🏗️ Jenga Build System

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

License | Proprietary

Python | 3.7+

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

✨ 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

📋 Table of Contents

✨ 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 Unitest 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

🔧 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`:**

```python
with workspace("HelloWorld"):
    configurations(["Debug", "Release"])

    with project("Hello"):
        consoleapp()
        language("C++")
        files(["main.cpp"])
        targetdir("Build/Bin/%{cfg.buildcfg}")
```

4. **Build and run:**

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

📦 Installation

Method 1: From PyPI (Recommended)

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

Method 2: From Source

```bash
# 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

```python
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:
```

```python
        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:

```python
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:

```python
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

🔧 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:

```python
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
```

Example 2: Plugin-Based Application

```python
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"])
```

🤝 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

Code Contributions

```
# 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