"IOC_Config Documentation"

"Michele Bigi"

IOC_Config Library - Library Structure

This document describes the architecture and organization of the IOC_Config OOP Parser library.

Project Structure

```
IOC_Config/
├── include/
│   └── ioc_config/
│       └── oop_parser.h          # Main public API header
├── src/
│   └── oop_parser.cpp            # Implementation
├── examples/
│   ├── CMakeLists.txt
│   ├── basic_usage.cpp           # Basic usage example
│   ├── oop_to_json_converter.cpp # OOP ↔ JSON converter tool
│   └── config_builder_example.cpp # Programmatic configuration
builder
├── tests/
│   ├── CMakeLists.txt
│   ├── test_parser.cpp           # Parser functionality tests
│   ├── test_builder.cpp          # Config builder tests
│   └── test_conversion.cpp       # JSON conversion tests
├── cmake/
│   └── [CMake helper modules]
├── build/                        # Build output directory
(generated)
├── CMakeLists.txt               # Main CMake configuration
├── ioc_config.pc.in             # pkg-config template
├── README.md                     # Full documentation
├── QUICKSTART.md                # Quick start guide
├── LICENSE                       # License file
└── .gitignore
```

Library Architecture

Main Components

1. OopParser Class

The main class for parsing and manipulating OOP files.

Responsibilities: - Load OOP files from disk - Parse OOP format with sections and parameters - Provide type-safe access to configuration data - Support JSON serialization/deserialization - Validate configurations - Modify and save configurations

Key Methods:

```
bool loadFromOop(const std::string& filepath);
bool saveToOop(const std::string& filepath);
bool loadFromJson(const std::string& filepath);
bool saveToJson(const std::string& filepath);
ConfigSectionData* getSection(const std::string& name);
ConfigParameter* findParameter(const std::string& key);
bool setParameter(const std::string& section, const std::string&
        key, const std::string& value);
bool validate(std::vector<std::string>& errors);
```

2. ConfigSectionData Structure

Represents a configuration section.

Members:

```
SectionType type;                                    // Section
        type enumeration
std::string name;                                    // Section
        name
std::map<std::string, ConfigParameter> parameters;   //
        Parameters in section
```

3. ConfigParameter Structure

Represents a single configuration parameter.

Members:

```
std::string key;           // Parameter key
std::string value;         // Parameter value (string
        representation)
std::string type;          // Data type: "string", "int",
        "float", "bool", "array"
```

Type Conversion Methods:

```
std::string asString() const;
double asDouble() const;
int asInt() const;
bool asBoolean() const;
std::vector<std::string> asStringVector() const;
```

4. SectionType Enumeration

Predefined section types for standard IOoccultCalc configuration sections.

```
enum class SectionType {
    UNKNOWN = 0,
    OBJECT = 1,           // Object/asteroid information
    PROPAGATION = 2,      // Propagation settings
    ASTEROIDS = 3,        // Asteroid selection
    TIME = 4,             // Time configuration
    SEARCH = 5,           // Search parameters
    DATABASE = 6,         // Database configuration
    GAIA = 7,             // GAIA catalog settings
    OBSERVER = 8,         // Observer location
    OUTPUT = 9,           // Output configuration
    PERFORMANCE = 10,     // Performance settings
    OCCULTATION = 11,     // Occultation parameters
    FILTERS = 12          // Filter settings
};
```

Design Patterns Used

1. Facade Pattern

The `OopParser` class acts as a facade, providing a simple interface to the complex parsing logic.

2. Builder Pattern

Users can programmatically build configurations using `setParameter()`.

3. Strategy Pattern

Different parsing and serialization strategies for OOP and JSON formats.

4. Error Handling

Functions return `bool` for success/failure, with detailed error messages available via `getLastError()`.

Build System

CMake Configuration

Main targets: - `ioc_config_static` - Static library (libioc_config.a on macOS/Linux) - `ioc_config_shared` - Shared library (libioc_config.dylib on macOS, .so on Linux) - `example_*` - Example executables - `test_*` - Test executables

Build options:

```
-DBUILD_EXAMPLES=ON|OFF    # Default: ON
-DBUILD_TESTS=ON|OFF       # Default: ON
-DCMAKE_BUILD_TYPE=Release|Debug
```

Compilation Process

1. **Header Processing**
   - C++ includes are processed
   - Declarations validated
2. **Source Compilation**
   - `oop_parser.cpp` compiled to object files
   - Optimizations applied (Release mode)
3. **Linking**
   - Static library: object files archived
   - Shared library: object files linked with shared dependencies
   - Examples and tests linked with appropriate libraries

Dependencies

External Libraries

- **nlohmann/json** (required)
  - Used for JSON serialization/deserialization
  - Header-only library
  - Found via CMake's find_package()

Internal Dependencies

- C++17 Standard Library
  - `<string>`, `<vector>`, `<map>`, `<memory>`
  - `<fstream>`, `<sstream>` for file I/O
  - `<algorithm>`, `<cctype>` for string utilities

File Format Specifications

OOP Format

Format specification:

```
! Comment line starts with !

section_name.
        .parameter_key = value_expression
        .another_key = value_expression

next_section.
        .param1 = value1
```

Value Types: - **String**: `'quoted string'` or `"quoted string"` - **Number**: `123`, `45.67` (auto-detected) - **Boolean**: `.TRUE.`, `.FALSE.`, `true`, `false`, `1`, `0` - **Array**: `['element1', 'element2', 'element3']`

JSON Format

Equivalent JSON representation:

```
{
  "section_name": {
    "parameter_key": "value",
    "another_key": "value"
  },
  "next_section": {
    "param1": "value1"
  }
}
```

API Usage Patterns

Pattern 1: Read Configuration

```
OopParser parser("config.oop");
auto section = parser.getSection("object");
if (section) {
    auto param = section->getParameter("id");
    std::string id = param->asString();
}
```

Pattern 2: Write Configuration

```
OopParser parser;
parser.setParameter("object", ".id", "'17030'");
parser.setParameter("object", ".name", "'Asteroid 17030'");
parser.saveToOop("output.oop");
```

Pattern 3: Format Conversion

```
convertOopToJson("input.oop", "output.json");
// or
convertJsonToOop("input.json", "output.oop");
```

Pattern 4: Find Across Sections

```
auto param = parser.findParameter("id");
if (param) {
    // Process parameter from any section
}
```

Testing Strategy

Unit Tests

- **test_parser.cpp** - Basic parsing functionality
- **test_builder.cpp** - Configuration builder
- **test_conversion.cpp** - Format conversion

Test Coverage

- Parser initialization
- Section retrieval
- Parameter access
- Type conversions
- JSON serialization
- Error handling

Running Tests

```
cd build
ctest --output-on-failure
```

Performance Characteristics

Time Complexity

- Loading OOP file: O(n) where n = number of lines
- Looking up section: O(m) where m = number of sections
- Looking up parameter: O(1) hash map lookup
- JSON serialization: O(p) where p = number of parameters

Space Complexity

- Storage: O(m * p) for m sections with p parameters each
- No temporary buffers during parsing

Optimization Notes

- Uses `std::map` for O(log n) section lookup
- Direct hash map access for parameters within section
- Efficient string manipulation with move semantics

Platform Support

Tested Platforms

- **macOS**
  - Compiler: AppleClang 17.0+
  - Output: `.a` (static), `.dylib` (shared)

- **Linux** (Ubuntu 20.04+, Debian 11+)
  - Compiler: GCC 9+, Clang 10+
  - Output: `.a` (static), `.so` (shared)
- **Windows** (Windows 10+)
  - Compiler: MSVC 2017+
  - Output: `.lib` (static), `.dll` (shared)

Installation and Distribution

Package Contents (Release)

```
IOC_Config-1.0.0/
├── lib/
│   ├── libioc_config.a           # Static library
│   ├── libioc_config.dylib       # Shared library (macOS)
│   └── pkgconfig/
│       └── ioc_config.pc         # pkg-config metadata
├── include/
│   └── ioc_config/
│       └── oop_parser.h
└── share/
    └── doc/
        ├── README.md
        └── QUICKSTART.md
```

Installation Methods

1. **From Source**

   ```
   mkdir build && cd build
   cmake .. -DCMAKE_BUILD_TYPE=Release
   cmake --build .
   sudo cmake --install .
   ```

2. **Package Managers**

   ```
   # macOS
   brew install ioc_config

   # Ubuntu/Debian
   sudo apt-get install libioc-config-dev
   ```

Version Information

Current Version: 1.0.0

Semantic Versioning: - MAJOR: Breaking API changes - MINOR: New features (backward compatible) - PATCH: Bug fixes

Maintenance and Support

Known Limitations

1. Single-threaded parser (use separate instances for parallel parsing)
2. Comments must be on separate lines
3. Parameter values cannot contain newlines

Future Enhancements

1. Multi-threaded parsing support
2. Schema validation
3. Configuration templates
4. More format conversions (YAML, TOML)
5. Binary format support

Contributing Guidelines

When contributing to the library:

1. **Code Style**
    - Follow C++17 standards
    - Use meaningful variable names
    - Document public API with Doxygen comments
2. **Testing**
    - Add tests for new features
    - Ensure all tests pass before PR
    - Maintain >80% code coverage
3. **Documentation**
    - Update README for new features
    - Add examples for complex functionality
    - Update CHANGELOG.md
4. **Compatibility**
    - Test on macOS, Linux, and Windows
    - Support C++17 and above
    - Maintain backward compatibility when possible