MCS Engine: SDK and C API Guide

Contents

Multi-model Reasoning, Deliberation, and Decision System Osinix (osinix.com)

This guide is for developers who want to run MCS deliberations from their own programs instead of the web interface. It covers the C API (`libmcs`), the Python SDK, installation, the session lifecycle, and error handling. For the concepts (panels, agendas, modes, the session config format), see the User Guide; this guide assumes you know them.

---

1. What the SDK is

The SDK lets your code start an MCS deliberation, watch its progress, and collect its results. It comes in two forms that sit on the same foundation:

- **The C API**, a shared library (`libmcs.so` on Linux, `libmcs.dylib` on macOS) and one header (`libmcs.h`).
- **The Python SDK**, a small package (`mcs`) that wraps the C library.

Under the hood, `libmcs` does not reimplement the engine. When you start a session, the library **forks and execs an installed MCS engine binary** of the matching tier, hands it your configuration, and then reports on its progress by reading the engine's status files. This means two things:

- An **installed MCS engine is required**; the SDK drives it, it does not replace it.
- The deliberation behaves exactly as it does from the web or command line, because it is the same engine.

The same single-session rule applies as everywhere else: one session runs at a time, enforced by a lockfile. Starting a second while one is running returns an error.

---

2. Requirements and licensing

- An **installed MCS engine** (the SDK forks and execs it).
- A license whose **SDK access is enabled**. The SDK is **included with Pro Max and Enterprise**; Basic and Pro can enable it with a license add-on. If your license does not grant SDK access, `mcs_session_start` fails with `MCS_ERR_FEATURE_NOT_LICENSED`.
- For Python: a standard CPython install (the package uses `ctypes`, no compilation).

At launch the SDK ships a **C API** and a **Python SDK**. C++ programs can use the C API directly, since the header is C-compatible. Bindings for other languages such as Go and Java may follow.

The SDK ships for both **Linux x86_64** and **macOS (Apple Silicon)**, installed the same way by the same `install_sdk.sh`. The only platform differences are the library name (`libmcs.so` on Linux, `libmcs.dylib` on macOS) and, on macOS, a one-time quarantine clear on the downloaded folder (see below).

---

3. Installing the SDK

The SDK ships as the native library (`libmcs.so` on Linux, `libmcs.dylib` on macOS), the header `libmcs.h`, and the Python package. Install everything with the included script, run from the unpacked SDK package.

macOS only: first clear the download quarantine on the unpacked folder, or macOS will block the library:

```
xattr -dr com.apple.quarantine mcs-sdk-<version>-macos_arm64
```

Then, on either platform:

```
./install_sdk.sh
```

It is interactive by default and asks where your engine is installed and where to place the files. By default it groups the SDK under the engine, so a normal install needs no arguments and copies:

- library -> `<engine-dir>/sdk/lib/libmcs.<ext>`

- header -> `<engine-dir>/sdk/include/libmcs.h`
- Python -> `<engine-dir>/sdk/python`

with the engine directory defaulting to `~/osinix/mcs`. Everything is copied into the install, so nothing depends on the unpacked download folder afterward.

Options:

```
./install_sdk.sh -y                   # accept all defaults
./install_sdk.sh --engine-dir=DIR     # where the engine is installed
./install_sdk.sh --lib-dir=DIR        # where to put the library
./install_sdk.sh --include-dir=DIR    # where to put libmcs.h
./install_sdk.sh --uninstall          # remove the installed SDK files
```

Using the Python SDK

After `install_sdk.sh`, pick one of two ways. Both use the installed copy under `<engine-dir>/sdk/python`, so the download folder is not needed.

A) No pip. Put the wrapper on your Python path:

```
# add to ~/.zshrc (macOS) or ~/.bashrc (Linux) to make it permanent
export PYTHONPATH="$HOME/osinix/mcs/sdk/python:$PYTHONPATH"
python3 -c "import mcs; print(mcs.version())"
```

B) pip install (needs pip and setuptools 61 or newer):

```
pip install --user "$HOME/osinix/mcs/sdk/python"
python3 -c "import mcs; print(mcs.version())"
```

If pip builds a package named `UNKNOWN`, your setuptools is old; run `pip install --user --upgrade pip setuptools` and reinstall.

The Python wrapper finds the native library automatically from the install location, and for a default engine install it also finds the license and engine on its own, so `mcs.init()` **needs no arguments and no environment variables**.

Where things are found, and how to override each:

- **License:** `<engine-dir>/config/license.key` (default `~/osinix/mcs/config/license.key`). If yours is elsewhere, pass `mcs.init(license_path="...")` or set `MCS_LICENSE_KEY` to the file.
- **Engine binary:** under `<engine-dir>/bin`. Override with `mcs.init(engine_path="...")`.
- **Native library:** found from its own install location. Override with `MCS_LIB_PATH` set to the library file.

(In C, the two arguments to `mcs_library_init(engine_path, license_path)` are the equivalent overrides.)

Using the C library

Build against the header and library (`<sdk>` is your `<engine-dir>/sdk`):

```
cc myprog.c -I"$HOME/osinix/mcs/sdk/include" -L"$HOME/osinix/mcs/sdk/lib" -lmcs -o myprog
```

At run time the dynamic loader must be able to find the library. For an install under `<engine-dir>/sdk/lib`, add it to the loader path:

```
export LD_LIBRARY_PATH="$HOME/osinix/mcs/sdk/lib:$LD_LIBRARY_PATH"     # Linux
export DYLD_LIBRARY_PATH="$HOME/osinix/mcs/sdk/lib:$DYLD_LIBRARY_PATH"  # macOS
```

For a **system-wide** install on Linux, point the installer at the system paths and run as root; it then runs `ldconfig`, so no

loader-path variable is needed:

```
sudo ./install_sdk.sh --lib-dir=/usr/lib --include-dir=/usr/include
```

Once the C library is loaded, it locates the license and engine automatically from the install (same as the Python SDK), so `mcs_library_init(NULL, NULL)` works without any environment variables.

---

4. The session lifecycle

Every SDK program, in C or Python, follows the same shape:

1. **Initialize** the library once, which loads and verifies your license.
2. **Provide a session configuration** (an `mcs.conf` and a session config file, the same formats described in the User Guide).
3. **Start** the session, which launches the engine subprocess.
4. **Poll** progress (turn, round, phase, status) and/or **wait** for it to finish.
5. **Read the results** from the session directory (transcript, produced files).
6. **Free** the session handle.

License and tier limits are checked at start: if the configuration exceeds your tier (too many participants, a mode your tier lacks, an unsupported file type), start fails with a specific error rather than running and failing later.

---

5. Python SDK

The Python package mirrors the lifecycle above with a context-managed `Session`.

```
import mcs

mcs.init()                      # finds the license and engine automatically
print("SDK version:", mcs.version())

with mcs.start("mcs.conf", "session.conf") as session:
    session.wait()              # block until the deliberation finishes
    print("status:", session.status())
    print("session dir:", session.session_dir())
    for path in session.output_files():
        print("output:", path)
    for path in session.media_files():
        print("produced file:", path)
```

Key calls:

- `mcs.version()` returns the SDK version string.
- `mcs.init(license_path=None, engine_path=None)` initializes the library. With no arguments it finds the license and engine automatically from the install location (no environment variables needed); pass paths only to override a non-standard layout.
- `mcs.start(mcs_conf, session_conf)` launches a session and returns a `Session`.
- `mcs.locations()` returns the engine's resolved paths (install, config, sessions, temp).
- `mcs.license_info()` returns the active license's licensee, tier, expiry, days remaining, and whether SDK access is granted (`has_sdk`).

A `Session` (use it as a context manager so it is freed automatically) provides:

- `wait(timeout=None)` blocks until the session ends; with a timeout it raises `MCSTimeoutError` if the session is still running

when the time elapses.

- `stop(force=False)` ends the session: a graceful stop lets the engine flush its final output; `force=True` kills it immediately.
- `is_done()`, `status()`, `phase()`, `current_turn()`, `last_completed_turn()`, `current_round()`, `last_completed_round()` for progress.
- `session_dir()` for the session's folder.
- `input_files()`, `output_files()`, `media_files()` list the session's files (the produced files are under `media`).

Errors are raised as exceptions: `MCSError` (base), `MCSLicenseError`, `MCSTimeoutError`, and `MCSConfigError`. Catch `MCSLicenseError` to handle an expired or non-SDK license cleanly.

The SDK includes runnable examples (a basic run, a batch reviewer, a CI quality gate) under the package's `examples/` directory.

For the complete per-function reference, including every constant and error code, see `SDK_API_REFERENCE.pdf`. Reading the examples alongside it is the fastest way to get familiar with the API.

---

6. C API

Link against `libmcs` and include the single header:

```
#include "libmcs.h"
```

```
cc myprog.c -I<sdk>/include -L<sdk>/lib -lmcs -o myprog
```

(where `<sdk>` is your `<engine-dir>/sdk`; see section 3 for the run-time loader-path note.)

A minimal session, with error checking via `mcs_error`:

```c
#include "libmcs.h"
#include <stdio.h>

int main(void) {
    if (mcs_library_init(NULL, NULL) != MCS_OK) {
        fprintf(stderr, "init failed: %s\n", mcs_error());
        return 1;
    }

    mcs_session_handle_t *s = mcs_session_start("mcs.conf", "session.conf");
    if (!s) {
        fprintf(stderr, "start failed: %s\n", mcs_error());
        return 1;
    }

    /* Block until the deliberation ends on its own. */
    if (mcs_session_wait(s, 0) != MCS_OK) {
        fprintf(stderr, "wait: %s\n", mcs_error());
    }

    char dir[4096];
    if (mcs_session_get_session_dir(s, dir, sizeof dir) == MCS_OK) {
        printf("results in: %s\n", dir);
    }

    mcs_session_free(s);
    return 0;
}
```

The core calls:

- `mcs_library_init(engine_path, license_path)` once at startup. Pass `NULL` for either to have the library find it automatically from the install location. Loads and verifies the license.
- `mcs_library_version()` returns the version string (safe to call before init).
- `mcs_session_start(mcs_conf_path, session_conf_path)` returns a handle, or `NULL` on failure (check `mcs_error` / `mcs_errno`).
- `mcs_session_wait(handle, timeout_sec)` blocks until the session ends naturally (use `0` to wait indefinitely).
- `mcs_session_stop(handle, mode)` ends it: `MCS_STOP_GRACEFUL` sends an interrupt and lets the engine flush final output within a 30-second grace, returning `MCS_ERR_TIMEOUT` if it does not exit in time; `MCS_STOP_FORCE` kills it immediately. Calling stop on a finished session is a no-op.
- `mcs_session_free(handle)` releases the handle.

Progress and results:

- `mcs_session_is_done(handle)`.
- `mcs_session_get_turn` / `_get_round` / `_get_phase` / `_get_status` (pass `MCS_CURRENT` for the current value).
- `mcs_session_get_session_dir(handle, buf, buflen)`.
- `mcs_session_get_file_count(handle, category)` and `mcs_session_get_file(handle, category, index, buf, buflen)`, with category `MCS_FILES_INPUT`, `MCS_FILES_OUTPUT`, or `MCS_FILES_MEDIA`.

License and locations:

- `mcs_license_info(&info)` fills licensee, tier, expiry, days remaining, and whether SDK access is granted.
- `mcs_locations(&loc)` fills the engine's resolved install, config, sessions, and temp paths.

Error handling: every call returns `MCS_OK` (0) on success and a negative `MCS_ERR_*` code on failure; functions that return a

pointer return `NULL` on failure. After any failure, `mcs_error()` gives a human-readable message and `mcs_errno()` gives the last code.

---

7. Building the session configuration

The SDK runs the same configuration files as the rest of MCS: an `mcs.conf` (engine settings) and a session config (the panel, agenda, and mode). You can author the session config by hand in the format documented in the User Guide, or build it programmatically as JSON and serialize it with `mcs_session_config_save_conf` (or `mcs_session_config_save_json`), which writes a file you then pass to `mcs_session_start`.

The three required elements are the same as everywhere: an agenda, at least two participants, and a moderator that matches a participant. Tier limits apply, and are enforced at start.

---

8. Error codes

All C calls share one set of codes. `MCS_OK` is `0`; failures are negative.

Code	Name	Meaning
`0`	`MCS_OK`	Success
`-1`	`MCS_ERR_NOT_INITIALIZED`	`mcs_library_init` has not succeeded
`-2`	`MCS_ERR_LICENSE_INVALID`	License missing or invalid
`-3`	`MCS_ERR_LICENSE_EXPIRED`	License past its expiry
`-4`	`MCS_ERR_ENGINE_BINARY_NOT_FOUND`	No matching engine binary installed
`-5`	`MCS_ERR_CONFIG_NOT_FOUND`	A config file path does not exist
`-6`	`MCS_ERR_CONFIG_INVALID`	A config file is malformed
`-7`	`MCS_ERR_SESSION_ALREADY_RUNNING`	Another session is active
`-8`	`MCS_ERR_SUBPROCESS_SPAWN_FAILED`	The engine process failed to start
`-9`	`MCS_ERR_HANDLE_INVALID`	NULL or invalid session handle
`-10`	`MCS_ERR_TIMEOUT`	Wait or graceful-stop time elapsed
`-11`	`MCS_ERR_STATUS_FILE_MISSING`	The engine status file was not found
`-12`	`MCS_ERR_SESSION_ABORTED`	The session ended abnormally
`-13`	`MCS_ERR_IO`	An I/O operation failed
`-14`	`MCS_ERR_INVALID_PARAMETER`	A bad argument was passed
`-15`	`MCS_ERR_INTERNAL`	Unexpected internal error
`-16`	`MCS_ERR_TIER_LIMIT_EXCEEDED`	Config exceeds your tier's limits
`-17`	`MCS_ERR_SESSION_ALREADY_DONE`	The session has already finished
`-18`	`MCS_ERR_FEATURE_NOT_LICENSED`	Valid license, but no SDK access
`-19`	`MCS_ERR_FILE_FORMAT_NOT_ALLOWED`	A file type your tier cannot use
`-20`	`MCS_ERR_NO_ACTIVE_SESSION`	No session is active
`-21`	`MCS_ERR_LOCATIONS_UNAVAILABLE`	The engine's locations could not be resolved

The Python SDK maps these onto exceptions (`MCSError` and its subclasses), so you handle them with `try` / `except` instead of checking return codes.

---

9. Support

- **Developer and integration help, and bug reports:** support@osinix.com
- **Licensing, SDK access, and add-ons:** licensing@osinix.com
- **Documentation:** https://osinix.com/docs
