-Containerstruktur (`[Content_Types].xml` , `_rels/.rels` , `xl/workbook.xml` , `xl/styles.xml` , `xl/worksheets/sheet1.xml`).
- `excel:read_file(sequence filepath)` : Liest eine vorhandene `.xlsx` -Datei ein, löst Shared Strings auf und liefert `{status, rows}` zurück.
- `excel:export_table(filepath, headers, rows, sheet_name)` : Komfortable Export-Funktion für Datenbankabfragen.

```
include std/excel.e as excel

sequence headers = {"KundenNr", "Name", "Ort", "Umsatz"}
sequence rows = {
    {1001, "Müller Maschinenbau", "München", 84500.50},
    {1002, "Schmidt IT", "Berlin", 42300.00}
}
excel:write_file("kunden.xlsx", headers, rows, "Kundenliste")
```

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

`std/pdf_report.e` erzeugt professionelle, druckreife PDF-Geschäftsberichte im Standard-Format PDF 1.4 direkt aus Euphorix-Datenstrukturen heraus:

- `pdf_report:create_report(filepath, title, metadata, headers, rows, options = {})` :
- **Kopfbereich:** Hochwertiges farbiges Banner mit Dokumententitel und Erstellungsdatum.
- **Metadaten:** Übersichtliche Schlüssel-Wert-Karten (z. B. Mandant, Berichtszeitraum, Kostenstelle).
- **Datentabelle:** Automatische Spaltenbreiten, dunkle Kopfzeile, elegante Zebrastreifen-Schattierung für optimale Lesbarkeit.
- **Zahlenformatierung:** Numerische Werte werden automatisch rechtsbündig ausgerichtet.

- **Paginierung:** Fußzeile mit automatischer Seitennummerierung ("Seite X von Y").
- `pdf_report:export_table(filepath, title, headers, rows)` : Schneller tabellarischer PDF-Export ohne Metadatenkasten.

Das vollständige Anwendungsbeispiel zum Zusammenspiel von CSV, Excel und PDF-Reporting befindet sich in examples/09_excel_und_csv_demo.ex.

---

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

9.1 Eingebettete SQLite 3 Datenbank (`std/sqlite.e`)

Die SQLite-Engine ist autark über JDBC eingebunden und unterstützt dateibasierte Datenbanken sowie `:memory:` -Instanzen.

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

Der PostgreSQL-Treiber bietet hochskalierbaren Enterprise-Zugriff inklusive Connection Pooling und SSL.

Beide Module bieten eine **identische, einheitliche API** über JDBC:

Funktion	SQLite 3 (`std/sqlite.e`)	PostgreSQL (`std/pgsql.e`)
Verbinden	`sqlite:open("daten.db")`	`pgsql:open("postgresql://user:pass@host:5432/db")`
Schließen	`sqlite:close(db)`	`pgsql:close(db)`
DDL/DML	`sqlite:exec(db, sql, params)`	`pgsql:exec(db, sql, params)`
SELECT (Zeilen)	`sqlite:query(db, sql, params)`	`pgsql:query(db, sql, params)`
SELECT (JSON-Objekte)	`sqlite:query_assoc(db, sql, params)`	`pgsql:query_assoc(db, sql, params)`
Einzelwert	`sqlite:query_value(db, sql, params)`	`pgsql:query_value(db, sql, params)`
Transaktionen	`sqlite:begin` , `commit` , `rollback`	`pgsql:begin` , `commit` , `rollback`

10. Web-Server, Microservices, Sicherheit & Realtime (`std/http.e`, `std/auth.e`, `std/session.e`, `std/sse.e`)

Euphorix bietet einen kompletten Web- und Microservice-Stack mit asynchronem Client, integriertem HTTP-Server, kryptografischer Sicherheit, Session-Handling und Live-Event-Streaming.

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

Basiert auf dem modernen 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`.
- Automatische TLS/SSL-Zertifikatsvalidierung, Timeouts, benutzerdefinierte Header und transparente HTTP/2-Aushandlung.

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

Leichtgewichtiger, integrierter HTTP-Server basierend auf Java 25 Virtual Threads – jede eingehende Verbindung wird nebenläufig ohne Thread-Erschöpfung verarbeitet:

- `http:server_create(port)`: Erzeugt eine Server-Instanz auf dem gewünschten Port.
- `http:route(srv, method, path, handler_rid)`: Registriert Endpunkte (`GET`, `POST`, `PUT`, `DELETE` oder `*`).
- `http:response_json(status, body)`: Hilfsfunktion zum Senden formatierter JSON-Antworten.
- `http:server_background(srv)`: Führt den Server asynchron im Hintergrund aus.

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

Für geschützte Web-APIs und Benutzerkonten integriert `std/auth.e` moderne kryptografische Standards:

- **Sicheres Passwort-Hashing (PBKDF2)**:
- 65.536 Iterationen mit HMAC-SHA256 und 16 Byte kryptografischem Zufalls-Salt (`SecureRandom`).
- `auth:hash_password(password)`: Erzeugt formatierte Hashes (`pbkdf2:sha256:65536:<salt>:<hash>`).
- `auth:verify_password(password, hash_str)`: Verifiziert Passwörter in konstanter Zeit (`MessageDigest.isEqual`), immun gegen Seitenkanal-Timing-Angriffe.
- **JSON Web Tokens (JWT)**:
- RFC 7519 konforme Tokens mit Signaturalgorithmus HMAC-SHA256 (HS256).
- `auth:create_jwt(payload_json, secret_key, expiry_secs = 3600)`: Erzeugt signierten JWT-Token mit automatischem Ablauf-Claim `exp`.
- `auth:verify_jwt(token, secret_key)`: Validiert Signatur und Ablaufzeit. Liefert `{1, payload_json}` oder `{0, "Fehler"}`.
- **Kryptografische Zufallstoken**:
- `auth:random_token(byte_count = 32)`: Erzeugt hoch-entropische Hex-Token für CSRF-Schutz und API-Keys.

```
include std/auth.e as auth

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

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

Das Modul `std/session.e` verwaltet serverseitige Benutzersitzungen mit konfigurierbarer Lebensdauer (TTL):

- `session:start(cookie_header, cookie_name = "EUI_SESSION")`: Liest eine bestehende Sitzungs-ID aus dem Cookie oder erzeugt eine neue (`{sid, is_new}`).
- `session:set(sid, key, val)` & `session:get(sid, key, default_val)`: Speichert und liest beliebige Euphorix-Datenstrukturen.
- `session:destroy(sid)`: Beendet die Sitzung und gibt Ressourcen frei.
- `session:cookie_header(sid, max_age, cookie_name, path, http_only)`: Erzeugt standardisierte `Set-Cookie`-Header mit gesetztem `HttpOnly`, `SameSite=Lax` und `Path=/`.

Ein vollständiges Code-Beispiel für Auth, JWT und Sessions ist in examples/10_auth_jwt_und_session_demo.ex verfügbar.

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

Server-Sent Events ermöglichen das unidirektionale Streamen von Live-Daten, Status-Updates und Telemetrie an Browser ohne Polling:

- `sse:event(event_name, data, id = "")`: Formatiert W3C-Standard-SSE-Nachrichten (`event: ...\ndata: ...\nid: ...\n\n`).
- `sse:message(data)`: Einfache Daten-Nachricht.

- `sse:comment(text)` : Kommentar-Frame als Keep-Alive-Heartbeat gegen Timeouts.
- `sse:headers()` : Korrekte HTTP-Streaming-Header (`text/event-stream` , `no-cache` , `keep-alive`).

---

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

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

NIO-Socketkanäle für performante IPv4- und IPv6-Netzwerkverbindungen:

- `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-Namensauflösung & URLs (`std/net/dns.e`, `std/net/url.e`)

Auflösung von Hostnamen in IP-Adressen und Parsing von URLs nach RFC 3986.

---

12. Demos, Beispiele & Ausführung

12.1 Praxisbeispiele & Demos im Überblick

Alle Beispielprogramme befinden sich in `demo/` und `examples/` und können direkt ausgeführt werden.

12.2 Ausführen der Demonstrationsprogramme

```
# 1. Interaktive Desktop-GUI starten (Swing / FlatLaf)
./euphorix demo/gui_demo.ex

# 2. REST-API Microservice mit SQLite & Schema-Validierung
./euphorix examples/05_rest_api_microservice.ex

# 3. Datenbank-Integrationstests (SQLite & PostgreSQL)
./euphorix examples/04_sqlite_datenbank.ex
./euphorix examples/06_postgresql_datenbank.ex

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

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

# 6. Autarke 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-Sicherheit: Auth, PBKDF2, JWT & HTTP-Sessions
./euphorix examples/10_auth_jwt_und_session_demo.ex

# 9. VSUI: Formularvalidierung, 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 für Linux-Dienste
./euphorix tools/make_service.ex examples/05_rest_api_microservice.ex euphorix-api "REST API" volkix

# 12. Alle Testsuiten ausführen (Java 25 Runner)
./tools/run_all_tests ./euphorix tests/t_gui.e tests/t_java_interop.e tests/t_concurrency.ex
```

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

Durch die Umstellung auf Java 25 Virtual Threads (Project Loom) überwindet OpenEuphoria die historische Beschränkung auf Single-Core-Multitasking. Blockierende Netzwerk-, Datenbank- oder Datei-Operationen blockieren nie wieder das Gesamtsystem. Drei eng integrierte Wege decken alle Anwendungsfälle ab:

13.1 Weg 1: Klassische Tasks mit Virtual Threads (`task:create_thread`)

Das Modul `std/task.e` behält 100 % Abwärtskompatibilität zum bisherigen kooperativen Scheduler, bietet aber mit `create_thread` den direkten Sprung auf 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) -- Blockiert NICHT das Hauptprogramm!
        printf(1, "Task %s: Schritt %d\n", {name, i})
    end for
end procedure

-- Startet als preemptiver Virtual Thread auf separaten CPU-Kernen:
atom t1 = task:create_thread(routine_id("worker"), {"A"})
atom t2 = task:create_thread(routine_id("worker"), {"B"})
```

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

Thread-sichere Röhren (CSP - Communicating Sequential Processes) ermöglichen synchronisationsfreie Datenübertragung ohne Locks oder Race Conditions:

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

atom ch = channel:new(10) -- Pufferkapazität 10

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

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

-- Verbraucher liest bis Channel geschlossen ist:
while 1 do
    sequence msg = channel:receive(ch)
    if msg[1] != channel:OK then exit end if
    printf(1, "Empfangen: %s\n", {msg[2]})
end while
```

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

Das ergonomischste Muster für Entwickler: Asynchrone Jobs starten und Ergebnisse synchron abholen, wenn sie benötigt werden:

```
include std/thread.e

function compute_pi_chunk(atom terms)
    atom sum = 0
    -- Aufwändige mathematische Berechnung...
    return sum
end function

-- Parallel im Hintergrund starten:
atom job1 = thread:async(routine_id("compute_pi_chunk"), {1000000})
atom job2 = thread:async(routine_id("compute_pi_chunk"), {2000000})

-- Hauptprogramm macht derweil andere Dinge (z.B. GUI-Events verarbeiten)
if not thread:is_done(job1) then
    puts(1, "Job 1 rechnet noch...\n")
end if

-- Auf Ergebnisse warten:
atom res1 = thread:await(job1)
atom res2 = thread:await(job2)

-- Oder mehrere Jobs auf einmal abholen:
-- sequence alle_ergebnisse = thread:await_all({job1, job2})
```

13.4 Weg 4: Hintergrundaufgaben & Task-Scheduler (`std/cron.e`)

Mit `std/cron.e` können zeitgesteuerte Hintergrundaufgaben direkt im Euphorix-Prozess ausgeführt werden, ohne externe Cronjobs oder Hilfsskripte zu benötigen. Aufgaben laufen in separaten Java 25 Virtual Threads und blockieren weder die Programmausführung noch laufende Web-Anfragen:

- `cron:schedule_interval(sequence name, atom interval_seconds, integer routine_id, sequence args = {})` : Führt eine Euphorix-Prozedur alle N Sekunden periodisch aus.
- `cron:schedule_daily(sequence name, integer hour, integer minute, integer routine_id, sequence args = {})` : Führt eine Aufgabe täglich zu einer festen Uhrzeit aus (z. B. täglicher Datenabgleich um 02:30 Uhr).
- `cron:start()` & `cron:stop()` : Startet bzw. stoppt den Scheduler-Hintergrunddienst geordnet.

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

procedure aufraeumen()
    puts(1, "Temporäre Dateien gelöscht.\n")
end procedure

-- Alle 60 Sekunden ausführen
integer job_id = cron:schedule_interval("cleanup", 60, routine_id("aufraeumen"))
cron:start()
```

Ein vollständiges Anwendungsbeispiel befindet sich in examples/12_cron_scheduler_demo.ex.

---

14. Web-Anwendungen mit SAP UI5 Web Components 2.29 (`std/ui5.e`)

Das Modul `std/ui5.e` bringt die standardisierten **SAP UI5 Web Components (targeting v2.29)** direkt in das Euphorix-Ökosystem:

14.1 Architektur & Fiori Horizon Design

- **Zero Node.js / Zero Build-Step**: Entwickler schreiben reine OpenEuphoria-Syntax. Keine `package.json`, kein Webpack/Vite oder Angular/React nötig.
- **SAP Fiori Themes**: Vollständige Unterstützung für `THEME_HORIZON` (*Morning Horizon*), `THEME_HORIZON_DARK` (*Evening Horizon*) und `THEME_FIORI_3`.
- **Standard HTML5 Web Components**: Standard-Elemente wie `<ui5-shellbar>`, `<ui5-card>`, `<ui5-table>`, `<ui5-button>`, `<ui5-input>`, `<ui5-dialog>`, `<ui5-toast>`, `<ui5-badge>`, `<ui5-datepicker>`, `<ui5-switch>` usw.
- **Offline-Fallback**: Eingebettete CSS-Stile stellen sicher, dass das Fiori-Layout auch ohne aktive Internetverbindung mit sauberer Typografie und Farben dargestellt wird.

14.2 Komponenten-Übersicht & Deklaration

```
include std/ui5.e as ui5

-- ShellBar (Kopfzeile mit Avatar, Suche, Benachrichtigungen)
sequence sb = ui5:shellbar("Meine Firma", "Enterprise ERP", {{"profile", "VS"}})

-- Fiori Card
sequence crd = ui5:card("Offene Tickets", "<p>3 Dringende Aufgaben</p>", {
    {"subtitle", "Status: Aktiv"},
    {"icon", "order-status"}
})

-- Interaktive Tabelle
sequence tbl = ui5:table(
    {{"ID", "80px"}, {"Titel", ""}, {"Status", "120px"}},
    {
        {"#1", "Server-Wartung", ui5:badge("Erledigt", 8)},
        {"#2", "DB Backup", ui5:badge("Dringend", 2)}
    }
)

-- Eingabeelemente
sequence inp = ui5:input("task_title", "", {{"placeholder", "Aufgabe..."}, {"icon", "edit"}})
sequence btn = ui5:button("btn_add", "Hinzufügen", {{"design", ui5:DESIGN_EMPHASIZED}, {"icon", "add"}, {"action", "add_item"}}
```

14.3 Reaktive Euphorix-Bridge & Live DOM-Updates

Klicks und Formulareingaben werden vom integrierten Bridge-Skript automatisch an serverseitige `routine_id`-Handler übergeben. Die Antwort aktualisiert gezielt den DOM ohne Seiten-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, "Titel erforderlich!"),
            ui5:toast_show("app_toast", "Fehler!")
        })
    end if

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

    -- Reaktive Updates zurück an den Browser
    return ui5:updates({
        ui5:patch("table_container", render_table()),
        ui5:set_val("task_title", ""),
        ui5:toast_show("app_toast", "Gespeichert!")
    })
end function
```

14.4 Vollständiges Anwendungsbeispiel (`examples/07_sapui5_webcomponents_demo.ex`)

Das vollständige, lauffähige Anwendungsbeispiel eines Fiori Horizon Enterprise Cockpits mit SQLite-Anbindung befindet sich in examples/07_sapui5_webcomponents_demo.ex. Es kann direkt über `./euphorix examples/07_sapui5_webcomponents_demo.ex` ausgeführt werden.

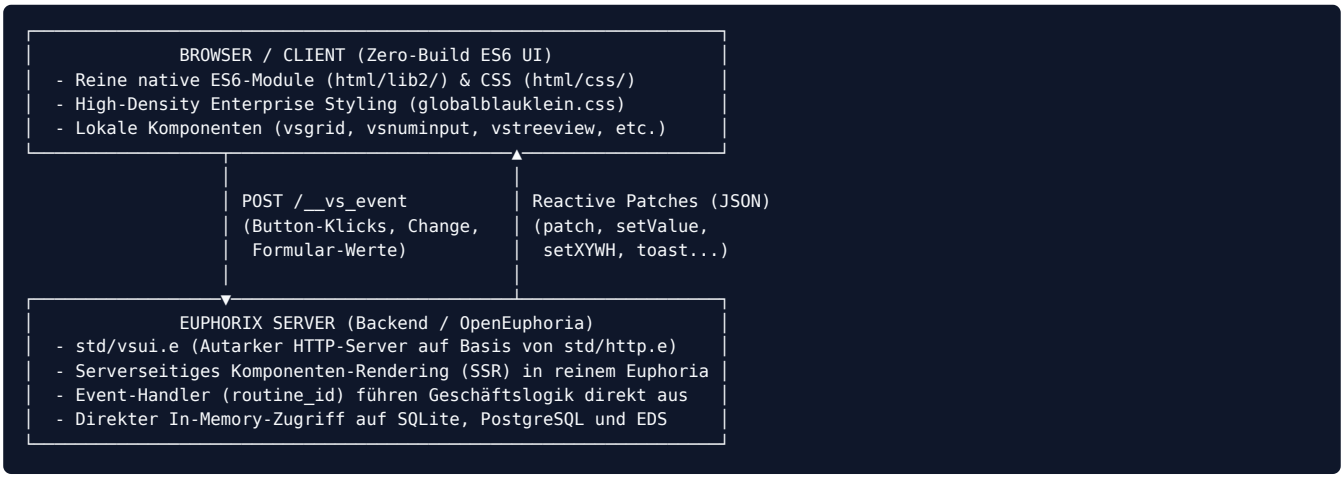
---

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

Als leichtgewichtige, autarke Alternative zu monolithischen Frontend-Frameworks integriert Euphorix mit `std/vsui.e` ein vollwertiges Enterprise-Webframework. Es verbindet zwei der produktivsten Architekturmodelle der Web-Entwicklung zu einer homogenen Einheit: die **hypermedia-getriebene Einfachheit von HTMX** und das **serverseitige, typisierte Komponentenmodell von Vaadin**.

15.1 Architektur & Philosophie: Die Symbiose aus HTMX und Vaadin

Moderne Enterprise-Webanwendungen leiden häufig unter immenser Build-Komplexität (Node.js, Webpack/Vite, npm-Abhängigkeiten) und redundanten REST/DTO-Abstraktionsschichten. `std/vsui.e` eliminiert diesen Overhead vollständig, indem es die Stärken zweier bewährter Ansätze vereint:



Was es mit HTMX teilt:

- **Hypermedia / Server-Driven UI**: Der Server liefert fertiges HTML und behält die Hoheit über Geschäftslogik, Validierung und Statusübergänge.
- **Kein SPA-Build-Overhead**: Völliger Verzicht auf Node.js, npm, Webpack, Vite, React oder Vue sowie riesige `node_modules`-Verzeichnisse. Der Browser lädt sauberes HTML5 und native ES6-Module.
- **Flackerfreie Partial DOM-Updates**: Nutzeraktionen lösen keine Full-Page-Reloads (`F5`) aus. Stattdessen sendet der Server zielgerichtete JSON-Patches (`vsui:updates`), die gezielt betroffene DOM-Knoten im Browser austauschen oder manipulieren (`vsui:patch` , `vsui:set_val`).

Was es mit Vaadin teilt:

- **Backend-Zentriertes Komponentenmodell**: Die gesamte Benutzeroberfläche wird deklarativ in der Hauptprogrammiersprache (reines OpenEuphoria statt Java) deklariert: `vsui:button()` , `vsui:grid()` , `vsui:tabs()` , `vsui:splitter()` , `vsui:numinput()` .
- **Kein REST-API / DTO Glue-Code**: Entwickler müssen keine manuellen REST-Endpunkte, DTO-Serialisierer, Validierungs-Pipelines und Frontend-Fetch-Aufrufe schreiben, nur um einen Button mit einer Datenbankaktion zu verknüpfen. Events rufen direkt Backend-Handler (`routine_id`) auf.
- **Direkte Datenbank- & Thread-Nähe**: Event-Handler laufen direkt auf dem Server unmittelbar neben transaktionalen Datenbanken (SQLite, PostgreSQL, EDS) und Java 25 Loom Virtual Threads. Datenänderungen werden atomar ausgeführt und die UI sofort aktualisiert.

Der Euphorix-Enterprise-Vorteil:

1. **Kompakte High-Density Arbeitsplatz-Ergonomie**: Im Gegensatz zu Consumer-Frameworks mit weitläufigen Weißräumen bieten globalblauklein.css und vs.css kompakte, monospace-optimierte Layouts für datenintensive ERP-, CRM-, Lager- und Dispositionssysteme.
2. **Kaufmännische Präzision**: Lokalisierte Zahlenformatierung (vsnumber.js & `vsnuminput.js`) mit konfigurierbaren Nachkommastellen (`nk`), deutscher DIN 5008 Formatierung (`50.000,00 €`), rechtsbündigen Monospace-Ziffern und automatischer Blur-Formatierung.
3. **Pixelgenaue Positionierung**: Desktop-artige Koordinatenplatzierung (`setXYWH` , `opt_xywh(x, y, w, h)`) und automatische Fenster- bzw. Parent-Zentrierung (`centerWinX` , `centerWinY` via ldom.js).
4. **100% Autark & Offline-fähig**: Sämtliche Assets liegen lokal im Verzeichnis `html/` . Keine externen CDNs, keine Telemetrie, kein Internetzugriff erforderlich.

15.2 Vollständige 22-Komponenten-Suite (`html/lib2/components/`)

`std/vsui.e` bietet Euphorix-Konstruktoren für ausnahmslos alle Komponenten der `lib2` -Bibliothek:

Komponente	Euphorix-Funktion	Beschreibung / Feature
Button	`vsui:button(id, text, opts)`	Nativer `.vs-button` mit Klick-Event-Weiterleitung & Aktions-Dispatch
Input	`vsui:input(id, val, opts)`	Standard-Texteingabefeld `.vs-input`
Zahlenfeld	`vsui:numinput(id, val, nk, opts)`	Kaufmännisches Zahlenfeld (`vs-number`) mit `nk` Nachkommastellen und `vsnumber.js` -Formatierung
Datumsfeld	`vsui:dateinput(id, val, opts)`	HTML5 Datumsselektor `.vs-date` mit zentrierter Ausrichtung
Zeitfeld	`vsui:timeinput(id, val, opts)`	HTML5 Zeitauswahl `.vs-time`
Checkbox	`vsui:check(id, label, checked, opts)`	Checkbox mit Textbeschriftung `.vs-check`
Textarea	`vsui:textarea(id, val, rows, opts)`	Mehrzeiliges Texteingabefeld `.vs-input`
Dropdown	`vsui:combo(id, items, selected, opts)`	Dropdown-Auswahlbox `.vs-combo`
Label	`vsui:ui_label(text, for_id, opts)`	Feldbeschriftung `.vs-label`
Fieldset	`vsui:group(legend, content, opts)`	Gruppierungsrahmen `.vs-group` mit Titellegende
Panel	`vsui:panel(id, title, content, opts)`	Karte/Container `.vs-panel` mit blauem Header
Collapse	`vsui:collapse(id, sections, open_first, opts)`	Klappbare Abschnitte (`<details>` / `<summary>`)
DataGrid	`vsui:grid(id, cols, rows, opts)`	Tabellenkomponente mit **Sticky-Header**, **Client-Live-Suche** und automatischem **Mobile-Card-View**
Tabs	`vsui:tabs(id, tabs_list, opts)`	Reiter-Navigation mit dynamischer Umschaltung und horizontalem Mobile-Swipe
Modal	`vsui:modal(id, title, content, footer, opts)`	Zentrierter Popup-Dialog mit Abdunklungs-Overlay (mobil Vollbild)
Splitter	`vsui:splitter(id, left, right, pct, opts)`	2-Spalten-Container mit anpassbarer Breitenaufteilung
Blocker	`vsui:blocker(id, zindex, opts)`	Vollflächiger UI-Overlay-Sperrbildschirm (`div.blocker`) mit dynamischem Z-Index
Treeview	`vsui:treeview(id, nodes, opts)`	Hierarchische Baumstruktur (`ul.vs-treeview`) mit aufklappbaren Knoten (▼)
Kontextmenü	`vsui:with_contextmenu(html, items)`	Rechtsklick-Kontextmenü mit Aktions-Triggern und Trennlinien
Tooltip	`vsui:with_tooltip(html, text)`	Schwebender Hinweistext mittig über dem Zielelement
Datei-Upload	`vsui:upload(id, target_url, opts)`	Drag & Drop Dropzone mit Fortschrittsanzeige und Dateidialog
SVG-Charts	`vsui:chart(id, type, lbls, data, opts)`	Server-gerenderte responsive Vektordiagramme (`bar` , `line` , `pie` , `doughnut`)

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

Der integrierte Server in `std/vsui.e` bedient Anfragen für statische Dateien automatisch direkt aus dem lokalen `html/` -Verzeichnis des Projekts:

- `/css/globalblauklein.css` , `/css/vs.css` und `/css/vs-mobile.css`
- `/lib2/ldom.js` , `/lib2/vsevent.js` , `/lib2/ui.js` , `/lib2/vsgrid.js` , `/lib2/vsnumber.js`
- `/lib2/components/*` (sämtliche 22 Modulkomponenten)
- `/img/` *(Logos, Icons) und* `/app/`

Die MIME-Types werden anhand der Dateiendung automatisch ermittelt (`application/javascript` , `text/css` , `image/png` , `font/woff2` , etc.).

15.4 Reaktive Event-Bridge mit `vsevent` & DOM-Patches

Das Frontend sammelt bei Nutzerinteraktionen (`button_click` , `changevalue`) alle Formularwerte ein und sendet sie per POST an `/__vs_event` . Dabei nutzt die Bridge intelligent die komponentenspezifische Methode `.getValue()` , wodurch formatierte Zahlen oder standardisierte Datumsangaben fehlerfrei typisiert übertragen werden.

Der Euphorix-Server antwortet mit einer Liste reaktiver Patches (`vsui:updates({...})`):

Patch-Funktion	Parameter	Wirkung im Browser
`vsui:patch(id, html)`	`id, html`	Ersetzt den `innerHTML` des Zielelements
`vsui:append_html(id, html)`	`id, html`	Hängt HTML am Ende des Elements an (`beforeend`)
`vsui:set_val(id, val)`	`id, val`	Setzt den Wert und ruft `.setValue(val)` auf
`vsui:grid_set_data(id, data)`	`id, data`	**Neu:** Aktualisiert DataGrid-Zeilen direkt per JSON ohne HTML-Neuaufbau
`vsui:set_focus(id)`	`id`	**Neu:** Setzt den Tastaturfokus direkt in das Zielfeld
`vsui:set_enabled(id, bool)`	`id, 0/1`	**Neu:** Aktiviert oder sperrt Eingabefelder und Buttons
`vsui:set_visible(id, bool)`	`id, 0/1`	**Neu:** Blendet Elemente ein oder aus (`display: none` / `''`)
`vsui:set_css(id, prop, val)`	`id, prop, val`	**Neu:** Dynamisches Inline-Styling (z. B. rote Rahmen bei Validierung)
`vsui:set_class(id, cls, act)`	`id, cls, act`	**Neu:** Fügt CSS-Klassen hinzu, entfernt sie oder schaltet sie um
`vsui:clear_form(container_id)`	`container_id`	**Neu:** Leert alle Formularfelder innerhalb des Containers
`vsui:toast_show(txt, type, ms)`	`txt, type, ms`	Blendet eine schwebende Erfolgs- oder Fehlermeldung ein
`vsui:modal_open(id) / close`	`id`	Öffnet oder schließt ein modales Dialogfenster
`vsui:redirect(url)`	`url`	Leitet den Browser auf eine neue URL weiter

```
function on_save_kunde(sequence ctx)
    sequence name = vsui:val(ctx, "txt_name")
    sequence kdnr = vsui:val(ctx, "txt_kdnr")
    atom     umsatz = vsui:val_num(ctx, "num_umsatz")

    -- In SQLite speichern...
    sqlite:exec(db, "INSERT INTO kunden (kdnr, name, umsatz) VALUES (?, ?, ?);", {kdnr, name, umsatz})

    -- Frische Kundenzeilen holen
    sequence frische_kunden = sqlite:query_all(db, "SELECT kdnr, name, umsatz FROM kunden ORDER BY kdnr DESC;")

    return vsui:updates({
        vsui:grid_set_data("grid_kunden", frische_kunden),
        vsui:clear_form("form_kunde"),
        vsui:set_focus("txt_kdnr"),
        vsui:toast_show("Kunde erfolgreich angelegt!", vsui:TOAST_SUCCESS)
    })
end function
```

15.5 Pixelgenaue Positionierung & Fensterzentrierung (`ldom.setXYWH` , `opt_xywh`)

Im Unterschied zu herkömmlichen Web-Frameworks, die Entwickler bei Desktop-ähnlichen Dialogen zu komplexen CSS-Regeln zwingen, integriert `std/vsui.e` eine direkte pixelgenaue Koordinatensteuerung über das ldom.js-Koordinatensystem:

- **In Euphoria-Komponenten-Deklarationen (`opt_xywh`)**:

Jeder Komponenten-Konstruktor akzeptiert `vsui:opt_xywh(x, y, w, h)` :

```
  -- Button bei X=100, Y=250 mit Breite 180px und Höhe 36px positionieren:
  sequence btn = vsui:button("btn_save", "Speichern", {vsui:opt_xywh(100, 250, 180, 36)})

  -- DataGrid in einem festen Workspace-Bereich platzieren:
  sequence grid = vsui:grid("grd_kunden", cols, rows, {vsui:opt_xywh(20, 80, 900, 500)})
```

- **Dynamische serverseitige Patches**:
- `vsui:set_xywh(id, x, y, w, h)` : Positioniert und skaliert beliebige DOM-Elemente zur Laufzeit via `ldom: setXYWH` .
- `vsui:center_win(id, axis = "both")` : Zentriert ein Element horizontal und/oder vertikal im Browserfenster (`centerWinX` , `centerWinY`).
- `vsui:center_parent(id, axis = "both")` : Zentriert ein Element relativ zu seinem übergeordneten Parent-Container (`centerX` , `centerY`).

15.6 Kaufmännische Präzision & Deutsche Zahlenformatierung (`vsnumber.js`)

Das Standard-HTML5 `<input type="number">` erzwingt den internationalen Punkt als Dezimaltrennzeichen und unterstützt keine Tausendertrennpunkte – für deutsche ERP- und Finanzsoftware unbrauchbar. `std/vsui.e` löst dieses Problem über die integrierte vsnumber.js-Engine:

- **Konfigurierbare Nachkommastellen (`nk`)**: `vsui:numinput(id, value, nk, options)` setzt die gewünschte Genauigkeit (`nk = 2` für Euro-Beträge, `0` für Ganzzahlen, `4` für Umrechnungskurse).
- **Automatische Blur-Formatierung**: Benutzer können `1234,5` oder `1234.5` eintippen; beim Verlassen des Feldes (`blur`) formatiert der Browser den Wert automatisch zu `1.234,50` via `vsnumber.fz(val, nk)` .

- **Kaufmännisches High-Density Styling**: Formatierte Felder erhalten die CSS-Klasse `.vs-number` aus `globalblauklein.css`, wodurch Zahlen rechtsbündig in einer lesefreundlichen Monospace-Schrift dargestellt werden.
- **Bidirektionale Konvertierung in Euphoria**:
- `vsui:format_num_de(val, nk)` : Formatiert numerische Atoms in lokalisierte Strings (z. B. `50000.5` -> `"50.000,50"`).
- `vsui:parse_num_de(str)` : Parst sowohl deutsche (`"1.234,56"`) als auch internationale (`"1234.56"`) Formatstrings robust in rohe Euphoria-Atoms.
- `vsui:val_num(ctx, id)` : Komfortfunktion, die übertragene Formularwerte direkt als Zahl ausliest und typisiert zurückgibt.

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

Über das automatisch eingebundene Stylesheet `/css/vs-mobile.css` passt sich die gesamte `vsui`-Suite nahtlos an Smartphones und Tablets (Breakpoints `<= 768px`) an, ohne die Desktop-Bedienung zu verändern:

- **Eingabe-Zoom-Stopp (iOS Safari)**: Alle Formularfelder (`.vs-input` , `.vs-combo` , `.vs-spreadsheet-search`) erhalten auf Mobilgeräten `font-size: 16px;` , wodurch das störende automatische Hineinzoomen auf iPhones unterbunden wird.
- **Touch-Ziele nach Fitts's Law**: Buttons und Eingabefelder bieten eine Mindesthöhe von 42px mit vergrößerten Klickflächen für Daumenbedienung.
- **DataGrid Card-View**: Auf Bildschirmen `<= 768px` transformiert CSS das tabellarische `vsgrid` automatisch in eine strukturierte Kartenansicht (Card-View). Die Spaltentitel werden über das Attribut `data-label` vor jedem Feld angezeigt.
- **Horizontales Wischen für Tabs**: `.vs-tabs` unterstützen flüssiges horizontales Swipen mit Touch-Momentum, ohne die Kopfleiste vertikal umzubrechen.
- **Splitter & Modale**: `.vs-splitter-container` stapelt Seitenbereiche vertikal untereinander, und Modaldialoge passen sich als Vollbild-Dialoge mit flüssigem Scrollbereich an.

15.8 Formularvalidierung & Fehlertooltips (`vsui:field_error`)

Für robuste Formulareingaben bietet `std/vsui.e` eine direkte reaktive Fehler-Feedback-Bridge zwischen Server und Browser. Wenn eine serverseitige Validierung fehlschlägt, kann das entsprechende Eingabefeld rot markiert und mit einem erklärenden Tooltip versehen werden:

- `vsui:field_error(id, error_msg)` : Erzeugt einen reaktiven DOM-Patch, der dem Eingabeelement die Klasse `.vs-input-error` zuweist und direkt darunter einen Fehlerhinweis `<div class="vs-field-error-tip">` einblendet.
- `vsui:field_clear_error(id)` : Entfernt die Fehlermarkierung und den Tooltip wieder.
- `vsui:validate_required(ctx, field_id, field_label)` : Hilfsfunktion zur schnellen Pflichtfeldprüfung. Gibt `{1, ""}` zurück, wenn das Feld ausgefüllt ist, oder `{0, "Das Feld '...' darf nicht leer sein."}` .

```
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", "Bitte geben Sie eine gültige E-Mail-Adresse an."),
            vsui:set_focus("txt_email")
        })
    end if

    return vsui:updates({
        vsui:field_clear_error("txt_email"),
        vsui:toast_show("Eingaben erfolgreich validiert!", vsui:TOAST_SUCCESS)
    })
end function
```

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

Mit `vsui:upload` steht eine einsatzbereite Datei-Upload-Komponente zur Verfügung, die Drag-and-Drop, Dateidialog-Auswahl und Fortschrittsanzeige nativ unterstützt:

- `vsui:upload(id, target_url, options)` : Erzeugt das Dropzone-Markup `.vs-upload-dropzone` mit Upload-Icon, Hinweistext und Ladebalken.
- **Optionen:**
- `"prompt"` : Benutzerdefinierter Beschriftungstext (z. B. `"Excel- oder CSV-Datei hier ablegen"`).
- `"accept"` : Erlaubte MIME-Typen oder Dateiendungen (z. B. `".xlsx,.csv,.pdf"`).

```
sequence upload_box = vsui:upload("my_dropzone", "/api/upload", {
    {"prompt", "Geschäftsbericht (XLSX, CSV) hier ablegen oder durchsuchen"},
    {"accept", ".xlsx,.csv"}
})
```

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

Euphorix enthält eine vollständig autarke SVG-Diagramm-Engine. Charts werden direkt auf dem Server als leichtgewichtiges, responsives SVG (`viewBox="0 0 600 300"`) gerendert – ganz ohne externe JavaScript-Bibliotheken (kein Chart.js, D3 oder Canvas):

- `vsui:chart(id, chart_type, labels, datasets, options)` : Erzeugt ein SVG-Diagramm.

- `chart_type` : `"bar"` (Balkendiagramm), `"line"` (Liniendiagramm), `"pie"` (Tortendiagramm) oder `"doughnut"` (Donut-Diagramm).
- `labels` : Sequenz von Beschriftungs-Strings, z. B. `{"Jan", "Feb", "Mär", "Apr"}` .
- `datasets` : Sequenz von Datenserien im Format `{ {name, {val1, val2, ...}}, ... }` .
- `options` : Zusätzliche Optionen wie `{"title", "Monatsumsatz 2026"}` .
- `vsui:chart_set_data(id, chart_type, labels, datasets)` : Erzeugt einen Live-Patch, um ein bestehendes Diagramm per SSE oder Event-Bridge ohne Seiten-Reload in Echtzeit zu aktualisieren.

15.11 Vollständige Anwendungsbeispiele & Demos (`demo/vsui_demo.ex` , `examples/08_vsui_cockpit_demo.ex` , `examples/11_vsui_charts_und_upload_demo.ex`)

Euphorix bietet drei abgestimmte Demonstrations- und Testprogramme, die 100% aller Komponenten und Funktionen aus Kapitel 15 abdecken:

1. **Interaktives All-in-One Flaggschiff-Cockpit**: demo/vsui_demo.ex

- **100% Komponentenabdeckung**: Führt sämtliche Komponenten aus Kapitel 15 in einer interaktiven Web-Anwendung auf Port 8089 zusammen (DataGrid mit Sticky-Header & Suche, Splitter-Details, Tab-Navigation, Formularfelder für Text, Datum, Uhrzeit, Zahlen und Notizen, serverseitige Pflichtfeld- und E-Mail-Validierung mit Fehlertooltips, interaktive responsive SVG-Diagramme mit Live-Update per Button sowie Drag & Drop Datei-Upload).
- **Starten & im Browser öffnen**:

```
./euphorix demo/vsui_demo.ex
# Im Browser öffnen: http://127.0.0.1:8089/
```

1. **Automatisierter End-to-End Test & Headless-Prüfung**: examples/08_vsui_cockpit_demo.ex

- Führt einen vollautomatischen Integrations- und Funktionstest durch: Startet den Hintergrundserver, lädt statische CSS- und JS-Assets per HTTP GET herunter, validiert die HTML- und DataGrid-Generierung, sendet synthetische Klick-Events an `/__vs_event` , prüft reaktive DOM-Patches und verifiziert Datenbankoperationen in SQLite:

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

1. **Fokussiertes Dashboard für SVG-Charts & Upload**: examples/11_vsui_charts_und_upload_demo.ex

- Kompaktes Codebeispiel für den gezielten Einsatz von server-gerenderten SVG-Balken- und Doughnut-Diagrammen, Drag & Drop Upload und E-Mail-Validierung:

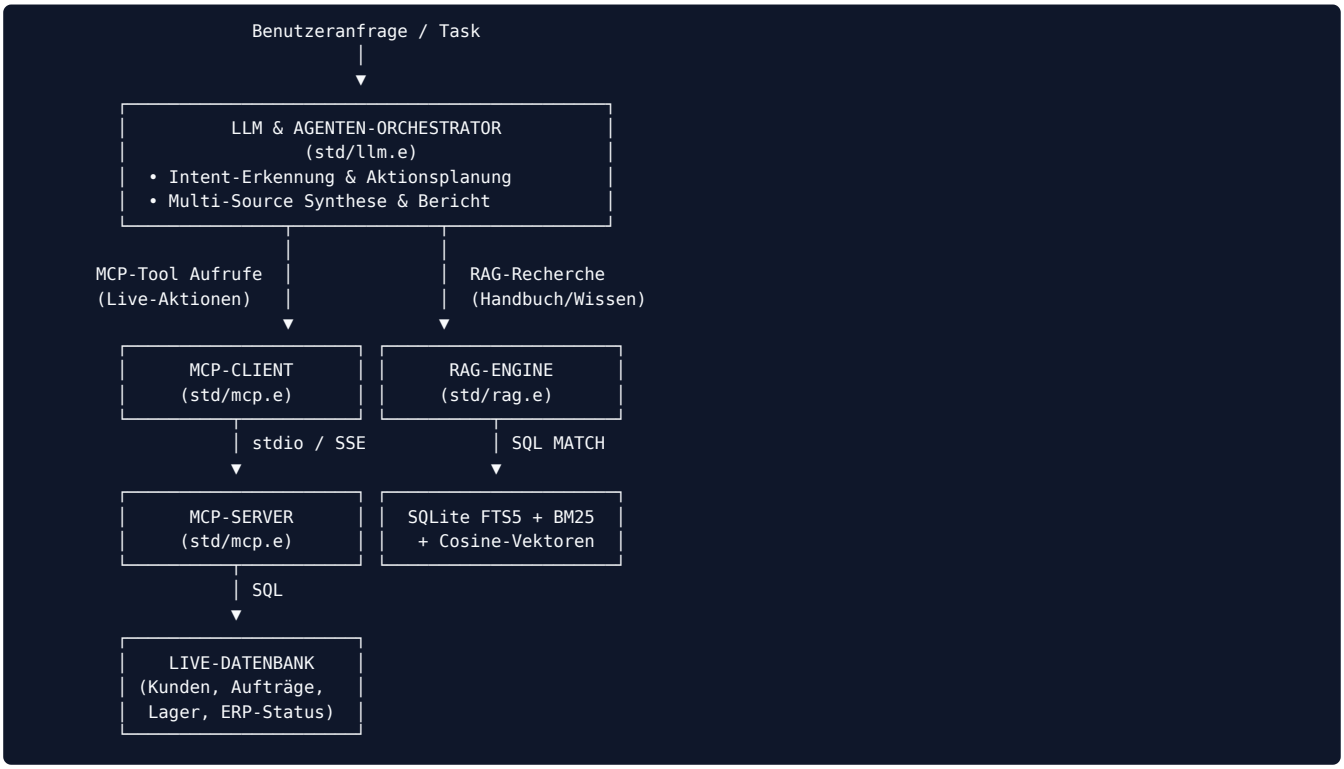
```
./euphorix examples/11_vsui_charts_und_upload_demo.ex --run
# Im Browser öffnen: http://127.0.0.1:8092/
```

16. Künstliche Intelligenz: Autonome Agenten, Model Context Protocol (MCP), RAG & LLMs (`std/mcp.e`, `std/rag.e`, `std/llm.e`)

Euphorix 1.0 integriert native Enterprise-Schnittstellen für moderne künstliche Intelligenz und autonome Agenten. Über drei aufeinander abgestimmte Module lässt sich jede Euphorix-Anwendung wahlweise als **MCP-Server** für KI-Assistenten (wie Claude Desktop, Antigravity oder Cursor) bereitstellen, als **autonomer Agent mit eigenem MCP-Client** betreiben oder mit **RAG (Retrieval-Augmented Generation)** an unternehmenseigene Wissensbestände anbinden.

16.1 Architektur: Das Zusammenspiel von LLM, MCP und RAG

Ein moderner KI-Agent stützt sich auf drei zentrale Säulen:



1. **LLM (`std/llm.e`) – Das Gehirn:**

Interpretiert Benutzeranfragen, entscheidet über den Aufruf externer Werkzeuge und führt am Ende strukturierte Daten und Regeltexte zu einer fundierten Antwort zusammen.

1. **MCP-Server & Client (`std/mcp.e`) – Die Hände:**

Führt kontrollierte Lese- und Schreibaktionen in Live-Systemen aus (z. B. Kunde suchen, Lagerbestand prüfen, Auftrag in SQLite anlegen).

1. **RAG-Engine (`std/rag.e`) – Das Langzeitgedächtnis:**

Durchsucht unstrukturierte Dokumente (Richtlinien, Verträge, Handbücher) per Volltext- (BM25) und Vektorsuche und liefert belegbare Textabschnitte mit Quellenangaben.

16.2 Wann LLM + MCP allein genügen (und wann RAG benötigt wird)

In der Praxis besteht oft der Irrglaube, dass jedes KI-System zwingend eine RAG-Pipeline benötigt. **Für klassische Datenbank-, ERP- und CRM-Anwendungen reichen LLM + MCP vollkommen aus!**

Kriterium	LLM + MCP (Direkter Werkzeug-Zugriff)	RAG (Dokumenten-Vektorsuche)
Primärer Datenbestand	Strukturierte relationale Daten (Tabellen, Spalten, IDs, Status, Fremdschlüssel).	Unstrukturierte Fließtexte (PDFs, Handbücher, AGBs, E-Mails, Support-Tickets).
Präzision & Berechnungen	**100% mathematisch exakt**: SQL-Queries berechnen Summen, Durchschnitte und Zählungen direkt auf der Datenbank.	**Probabilistisch**: Das LLM muss Zahlen aus Textabschnitten extrahieren und schätzen (Halluzinationsgefahr).
Aktualität	**Echtzeit (Zero-Latency)**: Jedes Tool liest den sekundengenauen Live-Stand der Datenbank.	**Verzögert**: Dokumente müssen vorab gechunkt, vektorisiert und in Indexen gespeichert werden.
Aktionen & Write-Operationen	**Schreibend & Ausführend**: Kann Datensätze anlegen (`INSERT`), ändern (`UPDATE`) oder E-Mails versenden.	**Rein lesend**: Dient ausschließlich der passiven Informationsbeschaffung.
Empfohlener Einsatz	**90% aller betrieblichen ERP/CRM-Workflows**: Auftragsstatus, Kundenhistorie, Bestandsabfragen.	**Ergänzung für unstrukturierte Texte**: Richtlinien nachschlagen, Fehlerdiagnose nach Handbuch.

> [!TIP]

> **RAG als MCP-Werkzeug:** Sie müssen sich nicht zwischen beiden Ansätzen entscheiden. Wenn Sie Dokumente durchsuchen wollen, implementieren Sie RAG einfach als ein weiteres MCP-Tool (`tool: handbuch_suchen(thema)`). Für das LLM ist RAG dann lediglich ein spezialisiertes Werkzeug unter vielen!

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

Das Model Context Protocol (MCP, offizielle Spezifikation 2024-11-05) ist der moderne Standard, um KI-Modelle sicher mit lokalen Datenquellen und Programmen zu verbinden.

16.3.1 Einen eigenen MCP-Server für Datenbanken & Werkzeuge erstellen

Ein Euphorix-MCP-Server deklariert Werkzeuge mit JSON-Schema-Parametern und registriert Handler-Routinen:

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

-- 1. Server-Instanz anlegen
sequence srv = mcp:server_create("Euphorix-CRM-Server", "1.0.0")

-- 2. Werkzeug-Handler definieren
function handle_kunden_suche(sequence args)
    sequence term = ""
    for i = 1 to length(args) do
        if equal(args[i][1], "suchbegriff") then term = args[i][2] end if
    end for

    -- SQLite-Query ausführen
    sequence sql = sprintf("SELECT kdnr, name, ort, status FROM kunden WHERE name LIKE '%%%s%%'", {term})
    sequence res = sqlite:query(db, sql)
    -- Formatierte Antwort an den Agenten zurückgeben
    return {1, "Gefundene Kunden: " & sprint(res), {}}
end function

-- 3. Werkzeug mit Parametern registrieren
srv = mcp:server_register_tool(srv,
    "kunden_suchen",
    "Durchsucht den Kundenstamm im SQLite-ERP nach Namen oder Ort",
    {
        {"suchbegriff", "string", "Firmenname oder Suchbegriff", 1}
    },
    routine_id("handle_kunden_suche")
)

-- 4. Server im gewünschten Modus starten
-- Für Claude Desktop / Subprozesse (stdio):
mcp:server_run_stdio(srv)

-- Für Web- und Netzwerk-Zugriff via HTTP/SSE:
-- mcp:server_run_http(srv, 8085)
```

16.3.2 Der MCP-Client: Werkzeuge dynamisch auflisten und ausführen

Ein Euphorix-Programm kann als autonomer MCP-Client beliebige MCP-Server steuern:

```
include std/mcp.e as mcp

-- 1. MCP-Server per stdio als Subprozess starten
sequence res = mcp:client_stdio("./euphorix examples/12_mcp_server_demo.ex --stdio")
sequence client = res[2]

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

-- 3. Verfügbare Werkzeuge abfragen
sequence tools_res = mcp:client_list_tools(client)
client = tools_res[3]

-- 4. Werkzeug gezielt ausführen
sequence call_res = mcp:client_call_tool(client, "kunden_suchen", {{"suchbegriff", "Schulze"}})
client = call_res[4]
puts(1, "Antwort vom Server:
" & call_res[2] & "
")

-- 5. Verbindung schließen
mcp:client_close(client)
```

16.4 Universeller LLM-Client (`std/llm.e`)

Das Modul `std/llm.e` bietet eine einheitliche, anbieterunabhängige Schnittstelle für Chat-Completions und Vektor-Embeddings.

16.4.1 Anbindung von lokalem Ollama, OpenAI & Claude

```
include std/llm.e as llm

-- 1. 100% Offline-Betrieb (Keine externen Daemons oder Internetverbindung nötig)
sequence cli_mock = llm:client(llm:PROVIDER_MOCK)

-- 2. Lokales Ollama (z. B. 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 Text-Generierung & Chat

```
-- Einfache Text-Vervollständigung
sequence reply = llm:complete(cli_ollama, "Fasse die Vorteile von SQLite in 2 Sätzen zusammen.")
puts(1, reply[2] & "
")

-- Strukturierter Chat mit Rollen (System, User, Assistant)
sequence messages = {
    {"system", "Du bist ein präziser ERP-Berater."},
    {"user", "Welche Zahlungskonditionen gelten für A-Kunden?"}
}
sequence chat_res = llm:chat(cli_ollama, messages, {{"temperature", 0.3}})
puts(1, chat_res[2] & "
")
```

16.4.3 Generierung von Vektor-Embeddings & Ähnlichkeitsberechnung

```
-- 64-dimensionaler normalisierter Einbettungsvektor
sequence v1 = llm:embed(cli_mock, "Kunde bezahlt Rechnung mit Skonto")
sequence v2 = llm:embed(cli_mock, "Zahlungseingang mit Skontoabzug verbucht")

-- Kosinus-Ähnlichkeit (1.0 = identisch, 0.0 = orthogonal)
atom sim = llm:cosine_similarity(v1, v2)
printf(1, "Semantische Ähnlichkeit: %.2f%%
", {sim * 100.0})
```

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

Die RAG-Engine verbindet SQLite Full-Text Search (FTS5 mit BM25-Ranking) und semantische Vektor-Embeddings zu einem **hybriden Retrieval-System**.

16.5.1 Smart Document Chunking mit Überlappung

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

-- Wissensdatenbank in Datei oder In-Memory anlegen
sequence kb = rag:create_kb("unternehmens_wissen", llm_cli, ":memory:")

-- Dokumente mit automatischem Smart-Chunking (Absatz- & Satzgrenzen) indizieren
integer chunks = rag:add_document(kb, "DOC-01", "Vertriebshandbuch 2026",
    "Kundenkategorie A: Erhalten 15% Sonderrabatt. Zahlungsziel 30 Tage netto oder 3% Skonto binnen 10 Tagen.

" &
    "Kundenkategorie B: Erhalten 5% Rabatt ab 25.000 € Umsatz. Zahlungsziel 14 Tage netto.")
```

16.5.2 Hybrides Retrieval & Quellenbasierte Antworten (`rag:ask`)

```
-- Hybride Suche ausführen und fundierte Antwort mit Zitaten generieren
sequence result = rag:ask(kb, "Welche Skonto-Regeln gelten für A-Kunden?", {{"top_k", 2}})

-- Generierte Antwort ausgeben
puts(1, "Antwort:
" & rag:answer(result) & "

")

-- Quellennachweise formatieren
puts(1, "Verwendete Quellen:
" & rag:format_sources(result) & "
")
```

16.6 Vollständiger Autonomer KI-Agent (`examples/15_autonomous_agent_mcp_rag_demo.ex`)

Das Demonstrationsprogramm examples/15_autonomous_agent_mcp_rag_demo.ex führt alle Komponenten zu einem vollständigen, praxiserprobten End-to-End-Workflow zusammen:

1. **Agent startet MCP-Server** als Subprozess (`stdio`) und bindet Live-ERP-Werkzeuge an.
2. **Agent indiziert Handbücher** in SQLite FTS5 via RAG.
3. **Aufgabe 1 (Lesen & Berichten):** Prüft den Kunden *Schulze Maschinenbau* (liest Live-Fertigungsauftrag über MCP aus SQLite und kombiniert ihn mit Zahlungszielen aus dem RAG-Handbuch zu einem vollständigen Bericht).
4. **Aufgabe 2 (Schreiben & Regeln):** Legt per MCP-Schreibaktion den Neukunden *Quantum Logistics GmbH* in der SQLite-Datenbank an und ermittelt gleichzeitig die passenden B-Kunden-Frachtbedingungen aus dem Handbuch.
5. **Sauberes Herunterfahren** aller Subprozesse und Datenbankhandles.

Ausführen mit:

```
# 100% Offline (deterministische Euphorix-Synthese)
./euphorix examples/15_autonomous_agent_mcp_rag_demo.ex

# Mit lokalem Ollama
./euphorix examples/15_autonomous_agent_mcp_rag_demo.ex --ollama

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