OmniLink

Python Library -- API Reference

Version 0.6.0

Table of Contents

Index

1. Installation

```
# From PyPI
pip install omnilink

# Or install from source
git clone https://github.com/omnilink/omnilink
pip install -e omnilink/omnilink-lib
```

Requirements

Requirement	Details
Python	3.9 or later
Dependencies	requests (installed automatically)
Optional -- Chess benchmark	pip install chess
Optional -- Robot Demo	pip install pygame
Optional -- Husky benchmarks	pip install pybullet

Module Structure

```
omnilink/
  __init__.py          # Core: OmniLinkEngine, TypeRegistry, OmniLinkHTTPBridge,
                       #        AgentFeedback, AgentQuestion, OmniLinkChatClient
  client.py            # OmniLinkClient (full REST API), OmniLinkAPIError
  tool_runner.py       # ToolRunner base class
  examples/
    hello_world.py         # Minimal engine demo (no key needed)
    chat_quickstart.py     # OmniLinkChatClient demo
    client_demo.py         # Full REST API walkthrough
    arithmetic_tools.py    # Tool-calling with agent profiles
    messaging_demo.py      # Messenger + operator confirmation
    industrial_gateway_demo.py  # Batching, retries, heartbeat
    pacman/                # BFS pathfinding benchmark
    chess/                 # Minimax + alpha-beta benchmark
    tetris/                # Pierre Dellacherie benchmark
    breakout/              # Ball trajectory benchmark
    pong/                  # Full simulation benchmark
    space_invaders/        # Lead-shot targeting benchmark
    montezuma/             # BFS + enemy avoidance benchmark
    go/                    # Heuristic evaluation benchmark
    asteroids/             # Multi-tool strategic benchmark
    husky/                 # Single robot navigation
    husky_fleet/           # 5-robot fleet coordination
    robot_demo/            # UI-controlled robot demo
```

Get Your Omni Key

Before using any platform features, generate an Omni Key at https://www.omnilink-agents.com/omnilink-api . Sign in, click Generate API key , and copy the key (starts with olink_). Set it as an environment variable:

```
export OMNI_KEY="olink_YOUR_KEY_HERE"
```

Troubleshooting Installation

ModuleNotFoundError: No module named 'omnilink' --
Ensure you installed the package in the correct Python environment.
If you use virtual environments, activate yours first: source venv/bin/activate && pip install omnilink .
Verify the install with python -c "import omnilink; print('OK')" .

OMNI_KEY not found or authentication failures --
Confirm the key is set in your current shell session by running echo $OMNI_KEY (Linux/macOS) or echo %OMNI_KEY% (Windows).
If the key starts with something other than olink_ , it may be invalid -- regenerate it at https://www.omnilink-agents.com/omnilink-api .
On Windows, use set OMNI_KEY=olink_YOUR_KEY instead of export .

requests.ConnectionError when calling the API --
This usually means the OmniLink platform is unreachable. Check your internet connection and verify you can reach https://www.omnilink-agents.com in a browser.
If you are behind a corporate proxy, configure the HTTPS_PROXY environment variable.

2. ToolRunner

Base class for cloud-orchestrated local tool execution. Subclass ToolRunner to create agents that run autonomously with
minimal API cost. The cloud AI makes a single tool-call to kick off;
your local code handles the rest.

Credit usage: 1 credit to kick off + 1 for final analysis.
A 30-minute session typically costs 1-2 credits total.

```
class omnilink.tool_runner.ToolRunner
```

Constructor

No constructor arguments. All configuration is done via class attributes.
Import with:

```
from omnilink.tool_runner import ToolRunner
```

Class Attributes

Attribute	Type	Default	Required	Description
agent_name	str	"tool-agent"	Yes (override)	Agent profile name on the OmniLink platform. Must be unique per agent.
display_name	str	"Tool"	Yes (override)	Human-readable name shown in banners, logs, and the web UI.
base_url	str	"https://www.omnilink-agents.com"	No	OmniLink API base URL.
omni_key	str	os.environ["OMNI_KEY"]	Yes	Your Omni Key for authentication. Read from environment by default.
engine	str	"g3-engine"	No	AI engine to use. Options: "g1-engine" (Gemini), "g2-engine" (GPT), "g3-engine" (Grok), "g4-engine" (Claude).
poll_interval	float	0.0	No	Seconds to sleep between ticks in the main loop. Set to 0.0 for maximum speed.
memory_every	int	60	No	Save state_summary() to agent memory every N seconds.
ask_every	int	2400	No	Periodic agent review interval in seconds. The AI is asked whether to continue or stop.
tool_name	str	"make_move"	No	Name of the action tool the cloud agent calls to trigger local execution.
tool_description	str	"Execute the next action."	No	Human-readable description of the action tool, shown to the AI.

Attribute	Type	Default	Required	Description
commands	str	"stop_game, pause_game, resume_game"	No	Comma-separated list of available UI commands.
game_server_url	str | None	None	No	URL of the local game/system server. Used for pause and resume commands.
tool_callback_url	str | None	None	No	URL the web UI calls for query tools. Auto-set when query_tools is defined.
query_tools	list[dict]	[]	No	List of query tool definitions. Each dict has "name" and "description" keys. When non-empty, a lightweight HTTP callback server starts automatically.

Required Methods (must override)

get_state()

Fetch the current state from the target system. Called every tick in the main loop. This method should return a dictionary representing the complete observable state of your target system (e.g. game board, robot sensors, API status). The returned dict is passed to execute_action() , state_summary() , is_game_over() , and log_events() , so include all fields those methods need. If the target system is unreachable, this method should raise an exception rather than returning partial data.

Parameter	Type	Description
No parameters.		

Returns: dict -- Current state of the target system. The structure is defined by your target system.

Sample Usage:

```
def get_state(self):
    return requests.get("http://localhost:5000/state").json()
```

execute_action(state)

Decide and send the next action based on the current state. Called every tick while the game is active and not paused. This is where your core AI logic lives -- read the state, decide what to do, and send the action to the target system via HTTP or another transport. The method should return None ; the result of the action will be visible on the next tick when get_state() is called again. If the game is paused (via a pause_game command from the UI), this method is skipped until the game resumes.

Parameter	Type	Required	Description
state	dict	Yes	The current state dict returned by get_state() .

Returns: None

Sample Usage:

```
def execute_action(self, state):
    if state.get("game_state") == "PLAY":
        action = my_engine.decide(state)
        requests.post("http://localhost:5000/callback", json={"action": action})
```

state_summary(state)

Return a concise text summary of the current state. Used for memory persistence (saved every memory_every seconds) and for the final analysis report. Keep summaries short (one or two lines) -- they are stored in the agent's conversation memory and sent to the AI for context. Include only the most important metrics (score, position, health) rather than dumping the entire state dict. The AI uses these summaries to understand progress over time, so consistency in format helps.

Parameter	Type	Required	Description
state	dict	Yes	The current state dict returned by get_state() .

Returns: str -- A human-readable summary, typically one line.

Sample Usage:

```
def state_summary(self, state):
    return f"Score: {state['score']} | Level: {state['level']} | Lives: {state['lives']}"
```

is_game_over(state)

Return True when the task or game has ended. The main loop exits when this returns True .

Parameter	Type	Required	Description
state	dict	Yes	The current state dict returned by get_state() .

Returns: bool -- True if the task is finished, False otherwise.

Sample Usage:

```
def is_game_over(self, state):
    return state.get("game_state") == "GAMEOVER"
```

Optional Methods

on_start()

Called once after the agent kicks off, before the main loop begins. Override to send an initial command (e.g. START or RESUME) to the game server.

Returns: None

Default: No-op.

Sample Usage:

```
def on_start(self):
    requests.post("http://localhost:5000/callback", json={"action": "START"})
```

log_events(state)

Print noteworthy events each tick (score changes, level-ups, lives lost, etc.). Called every tick after execute_action() .

Parameter	Type	Required	Description
state	dict	Yes	The current state dict.

Returns: None

Default: No-op.

Sample Usage:

```
def log_events(self, state):
    if state["score"] != self._last_score:
        print(f"Score changed: {self._last_score} -> {state['score']}")
        self._last_score = state["score"]
```

game_over_message(state)

Return text for the game-over banner displayed when the task ends.

Parameter	Type	Required	Description
state	dict	Yes	The final state dict.

Returns: str

Default: "GAME OVER"

execute_query_tool(tool_name, **kwargs)

Handle a query-tool call from the OmniLink web UI. When query_tools is defined, the ToolRunner starts an HTTP callback server; the UI POSTs tool calls directly to it, and this method is invoked.

Parameter	Type	Required	Description
tool_name	str	Yes	Name of the query tool being called (e.g. "get_score").
**kwargs	Any	No	Additional arguments passed by the UI in the tool call payload.

Returns: dict -- Result payload sent back to the UI.

Default: Returns {"error": f"Unknown tool: {tool_name}"} .

Sample Usage:

```python
class MyRunner(ToolRunner):
    query_tools = [
        {"name": "get_score", "description": "Returns the current score."},
        {"name": "get_position", "description": "Returns player XY position."},
    ]

    def execute_query_tool(self, tool_name, **kwargs):
        state = self.get_state()
        if tool_name == "get_score":
            return {"score": state.get("score", 0)}
        if tool_name == "get_position":
            return {"x": state.get("x"), "y": state.get("y")}
        return {"error": f"Unknown tool: {tool_name}"}
```

get_system_instruction()

Override to provide a custom system instruction for the initial tool-call kickoff.

Returns: dict -- Contains mainTask , availableTools , availableToolDetails , availableCommands , allowToolUse .

get_review_instruction()

Override to provide a custom system instruction for periodic agent reviews.

Returns: dict -- Same structure as get_system_instruction() .

get_profile_settings()

Override to provide custom agent profile settings stored on the platform.

Returns: dict -- Contains agentName , mainTask , availableTools , availableToolDetails , availableCommands , allowToolUse , and optionally toolCallbackUrl .

run()

Start the ToolRunner lifecycle. This is a blocking call that runs until is_game_over() returns True or the user sends a stop_game command.

Returns: None

Lifecycle

```
1. Profile setup       -- creates or updates the agent profile on OmniLink
2. Memory clear         -- clears any previous conversation memory
3. Tool-call kickoff    -- one API call to trigger the tool (1 credit)
4. on_start()           -- your custom initialisation hook
5. Main loop:
   a. get_state()         -- fetch current state from target system
   b. is_game_over()?     -- check termination condition
   c. execute_action()    -- decide and send the next action
   d. log_events()        -- print noteworthy events
   e. Memory persistence  -- save state_summary() every memory_every seconds
   f. UI command polling   -- check for stop_game / pause_game / resume_game
   g. Periodic review     -- ask agent to continue/stop every ask_every seconds
6. Final analysis       -- save final state, ask agent for summary (1 credit)
7. game_over_message()  -- display the final banner
```

Complete Example

Example 1 -- Minimal ToolRunner:

```python
from omnilink.tool_runner import ToolRunner
import requests

class MyRunner(ToolRunner):
    agent_name = "my-agent"
    display_name = "My Task"

    def get_state(self):
        return requests.get("http://localhost:8000/state").json()

    def execute_action(self, state):
        action = "UP" if state["y"] < state["target_y"] else "DOWN"
        requests.post("http://localhost:8000/action", json={"action": action})

    def state_summary(self, state):
        return f"Position: ({state['x']}, {state['y']})"

    def is_game_over(self, state):
        return state.get("done", False)

if __name__ == "__main__":
    MyRunner().run()
```

Example 2 -- ToolRunner with query tools and custom hooks:

```python
from omnilink.tool_runner import ToolRunner
import requests

class PacmanRunner(ToolRunner):
    agent_name = "pacman-agent"
    display_name = "Pac-Man"
    tool_description = "Run Pac-Man AI."
    game_server_url = "http://127.0.0.1:5000"
    query_tools = [
        {"name": "get_score", "description": "Returns the current score."},
        {"name": "get_position", "description": "Returns player position."},
        {"name": "get_ghosts", "description": "Returns ghost positions."},
    ]

    def __init__(self):
        self._last_score = 0

    def get_state(self):
        return requests.get("http://localhost:5000/state").json()

    def execute_action(self, state):
        if state.get("game_state") == "PLAY":
            action = decide_action(state)  # your BFS / pathfinding logic
            requests.post("http://localhost:5000/callback", json={"action": action})

    def execute_query_tool(self, tool_name, **kwargs):
        state = self.get_state()
        if tool_name == "get_score":
            return {"score": state.get("score", 0)}
        if tool_name == "get_position":
            return {"x": state["pacman"]["x"], "y": state["pacman"]["y"]}
        if tool_name == "get_ghosts":
            return {"ghosts": state.get("ghosts", [])}
        return {"error": f"Unknown tool: {tool_name}"}

    def state_summary(self, state):
        return f"Score: {state['score']} | Level: {state['level']} | Lives: {state['lives']}"

    def is_game_over(self, state):
        return state.get("game_state") == "GAMEOVER"

    def on_start(self):
        requests.post("http://localhost:5000/callback", json={"action": "START"})

    def log_events(self, state):
        if state["score"] != self._last_score:
            print(f"  Score: {self._last_score} -> {state['score']}")
            self._last_score = state["score"]

    def game_over_message(self, state):
        return f"GAME OVER | Final Score: {state.get('score', 0)}"

if __name__ == "__main__":
    PacmanRunner().run()
```

Troubleshooting ToolRunner

The agent starts but no actions are sent --
Ensure your game server is running and reachable at the URL used by get_state() . If get_state() raises a ConnectionError , the ToolRunner will keep retrying silently.
Add a print() inside execute_action() to confirm
it is being called. Also verify that is_game_over() does not

return True immediately -- this would cause the loop to exit
on the first tick.

Query tools are not responding --
When query_tools is defined, the ToolRunner starts an HTTP
callback server on a random port. Ensure no firewall is blocking the
port. The URL is printed to stdout at startup (look for "Tool callback server listening on ...").
Also verify that execute_query_tool() returns a dict ,
not None . A None return causes a silent failure.

Agent uses too many credits --
The ToolRunner is designed to use only 1-2 credits per session.
If you see higher usage, check that ask_every is not set too low
(each review triggers an API call). The default of 2400 seconds (40 minutes)
keeps costs minimal.

3. OmniLinkEngine

The pattern-driven command engine. Matches input strings against registered templates, extracts typed variables, and dispatches to handler functions. Includes middleware support, metrics tracking, and event history.

```
class omnilink.OmniLinkEngine(patterns, types=None, keep_history=200)
```

Constructor

Parameter	Type	Default	Required	Description
patterns	Iterable[str]	--	Yes	Template strings with optional [varname:typename] placeholders. Example: "set speed to [speed:float]" .
types	TypeRegistry | None	None	No	Custom type registry. If None , a default registry with all built-in types is used.
keep_history	int	200	No	Maximum number of events to retain in the history deque.

Sample Usage:

```
from omnilink import OmniLinkEngine

engine = OmniLinkEngine([
    "hello",
    "echo [message:any]",
    "set speed to [speed:float]",
    "move [direction] [distance:int] meters",
])
```

Attributes

Attribute	Type	Description
types	TypeRegistry	The type registry used for variable extraction.
metrics	Counter	Command execution counts, keyed by normalised template.
history	Deque[Event]	Bounded event history (most recent events at the end).

handle(text, meta=None, messenger=None)

Match a command string against all registered templates, extract variables, run middleware and handlers, update metrics and history. This is the primary entry point for processing commands. The engine normalises whitespace and tests the input against each template in registration order; the first match

wins. If a handler is registered for the matched template, it is called with
an event dict containing the parsed variables, original command text, and
optional metadata. Before- and after-middleware run around every handler call.
If no template matches, the result has ok: false and the errors list explains what was tried.

Parameter	Type	Default	Required	Description
text	str	--	Yes	The raw command text to process.
meta	dict | None	None	No	Arbitrary metadata passed through to handlers via the event.
messenger	AgentMessenger | None	None	No	Messenger for sending feedback or questions back to the caller.

Returns: dict -- Result with keys: ok (bool), template (str|None), normalized_template (str|None), vars (dict), result (any), errors (list).

Sample Usage:

```
result = engine.handle("echo good morning")
print(result["ok"])       # True
print(result["template"]) # "echo [message:any]"
print(result["vars"])     # {"message": "good morning"}
print(result["result"])   # return value of the handler
```

parse(text)

Parse a command against templates without executing handlers. Useful for validation or preview.

Parameter	Type	Required	Description
text	str	Yes	The command text to parse.

Returns: dict -- Result with keys: ok , template , normalized_template , vars , errors .

Sample Usage:

```
parsed = engine.parse("set speed to 42.5")
print(parsed["ok"])    # True
print(parsed["vars"])  # {"speed": 42.5}
```

on_template(template, handler)

Register a handler function for a specific template. The handler receives the event dict and should return the result.

Parameter	Type	Required	Description
template	str	Yes	The template string to match (must be one of the registered patterns).
handler	Callable[[dict], Any]	Yes	Handler function. Receives the event dict with command , vars , meta , messenger .

Returns: None

Sample Usage:

```
engine.on_template("hello", lambda e: {"message": "Hello, world!"})

engine.on_template("set speed to [speed:float]", lambda e: {
    "new_speed": e["vars"]["speed"]
})
```

on(predicate, handler)

Register a handler with a custom matching predicate. The predicate is checked after template matching succeeds.

Parameter	Type	Required	Description
predicate	Callable[[dict], bool]	Yes	Function that receives the event dict and returns True to trigger the handler.
handler	Callable[[dict], Any]	Yes	Handler function.

Returns: None

Sample Usage:

```
engine.on(
    lambda e: e["vars"].get("speed", 0) > 100,
    lambda e: {"warning": "Speed exceeds safe limit!"}
)
```

before(callback) / after(callback)

Register middleware that runs before or after every command execution.

Parameter	Type	Required	Description
callback	Callable[[dict], Any]	Yes	Middleware function. Receives the event dict.

Returns: None

Sample Usage:

```
# Log every command before processing
engine.before(lambda e: print(f"[CMD] {e['command']}"))

# Log results after processing
engine.after(lambda e: print(f"[DONE] {e.get('template', 'unknown')}"))
```

add_template(template)

Add a new command template at runtime.

Parameter	Type	Required	Description
template	str	Yes	The template string to add.

Returns: None

templates (property)

Returns a list of all registered template strings.

Returns: list[str]

get_recent_events(limit=None, *, newest_first=True, include_messenger=False)

Return JSON-serializable snapshots of recent events from the history.

Parameter	Type	Default	Required	Description
limit	int | None	None	No	Max events to return. None returns all.
newest_first	bool	True	No	Sort order.
include_messenger	bool	False	No	Include messenger reference in output.

Returns: list[dict]

get_metrics_snapshot()

Return a copy of the current command execution metrics.

Returns: dict[str, int] -- Template names mapped to execution counts.

get_context(*, history_limit=10, include_metrics=True, include_history=True, newest_first=True, include_messenger=False)

Return a combined snapshot of metrics and history, useful for publishing as context.

Parameter	Type	Default	Required	Description
history_limit	int | None	10	No	Max events in the history slice.
include_metrics	bool	True	No	Include metrics in the output.
include_history	bool	True	No	Include event history in the output.
newest_first	bool	True	No	Sort order for events.
include_messenger	bool	False	No	Include messenger references.

Returns: dict -- Contains metrics and/or history keys.

set_messenger(messenger)

Register a default AgentMessenger used when no per-command messenger is provided.

Parameter	Type	Required	Description
messenger	AgentMessenger | None	Yes	The messenger instance, or None to clear.

Returns: None

create_question_waiter()

Create and track a PendingQuestion that can be resolved later via receive_agent_reply() .

Returns: PendingQuestion

receive_agent_reply(reply)

Resolve a pending question from an external reply payload.

Parameter	Type	Required	Description
reply	dict	Yes	Reply payload containing a correlationId key matching a pending question.

Returns: bool -- True if a pending question was resolved, False otherwise.

Troubleshooting OmniLinkEngine

Command returns {"ok": false} --
This means no registered template matched the input. Run engine.parse("your command") to see the errors list, which reports which templates were tried and why they failed.
Matching is case-insensitive but whitespace-sensitive after normalisation.

Variable not extracted --
Ensure the variable syntax is correct: [name:type] .
If using a custom type, verify it is registered in the TypeRegistry before constructing the engine. Call engine.types.available() to list all registered types.

4. TypeRegistry

Maps type names to regex patterns and optional converter functions for template variable extraction. A default registry with all built-in types is created automatically when OmniLinkEngine is instantiated without a custom registry.

```
class omnilink.TypeRegistry()
```

register(name, regex, converter=None)

Register a new named type.

Parameter	Type	Default	Required	Description
name	str	--	Yes	Type name used in templates as [var:name] .
regex	str	--	Yes	Regex pattern to match values of this type.
converter	Callable | None	None	No	Optional function to convert the matched string (e.g. int , float).

Returns: None

Sample Usage:

```
from omnilink import TypeRegistry

types = TypeRegistry()
types.register("room", r"(?:living_room|kitchen|bedroom|office)")
types.register("speed", r"\d+(?:\.\d+)?", float)
types.register("color", r"(?:red|green|blue|yellow)")
```

get(name)

Retrieve a type specification by name.

Parameter	Type	Required	Description
name	str	Yes	The type name to look up.

Returns: tuple[str, Callable | None] | None -- The (regex, converter) pair, or None if not found.

available()

Return all registered types as a dictionary.

Returns: dict[str, str] -- Type names mapped to their regex patterns.

Built-in Types

Type Name	Pattern	Conversion	Description
int	Integer digits	int()	Matches whole numbers (e.g. 42 , -7).
float	Decimal number	float()	Matches decimals (e.g. 3.14 , -0.5).
num / digit / digits	Numeric digits	Auto	Matches numeric strings.
any	One or more words (greedy)	String	Captures everything remaining in the input.
alpha / letters	Letters only	String	Matches alphabetic characters.
word	Single word	String	Matches one whitespace-delimited word.
lower	Lowercase letters	String	Matches lowercase-only strings.
upper	Uppercase letters	String	Matches uppercase-only strings.
slug	Slug format	String	Matches URL-slug-style strings (letters, digits, hyphens).
alnum / alphanumeric	Letters + digits	String	Matches alphanumeric strings.
bool	Boolean	Auto	Matches true / false , yes / no , on / off , 1 / 0 .
uuid	UUID format	String	Matches UUID strings (e.g. 550e8400-e29b-41d4-a716-446655440000).
date	Date format	String	Matches date strings (e.g. 2026-04-04).
time	Time format	String	Matches time strings (e.g. 14:30:00).
datetime	DateTime format	String	Matches ISO datetime strings.

Template Variable Syntax

Syntax	Example	Description
[name]	[vehicle]	Captures a single word into variable vehicle .
[name:type]	[speed:float]	Captures and converts using the named type.
[:type]	[:int]	Captures with type but no named variable.
[name:any]	[message:any]	Greedy capture of one or more words.
[name:/regex/]	[code:/[A-Z]{3}/]	Captures using a custom inline regex.

Sample Usage:

```python
from omnilink import OmniLinkEngine, TypeRegistry

# Custom types
types = TypeRegistry()
types.register("room", r"(?:living_room|kitchen|bedroom|office)")

engine = OmniLinkEngine([
    "turn [state:bool] the lights in [room:room]",
    "set temperature to [temp:float] in [room:room]",
], types=types)

engine.on_template(
    "turn [state:bool] the lights in [room:room]",
    lambda e: {"lights": e["vars"]["state"], "room": e["vars"]["room"]}
)

result = engine.handle("turn on the lights in kitchen")
# {"ok": True, "vars": {"state": True, "room": "kitchen"}, ...}
```

5. OmniLinkClient

Full synchronous REST API client for the OmniLink platform. All methods are thread-safe. Provides access to chat, agent profiles, short-term memory, speech-to-text, text-to-speech, and translation.

```
class omnilink.client.OmniLinkClient(omni_key, base_url=None, *, timeout=30)
```

Constructor

Parameter	Type	Default	Required	Description
omni_key	str	--	Yes	Your Omni Key for authentication (starts with olink_).
base_url	str	"https://www.omnilink-agents.com"	No	Root URL of the OmniLink deployment.
timeout	int	30	No	HTTP request timeout in seconds.

Raises:

- ValueError -- If omni_key is empty or not a string.
- ImportError -- If the requests package is not installed.

Sample Usage:

```
from omnilink.client import OmniLinkClient

client = OmniLinkClient(omni_key="olink_YOUR_KEY")
```

chat(prompt=None, *, agent_name, messages=None, engine="g2-engine", temperature=None, max_tokens=None, system_instruction=None, system_instruction_suffix=None, use_prompt_pipeline=False, **extra)

Send a chat message to an AI engine through the OmniLink platform.

Parameter	Type	Default	Required	Description
prompt	str | None	None	No*	Shorthand for a single user message. Either prompt or messages must be provided.
agent_name	str	--	Yes	Target agent name on the platform.
messages	list[dict] | None	None	No*	OpenAI-style message list. Takes precedence over prompt if both provided.
engine	str	"g2-engine"	No	AI engine: "g1-engine" (Gemini), "g2-engine" (GPT), "g3-engine" (Grok), "g4-engine" (Claude).

Parameter	Type	Default	Required	Description
temperature	float | None	None	No	Sampling temperature. Higher values produce more creative responses.
max_tokens	int | None	None	No	Maximum tokens in the response.
system_instruction	dict | None	None	No	Structured system instruction object for the AI.
system_instruction_suffix	str | None	None	No	Text appended to the system instruction.
use_prompt_pipeline	bool	False	No	When True , the server composes system instructions from the agent context.
**extra	Any	--	No	Additional fields (e.g. agentPersona).

Returns: dict -- {"ok": True, "text": "...", "requestId": "..."}

Raises:

- ValueError -- If neither prompt nor messages is provided.
- OmniLinkAPIError -- On non-2xx HTTP response.
- requests.RequestException -- On network failure.

Sample Usage:

```
# Example 1: Simple prompt
reply = client.chat("What is 2 + 2?", agent_name="math-tutor")
print(reply["text"])

# Example 2: With messages and custom engine
reply = client.chat(
    agent_name="assistant",
    engine="g4-engine",
    temperature=0.3,
    messages=[
        {"role": "user", "content": "Summarize quantum computing"},
    ],
)
print(reply["text"])
```

list_profiles()

Retrieve all agent profiles for the authenticated user.

Returns: list[dict] -- Each dict contains id , name , settings , appearance , plan_locked , updated_at .

Raises: OmniLinkAPIError on non-2xx response.

Sample Usage:

```
profiles = client.list_profiles()
for p in profiles:
    print(f"{p['name']} (id: {p['id']})")
```

create_profile(name, *, settings=None, appearance=None)

Create a new agent profile on the platform.

Parameter	Type	Default	Required	Description
name	str	--	Yes	Display name for the profile (e.g. "door-controller").
settings	dict | None	None	No	JSONB settings object with fields like agentName , mainTask , allowToolUse , availableTools , availableToolDetails , availableCommands .
appearance	dict | None	None	No	JSONB appearance object for UI customization.

Returns: dict -- Created profile with id , name , settings , appearance , plan_locked , updated_at .

Sample Usage:

```
profile = client.create_profile("my-agent", settings={
    "agentName": "my-agent",
    "mainTask": "You are a helpful assistant.",
    "allowToolUse": True,
    "availableTools": "make_move",
    "availableToolDetails": [
        {"name": "make_move", "description": "Execute the next action."}
    ],
    "availableCommands": "stop_game, pause_game, resume_game",
})
print(f"Created profile: {profile['id']}")
```

update_profile(profile_id, *, name=None, settings=None, appearance=None)

Update an existing agent profile.

Parameter	Type	Default	Required	Description
profile_id	str	--	Yes	UUID of the profile to update.
name	str | None	None	No	New display name (omit to keep current).
settings	dict | None	None	No	New settings object (omit to keep current).
appearance	dict | None	None	No	New appearance object (omit to keep current).

Returns: dict -- Updated profile dict.

Raises: OmniLinkAPIError on non-2xx response.

Sample Usage:

```
client.update_profile(profile["id"], settings={
    "mainTask": "Updated instructions for the agent.",
})
```

delete_profile(profile_id)

Delete an agent profile from the platform.

Parameter	Type	Required	Description
profile_id	str	Yes	UUID of the profile to delete.

Returns: str -- ID of the deleted profile.

Sample Usage:

```
deleted_id = client.delete_profile(profile["id"])
print(f"Deleted: {deleted_id}")
```

list_agents()

Retrieve all agents and their memory status.

Returns: dict -- {"agents": [...], "memoryAgents": [...]} . Each agent has id , name , updated_at , and hasMemory boolean.

Sample Usage:

```
data = client.list_agents()
for agent in data["agents"]:
    print(f"{agent['name']} - has memory: {agent['hasMemory']}")
```

get_memory(agent_name)

Retrieve the short-term conversation memory for an agent.

Parameter	Type	Required	Description
agent_name	str	Yes	Agent name or profile ID.

Returns: list[dict] | None -- List of conversation entries, or None if no memory exists.

Sample Usage:

```
memory = client.get_memory("my-agent")
if memory:
    for entry in memory:
        print(f"[{entry['role']}] {entry['parts'][0]['text'][:80]}")
```

set_memory(agent_name, conversation)

Write short-term conversation memory for an agent. Overwrites any existing memory.

Parameter	Type	Required	Description
agent_name	str	Yes	Target agent name.
conversation	list[dict]	Yes	List of entries. Each entry has "role" ("user" or "model") and "parts" (list of {"text": "..."} dicts).

Returns: list[dict] -- The sanitized conversation that was persisted.

Sample Usage:

```
client.set_memory("my-agent", [
    {"role": "user",  "parts": [{"text": "What is the current score?"}]},
    {"role": "model", "parts": [{"text": "The current score is 42."}]},
])
```

clear_memory(agent_name)

Clear all short-term memory for an agent by writing an empty conversation list.

Parameter	Type	Required	Description
agent_name	str	Yes	Target agent name.

Returns: None

Sample Usage:

```
client.clear_memory("my-agent")
```

transcribe(audio, *, mime_type="audio/webm", language="", duration_sec=0.0)

Transcribe audio to text using the OmniLink speech-to-text service (Whisper).

Parameter	Type	Default	Required	Description
audio	bytes	--	Yes	Raw audio bytes.
mime_type	str	"audio/webm"	No	Audio MIME type. Supported: "audio/webm" , "audio/mp4" , "audio/wav" , "audio/mpeg" .
language	str	""	No	ISO-639-1 language hint (e.g. "en" , "fr"). Empty string for auto-detection.
duration_sec	float	0.0	No	Audio duration in seconds (for tracking/billing only).

Returns: dict -- Typically {"text": "transcribed text..."} .

Raises: OmniLinkAPIError on non-2xx response.

Sample Usage:

```
with open("recording.webm", "rb") as f:
    result = client.transcribe(f.read(), mime_type="audio/webm", language="en")
print(result["text"])
```

synthesize(text, *, language_code="en-US", voice_name=None, audio_encoding="MP3", speaking_rate=1.0, sample_rate_hertz=None)

Synthesize text to speech audio.

Parameter	Type	Default	Required	Description
text	str	--	Yes	The text to synthesize into speech.
language_code	str	"en-US"	No	BCP-47 language code (e.g. "en-US" , "fr-FR" , "de-DE").
voice_name	str | None	None	No	Chirp3-HD voice name. When None , the server selects the default voice for the language.
audio_encoding	str	"MP3"	No	Output format: "MP3" , "LINEAR16" , "OGG_OPUS" , "MULAW" , or "ALAW" .
speaking_rate	float	1.0	No	Speed multiplier. 1.0 is normal speed; 0.5 is half speed; 2.0 is double speed.
sample_rate_hertz	int | None	None	No	Output sample rate in Hz. Omit to use the encoder default.

Returns: dict -- {"audioContent": "<base64>", "audioMimeType": "audio/mpeg", "meta": {...}} .

Raises: OmniLinkAPIError on non-2xx response.

Sample Usage:

```
result = client.synthesize(
    "Hello from OmniLink!",
    language_code="en-US",
    audio_encoding="MP3",
    speaking_rate=1.2,
)
import base64
audio_bytes = base64.b64decode(result["audioContent"])
```

synthesize_to_bytes(text, **kwargs)

Convenience wrapper that synthesizes text and returns raw audio bytes directly.

Parameter	Type	Default	Required	Description
text	str	--	Yes	The text to synthesize.
**kwargs	Any	--	No	All keyword arguments are forwarded to synthesize() .

Returns: bytes -- Raw audio bytes (MP3 by default).

Raises: ValueError if the server returns no audioContent field.

Sample Usage:

```
audio = client.synthesize_to_bytes("Hello from OmniLink!")
with open("output.mp3", "wb") as f:
    f.write(audio)
```

translate(text, *, target_language="English", source_language=None, model="gpt-5", max_tokens=None)

Translate text between languages using the OmniLink translation service.

Parameter	Type	Default	Required	Description
text	str	--	Yes	The text to translate.
target_language	str	"English"	No	Target language name or code (e.g. "French" , "Spanish" , "ar").
source_language	str | None	None	No	Source language. Omit for automatic detection.
model	str	"gpt-5"	No	Underlying model for translation.
max_tokens	int | None	None	No	Upper bound on translation length.

Returns: dict -- {"translation": "translated text"} .

Raises: OmniLinkAPIError on non-2xx response.

Sample Usage:

```
# Example 1: Auto-detect source language
result = client.translate("Bonjour le monde", target_language="English")
print(result["translation"])  # "Hello world"

# Example 2: Specify source language
result = client.translate(
    "Hello world",
    source_language="English",
    target_language="Japanese",
)
print(result["translation"])
```

get_usage(limit=100)

Return usage events and credit rollup for the current Omni Key.

Parameter	Type	Default	Required	Description
limit	int	100	No	Maximum number of recent usage events to return (1-500).

Returns: dict -- {"omniKey": {...}, "rollup": {"total_credits": ...}, "events": [...]} .

Raises: OmniLinkAPIError on non-2xx response.

Sample Usage:

```
usage = client.get_usage(limit=10)
print(f"Total credits: {usage['rollup']['total_credits']}")
for event in usage["events"]:
    print(f"  {event['engine']}: {event['input_units']} in / {event['output_units']} out")
```

Chat response -- engine fallback fields (new in v0.4.3)

When a chat request is served by a fallback engine (e.g., the requested
engine was rate-limited), the response includes additional metadata:

- engine -- the engine that actually served the request.
- fallbackFrom -- the originally requested engine that failed (present only on fallback).
- attempts -- list of {"engine": "...", "error": "..."} for each engine tried (present only on fallback).

```
resp = client.chat(prompt="Hi", agent_name="demo", engine="g4-engine")
if resp.get("fallbackFrom"):
    print(f"Fell back from {resp['fallbackFrom']} to {resp['engine']}")
```

Troubleshooting OmniLinkClient

OmniLinkAPIError with status 401 --
Your Omni Key is invalid or expired. Generate a new key at https://www.omnilink-agents.com/omnilink-api .

OmniLinkAPIError with status 429 --
You have exceeded the rate limit. Wait a few seconds and retry.
For high-throughput applications, add a small delay between calls
or implement exponential backoff.

Timeout errors on chat() --
Large prompts or slow engines may exceed the default 30-second timeout.
Increase it when constructing the client: OmniLinkClient(omni_key="...", timeout=120) .

OmniLinkAPIError with status 400 --
The request body is malformed. Common causes: passing prompt as None without providing messages , or passing
an invalid engine name. Valid engines are "g1-engine" , "g2-engine" , "g3-engine" , and "g4-engine" .

6. OmniLinkChatClient

Lightweight HTTP client for the OmniLink /api/chat endpoint only.
Use this when you only need chat functionality without the full REST client.

```
class omnilink.OmniLinkChatClient(base_url, omni_key, *, timeout=30)
```

Constructor

Parameter	Type	Default	Required	Description
base_url	str	--	Yes	Root URL of the OmniLink deployment (no trailing slash).
omni_key	str	--	Yes	Bearer token for authentication.
timeout	int	30	No	HTTP timeout in seconds.

Raises: ImportError if requests is not installed.

chat(prompt=None, *, agent_name, messages=None, engine="g2-engine", temperature=None, max_tokens=None, system_instruction_request=None, system_instruction_suffix=None, use_prompt_pipeline=False, **extra)

Send a chat request. Either prompt or messages must be provided.

Parameter	Type	Default	Required	Description
prompt	str | None	None	No*	Shorthand for a single user message.
agent_name	str	--	Yes	Target agent name.
messages	list[dict] | None	None	No*	OpenAI-style message list.
engine	str	"g2-engine"	No	AI engine to use.
temperature	float | None	None	No	Sampling temperature.
max_tokens	int | None	None	No	Token limit.
system_instruction_request	dict | None	None	No	Structured system instruction object.
system_instruction_suffix	str | None	None	No	Text appended to the system instruction.
use_prompt_pipeline	bool	False	No	Let the server compose system instructions from agent context.
**extra	Any	--	No	Additional payload fields.

Returns: dict -- {"ok": True, "text": "...", "requestId": "..."} .

Raises:

- requests.RequestException on network failure.
- RuntimeError on non-2xx status code.

Sample Usage:

```python
from omnilink import OmniLinkChatClient

chat = OmniLinkChatClient(
    base_url="https://www.omnilink-agents.com",
    omni_key="olink_YOUR_KEY",
)
reply = chat.chat("Hello!", agent_name="assistant")
print(reply["text"])
```

7. OmniLinkHTTPBridge

Turns an OmniLinkEngine into a local HTTP REST server with
no external broker required. Exposes endpoints for command submission,
feedback retrieval, context publishing, and inline code execution.

```
class omnilink.OmniLinkHTTPBridge(engine, host="0.0.0.0", port=5000, log=True, *, inline_code_handler=None)
```

Constructor

Parameter	Type	Default	Required	Description
engine	OmniLinkEngine	--	Yes	The engine instance to expose over HTTP.
host	str	"0.0.0.0"	No	Bind address. Use "0.0.0.0" for all interfaces, "127.0.0.1" for localhost only.
port	int	5000	No	Bind port number.
log	bool	True	No	Enable HTTP request logging to stdout.
inline_code_handler	Callable | None	None	No	Handler function for POST /inline-code requests. Receives an InlineCodeMessage object.

Endpoints

Method	Path	Request Body	Description
POST	/command	{"command": "..."}	Submit a command for the engine to process. Returns the handler result.
GET	/feedback	--	Retrieve the latest feedback from the last processed command.
GET	/context	--	Retrieve the latest published context or state snapshot.
POST	/inline-code	{"command": "...", "snippets": [...]}	Submit inline code snippets for execution (requires inline_code_handler).

start()

Start the HTTP server in a background thread. Non-blocking -- returns immediately.

Returns: OmniLinkHTTPBridge -- Returns self for chaining.

Sample Usage:

```
bridge = OmniLinkHTTPBridge(engine, port=8080)
bridge.start()
# ... continue with other work ...
bridge.stop()
```

loop_forever()

Start the HTTP server and block until KeyboardInterrupt (Ctrl+C). Ideal for standalone scripts.

Returns: None

Sample Usage:

```
from omnilink import OmniLinkEngine, OmniLinkHTTPBridge

engine = OmniLinkEngine(["launch [vehicle]"])
engine.on_template("launch [vehicle]", handle_launch)

bridge = OmniLinkHTTPBridge(engine, host="0.0.0.0", port=8080)
bridge.loop_forever()  # Blocks until Ctrl+C
```

stop()

Shutdown the HTTP server gracefully.

Returns: None

publish_context(context_str)

Publish a context string that will be served by GET /context .

Parameter	Type	Required	Description
context_str	str	Yes	The context text to publish.

Returns: None

publish_state_snapshot(*, history_limit=None, include_metrics=True, include_history=True, include_messenger=False)

Publish a snapshot of the engine's metrics and event history as the current context.

Parameter	Type	Default	Required	Description
history_limit	int | None	None	No	Max events to include in the snapshot.
include_metrics	bool	True	No	Include command execution metrics.
include_history	bool	True	No	Include event history.
include_messenger	bool	False	No	Include messenger references.

Returns: dict -- The snapshot that was published.

Environment Variables

Variable	Default	Description
HTTP_BRIDGE_HOST	0.0.0.0	Override bind address (takes precedence over constructor).
HTTP_BRIDGE_PORT	8080	Override bind port (takes precedence over constructor).

8. AgentFeedback & AgentQuestion

Structured messaging objects for bidirectional handler communication.
Used within command handlers to send progress updates, ask confirmation questions, and receive operator replies.

AgentFeedback

```
from omnilink import AgentFeedback
```

Field	Type	Default	Required	Description
message	str	--	Yes	The feedback text to display.
kind	str	"info"	No	Feedback level: "info" , "success" , "warning" , or "error" .
ok	bool | None	None	No	Whether the operation succeeded. None means no explicit status.
progress	float | None	None	No	Progress indicator from 0.0 to 1.0 . None means no progress tracking.
data	dict | None	None	No	Optional structured data payload.

Methods

to_payload(*, command=None) -- Create a JSON-serializable payload dict with optional command context.

to_legacy_payload() -- Create a legacy feedback format dict for backward compatibility.

AgentQuestion

```
from omnilink import AgentQuestion
```

Field	Type	Default	Required	Description
prompt	str	--	Yes	The question text to display to the operator.
choices	list[str] | None	None	No	Optional list of choices (e.g. ["yes", "no"]).
allow_multiple	bool	False	No	Allow selecting multiple choices.
inline_code	InlineCodeMessage | None	None	No	Optional inline code attached to the question.
data	dict | None	None	No	Optional structured data payload.

Methods

to_payload(*, command=None) -- Create a JSON-serializable payload dict.

PendingQuestion

Represents a question that has been sent and is awaiting an external reply.

wait(timeout=None, *, cancel_on_timeout=False, cancel_message=None)

Block until the answer is received or the timeout expires.

Parameter	Type	Default	Required	Description
timeout	float | None	None	No	Maximum seconds to wait. None for indefinite.
cancel_on_timeout	bool	False	No	Automatically cancel the question if it times out.
cancel_message	str | None	None	No	Message sent if the question is auto-cancelled.

Returns: dict -- The reply payload.

Raises: TimeoutError if the timeout expires and cancel_on_timeout is False .

cancel(message=None)

Cancel the question so that any late reply is ignored.

Parameter	Type	Required	Description
message	str | None	No	Optional cancellation message.

done()

Check whether a reply or cancellation has been recorded.

Returns: bool

result()

Get the reply payload if received.

Returns: dict | None

AgentMessenger (Protocol)

Interface for emitting structured agent updates from within handlers.

Method	Description
send_feedback(feedback: AgentFeedback)	Emit a feedback update.
ask_question(question: AgentQuestion) -> PendingQuestion	Ask a question and get a pending object for the reply.
acknowledge(status, *, message=None, data=None)	Send a final acknowledgement.

Complete Example

```python
from omnilink import AgentFeedback, AgentQuestion

def handle_deploy(evt):
    messenger = evt.get("messenger")
    if not messenger:
        return {"result": "deployed"}

    # Step 1: Send progress feedback
    messenger.send_feedback(AgentFeedback(
        message="Building container image...",
        kind="info",
        progress=0.25,
        ok=True,
    ))

    # Step 2: Ask for confirmation before deploying
    pending = messenger.ask_question(AgentQuestion(
        prompt="Deploy to production?",
        choices=["yes", "no"],
    ))
    reply = pending.wait(timeout=60, cancel_on_timeout=True)

    if reply.get("answer") != "yes":
        messenger.acknowledge("cancelled", message="Deployment cancelled by operator.")
        return {"result": "cancelled"}

    # Step 3: Deploy and send final feedback
    messenger.send_feedback(AgentFeedback(
        message="Deploying to production...",
        kind="info",
        progress=0.75,
    ))

    # ... perform deployment ...

    messenger.acknowledge("complete", message="Deployment successful.")
    return {"result": "deployed"}
```

InlineCodeSnippet

Represents a single labelled code snippet submitted via the POST /inline-code endpoint.

Field	Type	Description
label	str	A short label describing the snippet (e.g. "Setup" , "Main Loop").
content	str	The raw code text.

Class method: InlineCodeSnippet.from_payload(payload) -- Create an instance from a raw dict or string payload. Returns None if the payload is invalid.

InlineCodeMessage

A collection of code snippets submitted together with an associated command. Used with the inline_code_handler parameter of OmniLinkHTTPBridge .

Field	Type	Default	Description
command	str	--	The command string associated with the snippets.
snippets	list[InlineCodeSnippet]	--	Ordered list of code snippets.
response	str | None	None	Optional response text set by the handler.
timestamp	str | None	None	ISO timestamp of when the message was received.

Methods

from_payload(cls, payload) -- Class method. Create an instance from a raw dict payload. Returns None if the payload is invalid.

to_script(*, comment_prefix="# ") -- Combine all snippets into a single executable script string with label comments separating each snippet.

to_payload() -- Serialize back to a JSON-compatible dict.

Sample Usage:

```
from omnilink import OmniLinkHTTPBridge, OmniLinkEngine

def handle_code(msg):
    print(f"Received {len(msg.snippets)} snippets for: {msg.command}")
    script = msg.to_script()
    exec(script)  # execute the combined code

engine = OmniLinkEngine(["run code"])
bridge = OmniLinkHTTPBridge(engine, inline_code_handler=handle_code)
bridge.loop_forever()
```

Event

Represents a single processed command in the engine history. Events are stored in engine.history and returned by get_recent_events() .

Field	Type	Description
command	str	The original command text.
text	str	The normalised command text.
template	str | None	The matched template, or None if no match.
normalized_template	str | None	The normalised version of the matched template.
vars	dict	Extracted variables with type-converted values.
errors	list[dict]	List of match-failure details (empty on success).
meta	dict	Metadata passed via handle(text, meta=...) .
timestamp	float	Unix timestamp of when the event was processed.
messenger	AgentMessenger | None	The messenger used for this command, if any.

9. Utility Functions

give_context(context_str)

Publish a context string via the active OmniLinkHTTPBridge . Useful for pushing state updates from anywhere in your code.

Parameter	Type	Required	Description
context_str	str	Yes	The context text to publish.

Returns: None

Raises: RuntimeError if no HTTP bridge is currently active.

Sample Usage:

```
from omnilink import give_context

give_context("Robot is at position (3, 7). Battery: 85%.")
```

start_periodic_context(interval_seconds, message)

Start a background thread that calls give_context() at a regular interval. If called again, the previous thread is stopped and replaced.

Parameter	Type	Required	Description
interval_seconds	float	Yes	Seconds between context publishes.
message	str | Callable[[], str]	Yes	Static string or zero-argument callable that returns a string.

Returns: None

Raises: RuntimeError if no HTTP bridge is active.

Sample Usage:

```
from omnilink import start_periodic_context

# Static message
start_periodic_context(10.0, "System healthy.")

# Dynamic message via callable
start_periodic_context(5.0, lambda: f"CPU: {get_cpu()}%, Memory: {get_mem()}%")
```

stop_periodic_context()

Stop the periodic context publishing thread. Safe to call even if no thread is running.

Returns: None

load_patterns_from_file(path, types=None)

Load command template patterns from a text file. Lines starting with # are treated as comments and blank lines are skipped. Quoted lines are automatically unquoted.

Parameter	Type	Default	Required	Description
path	str | Path	--	Yes	Path to the patterns file. Relative paths are resolved against the caller's directory.
types	TypeRegistry | None	None	No	Optional type registry for validation.

Returns: list[str] -- List of template strings.

Raises: FileNotFoundError if the file does not exist.

Sample Usage:

```
from omnilink import load_patterns_from_file, OmniLinkEngine

# patterns.txt:
# move [direction] [distance:int] meters
# set speed to [speed:float]
# hello

patterns = load_patterns_from_file("patterns.txt")
engine = OmniLinkEngine(patterns)
```

OmniLinkAPIError

Exception raised when the OmniLink API returns a non-2xx HTTP response.

```
class omnilink.client.OmniLinkAPIError(status_code, body)
```

Attribute	Type	Description
status_code	int	The HTTP status code returned by the server.
body	dict	Parsed JSON error body. Falls back to {"raw": ...} if the response is not valid JSON.

Sample Usage:

```
from omnilink.client import OmniLinkClient, OmniLinkAPIError

client = OmniLinkClient(omni_key="olink_YOUR_KEY")

try:
    reply = client.chat("Hello", agent_name="my-agent")
except OmniLinkAPIError as e:
    print(f"API error {e.status_code}: {e.body}")
except Exception as e:
    print(f"Network error: {e}")
```

10. Benchmark Examples

The library ships with game and robotics benchmarks demonstrating the ToolRunner pattern. Each follows a consistent three-file architecture:

- play_*.py -- ToolRunner subclass (the agent entry point)
- *_api.py -- HTTP client for the game/system server
- *_engine.py -- Pure local AI logic (no network calls)

Game Benchmarks

S.No.	Example	Agent Name	AI Strategy	Query Tools
1.	Pac-Man	pacman-agent	BFS pathfinding with ghost avoidance and dead-end detection	get_score, get_position, get_ghosts, get_pellets
2.	Chess	chess-agent	Minimax with alpha-beta pruning and piece-square tables	get_position, get_material, get_moves, get_status
3.	Tetris	tetris-agent	Pierre Dellacherie evaluation with macro action batching	get_score, get_piece, get_board, get_status
4.	Breakout	breakout-agent	Ball trajectory simulation with brick collision physics	get_score, get_ball, get_paddle, get_bricks
5.	Pong	pong-agent	Full trajectory simulation with opponent AI modelling	get_score, get_ball, get_paddles
6.	Space Invaders	space-invaders-agent	Lead-shot targeting with stateful direction tracking	get_score, get_ship, get_aliens, get_status
7.	Montezuma	montezuma-agent	BFS pathfinding with enemy avoidance radius	get_score, get_player, get_status
8.	Go	go-agent	Heuristic evaluation with capture priority and territory estimation	get_score, get_moves, get_board_state
9.	Asteroids	asteroids-agent	Multi-tool strategic targeting	get_score, get_ship, get_asteroids, get_status

Robotics Benchmarks

S.No.	Example	Agent Name	Description	Query Tools
1.	Husky	husky-agent	Single robot waypoint navigation in PyBullet simulation	get_pose, get_waypoint, get_progress
2.	Husky Fleet	husky-fleet-agent	5-robot fleet coordination with task assignment	get_fleet_status, get_robot, get_all_positions

Running a Benchmark

```
# Step 1: Start the game server (from omnilink-benchmarks/)
python -m omnilink_benchmarks.pacman.server  # Port 5000

# Step 2: Start the agent (requires OMNI_KEY)
export OMNI_KEY="olink_YOUR_KEY"
python -m omnilink.examples.pacman.play_pacman

# Step 3: Open the OmniLink web UI
# Navigate to https://www.omnilink-agents.com
# Select your agent profile from the dropdown
```

All Benchmark Commands

```
python -m omnilink.examples.pacman.play_pacman
python -m omnilink.examples.chess.play_chess
python -m omnilink.examples.tetris.play_tetris
python -m omnilink.examples.breakout.play_breakout
python -m omnilink.examples.pong.play_pong
python -m omnilink.examples.space_invaders.play_space_invaders
python -m omnilink.examples.montezuma.play_montezuma
python -m omnilink.examples.go.play_go
python -m omnilink.examples.asteroids.play_asteroids
python -m omnilink.examples.husky.play_husky
python -m omnilink.examples.husky_fleet.play_fleet
```

UI-Controlled Robot Demo

The robot demo uses a different architecture -- the user controls the robot
through the OmniLink web UI instead of the AI running autonomously.
It showcases both commands (actions that change simulation state)
and tools (read-only queries) working together through the platform.

Running the Demo

```
# Terminal 1: Start the Pygame simulation (port 5050)
python -m omnilink.examples.robot_demo.robot_sim

# Terminal 2: Start the UI bridge
OMNI_KEY=olink_... python -m omnilink.examples.robot_demo.run_demo
```

Then open https://www.omnilink-agents.com ,
select the robot-demo agent, and start chatting. The AI translates
your natural language into the appropriate command or tool call automatically.

Commands vs. Tools

The robot demo has two distinct execution paths. You do not need to choose which to use -- the AI decides based on your message.

Commands change simulation state. When the AI outputs a Command: tag (e.g. Command: move_forward), the bridge process (run_demo.py) polls platform memory, finds the tag, and sends the action to the simulation via POST /callback . Commands have a latency of ~3 seconds (the polling interval).

10 Commands: move_forward, move_backward, turn_left, turn_right, scan_area, pick_up, drop_item, report_status, set_speed_to_[speed], go_to_[x]_[y]

Tools are read-only queries. When the AI outputs a Tool: tag (e.g. Tool: get_position), the OmniLink web UI sends a direct HTTP request to the simulation's /tool endpoint via the toolCallbackUrl . Tools execute instantly with no polling delay.

10 Tools: get_position, get_battery, get_inventory, get_obstacles, get_items, calculate_distance, find_path, check_collision, get_map_info, analyze_surroundings

	Commands	Tools
Purpose	Change simulation state	Read simulation state
AI output	Command: move_forward	Tool: get_position
Executed by	Bridge (run_demo.py)	Browser UI (direct HTTP)
Endpoint	POST /callback	POST /tool
Latency	~3s (polling)	Instant

The toolCallbackUrl

The toolCallbackUrl is the key setting that enables real-time tool execution. When the bridge starts, it registers this URL in the agent profile:

```
# In run_demo.py _build_instruction()
{
    "allowToolUse": True,
    "toolCallbackUrl": "http://127.0.0.1:5050/tool",
}
```

When the AI responds with a Tool: tag, the web UI POSTs directly to this URL from the browser, receives the result as JSON, passes it back to the AI for a human-readable summary, and displays that summary to the user. The server must return CORS headers (Access-Control-Allow-Origin: *) and accept POST with {"tool": "name", ...args} .

Controlling from Your Phone

Since the OmniLink web UI runs in any browser, you can control the robot from a phone or tablet. Open https://www.omnilink-agents.com on your mobile browser and select the robot-demo agent.

Commands work immediately from any device because they flow through
the OmniLink cloud (browser -> platform -> bridge -> sim). Tools require the browser to reach the simulation directly via toolCallbackUrl . The default http://127.0.0.1:5050/tool only works when the browser is on the same machine. For mobile access, use one of
these options:

Option A -- LAN IP (same Wi-Fi):

. Find your PC's local IP (ipconfig on Windows, ifconfig on macOS/Linux)
. Update toolCallbackUrl in run_demo.py to http://YOUR_IP:5050/tool
. Allow inbound connections on port 5050 through your firewall
. Restart the bridge

Option B -- ngrok tunnel (any network):

. Install ngrok: choco install ngrok (Windows) or brew install ngrok (macOS) or download from https://ngrok.com/download
. Sign up at https://dashboard.ngrok.com/signup and authenticate: ngrok config add-authtoken YOUR_TOKEN
. Start the tunnel: ngrok http 5050
. Copy the Forwarding URL (e.g. https://a1b2c3d4.ngrok-free.app)
. Update toolCallbackUrl in run_demo.py to https://a1b2c3d4.ngrok-free.app/tool
. Restart the bridge

```
# Complete 3-terminal setup with ngrok:

# Terminal 1 - Simulation
python -m omnilink.examples.robot_demo.robot_sim

# Terminal 2 - ngrok tunnel
ngrok http 5050
# Note the https://xxxx.ngrok-free.app URL

# Terminal 3 - Bridge (update toolCallbackUrl first!)
OMNI_KEY=olink_... python -m omnilink.examples.robot_demo.run_demo
```

ngrok tips: The free tier generates a new URL each restart
(update toolCallbackUrl accordingly). Visit http://127.0.0.1:4040 on your PC to inspect tunnel traffic.
The ngrok URL is publicly accessible -- stop it when done.

File Reference

File	Purpose
robot_sim.py	Pygame simulation + HTTP server (port 5050)
run_demo.py	Bridge -- connects the OmniLink platform to the local sim
robot_engine.py	Defines the 10 commands and 10 tools
robot_api.py	HTTP client wrappers for the sim server
robot_bridge.py	Alternative bridge (AI in-process, no UI)
play_robot.py	Autonomous player (AI controls robot without user)

11. Environment Variables

Variable	Default	Used By	Description
OMNI_KEY	--	ToolRunner , OmniLinkClient	Your Omni Key for authentication. Required for all platform features.
HTTP_BRIDGE_HOST	0.0.0.0	OmniLinkHTTPBridge	Override the HTTP bridge bind address.
HTTP_BRIDGE_PORT	8080	OmniLinkHTTPBridge	Override the HTTP bridge bind port.

12. Practical Examples & Recipes

End-to-end patterns that combine multiple parts of the library.
Each recipe is self-contained -- copy it, fill in your Omni Key,
and run.

Recipe 1 -- Chat Bot in 10 Lines

The fastest way to get a response from the OmniLink platform.
This uses OmniLinkClient.chat() with a simple prompt string.
The agent name determines which profile (and therefore which system
instruction) the AI uses. If the profile does not exist yet, the
platform creates a default one automatically.

```
from omnilink.client import OmniLinkClient
import os

client = OmniLinkClient(omni_key=os.environ["OMNI_KEY"])

while True:
    user_input = input("You: ")
    if user_input.lower() in ("quit", "exit"):
        break
    reply = client.chat(user_input, agent_name="my-chatbot")
    print(f"Bot: {reply['text']}")
```

Recipe 2 -- Command Engine with HTTP Bridge

Build a local REST API that accepts natural-language commands,
parses them into structured actions, and returns JSON results.
The bridge runs on 0.0.0.0:8080 by default and
accepts POST /command with a JSON body {"command": "your text here"} .

```
from omnilink import OmniLinkEngine, OmniLinkHTTPBridge

engine = OmniLinkEngine([
    "turn [state:bool] the [device]",
    "set [device] brightness to [level:int]",
    "what is the status of [device:any]",
])

engine.on_template("turn [state:bool] the [device]", lambda e: {
    "action": "power",
    "device": e["vars"]["device"],
    "state": e["vars"]["state"],
})

engine.on_template("set [device] brightness to [level:int]", lambda e: {
    "action": "brightness",
    "device": e["vars"]["device"],
    "level": e["vars"]["level"],
})

bridge = OmniLinkHTTPBridge(engine, port=8080)
bridge.loop_forever()

# Test with: curl -X POST http://localhost:8080/command \
#   -H "Content-Type: application/json" \
#   -d '{"command": "turn on the kitchen light"}'
```

Recipe 3 -- Multi-Engine Chat Comparison

Send the same prompt to all four AI engines and compare their responses side by side. Useful for evaluating which engine works best for your use case. Each engine has different strengths: g1-engine (Gemini) excels at factual knowledge, g2-engine (GPT) is a strong all-rounder, g3-engine (Grok) offers real-time information, and g4-engine (Claude) is strong at reasoning and code.

```
from omnilink.client import OmniLinkClient
import os

client = OmniLinkClient(omni_key=os.environ["OMNI_KEY"])
prompt = "Explain how a neural network learns in two sentences."

for engine in ["g1-engine", "g2-engine", "g3-engine", "g4-engine"]:
    reply = client.chat(prompt, agent_name="comparison", engine=engine)
    print(f"\n[{engine}]")
    print(reply["text"])
```

Recipe 4 -- Voice-Powered Agent

Combine speech-to-text, chat, and text-to-speech into a voice pipeline. Record audio, transcribe it, send the text to the AI, and synthesize the response back to audio. This pattern is the foundation for voice assistants and hands-free interfaces.

```python
from omnilink.client import OmniLinkClient
import os

client = OmniLinkClient(omni_key=os.environ["OMNI_KEY"])

# Step 1: Transcribe recorded audio
with open("question.webm", "rb") as f:
    transcript = client.transcribe(f.read(), mime_type="audio/webm")
print(f"You said: {transcript['text']}")

# Step 2: Send to AI
reply = client.chat(transcript["text"], agent_name="voice-assistant")
print(f"AI says: {reply['text']}")

# Step 3: Synthesize response to audio
audio = client.synthesize_to_bytes(reply["text"], language_code="en-US")
with open("response.mp3", "wb") as f:
    f.write(audio)
print("Saved response.mp3")
```

Recipe 5 -- Custom ToolRunner with Error Recovery

Production ToolRunners should handle network failures gracefully.
This pattern wraps get_state() in a retry loop and adds
connection-error handling to execute_action() . The on_start() hook waits for the game server to become available before entering the main loop.

```python
from omnilink.tool_runner import ToolRunner
import requests
import time

class RobustRunner(ToolRunner):
    agent_name = "robust-agent"
    display_name = "Robust Task"
    game_server_url = "http://127.0.0.1:5000"

    def get_state(self):
        """Fetch state with automatic retry on connection failure."""
        for attempt in range(5):
            try:
                return requests.get(f"{self.game_server_url}/state", timeout=5).json()
            except requests.ConnectionError:
                wait = 2 ** attempt
                print(f"  Connection failed, retrying in {wait}s...")
                time.sleep(wait)
        raise RuntimeError("Game server unreachable after 5 attempts")

    def execute_action(self, state):
        if state.get("game_state") != "PLAY":
            return
        action = self._decide(state)
        try:
            requests.post(
                f"{self.game_server_url}/callback",
                json={"action": action},
                timeout=5,
            )
        except requests.RequestException as e:
            print(f"  Action failed: {e}")

    def on_start(self):
        """Wait for the game server to become available."""
        print("Waiting for game server...")
        for _ in range(30):
            try:
                requests.get(f"{self.game_server_url}/state", timeout=2)
                print("Game server is ready.")
                return
            except requests.ConnectionError:
                time.sleep(1)
        raise RuntimeError("Game server did not start in time")

    def _decide(self, state):
        return "UP" if state.get("y", 0) < state.get("target_y", 0) else "DOWN"

    def state_summary(self, state):
        return f"Position: ({state.get('x', 0)}, {state.get('y', 0)})"

    def is_game_over(self, state):
        return state.get("done", False)

if __name__ == "__main__":
    RobustRunner().run()
```

Recipe 6 -- Agent Profile Lifecycle Management

Demonstrates the full create -> update -> list -> delete
lifecycle for agent profiles. Profiles control how the AI behaves in
the web UI: the system instruction, available tools, and commands are

all configured through the profile settings.

```python
from omnilink.client import OmniLinkClient
import os

client = OmniLinkClient(omni_key=os.environ["OMNI_KEY"])

# Create a profile
profile = client.create_profile("demo-agent", settings={
    "agentName": "demo-agent",
    "mainTask": "You are a helpful demo assistant.",
    "allowToolUse": False,
    "availableCommands": "help, status",
})
print(f"Created: {profile['id']}")

# Update the profile to enable tools
client.update_profile(profile["id"], settings={
    "allowToolUse": True,
    "availableTools": "get_info",
    "availableToolDetails": [
        {"name": "get_info", "description": "Retrieve system information."}
    ],
})
print("Updated: tools enabled")

# List all profiles
profiles = client.list_profiles()
print(f"Total profiles: {len(profiles)}")
for p in profiles:
    print(f"  - {p['name']} ({p['id'][:8]}...)")

# Clean up
client.delete_profile(profile["id"])
print(f"Deleted: {profile['id']}")
```

Recipe 7 -- Multilingual Translation Pipeline

Translate text through a chain of languages using the translate() method. This is useful for detecting translation drift or for building multilingual content pipelines.

```python
from omnilink.client import OmniLinkClient
import os

client = OmniLinkClient(omni_key=os.environ["OMNI_KEY"])

original = "The quick brown fox jumps over the lazy dog."
languages = ["French", "Japanese", "Arabic", "English"]

text = original
for lang in languages:
    result = client.translate(text, target_language=lang)
    text = result["translation"]
    print(f"[{lang}] {text}")

print(f"\nOriginal:    {original}")
print(f"Round-trip:  {text}")
```