An introduction to C++26 reflection

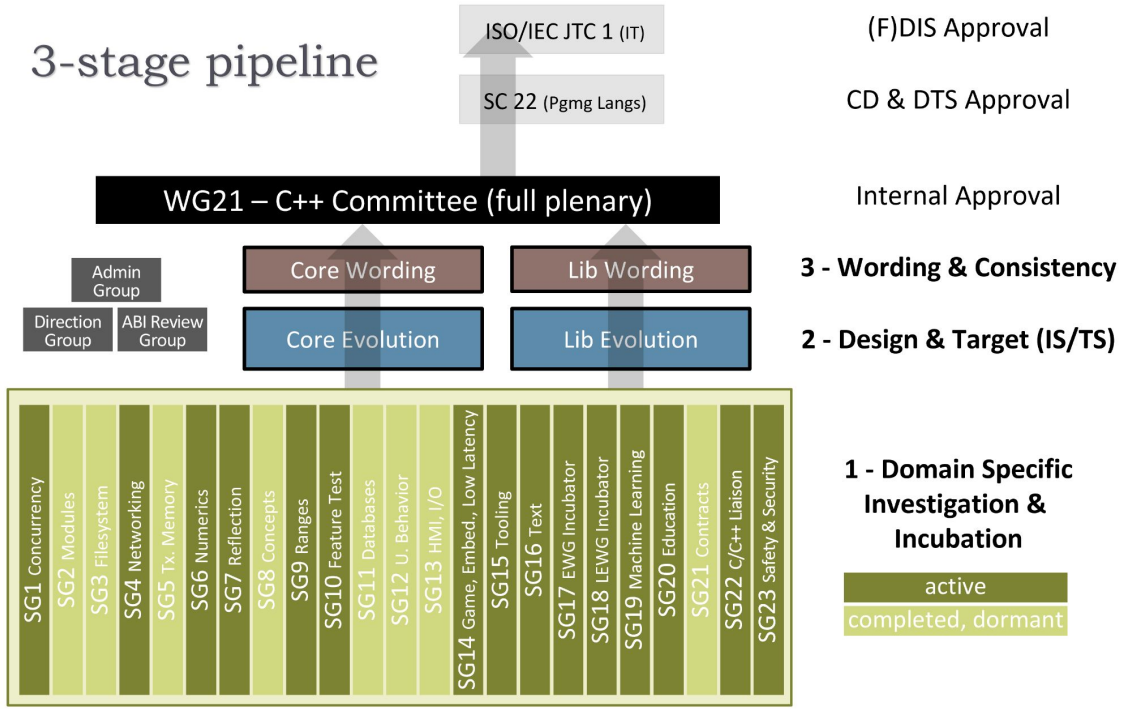
Bastien DOIGNIES

Le C++ comment ça marche ?

Le C++ marche un peu comme un journal:

Des articles sont soumis par la communauté puis approuvé par des reviewer: le comité C++.

Tous les 3 ans, ils publient les nouveautés acceptés : une nouvelle version du C++



C++26

- It will probably be finalised by 28 March. The expected official publication by ISO/IEC is around September 2027.

- <linalg>: BLAS interface
- <simd>: SIMD interface
- <inplace_vector>
- <hive>
- <hazard_pointer>
- <debugging>
- <meta>
- <contracts>

- Contracts
- **Reflection (and annotation)**
- Pack indexing
- Forbidding returning references to temporaries

Practical reflection:

What is possible with reflection, and what we will see today:

- Indexing into types (warning: this is compile time !)
- Enum to string (and string to enum)
- Serializing classes
- Option parser from structure
- SOA from Struct

Feature	Standard	Paper(s)	GCC	Clang	MSVC	Xcode
Reflection for C++26 (<meta>)	C++26	P2996	16*	—	—	—
Annotations for Reflection	C++26	P3394	16	—	—	—
Splicing a base class subobject	C++26	P3293	16	—	—	—
define_static_{string,object,array}	C++26	P3491	16	—	—	—
Expansion Statements (template for)	C++26	P1306	16	—	—	—
Function Parameter Reflection in Reflection for C++26	C++26	P3096	16	—	—	—
Error Handling in Reflection	C++26	P3560	16	—	—	—
Library Support for Expansion Statements	C++26	P1789	—	22*	—	—

C++26 reflection [P2996R13]

- Cat ears operator ^^X: obtain informations about anything (type, function, namespace, …)
- Splicing [:x:] : obtain object from informations (not its value)

Operators

```
1  constexpr auto r = ^^int;     // r contains info about 'int'
2  typename[:r:] x = 42;         // Same as: int x = 42;
3  typename[:^^char:] c = '*';   // Same as: char c = '*';
```

C++26 reflection [P2996R13]

Operators on members

```cpp
struct S { unsigned int i, j; };

consteval auto member_number(int n) {
  if (n == 0) return ^^S::i;
  else if (n == 1) return ^^S::j;

  std::unreachable();
}

int main(int argc, char** argv)
{
    S s{0, 0};
    s.[:member_number(1):] = 42;  // Same as: s.j = 42;
}
```

C++26 reflection header <meta>

Most important:

- identifier_of: returns the name of the object (class or member)
- type_of: extract the type of a meta::info object
- extract: extract the value of a meta::info object
- access_context::current/::unpriviledged/::unchecked : access context for members
- nonstatic_data_members_of: list of non static member variables
- enumerators_of: list members of enums
- template_arguments_of: list template parameters
- substitute: allow to set template parameters
- data_member_of: create a variable from a reflected one and additional infos.
- define_aggregates: fills an incomplete type with members
- NO WAY TO ADD FUNCTIONS OR DERIVE A TYPE (FOR NOW ?)!

Enum to string:

EnumToString

```cpp
template<typename E>
  requires std::is_enum_v<E> // C++23 Concepts to filter out non enums :) !
constexpr std::string_view enum_to_string(E&& value) {
    // template for is new: compile time for (unrolled !!!) loops (P1306R5)
    template for (constexpr auto e : define_static_array(enumerators_of(^^E)))
    {
      if (value == [:e:])
        return identifier_of(e);
    }
  return "<unnamed>";
}

```

https://godbolt.org/z/KP6hPbE9o

Serialization:

Serializing

```cpp
template <typename T>
void debugPrint(const T& obj)
{
    constexpr auto ctx = std::meta::access_context::unchecked();
    constexpr auto members = define_static_array(
        nonstatic_data_members_of(^^T, ctx)
    );

    bool first = true;
    std::cout << "(";
    template for (constexpr auto m : define_static_array(members))
    {
        if (!first)
            std::cout << ", ";

        std::cout << std::meta::identifier_of(m) << " = ";
        std::cout << obj.[:m:];
        first = false;
    }

    std::cout << ")";
}
```

Serializing - Result

```cpp
struct A { unsigned int i, j; };
class B  { unsigned int i, j; };

int main(int argc, char** argv)
{
    debugPrint(A{50, 12}); std::cout << '\n';
    debugPrint(B{});
}
```

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
   (i = 50, j = 12)
   (i = 0, j = 0)
```

https://godbolt.org/z/G5Y1cvh8Y

(Simple) Command line parser

Cmdline Parser

```cpp
template<typename Opts>
auto parse_options(std::span<const std::string_view> args) -> Opts {
  Opts opts;

  constexpr auto ctx = std::meta::access_context::current();
  constexpr static auto members = define_static_array(
    nonstatic_data_members_of(^^Opts, ctx)
   );
  template for (constexpr auto dm : members) {
    auto it = std::ranges::find_if(args,[&](std::string_view arg){
        return arg.starts_with("--") && arg.substr(2) == identifier_of(dm);
      });

    if (it == args.end()) continue;
    else if (it + 1 == args.end()) {
      std::print(stderr, "Option {} is missing a value\n", *it);
      std::exit(EXIT_FAILURE);
    }

    using T = typename[:type_of(dm):];
    auto iss = std::ispanstream(it[1]);
    if (iss >> opts.[:dm:]; !iss) {
      std::print(stderr,
           "Failed to parse option {} into a {}\n", *it, display_string_of(^^T)
      );
      std::exit(EXIT_FAILURE);
    }
  }
  return opts;
}
```

Cmdline Parser Usage

```cpp
struct MyOpts {
  std::string file_name = "input.txt";  // Option "--file_name <string>"
  int    count = 1;                     // Option "--count <int>"
};

int main(int argc, char *argv[]) {
  // std::vector<std::string_view> cmdline(argv+1, argv+argc);
  std::vector<std::string_view> cmdline = { "--file_name", "toto.txt", "--count", "3"};
  MyOpts opts = parse_options<MyOpts>(cmdline);

  std::cout << opts.file_name << std::endl;
  std::cout << opts.count << std::endl;
}
```

https://godbolt.org/z/1o4E7fo5T

SOA

SOA

```
template <typename T>
struct SOA {
  struct Storage;

  consteval {
    auto ctx = std::meta::access_context::current();

    std::vector<std::meta::info> old_members = nonstatic_data_members_of(^^T, ctx);
    std::vector<std::meta::info> new_members = {};
    for (std::meta::info member : old_members) {
        auto array_type = substitute(^^std::vector, {type_of(member)});
        auto mem_descr = data_member_spec(array_type, {.name = identifier_of(member)});
        new_members.push_back(mem_descr);
    }

    define_aggregate(^^Storage, new_members);
  }

  Storage s{};
  void push(const T& object)
  {
    auto& [...impl] = s;
    auto& [...vals] = object;
    (impl.push_back(vals), ...);
  }
};
```

SOA - Usage

```
struct Point
{
    int x;
    int y;
    int z;
};

int main(int argc, char** argv)
{
    SOA<Point> points;
    points.push(Point{2, 3, 8});

    std::cout << points.s.x[0] << ", "
              << points.s.y[0] << ", "
              << points.s.z[0];
}
```

https://godbolt.org/z/69axx3qeE

Annotations for reflections (P3394R2)

We can add annotation to fields within [[=]], in order to alter generation through reflection.

Motivation

```cpp
struct Args {
    [[=parser::Help("Name of the person to greet")]]
    [[=parser::Short, =parser::Long]]
    std::string name;

    [[=parser::Help("Number of times to greet")]]
    [[=parser::Short, =parser::Long]]
    int count = 1;
};


int main(int argc, char** argv) {
    Args args = parser::parse<Args>(argc, argv);

    for (int i = 0; i < args.count; ++i) {
        std::cout << "Hello " << args.name << '\n';
    }
}
```

- annotations_of: list all annotation of a reflected field
- annotation_with_type: list all annotations with a type
- With extract, we can recover and use the value of the annotation

FIN