CMLABS · PSYCLONE AIOS

Psyclone 2.3 Quickstart

Install or build Psyclone, run your first system and watch it in PsyProbe.

From the Psyclone User Guide, chapter 2 · 18 November 2026

The full User Guide and System Guide are in **html/docs.html** of the distribution, and on every running node.

CMLABS · PSYCLONE AIOS

2. Quick Start

Get Psyclone running in minutes: install or build it, run the classic **ping-pong** example, and take your first look at the live system in **PsyProbe**.

Obtain Psyclone

Psyclone ships as **source code** in the Psyclone distribution, dual-licensed LGPL / commercial (see Introduction § Licensing). Obtain the distribution from CMLabs. After unpacking (or checking out) you get a tree containing:

Directory	What it is
`Psyclone/`	The engine — builds the `Psyclone` executable (Windows: `Psyclone.exe`).
`CMSDK/`	The base library your crank libraries compile and link against.
`html/`	The PsyProbe web root (also contains this guide).
example specs	PsySpec XML files such as the ping-pong example used below.

Two placement rules to know before your first run: Psyclone serves PsyProbe from the `html/` tree, so **run it from a directory that contains `html/`** (or point it elsewhere with `html=<dir>`); and your crank libraries (`.dll` / `.so`) are **looked up in the current directory** — put them where you start Psyclone.

The system at a glance

Every Psyclone system follows the same five steps, whatever its size:

1. **Write a PsySpec** — one XML file declaring the components (modules, whiteboards, catalogs) and the dataflow between them.
2. **Start Psyclone** with `spec=<file>`. Psyclone reads the PsySpec, loads any crank libraries, and creates every component.
3. **The system boots**: when everything is ready, Psyclone posts the `Psyclone.Ready` message and prints `SYSTEM READY`.
4. **Components run**: triggers match incoming messages, cranks process them, posts publish results — until something posts `Psyclone.Shutdown`.

5. **Watch it live** in PsyProbe by pointing a browser at the main port (default `http://localhost:10000/`).

Running Psyclone

Windows

```
<path>\Psyclone.exe spec=<PsySpec XML file>
```

UNIX / Linux

```
<path>/Psyclone spec=<PsySpec XML file>
```

The PsySpec file holds the full system specification. Two convenience command-line parameters are available in addition:

Parameter	Meaning	Default
`port=<n>`	Main network port (inter-system communication and the PsyProbe URL)	10000
`html=<dir>`	Path to the PsyProbe HTML directory	`./html`

Beyond these, any `key=value` argument on the command line overrides a PsySpec variable of the same name — the command line wins over the spec's `<variable>` default.

Satellite mode

To use a computer as an extra node in a distributed system, start Psyclone with *no* parameters (optionally `port=<n>`). It idles until a master system's PsySpec claims it with a `<node>` entry — see Distributed Systems.

Building from source

The source tree contains five projects:

Project	What it is
CMSDK	Core library; all OS-dependent base code (threads, networking, shared memory, timing…).
Psyclone	The main executable; links CMSDK.
System	Whiteboards, Catalogs and other built-ins in a single DLL/SO; links CMSDK only; loaded when the PsySpec uses them.
Examples	Example modules in a single DLL/SO.
External Modules	Third-party programs that link CMSDK and connect to a running Psyclone from inside their own executable.

Windows (Visual Studio 2022, x64)

Open `Psyclone.vs2022.sln` and build the **x64** platform in one of four configurations: **Release**, **Debug**, **Release SSL** or **Debug SSL**. The SSL configurations (needed for HTTPS and PsyProbe 2 accounts) link OpenSSL 3.5.9 (LTS) as static libraries from `..\_libs\OpenSSL\openssl-3.5.9-vs2022`; `tools\windows\build-openssl-3.5.9.bat` downloads, checks and builds it there (needs Perl and NASM). Binaries land in `bin\x64`. 32-bit Windows builds were retired in v2.3 (the Win32 configurations are kept but unmaintained).

Linux and macOS (make)

You need a C++ compiler and `make`; add OpenSSL development headers for the SSL variants (Homebrew OpenSSL 3 on macOS), and Python development headers plus SWIG for the Python link. Run `make` in the source root:

```
make            # release
make debug      # debug
make ssl        # SSL-enabled: HTTPS and PsyProbe 2 accounts (needs libssl /
libcrypto dev packages)
make ssldebug
make clean      # always before switching between release, debug and ssl builds
```

Expected result: the `Psyclone` binary in `bin/linux64` (Linux) or `bin/macosx64` (macOS). On macOS run `make sign` once after a build so Gatekeeper does not stop the unsigned binaries.

Python module support

Python crank support is built separately: on Windows build the **Python3Link** DLL project (the SWIG-generated CMSDKPY3 binding expects Python 3.11 at `C:\Python311`; other versions by editing the

project file). On Linux and macOS run `make python3` (optionally `debug`), which finds the headers and library of the `python3` on your path; edit `CMSDK/Makefile.sys` for other locations. Python 2 is end-of-life — treat `make python2` and `language="python2"` as legacy.

> **Note.**
> v2.3 is built and tested with Visual Studio 2022 / OpenSSL 3.5.9 on Windows x64, GCC and OpenSSL 3 on Linux, and Apple clang with Homebrew OpenSSL 3 on macOS. Run `Psyclone test=list` for the built-in tests and `Psyclone test=<name>` to run one.

> **Warning.**
> Recent Psyclone builds verify SSL certificates by default (`SSL_VERIFY_PEER` plus hostname checks). Self-signed certificates that "just worked" with older builds now fail unless you explicitly set `allowselfsigned="yes"` or point `cafile` / `capath` at your own CA — see the System Guide's security chapter.

Your first system: ping-pong

The canonical first PsySpec defines two modules that bat a message back and forth using the built-in `Ping` crank — no user code required:

```
<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>
```

Run it:

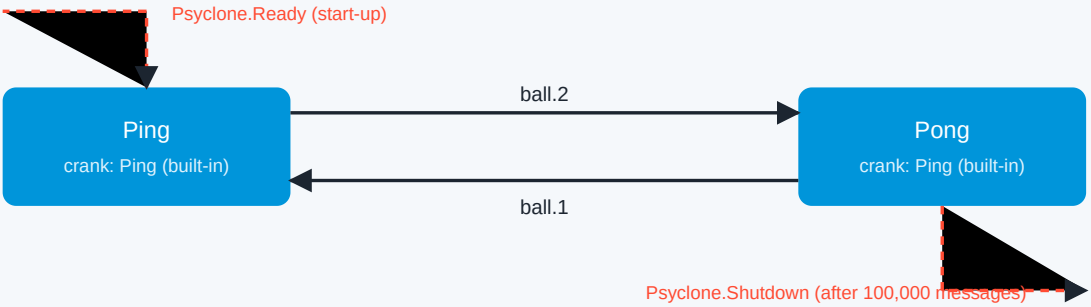
```
Psyclone spec=pingpong.xml
```

What happens: at start-up Psyclone posts `Psyclone.Ready`, which triggers **Ping**; Ping posts a `ball.2` message, which triggers **Pong**; Pong posts `ball.1` back — and so on for 100,000 messages, after which Pong posts `Psyclone.Shutdown`. The 100,000-message count is baked into the built-in `Ping` crank code — it is not set anywhere in the XML. Every 10,000 messages the crank logs throughput statistics (average message time/age in µs). The console shows the start-up configuration, `SYSTEM READY`, the statistics batches, and finally `SYSTEM SHUTDOWN`.

Note the two attributes on `<crank name="Pong" function="Ping"/>`: `name` is *your label* for this crank instance, while `function` selects *which crank code runs* — here both modules run the same built-in `Ping` function under different names.

> **Smoke test.**
> Before moving on, confirm the run actually worked: you should see `SYSTEM READY` in the console followed by the periodic throughput lines. If either is missing, see When things go wrong below.

Ping-pong dataflow



Ping-pong: `Psyclone.Ready` starts the rally; the ball bounces as `ball.2` / `ball.1` until Pong posts `Psyclone.Shutdown`.

> **Tip.**
> Notice that the modules never mention each other — each just posts and subscribes to *message types*. That decoupling is the core of the Psyclone mental model, explained in Core Concepts.

First look at PsyProbe

While a system runs, point any browser at the main port — by default `http://localhost:10000/` — for a dynamic, live-updating view of all nodes, spaces and components. The overview page shows the current time and uptime, the node list with activity levels, and expandable sections for System Info, Nodes, Catalogs (including whiteboard stored messages) and Modules with per-component performance, activity and

subscription tabs. Try the ping-pong system: open the **Pong** module's **Activity** tab and watch the last ten in/out messages tick past. PsyProbe is covered in depth in chapter 12.

> **Note.**
> Ping-pong at full speed posts messages faster than you can read; PsyProbe samples the activity view, so don't worry if counters jump.

When things go wrong (first run)

The five most common first-run failures:

Symptom	Cause & fix
Spec parse error at start-up	Bad XML in the PsySpec — the log prints the file, line and column of the problem; fix the XML there.
Crank library not found	A crank uses `function="Lib::Fn"` but the `.so` / `.dll` is not in the current directory / library path. Put it next to where you start Psyclone, or set the library path.
Port already in use	Another process (or another Psyclone) holds the main port. Start with a different `port=<n>`.
PsyProbe page blank or 404	You are not running from a directory containing the `html/` tree. Start Psyclone there, or pass `html=<dir>`.
`SYSTEM READY` never appears	A module's crank blocked at start-up, or a required node/space did not come up. Raise the verbosity level and check the log.

For deeper diagnosis — log levels, tracing dataflow, common crank bugs — see the System Guide's Debugging chapter.