

# MetaFront

Standalone C++20 Reflection Tool

Developer

2025-09-20

**MetaFront**

*Standalone C++20 Reflection Tool*

# What is MetaFront?

## *C++ Header Analysis Tool*

A command-line tool that analyzes C++ header files and generates reflection metadata.

- Single command execution on header files
- No modifications to existing source code
- Generates standard C++20 template code
- Standalone executable with no dependencies
- Works with any build system
- Output integrates with existing projects
- Compile-time reflection metadata
- Input: headers → Output: reflection code

***"Run once on your headers to add reflection capabilities"***

# Your Existing C++ Class

*Standard C++ - No Changes Required*

```
#include  
  
class Person {  
public:  
    int id;  
    std::string name;  
    std::string email;  
};
```

# Run MetaFront Once

*Single Command - Complete Reflection Generation*

```
$ metafront --input ./include --output ./generated
```

```
✓ Analyzing headers...  
✓ Found 1 class: Person  
✓ Generating reflection metadata...  
✓ Generated person_reflection.hpp  
✓ Complete! Your classes now have reflection capabilities.
```

*Note: That's it! One command generates all reflection code for your entire project.*

# Generated Reflection Code

*Standard C++20 Templates - Ready to Use*

```
// Generated by MetaFront - never edit this file
#pragma once
#include "person.hpp"

// Compile-time reflection metadata
template<>
struct metafront::reflection {
    static constexpr auto fields = std::tuple{
        Field{"int", "id"},
        Field{"std::string", "name"},
        Field{"std::string", "email"}
    };
};

// Now Person has reflection capabilities!
```

*Note: Generated code is pure C++20 - works with any compiler, no MetaFront needed*

# Simple Class

## *C++ Class Definition*

```
class Person {  
public:  
    int id;  
    std::string name;  
    std::string email;  
};
```

# Generated Metadata

## *Automatic Reflection Data*

```
static constexpr auto Person_meta = std::tuple<
Field,
Field,
Field
>{
{"int", "id"},
{"std::string", "name"},
{"std::string", "email"}
};
```

# Reflection Usage

*Using the Metadata*

```
Person p{123
```

# Serialization Formats

## *Multiple Output Formats*

### XMI

```
123  
John Doe  
john@email.com
```

### CSV

```
id,name,email  
123,"John Doe","john@email.com"
```

### JSON

```
{  
  "id": 123,  
  "name": "John Doe",  
  "email": "john@email.com"  
}
```

# Database Operations

## *Complete Database Operations*

### CREATE

```
CREATE TABLE Person (  
  id INTEGER PRIMARY KEY,  
  name TEXT NOT NULL,  
  email TEXT NOT NULL  
);
```

### INSERT

```
INSERT INTO Person (id, name, email)  
VALUES (123, 'John Doe', 'john@email.com');
```

### SELECT

```
SELECT id, name, email  
FROM Person  
WHERE id = 123;
```

### UPDATE

```
UPDATE Person SET  
  name = 'John Smith',  
  email = 'johnsmith@email.com'  
WHERE id = 123;
```

### DELETE

```
DELETE FROM Person  
WHERE id = 123;
```

# Reflection Usage

## *Using the Metadata*

```
Person p{123, "John Doe", "john@email.com"};

// Access fields via reflection
for_each_field(p, [](auto& field, const char* name) {
    std::cout << name << ": " << field << std::endl;
});
```

# Serialization Formats

## *Multiple Output Formats*

### XMI

```
123  
John Doe  
john@email.com
```

### CSV

```
id,name,email  
123,"John Doe","john@email.com"
```

### JSON

```
{  
  "id": 123,  
  "name": "John Doe",  
  "email": "john@email.com"  
}
```

# Database Operations

## *Complete Database Operations*

### CREATE

```
CREATE TABLE Person (  
  id INTEGER PRIMARY KEY,  
  name TEXT NOT NULL,  
  email TEXT NOT NULL  
);
```

### INSERT

```
INSERT INTO Person (id, name, email)  
VALUES (123, 'John Doe', 'john@email.com');
```

### SELECT

```
SELECT id, name, email  
FROM Person  
WHERE id = 123;
```

### UPDATE

```
UPDATE Person SET  
  name = 'John Smith',  
  email = 'johnsmith@email.com'  
WHERE id = 123;
```

### DELETE

```
DELETE FROM Person  
WHERE id = 123;
```