DAP SDK Test Framework - Complete Guide

Async Testing, Mocking, and Test Automation

Cellframe Development Team

October 27, 2025

Contents

1 Document Information

Version: 1.0.0
Date: October 27, 2025
Status: Production Ready
Language: English

1.1 Revision History

Version	Date	Changes	Author
1.0.0	2025-10-27	Initial comprehensive guide	Cellframe Team

1.2 Copyright

This document describes the DAP SDK Test Framework, part of the Cellframe Network project.

1.3 License

2 Part I: Introduction

2.1 1. Overview

The DAP SDK Test Framework is a production-ready testing infrastructure designed for the Cellframe blockchain ecosystem. It provides comprehensive tools for testing asynchronous operations, mocking external dependencies, and ensuring reliable test execution across platforms.

2.1.1 1.1 What is DAP SDK Test Framework?

A complete testing solution that includes:

- **Async Testing Framework** - Tools for testing asynchronous operations with timeouts
- **Mock Framework V4** - Function mocking without code modification
- **Auto-Wrapper System** - Automatic linker configuration
- **Self-Tests** - 21 tests validating framework reliability

2.1.2 1.2 Why Use This Framework?

Problem: Testing asynchronous code is hard - Operations complete at unpredictable times - Network delays vary - Tests can hang indefinitely - External dependencies complicate testing

Solution: This framework provides - □ Timeout protection (global + per-operation) - □ Efficient waiting (polling + condition variables) - □ Dependency isolation (mocking) - □ Realistic simulation (delays, failures) - □ Thread-safe operations - □ Cross-platform support

2.1.3 1.3 Key Features at a Glance

Feature	Description	Benefit
Global Timeout	alarm + siglongjmp	Prevents CI/CD hangs
Condition Polling	Configurable intervals	Efficient async waiting
pthread Helpers	Condition variable wrappers	Thread-safe coordination
Mock Framework	Linker-based (`--wrap`)	Zero technical debt
Delays	Fixed, Range, Variance	Realistic simulation
Callbacks	Inline + Runtime	Dynamic mock behavior
Auto-Wrapper	Bash/PowerShell scripts	Automatic setup
Self-Tests	21 comprehensive tests	Validated reliability

2.1.4 1.4 Quick Comparison

Traditional Approach:

```
// □ Busy waiting, no timeout, CPU waste
while (!done) {
    usleep(10000);  // 10ms sleep
}
```

With DAP Test Framework:

```
// ✅ Efficient, timeout-protected, automatic logging
DAP_TEST_WAIT_UNTIL(done == true, 5000, "Should complete");
```

2.1.5 1.5 Target Audience

- DAP SDK developers
- Cellframe SDK contributors
- VPN Client developers
- Anyone testing async C code in Cellframe ecosystem

2.1.6 1.6 Prerequisites

Required Knowledge: - C programming - Basic understanding of async operations - CMake basics - pthread concepts (for advanced features)

Required Software: - GCC 7+ or Clang 10+ (or MinGW on Windows) - CMake 3.10+ - pthread library - Linux, macOS, or Windows (partial support)

2.2 2. Quick Start

2.2.1 2.1 Your First Test (5 minutes)

Step 1: Create test file

```c
// my_test.c
#include "dap_test.h"
#include "dap_common.h"

#define LOG_TAG "my_test"

int main() {
    dap_common_init("my_test", NULL);

    // Test code
    int result = 2 + 2;
    dap_assert_PIF(result == 4, "Math should work");

    log_it(L_INFO, "✓ Test passed!");

    dap_common_deinit();
    return 0;
}
```

Step 2: Create CMakeLists.txt

```
add_executable(my_test my_test.c)
target_link_libraries(my_test dap_core)
add_test(NAME my_test COMMAND my_test)
```

Step 3: Build and run

```
cd build
cmake ..
make my_test
./my_test
```

2.2.2 2.2 Adding Async Timeout (2 minutes)

```c
#include "dap_test.h"
#include "dap_test_async.h"
#include "dap_common.h"

#define LOG_TAG "my_test"
#define TIMEOUT_SEC 30

int main() {
    dap_common_init("my_test", NULL);

    // Add global timeout
    dap_test_global_timeout_t timeout;
    if (dap_test_set_global_timeout(&timeout, TIMEOUT_SEC, "My Test")) {
        return 1;  // Timeout triggered
    }
```

```
    // Your tests here

    dap_test_cancel_global_timeout();
    dap_common_deinit();
    return 0;
}
```

Update CMakeLists.txt:

```
target_link_libraries(my_test dap_test dap_core pthread)
```

2.2.3 2.3 Adding Mocks (5 minutes)

```
#include "dap_test.h"
#include "dap_mock_framework.h"
#include "dap_common.h"

#define LOG_TAG "my_test"

// Declare mock
DAP_MOCK_DECLARE(external_api_call);

int main() {
    dap_common_init("my_test", NULL);
    dap_mock_framework_init();

    // Configure mock
    DAP_MOCK_SET_RETURN(external_api_call, (void*)42);

    // Run code that calls external_api_call
    int result = my_code_under_test();

    // Verify
    assert(DAP_MOCK_GET_CALL_COUNT(external_api_call) == 1);

    dap_mock_framework_deinit();
    dap_common_deinit();
    return 0;
}
```

Update CMakeLists.txt:

```
include(${CMAKE_CURRENT_SOURCE_DIR}/../test-framework/mocks/DAPMockAutoWrap.cmak

target_link_libraries(my_test dap_test dap_test_mocks dap_core pthread)

dap_mock_autowrap(TARGET my_test SOURCE my_test.c)
```

2.3 3. API Reference

2.3.1 3.1 Async Testing API

2.3.1.1 Global Timeout

```
int dap_test_set_global_timeout(
    dap_test_global_timeout_t *a_timeout,
    uint32_t a_timeout_sec,
    const char *a_test_name
);
// Returns: 0 on setup, 1 if timeout triggered

void dap_test_cancel_global_timeout(void);
```

2.3.1.2 Condition Polling

```
bool dap_test_wait_condition(
    dap_test_condition_cb_t a_condition,
    void *a_user_data,
    const dap_test_async_config_t *a_config
);
// Returns: true if condition met, false on timeout
```

2.3.1.3 pthread Helpers

```
void dap_test_cond_wait_init(dap_test_cond_wait_ctx_t *a_ctx);
bool dap_test_cond_wait(dap_test_cond_wait_ctx_t *a_ctx, uint32_t a_timeout_ms);
void dap_test_cond_signal(dap_test_cond_wait_ctx_t *a_ctx);
void dap_test_cond_wait_deinit(dap_test_cond_wait_ctx_t *a_ctx);
```

2.3.1.4 Time Utilities

```
uint64_t dap_test_get_time_ms(void);  // Monotonic time in ms
void dap_test_sleep_ms(uint32_t a_delay_ms);  // Cross-platform sleep
```

2.3.1.5 Macros

```
DAP_TEST_WAIT_UNTIL(condition, timeout_ms, msg)
// Quick inline condition waiting
```

2.3.2 3.2 Mock Framework API

2.3.2.1 Declaration

```
DAP_MOCK_DECLARE(func_name);
DAP_MOCK_DECLARE(func_name, {.return_value.i = 42});
DAP_MOCK_DECLARE(func_name, {.return_value.i = 0}, { /* callback */ });
```

2.3.2.2 Control Macros

```
DAP_MOCK_ENABLE(func_name)
DAP_MOCK_DISABLE(func_name)
```

```
DAP_MOCK_RESET(func_name)
DAP_MOCK_SET_RETURN(func_name, value)
DAP_MOCK_GET_CALL_COUNT(func_name)
```

2.3.2.3 Delay Configuration

```
DAP_MOCK_SET_DELAY_FIXED(func_name, microseconds)
DAP_MOCK_SET_DELAY_FIXED_MS(func_name, milliseconds)
DAP_MOCK_SET_DELAY_RANGE(func_name, min_us, max_us)
DAP_MOCK_SET_DELAY_VARIANCE(func_name, center_us, variance_us)
DAP_MOCK_CLEAR_DELAY(func_name)
```

2.3.2.4 Callback Configuration

```
DAP_MOCK_SET_CALLBACK(func_name, callback_func, user_data)
DAP_MOCK_CLEAR_CALLBACK(func_name)
```

2.4 4. Complete Examples

2.4.1 4.1 State Machine Test

```c
#include "dap_test.h"
#include "dap_test_async.h"
#include "vpn_state_machine.h"

#define LOG_TAG "test_vpn_sm"
#define TIMEOUT_SEC 30

bool check_connected(void *data) {
    return vpn_sm_get_state((vpn_sm_t*)data) == VPN_STATE_CONNECTED;
}

void test_connection() {
    vpn_sm_t *sm = vpn_sm_init();
    vpn_sm_transition(sm, VPN_EVENT_USER_CONNECT);

    dap_test_async_config_t cfg = DAP_TEST_ASYNC_CONFIG_DEFAULT;
    cfg.timeout_ms = 10000;
    cfg.operation_name = "VPN connection";

    bool ok = dap_test_wait_condition(check_connected, sm, &cfg);
    dap_assert_PIF(ok, "Should connect within 10 sec");

    vpn_sm_deinit(sm);
}

int main() {
    dap_common_init("test_vpn_sm", NULL);

    dap_test_global_timeout_t timeout;
    if (dap_test_set_global_timeout(&timeout, TIMEOUT_SEC, "VPN Tests")) {
        return 1;
    }

    test_connection();

    dap_test_cancel_global_timeout();
    dap_common_deinit();
    return 0;
}
```

2.4.2 4.2 Mock with Callback

```c
#include "dap_mock_framework.h"

DAP_MOCK_DECLARE(dap_hash_fast, {.return_value.i = 0}, {
    if (a_arg_count >= 2) {
        uint8_t *data = (uint8_t*)a_args[0];
        size_t size = (size_t)a_args[1];
```

```
        uint32_t hash = 0;
        for (size_t i = 0; i < size; i++) {
            hash += data[i];
        }
        return (void*)(intptr_t)hash;
    }
    return (void*)0;
});

void test_hash() {
    uint8_t data[] = {1, 2, 3};
    uint32_t hash = dap_hash_fast(data, 3);
    assert(hash == 6);  // Callback sums bytes
}
```

2.4.3 4.3 Network Test with Retry

```
void test_http_with_retry() {
    const char *hosts[] = {"httpbin.org", "postman-echo.com", NULL};

    for (int i = 0; hosts[i]; i++) {
        http_ctx_t ctx = {0};
        http_request_async(hosts[i], &ctx);

        dap_test_async_config_t cfg = {
            .timeout_ms = 30000,
            .poll_interval_ms = 500,
            .fail_on_timeout = false,
            .operation_name = "HTTP request"
        };

        if (dap_test_wait_condition(check_complete, &ctx, &cfg)) {
            log_it(L_INFO, "✓ Host %s responded", hosts[i]);
            return;
        }
        log_it(L_WARNING, "Host %s failed, trying next", hosts[i]);
    }
}
```

2.5 5. Glossary

Async Operation - Operation completing at unpredictable future time

Auto-Wrapper - System auto-generating linker `--wrap` flags from source

Callback - Function pointer executed on event

Condition Polling - Repeatedly checking condition until met or timeout

Condition Variable - pthread primitive for thread synchronization

Constructor Attribute - GCC attribute running function before main()

Designated Initializers - C99 struct init: `{.field = value}`

Global Timeout - Time limit for entire test suite via SIGALRM

Linker Wrapping - `--wrap=func` redirects calls to `__wrap_func`

Mock - Fake function implementation for testing

Monotonic Clock - Time source unaffected by system time changes

Poll Interval - Time between condition checks

pthread - POSIX threads library

Return Value Union - Tagged union for type-safe mock returns

Self-Test - Test validating the testing framework itself

siglongjmp/sigsetjmp - Signal-safe non-local jump

Thread-Safe - Works correctly with concurrent access

Timeout - Maximum wait time before giving up

Union - C type holding different types in same memory

2.6 6. Troubleshooting

2.6.1 Issue: Test Hangs

Symptom: Test runs forever
Solution: Add global timeout

```
dap_test_set_global_timeout(&timeout, 30, "Tests");
```

2.6.2 Issue: High CPU

Symptom: 100% CPU during test
Solution: Increase poll interval or use pthread helpers

```
cfg.poll_interval_ms = 500;  // Less frequent polling
```

2.6.3 Issue: Mock Not Called

Symptom: Real function executes
Solution: Check linker flags

```
make VERBOSE=1 | grep -- "--wrap"
```

2.6.4 Issue: Wrong Return Value

Symptom: Mock returns unexpected value
Solution: Use correct union field

```
.return_value.i = 42      // int
.return_value.l = 0xDEAD  // pointer
.return_value.ptr = ptr   // void*
```

2.6.5 Issue: Flaky Tests

Symptom: Sometimes pass, sometimes fail
Solution: Increase timeout, add tolerance

```
cfg.timeout_ms = 60000;  // 60 sec for network
assert(elapsed >= 90 && elapsed <= 150);  // ±50ms tolerance
```