

Jenga Build System v1.1.0

 **Un système de build moderne et puissant pour C/C++**

License **Proprietary**

Python **3.7+**

Platforms **Windows | Linux | macOS | Android | iOS | WebAssembly**

Jenga Build System

Modern Multi-Platform C/C++ Build System with Unified Python DSL

✦ What's New in v1.1.0

Enhanced Creation Tools

- **Intelligent File Creation:** Create classes, structs, enums, interfaces with auto-configuration
- **Smart Project Attachment:** Attach existing projects to workspaces
- **Template System:** Custom file templates for rapid development
- **Auto-configuration:** Files automatically added to project **.jenga** configuration









Advanced Dependency Management

- **Context-Based Inclusion:** **include()** context manager for clean external project integration
- **Project Filtering:** Include specific projects from external **.jenga** files
- **Dependency Validation:** Automatic dependency graph validation
- **Path Resolution:** Smart path handling for external projects

Advanced Documentation Extraction

- **Automatic API Documentation:** Extract documentation from C/C++ source files
- **Multi-Format Output:** Generate Markdown with links compatible across all platforms
- **Intelligent Parsing:** Support for Doxygen, JavaDoc, Qt-style comments, and NK sections
- **Comprehensive Statistics:** Detailed metrics and insights about your codebase

Table of Contents

- ✦ [Features](#)
-  [Quick Start](#)
-  [Installation](#)
-  [Basic Usage](#)
-  [Project Creation & Management](#)
-  [Advanced File Creation](#)
-  [External Project Integration](#)
-  [Documentation](#)
-  [Documentation Extraction](#)

-  [Advanced Features](#)
-  [Project Examples](#)
-  [Contributing](#)
-  [License](#)
-  [Disclaimer](#)

✦ Features

🎯 Core Capabilities

- **Unified Python DSL** - Clean, readable configuration syntax
- **Multi-Platform Support** - Windows, Linux, macOS, Android, iOS, WebAssembly
- **Intelligent Cache** - 20x faster incremental builds
- **Integrated Testing** - Built-in unittest framework
- **Zero Dependencies** - Pure Python 3, no external tools required

🔧 Advanced Creation Tools

- **Smart File Creation** - Automatic `.jenga` configuration updates
- **Multi-File Templates** - Class (.h + .cpp), Struct, Enum, Interface
- **Custom Templates** - User-defined file templates
- **Namespace Support** - Automatic namespace generation
- **Platform Detection** - Smart file placement based on type

🔗 External Project Management

- **Context-Based Inclusion** - `include()` context manager
- **Project Filtering** - Select specific projects to include
- **Dependency Resolution** - Automatic path and dependency handling
- **Workspace Attachment** - Attach existing projects to any workspace

📖 Documentation Tools

- **Multi-Format Comment Parsing** - Doxygen, Javadoc, Qt, NK sections, inline `///`
- **Automatic Markdown Generation** - Fully linked documentation with statistics
- **Cross-Platform Compatibility** - Links work in VS Code, GitHub, MkDocs, GitBook
- **Advanced Statistics** - Codebase metrics, type distribution, namespace analysis
- **Intelligent Navigation** - Multiple views (by file, namespace, type, search)

🔑 Build System

- **C/C++ Toolchains** - GCC, Clang, MSVC support
- **Cross-Compilation** - Android NDK, Emscripten
- **Parallel Builds** - Multi-core optimization
- **Dependency Graph** - Automatic build ordering
- **Smart File Tracking** - Changed files detection

🚀 Quick Start

Hello World in 60 Seconds

1. Create project structure:

```
mkdir hello-world  
cd hello-world
```

2. Create `main.cpp`:

```
#include <iostream>  
  
int main() {  
    std::cout << "Hello, Jenga!" << std::endl;  
    return 0;  
}
```

3. Create `hello.jenga`:

```
with workspace("HelloWorld"):  
    configurations(["Debug", "Release"])  
  
    with project("Hello"):  
        consoleapp()  
        language("C++")  
        files(["main.cpp"])  
        targetdir("Build/Bin/{cfg.buildcfg}")
```

4. Build and run:

```
jenga build  
jenga run  
# Output: Hello, Jenga!
```

Installation

Method 1: From PyPI (Recommended)

```
pip install jenga-build-system
```

Method 2: From Source

```
# Clone repository
git clone https://github.com/RihenUniverse/Jenga.git
cd Jenga

# Install in development mode
pip install -e .

# Or install globally
pip install .
```

💡 Basic Usage

Project Configuration

```
with workspace("MyApplication"):
    # Global settings
    configurations(["Debug", "Release", "Dist"])
    platforms(["Windows", "Linux", "Android"])
    startproject("MainApp")

    # Compiler toolchain
    with toolchain("gcc", "g++"):
        cppcompiler("g++")
        cppdialect("C++20")

    # Library project
    with project("CoreLibrary"):
        staticlib()
        files(["src/core/**/*.cpp", "include/**/*.h"])
        includedirs(["include"])

    # Application project
    with project("MainApp"):
        consoleapp()
        files(["src/app/**/*.cpp"])
        dependson(["CoreLibrary"])

    # Unit tests
    with test("Unit"):
        testfiles(["tests/**/*.cpp"])
```

Common Commands

```
# Build default project
jenga build

# Build specific configuration
```

```
jenga build --config Release --platform Windows
```

```
# Run application
```

```
jenga run
```

```
jenga run --project MyApp
```

```
# Clean build artifacts
```

```
jenga clean
```

```
jenga clean --all
```

```
# Show project info
```

```
jenga info
```

```
# Generate project files (VS, Xcode, etc.)
```

```
jenga gen
```

Project Creation & Management

Creating New Projects

```
# Interactive project creation
```

```
jenga create project
```

```
# Quick creation with options
```

```
jenga create project MyLibrary --type staticlib --language C++ --std C++20
```

```
# Create in specific location
```

```
jenga create project Tools --location utils/ --type consoleapp
```

Attaching Existing Projects

```
# Attach existing project to current workspace
```

```
jenga create attach-existing Core/ExistingLibrary
```

```
# Attach with custom name
```

```
jenga create attach-existing ../External/Engine --name GameEngine
```

Workspace Management

```
# Create new workspace
```

```
jenga create workspace MyGame
```

```
# Create workspace with main project
```

```
jenga create workspace MyApp --type windowedapp --platforms Windows,Linux
```

```
# Interactive workspace creation
jenga create workspace
```

📁 Advanced File Creation

Creating Source Files with Auto-Configuration

```
# Create a C++ class (header + source)
jenga create file Player --type class --namespace game

# Create a struct
jenga create file Vector3 --type struct --namespace math

# Create an enum
jenga create file ErrorCode --type enum --namespace utils

# Create a header-only file
jenga create file Constants --type header --namespace app

# Create source file
jenga create file Utilities --type source

# Create Objective-C file
jenga create file IOSAppDelegate --type m

# Create Objective-C++ file
jenga create file IOSBridge --type mm
```

Advanced File Creation with Templates

```
# Use custom utility template
jenga create file-advanced StringUtils --template custom_util --namespace utils

# Create template class
jenga create file-advanced Container --template custom_class_template

# Create with custom content
jenga create file-advanced Specialized --type custom_cpp --custom-content "//
Custom implementation"
```

File Creation Options

```
# Specify project
jenga create file MyClass --type class --project CoreLibrary
```

```
# Specify Location
jenga create file Config --type header --location config/ --namespace config

# Disable auto-configuration (for manual control)
jenga create file-advanced ManualFile --type header --auto-update false
```

🔗 External Project Integration

Using `include()` Context Manager

The `include()` context manager provides clean, safe external project integration:

```
with workspace("MyApp"):
    # Include all projects from external .jenga file
    with include("libs/logger/logger.jenga"):
        pass # All projects included automatically

    # Include specific projects only
    with include("libs/math/math.jenga") as math_inc:
        math_inc.only(["MathLib", "VectorMath"]) # Include only these projects

    # Exclude specific projects
    with include("libs/network/network.jenga") as net_inc:
        net_inc.skip(["Tests", "Examples"]) # Skip these projects

    # Your main project
    with project("MyApp"):
        consoleapp()
        dependson(["Logger", "MathLib", "VectorMath", "NetworkCore"])
```

Legacy `addprojects()` Function

For backward compatibility or simple use cases:

```
with workspace("MyApp"):
    # Include all projects from external file
    addprojects("external/lib.jenga")

    # Include specific projects only
    addprojects("external/engine.jenga", ["Core", "Renderer"])
```

Smart Path Resolution

Jenga automatically handles:

- Relative and absolute paths
- Project location resolution

- Include directory adjustment
- Dependency validation
- Toolchain inheritance

Project Properties Access

Access external project properties for configuration:

```
with workspace("MyApp"):
    with include("libs/logger/logger.jenga"):
        pass













    with project("MyApp"):
        # Access included project properties
        logger_props = get_project_properties("Logger")

        # Use properties in your project
        includedirs(logger_props['includedirs'])
        links(logger_props['links'])
```

Documentation

Complete Documentation

All documentation is included in the **Docs/** directory:

Document	Description
 BOOK_PART_1.md	Introduction & Installation
 BOOK_PART_2.md	Core Concepts
 BOOK_PART_3.md	Advanced Features
 QUICKSTART.md	Quick Start Guide
 API_REFERENCE.md	Complete API Reference
 ANDROID_EMSCRIPTEN_GUIDE.md	Android & WebAssembly
 MSVC_GUIDE.md	Windows/Visual Studio Guide
 TESTING_GUIDE.md	Testing Framework
 PACKAGING_SIGNING_GUIDE.md	Packaging & Signing
 MIGRATION_GUIDE.md	Migration from CMake/Make
 TROUBLESHOOTING.md	Troubleshooting Guide
 CHANGELOG.md	Version History

Documentation Extraction

Jenga includes a powerful documentation extractor that automatically generates comprehensive API documentation from your C/C++ source code comments.

Quick Start with Documentation

```
# Extract documentation from all projects in workspace
jenga docs extract

# Extract from specific project
jenga docs extract --project Engine

# Include private members
jenga docs extract --include-private

# Show documentation statistics
jenga docs stats

# Clean generated documentation
jenga docs clean
```

Features

Intelligent Comment Parsing

- **Multiple Comment Styles:** Doxygen (`/** ... */`), JavaDoc (`/*! ... */`), Qt (`/*! ... */`), NK sections (`// -----`), inline (`///`)
- **Automatic Element Detection:** Classes, structs, enums, functions, methods, variables, namespaces
- **Tag Support:** `@brief`, `@param`, `@return`, `@throws`, `@example`, `@note`, `@warning`, `@see`, etc.
- **Template Parameter Detection:** Automatic extraction of template parameters

Multi-View Organization

Generated documentation is organized in multiple views:

- **By File:** All elements grouped by source file
- **By Namespace:** Organized by C++ namespace
- **By Type:** Classes, structs, enums, functions separately
- **Alphabetical Search:** Complete alphabetical index
- **API Overview:** Complete API listing

Comprehensive Statistics

- **Element Counts:** Total documented elements by type
- **File Analysis:** Files with most documentation
- **Namespace Distribution:** Elements per namespace

- **Code Insights:** Automatic insights about your codebase
- **Progress Metrics:** Documentation coverage metrics

Documentation Commands

```
# Extract documentation with various options
jenga docs extract --project MyLibrary --output ./docs --include-private
jenga docs extract --format markdown --no-split-namespace
jenga docs extract --exclude-dirs "ThirdParty" "Tests" --exclude-projects
Sandbox

# View statistics
jenga docs stats --project Engine --json
jenga docs stats # Text format

# List projects available for documentation
jenga docs list

# Clean generated files
jenga docs clean --project Engine
jenga docs clean # Clean all documentation
```

Comment Examples

Doxygen Style

```
/**
 * @brief Calculates the length of a 3D vector
 *
 * Computes the Euclidean norm (magnitude) of the vector.
 *
 * @param[in] x X component
 * @param[in] y Y component
 * @param[in] z Z component
 *
 * @return Vector Length (magnitude)
 * @retval 0.0f For zero vector
 *
 * @complexity O(1)
 * @threadsafe
 *
 * @example Basic usage
 * @code
 * float Len = VectorLength(3.0f, 4.0f, 0.0f); // Returns 5.0f
 * @endcode
 *
 * @see Normalize()
 * @note This function is constexpr in C++17 and later
 */
```

```

* @author Rihen
* @date 2026-02-07
*/
constexpr float VectorLength(float x, float y, float z);

```

NK Section Style

```

// -----
// CLASSE: NkVector3
// DESCRIPTION: 3D vector class for positions and directions
// AUTEUR: Rihen
// DATE: 2026-02-07
// -----
class NK_API NkVector3 {
public:
    /// @brief X component
    /// X coordinate in right-handed coordinate system
    float x;

    /// @brief Y component
    /// Y coordinate in right-handed coordinate system
    float y;

    /// @brief Z component
    /// Z coordinate in right-handed coordinate system
    float z;

    /**
     * @brief Default constructor
     *
     * Initializes all components to zero.
     *
     * @example
     * @code
     * NkVector3 v; // (0, 0, 0)
     * @endcode
     */
    NkVector3() : x(0), y(0), z(0) {}
};

```

Generated Documentation Structure

```

docs/
├── MyProject/
│   └── markdown/
│       ├── index.md           # Home page with overview
│       ├── SUMMARY.md        # Complete table of contents
│       └── api.md             # Complete API listing

```

```
├── search.md           # Alphabetical search index
├── stats.md           # Detailed statistics
├── files/             # Documentation by file
│   ├── index.md
│   ├── NkVector3_h.md
│   └── NkMatrix4_h.md
├── namespaces/       # Documentation by namespace
│   ├── index.md
│   ├── nk.md
│   └── nk_math.md
├── types/            # Documentation by type
│   ├── index.md
│   ├── classes.md
│   ├── structs.md
│   └── functions.md
└── AnotherProject/
    ├── markdown/
    └── ...
```

Cross-Platform Compatibility

The generated documentation works perfectly with:

Platform	Support	Features
VS Code	☑ Perfect	Ctrl+click navigation, preview mode
GitHub/GitLab	☑ Perfect	Native web navigation
MkDocs	☑ Perfect	Direct integration
GitBook	☑ Perfect	Full compatibility
Obsidian/Typora	☑ Perfect	Advanced markdown editors
GitHub Pages	☑ Perfect	Ready to deploy

Integration with Development Workflow

Add to Build Tasks

```
# In your .jenga workspace file
addtarget("docs", "jenga docs extract --include-private")
addtarget("docs-clean", "jenga docs clean")
```

CI/CD Integration



```
# .github/workflows/docs.yml
name: Documentation
on: [push, pull_request]

jobs:
  docs:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v4
        with:
          python-version: '3.11'
      - run: pip install jenga-build-system
      - run: jenga docs extract --verbose
      - run: touch docs/.nojekyll # For GitHub Pages
```

Script Automation

```
#!/usr/bin/env python3
"""Automated documentation generation"""
import subprocess
import sys

def generate_docs():
    """Generate documentation for all projects"""
    print("📁 Generating documentation...")

    result = subprocess.run([
        "jenga", "docs", "extract",
        "--exclude-projects", "Tests", "Sandbox",
        "--include-private",
        "--verbose"
    ])

    if result.returncode == 0:
        print("✅ Documentation generated successfully")

        # Generate stats
        subprocess.run(["jenga", "docs", "stats"])

        print("\n📁 Open docs/[project]/markdown/index.md")
        return 0
    else:
        print("❌ Failed to generate documentation")
        return result.returncode

if __name__ == "__main__":
    sys.exit(generate_docs())
```

Complex Class Documentation

```
/**
 * @brief Template container class
 *
 * Implements a dynamic array with value semantics.
 *
 * @tparam T Type of elements (must be copyable)
 * @tparam Allocator Memory allocator type (default: std::allocator<T>)
 *
 * @example Basic usage
 * @code
 * NkArray<int> numbers;
 * numbers.push_back(42);
 * numbers.push_back(100);
 *
 * for (int n : numbers) {
 *     std::cout << n << std::endl;
 * }
 * @endcode
 *
 * @example With custom allocator
 * @code
 * NkArray<int, MyCustomAllocator> customArray;
 * customArray.reserve(1000);
 * @endcode
 *
 * @complexity
 * - Access:  $O(1)$ 
 * - Insertion at end:  $O(1)$  amortized
 * - Insertion elsewhere:  $O(n)$ 
 * - Deletion:  $O(n)$ 
 *
 * @threadsafe No (external synchronization required)
 *
 * @warning Not exception safe for non-trivial types
 *
 * @since Version 2.0.0
 * @deprecated Will be replaced by NkVector<T> in version 3.0
 */
template<typename T, typename Allocator = std::allocator<T>>
class NkArray {
    // Implementation...
};
```

Function with Exception Specification

```

/**
 * @brief Loads a texture from file
 *
 * @param[in] filepath Path to texture file
 * @param[in] async Load asynchronously
 * @param[in] generateMipmaps Generate mipmap chain
 *
 * @return Texture handle
 *
 * @throw NkFileNotFoundException If file doesn't exist
 * @throw NkInvalidFormatException If unsupported format
 * @throw NkOutOfMemoryException If insufficient GPU memory
 *
 * @warning In async mode, texture is not immediately usable
 * @warning generateMipmaps=true increases load time significantly
 *
 * @note Supported formats: PNG, JPEG, BMP, TGA, DDS
 * @note For DDS files, mipmaps are loaded from file
 *
 * @see LoadFromMemory()
 * @see Unload()
 *
 * @example Error handling
 * @code
 * try {
 *     NkTextureHandle tex = NkTexture::Load("character.png");
 *     Renderer::Draw(tex, position);
 * } catch (const NkFileNotFoundException& e) {
 *     Logger::Error("Texture not found: {}", e.what());
 *     return;
 * }
 * @endcode
 */
static NkTextureHandle Load(
    const std::string& filepath,
    bool async = false,
    bool generateMipmaps = true
);

```

Statistics and Insights

The documentation extractor provides valuable insights:



EXTRACTION SUMMARY



Projects processed: 3



Total files: 156



Total elements: 1,247



Documentation generated in: docs/

Each project has its own subdirectory

Sample Statistics Page Content:

- 45.3% Classes, 22.1% Functions, 15.8% Methods, 8.4% Structs
- Top namespace: nk (312 elements)
- Most documented file: Core/Nkentseu.h (89 elements)
- Documentation density: 8.0 elements/file

Troubleshooting Documentation Extraction

Problem: No comments extracted

Solution: Ensure your comments use supported formats:

- Doxygen: `/** ... */`
- NK sections: `// -----` with proper headers
- Inline: `///` for brief documentation

Problem: Links don't work in VS Code

Solution: Update to v1.1.0+ which fixes all relative paths.

Problem: Missing files

Solution: Check that source directories exist in project configuration.

Problem: Encoding issues

Solution: Ensure source files are UTF-8 encoded.

Best Practices

1. **Consistent Formatting:** Use consistent comment style throughout project
2. **Complete Documentation:** Document all public APIs
3. **Examples:** Include usage examples for complex functions
4. **Error Handling:** Document exceptions and error conditions
5. **Performance Notes:** Include complexity and thread safety information
6. **Version Information:** Use `@since` and `@deprecated` tags
7. **Cross-References:** Use `@see` to link related functions

Advanced Features

Multi-Platform Configuration

```
with workspace("CrossPlatformGame"):
    platforms(["Windows", "Linux", "Android", "iOS"])

    with project("GameEngine"):
        staticlib()
```



```

# Common code
files(["src/engine/**/*.cpp"])

# Platform-specific
with filter("system:Windows"):
    links(["d3d11", "dxgi"])

with filter("system:Android"):
    androidminsdk(21)
    links(["log", "android", "EGL"])

with filter("system:iOS"):
    framework("UIKit")
    framework("OpenGLES")

```

Advanced Dependency Management

```

with workspace("LargeProject"):
    # Batch include multiple libraries
    with include("libs/core.jenga"):
        pass

    with include("libs/graphics.jenga") as gfx:
        gfx.only(["Renderer", "ShaderSystem"])

    with include("libs/physics.jenga") as phys:
        phys.skip(["Tests", "DebugTools"])

    # Complex dependency chain
    with project("Game"):
        consoleapp()
        dependson([
            "CoreSystem",
            "Renderer",
            "ShaderSystem",
            "PhysicsEngine"
        ])

    # Auto-configure based on dependencies
    useproject("Renderer", copy_includes=True)
    useproject("PhysicsEngine", copy_defines=True)

```

Project Examples

Example 1: Modular Game Engine

```

game-engine/
├─ engine.jenga
├─ Core/           # Core systems
├─ Math/           # Mathematics library
├─ Render/         # Rendering system
├─ Audio/          # Audio system
├─ Physics/        # Physics engine
└─ Game/           # Game-specific code

```

engine.jenga:

```

with workspace("GameEngine"):
    configurations(["Debug", "Release", "Profile"])
    platforms(["Windows", "Linux", "Android"])

    # Include external math library
    with include("third_party/glm/glm.jenga"):
        pass

    # Core engine systems
    with project("CoreSystem"):
        staticlib()
        files(["Core/src/**/*.cpp"])
        includedirs(["Core/include"])

    with project("Renderer"):
        sharedlib()
        files(["Render/src/**/*.cpp"])
        includedirs(["Render/include"])
        dependson(["CoreSystem", "glm"])

    # Game project
    with project("MyGame"):
        windowedapp()
        files(["Game/src/**/*.cpp"])
        dependson(["CoreSystem", "Renderer"])

    # Auto-create files as needed
    # jenga create file Player --type class --namespace game

    # Generate documentation
    # jenga docs extract --project MyGame

```

Example 2: Plugin-Based Application

```

with workspace("PluginApp"):
    # Main application

```

```

with project("AppCore"):
    staticlib()
    files(["core/src/**/*.cpp"])

# Plugins as separate projects
with project("ImagePlugin"):
    sharedlib()
    files(["plugins/image/src/**/*.cpp"])
    dependson(["AppCore"])

with project("AudioPlugin"):
    sharedlib()
    files(["plugins/audio/src/**/*.cpp"])
    dependson(["AppCore"])

# Main executable
with project("Application"):
    consoleapp()
    files(["app/src/**/*.cpp"])
    dependson(["AppCore", "ImagePlugin", "AudioPlugin"])

```

Example 3: Cross-Platform Library

```

with workspace("CrossPlatformLib"):
    platforms(["Windows", "Linux", "macOS", "Android", "iOS"])

with project("PlatformAbstraction"):
    staticlib()
    files(["src/common/**/*.cpp"])

# Platform-specific implementations
with filter("system:Windows"):
    files(["src/windows/**/*.cpp"])
    defines(["PLATFORM_WINDOWS"])

with filter("system:Linux"):
    files(["src/linux/**/*.cpp"])
    defines(["PLATFORM_LINUX"])

with filter("system:Android"):
    files(["src/android/**/*.cpp"])
    defines(["PLATFORM_ANDROID"])

```

Tests Unitaires Avancés

Framework de Test Intégré

Jenga inclut un framework de tests unitaires puissant avec des assertions riches :

```

#include <Unitest/Unitest.h> // Macros de test de Jenga

// Tests basiques
TEST(Calculator_Addition) {
    ASSERT_EQUAL(5, Calculator::add(2, 3));
    ASSERT_EQUAL(0, Calculator::add(-1, 1));
    ASSERT_EQUAL(-5, Calculator::add(-2, -3));
}

TEST(Calculator_Multiplication) {
    ASSERT_EQUAL(6, Calculator::multiply(2, 3));
    ASSERT_EQUAL(0, Calculator::multiply(0, 100));
    ASSERT_EQUAL(-6, Calculator::multiply(2, -3));
}

TEST(Calculator_Division) {
    ASSERT_NEAR(5.0, Calculator::divide(10.0, 2.0), 0.001);
    ASSERT_NEAR(-2.5, Calculator::divide(5.0, -2.0), 0.001);

    // Test division par zéro
    ASSERT_THROWS(std::invalid_argument, Calculator::divide(1.0, 0.0));
}

TEST(Calculator_EdgeCases) {
    // Test avec grands nombres
    ASSERT_EQUAL(2000000000, Calculator::add(1000000000, 1000000000));

    // Test avec nombres négatifs
    ASSERT_EQUAL(1, Calculator::add(-10, 11));

    // Performance test
    ASSERT_EXECUTION_TIME_LESS([]() {
        for (int i = 0; i < 1000; ++i) {
            Calculator::add(i, i);
        }
    }, 10.0); // Doit prendre moins de 10ms
}

```

Macros de Test Disponibles

Assertions Basiques

```

// Assertions simples
ASSERT_EQUAL(expected, actual)
ASSERT_NOT_EQUAL(expected, actual)
ASSERT_TRUE(condition)
ASSERT_FALSE(condition)
ASSERT_NULL(ptr)
ASSERT_NOT_NULL(ptr)

```

```
// Avec messages personnalisés
ASSERT_EQUAL_MSG(expected, actual, "Message personnalisé")
ASSERT_TRUE_MSG(condition, "Doit être vrai")
```

Comparaisons Numériques

```
// Comparaisons avec tolérance
ASSERT_LESS(left, right)
ASSERT_LESS_EQUAL(left, right)
ASSERT_GREATER(left, right)
ASSERT_GREATER_EQUAL(left, right)
ASSERT_NEAR(expected, actual, tolerance)
ASSERT_EQUAL_TOLERANCE(expected, actual, tolerance)
```

Gestion des Exceptions

```
// Tests d'exceptions
ASSERT_THROWS(std::exception, expression)
ASSERT_NO_THROW(expression)
ASSERT_THROWS_MSG(std::exception, expression, "Message")
ASSERT_NO_THROW_MSG(expression, "Message")
```

Collections et Conteneurs

```
// Tests sur collections
ASSERT_CONTAINS(container, value)
ASSERT_NOT_CONTAINS(container, value)
ASSERT_CONTAINS_MSG(container, value, "Message")
ASSERT_NOT_CONTAINS_MSG(container, value, "Message")
```

Performance et Benchmarking

```
// Tests de performance
ASSERT_EXECUTION_TIME_LESS(expression, maxTimeMs)
ASSERT_EXECUTION_TIME_BETWEEN(expression, minTimeMs, maxTimeMs)

// Benchmarks
RUN_BENCHMARK("nom", fonction, iterations)
ASSERT_BENCHMARK_FASTER(benchmarkA, benchmarkB)
ASSERT_BENCHMARK_FASTER_WITH_LIMIT(benchmarkA, benchmarkB, limite)
```

```
// Profiling
BEGIN_PROFILING_SESSION("session")
END_PROFILING_SESSION_AND_REPORT("session")
PROFILE_TEST_SCOPE(testName, code_a_profiler)
```

Exemple Complet de Suite de Tests

```
// tests/MathTest.cpp
#include <Unitest/Unitest.h>
#include "../src/math/Calculator.h"

// Test de base
TEST(Math_BasicOperations) {
    ASSERT_EQUAL(4, Calculator::add(2, 2));
    ASSERT_EQUAL(6, Calculator::multiply(2, 3));
    ASSERT_NEAR(2.0, Calculator::divide(6.0, 3.0), 0.001);
}

// Test avec fixture
class CalculatorFixture : public TestFixture {
protected:
    Calculator* calc;

    void SetUp() override {
        calc = new Calculator();
    }

    void TearDown() override {
        delete calc;
    }
};

TEST_FIXTURE(CalculatorFixture, AdditionWithFixture) {
    ASSERT_EQUAL(5, calc->add(2, 3));
    ASSERT_EQUAL(0, calc->add(-1, 1));
}

// Test de performance
TEST_BENCHMARK_SIMPLE(Performance_Addition, "AdditionBenchmark", []() {
    volatile int result = 0;
    for (int i = 0; i < 10000; ++i) {
        result += Calculator::add(i, i);
    }
}, 1000)

// Test avec profiling
PROFILE_TEST_SCOPE(Profile_Addition, {
    for (int i = 0; i < 1000; ++i) {
        Calculator::add(i, i + 1);
    }
})
```

```

})

// Test de régression
TEST_BENCHMARK_WITH_BASELINE(Regression_Addition, "Addition", []() {
    Calculator::add(100, 200);
}, 1000, baseline_benchmark)

// Test avec comparaison
COMPARE_BENCHMARKS(Comparison_Operations,
    "Addition", []() { Calculator::add(1, 2); },
    "Multiplication", []() { Calculator::multiply(1, 2); },
    1000, 1.5)

```

Configuration des Tests dans .jenga

```

with workspace("MyProject"):
    configurations(["Debug", "Release"])

# Projet principal
with project("Calculator"):
    staticlib()
    files(["src/**/*.cpp", "src/**/*.h"])
    includedirs(["src"])
    targetdir("Build/Lib/{cfg.buildcfg}")

# Suite de tests
with test("CalculatorTests"):
    testfiles(["tests/**/*.cpp"])
    testmainfile("src/main.cpp") # Exclure le main de l'appli

# Options de test
testoptions([
    "--verbose",
    "--stop-on-failure",
    "--filter=Math*"
])

# Configuration spécifique aux tests
with filter("configurations:Debug"):
    defines(["ENABLE_TESTING", "DEBUG_TESTS"])

# Répertoires de sortie pour les tests
targetdir("Build/Tests/{cfg.buildcfg}")

# Dépendances des tests
dependson(["Calculator"])
includedirs(["tests/include"])

# Fichiers de test supplémentaires
dependfiles([

```

```
        "tests/data/**",  
        "tests/config/test.conf"  
    ])
```

Commandes de Test Avancées

```
# Exécuter tous les tests  
jenga test  
  
# Exécuter avec débogage  
jenga test --debug=gdb  
jenga test --debug=valgrind # Détection de fuites mémoire  
jenga test --debug=helgrind # Détection de courses  
  
# Exécuter un test spécifique  
jenga test --project CalculatorTests  
  
# Exécuter avec options personnalisées  
jenga test -- --verbose --filter=Math* --parallel=4  
  
# Lister les tests disponibles  
jenga test --list  
  
# Construire seulement les tests  
jenga test --build  
  
# Tests avec couverture  
jenga test --coverage  
  
# Tests avec profiling  
jenga test --profile
```

Contributing

We welcome contributions! Here's how you can help:

Reporting Issues

1. Check existing issues in GitHub
2. Use the issue template
3. Include system info and reproduction steps

Feature Requests

1. Describe the use case
2. Show example syntax
3. Discuss implementation


```
# Development setup
git clone https://github.com/RihenUniverse/Jenga.git
cd Jenga
pip install -e .[dev]

# Run tests
pytest

# Format code
black .

# Check code quality
flake8 Jenga/
mypy Jenga/
```

License

Proprietary License - Rihen

Copyright © 2026 Rihen. All rights reserved.

Permissions

- ☒ **Free to Use** - No cost for personal or commercial use
- ☒ **Modification Rights** - You may modify the source code
- ☒ **Distribution** - You may distribute modified versions
- ☒ **Integration** - Can be used in proprietary projects

Conditions

1. **Attribution Required** - Must include this license in distributions
2. **Copyright Notice** - Must preserve Rihen copyright
3. **No Removal** - Cannot remove license headers from source files
4. **No Sublicensing** - Cannot grant additional rights to others

Restrictions

- ☒ **No Resale** - Cannot sell Jenga as a standalone product
- ☒ **No Warranty** - Provided "as is" without guarantees
- ☒ **Liability** - Rihen not liable for damages
- ☒ **Patent Claims** - No patent licenses granted

Disclaimer

NO WARRANTY: Jenga Build System is provided "AS IS" without any warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

NO LIABILITY: In no event shall Rihen or its contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

Built with  by [Rihen](#)

Jenga Build System - Making C++ builds simple across all platforms

 Documentation Extractor v1.1.0 - Automatic API documentation generation