CMLABS · PSYCLONE AIOS

Psyclone 2.3

Technical Whitepaper

By Thor List, CTO

18 November 2026

Contents

1. Introduction
2. About this release
3. System overview
4. Messages and signals
5. Whiteboards and catalogs
6. Data communication
7. Services: the edge of the system
8. PsyProbe 2
9. Distributed systems and federation
10. Robustness
11. Builders, Supervisors and LLM connections
12. Platforms and building
13. Roadmap

1. Introduction

Psyclone is a platform for building large systems out of many heterogeneous modules that exchange heterogeneous data, and for changing how data flows through such a system while it runs. Its full name, Psyclone AIOS, describes the role it plays: not an operating system you install on hardware, but the runtime that hosts, connects and monitors the parts of an application in the way an operating system hosts processes.

The purpose is the same as it has always been: to make it practical for one person or a team to design, build, deploy, test and maintain systems with many interacting parts — perception, cognition and action pipelines, robots, multi-modal dialogue systems, live data analysis — without writing the plumbing between those parts. A Psyclone system is described in one XML file, the **PsySpec**. Components never call each other; they post typed messages and subscribe to the types they care about, so the dataflow can be rerouted by editing the PsySpec, or at run time, without touching module code.

This whitepaper describes the architecture of Psyclone 2.3 and what has changed since the 2.0 series. It assumes some familiarity with networking and software development, but not with earlier versions of Psyclone.

2. About this release

Psyclone 2.3 is a production release for Linux, macOS and Windows (x64). It is the result of a sustained hardening programme on all three platforms and adds four major capabilities:

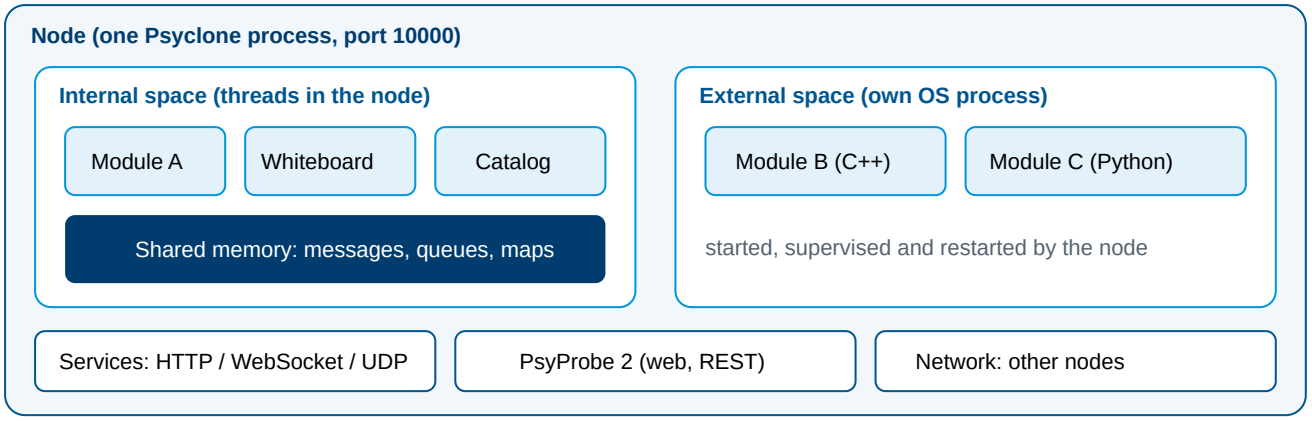
- **PsyProbe 2.0**, a new web application for monitoring and operating a running system, with accounts, roles, HTTPS, live data, control actions, alerts, federation of several systems, and a 3D view. PsyProbe 1.x has been retired.
- **Services**: custom HTTP, WebSocket and UDP endpoints declared in the PsySpec, through which phones, browsers and devices join the message bus without any module owning a socket.
- **Native Windows x64** built with Visual Studio 2022 and OpenSSL 3.5.9, passing the same test suites as Linux and macOS.
- **Crash-safe shared memory**: a process that crashes, or exits, can no longer leave a lock, a shared segment or a wake-up signal in a state that stalls the rest of the node.

Alongside these come ordered message delivery per poster, wildcard type matching, component stop/start/restart and a live parameter editor, a byte-exact Kotlin codec for Android clients, and a long list of fixes. Each release is gated on a suite of more than one hundred engine and library tests and on browser tests of PsyProbe, run on every platform.

3. System overview

A running Psyclone system consists of one or more **nodes**. A node is one Psyclone process on one computer, listening on one main port (10000 by default). A system may span many nodes on many computers and operating systems; the node that loads the PsySpec is the master, and other nodes join it as satellites.

Within a node, components run in **spaces**. A space is a pool of threads with access to the node's shared memory, either inside the node process or in a separate operating-system process. External spaces give crash isolation: a module that crashes takes down only its own space, which the node restarts, while the rest of the system carries on. External spaces may also host modules written as stand-alone executables that link the CMSDK library; the node starts them, supervises them, restarts them with back-off after a crash, and stops them when it stops.



There are three kinds of **component**:

Component	What it does
Module	Subscribes to message types with *triggers*, processes each message in a *crank* — a C++ function in a shared library, or Python code in a file or inline in the PsySpec — and publishes results through *posts*. A crank runs once per trigger, or stays resident in a loop.
Whiteboard	A message store with a retrieval query language: keeps recent messages of chosen types for later retrieval.
Catalog	A module that also answers synchronous queries: a data store, a conduit to another system, a recorder that plays a session back.

Crank code refers only to the *names* of its triggers and posts, never to message types. The PsySpec maps names to types, so a system is rewired by changing the PsySpec. Components can also be created, re-subscribed, stopped and restarted at run time.

```
<psySpec>
  <module name="Ping">
    <trigger name="Ready" type="Psyclone.Ready" />
    <trigger name="Ball" type="ball.1" />
    <crank name="Ping" function="Ping" />
    <post name="Ball" type="ball.2" />
  </module>
  <module name="Pong">
    <trigger name="Ball" type="ball.2" />
    <crank name="Pong" function="Ping" />
    <post name="Ball" type="ball.1" />
    <post name="Done" type="Psyclone.Shutdown" />
  </module>
</psySpec>
```

The classic first system, from the Quickstart: two modules bat a message back and forth with the built-in Ping crank, then shut the system down.

4. Messages and signals

Everything in Psyclone travels as a **DataMessage**: one flat block of memory holding a header (type, sender, receiver, tag, creation time, time-to-live, priority) and any number of named, typed entries — strings, integers, floats, times, binary blobs of any size, nested messages, and arrays and maps of each. Because a message is a single block, it is copied verbatim into shared memory or onto a socket: there is no serialisation step and no schema compiler. A compact **packed form** serves bandwidth-bound links such as phones on mobile networks, and a byte-exact Kotlin implementation lets Android and JVM programs read and write messages without any native code. Python programs use the same class through the SWIG binding.

Message types are dot-separated paths such as `input.audio.raw`. A trigger matches a type exactly or with wildcards (`input.audio.*`, `input.*.raw`); since 2.3 a trailing `*` matches every deeper level. **Contexts** switch whole groups of triggers on and off system-wide, so one PsySpec can describe several modes of operation, and the active set changes at run time. A poster's messages reach each subscriber in the order they were posted.

Signals are a lighter mechanism: a broadcast that bypasses subscription matching, used like a conductor's baton to beat time across a system, for example in simulations.

5. Whiteboards and catalogs

Whiteboards keep recent messages so that components that start late, or need history, can retrieve them by type, time and content. Catalogs answer queries: a module sends a query message and receives a reply, synchronously, from the catalog's crank. Psyclone ships a set of built-in catalogs — among them a FileCatalog, a **RequestStore** that serves the latest messages of chosen types to dashboards, a time-series catalog, and a **ReplayCatalog** that records part or all of the dataflow of a live system and plays it back for analysis and repeatable testing. Custom catalogs are ordinary crank libraries, which is how larger applications build shared world models, task managers and bridges to external services.

6. Data communication

Within a node, messages never leave memory. Components in all spaces of a node share a set of memory segments — the message store, the process and queue tables, the component registry and the name maps

— and a post is a copy into the shared store plus a wake-up of every subscriber's queue. Between nodes, messages for components elsewhere travel over Psyclone's own message protocol on the main port, and the nodes keep their type, context and component tables in step.

The CMSDK library underneath provides the same API on all three operating systems for threads, synchronisation, processes, sockets, an HTTP and WebSocket server and client, timers and shared memory. The 2.3 release reworked the shared-memory primitives so that their behaviour is identical on every platform, including when processes crash (section 10).

7. Services: the edge of the system

Edge devices — phones, browsers, sensors, other programs — reach a Psyclone system through **services** declared in the PsySpec. A service owns the network endpoint, so no module ever handles a socket:

- **HTTP ingestion**: each request becomes a post and is answered immediately.
- **HTTP request/reply**: the request becomes a post, and the message that arrives on the service's trigger becomes the reply, with a declared timeout status if nothing comes. A service can also host static files.
- **WebSocket and UDP streams**: every incoming frame or datagram becomes a post, and every message on the service's trigger is sent out on the connection it belongs to.

Clients choose which declared post a frame goes to by its *name*, never by a raw type, so the PsySpec stays in control of what an outside party can inject. Reliability on lossy links — sequence numbers, acknowledgements, retransmission — is the business of modules, where it can be measured, logged and replayed, rather than of the service.

```
<service name="Phone">
  <post name="AudioFrame" type="Mimir.Audio.Frame" />
  <trigger name="Playout" type="Mimir.Playout.Chunk" />
</service>
<!-- WebSocket arrives as an HTTP upgrade, so it uses the HTTP interface -->
<interface name="PhoneTCP" port="18600" protocol="HTTP" service="Phone" />
<interface name="PhoneUDP" port="18600" ip="UDP" protocol="Message" service="Phone" />
```

A phone streams audio up over UDP or WebSocket while playout comes down on the same connection; replies leave from the service's own port, which is what NAT and mobile networks let back through.

8. PsyProbe 2

Every node serves **PsyProbe 2** on its main port: a web application for developers and operators, and the REST API it is built on.

- **See the system live**: a graph of nodes, processes and components with traffic between them, in 2D or 3D; a dense table of every component's rates, queues and CPU; ten minutes of history behind every chart; the messages a component received and sent in the last minutes; subscriptions, contexts, logs and private data.
- **Operate it**: change parameters with editors generated from the PsySpec, stop, start and restart components, inject test messages, switch contexts, reload the PsySpec, and use a web console — with a confirmation step for anything destructive.
- **Secure it**: accounts with roles (admin, operator, monitoring, user, developer), API keys for scripts, an audit log of every action, HTTPS with a built-in local certificate authority or free certificates by ACME, and masking of secret-looking values.

- **Watch it**: alerts on queue lengths, CPU, stalled cranks, unreachable nodes and error rates, a `/health` endpoint and Prometheus `/metrics`.
- **Extend it**: applications and component views declared in the PsySpec, plain HTML pages that use a small SDK to read data, follow live topics and talk to services, inside PsyProbe's login.
- **Federate it**: one PsyProbe can show and operate several Psyclone systems, with signed identity assertions, certificate pinning and per-peer role caps.

The application is written in Svelte and TypeScript and compiled into static files that the node serves; it needs no server-side runtime. Its source is part of the source distribution.

9. Distributed systems and federation

A PsySpec can place components on named nodes. When it is loaded, the master brings the whole system up across the nodes under the control of a Startup Supervisor, negotiates the shared name and type tables, and confirms each component before announcing `Psyclone.Ready` — exactly once — to the whole system. Satellites can be started with no parameters and wait to be claimed. Separate systems, each with its own PsySpec, can be linked at the PsyProbe level through federation, as in multi-robot deployments where each robot runs its own system.

10. Robustness

Process separation only helps if a crashed process cannot damage the survivors through the memory and locks they share. Psyclone 2.3 closes the ways this could happen on POSIX systems:

- **Locks held by a dead process are recovered.** On Linux, shared mutexes are robust mutexes. On macOS, which has none, a shared mutex is a lock word holding its owner's process id, taken with a single atomic operation; a waiter that finds the owner gone, or exited but not yet reaped, takes the lock over. The macOS design is also faster than the native mutex it replaces (8% lower latency idle, 15% under heavy load).
- **Shared names belong to their creator.** Only the process that created a shared segment, mutex, semaphore or event removes its name; the node removes the names it owns when it shuts down. Before, a component that exited cleanly could remove a name for everyone, and the next process to open it got a new, unconnected object.
- **Wake-up signals cannot be wedged.** Named semaphores on macOS and named events on all POSIX systems are now counters in shared memory that no process ever holds, so a process can be killed at any point inside them.
- **External programs are supervised**, restarted with back-off up to a limit, and tied to the node so that they never outlive it.

Each of these is covered by a test that kills processes at random moments and requires the rest of the system to carry on. The external-process stress test that failed in 18 of 20 runs on macOS at the start of the hardening work now passes 40 of 40.

11. Builders, Supervisors and LLM connections

Each node carries a **Builder**: a declarative pipeline that creates components from recipes, with a stable recipe identity, separate progress and data streams with gap detection, and control verbs (pause, cancel, restart) that

are checked against an allow-list of issuers. **Supervisors** orchestrate it; the Startup Supervisor owns system bring-up today. **LLM connections** are named, node-owned connections to language-model endpoints, declared in the PySpec with credentials taken from files or the environment (never inline), and called in request/reply or streaming mode. In 2.3 these are supervised tools: code and operators drive them, and the model never decides on its own to change the system.

12. Platforms and building

Platform	Toolchain	Notes
Linux x64	GCC, make, OpenSSL 3	Reference platform; binary distribution
macOS	Apple clang, make, Homebrew OpenSSL 3	Apple Silicon and Intel
Windows x64	Visual Studio 2022, OpenSSL 3.5.9 (static)	32-bit Windows retired in 2.3

Modules are written in C++ against the CMSDK library, or in Python 3 through its binding; Android and JVM clients use the Kotlin codec. Psyclone is distributed as source — the engine under the LGPL, the CMSDK library under a BSD licence — and as binary distributions for each platform. The User Guide and System Guide are included and are also served by every running node.

13. Roadmap

The direction after 2.3 builds on the Builder and Supervisor framework: supervisors that observe a running system and decide — for simulation management, performance tuning and testing — and, further out, an LLM tier that proposes and builds modules under supervision. Other planned work includes enforcement of the token and cost limits that LLM connections already declare, global proxying of LLM connections through the master node, and continued work on running Psyclone as a long-lived production service. The System Guide's roadmap chapter lists the status of each item and is updated with every release.

Psyclone is a trademark of Communicative Machines Limited. © 2026 Communicative Machines Limited (CMLabs). Contact: info@cmlabs.com · https://cmlabs.com/psyclone