The Power of Reflection

with Facebook's Thrift

Marcelo Juchem
<marcelo@fb.com>
CppCon '16
September 22, 2016

Goals

Goals

- Showcase what can be done with reflection
  - RealWorld™ problems

Goals

- Showcase what can be done with reflection
  - RealWorld™ problems
  - For users
    - see what’s possible, follow examples

Goals

- Showcase what can be done with reflection
  - RealWorld™ problems
  - For users
    - see what’s possible, follow examples
  - For library writers
    - see the problem from yet another perspective

Goals

- Showcase what can be done with reflection
  - RealWorld™ problems
  - For users
    - see what’s possible, follow examples
  - For library writers
    - see the problem from yet another perspective

- Present a working framework that can be used today
  - Where Thrift applies

Goals

- Showcase what can be done with reflection
  - RealWorld™ problems
  - For users
    - see what's possible, follow examples
  - For library writers
    - see the problem from yet another perspective

- Present a working framework that can be used today
  - Where Thrift applies

- We won't look at how reflection library is implemented
  - We're just interested in how to use it
  - You can later check the source code
    - Or come find me after the talk

Agenda

Agenda

- Before we start
- Examples using static reflection
  - Pretty printer
  - Serialization
  - Untyped -> Typed Data conversion
- Enabling reflection in Thrift
- Closing words
- Before we go

Before we start

Assumptions

- Familiarity with **reflection**

Assumptions

- Familiarity with reflection
- High level familiarity with **serialization**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with **search algorithms**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
    - **Binary search**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - **Trie** data structure **(dictionary lookup)**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with **meta-programming**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - **Metafunctions**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - Metafunctions
  - **Template specialization**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - Metafunctions
  - Template specialization
  - **Traits classes**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - Metafunctions
  - Template specialization
  - Traits classes
  - **Type lists**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - Metafunctions
  - Template specialization
  - Traits classes
  - Type lists -> `list<int, bool, double>`

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - Metafunctions
  - Template specialization
  - Traits classes
  - Type lists -> `list<int, bool, double>`
    - **List transforms**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - Metafunctions
  - Template specialization
  - Traits classes
  - Type lists -> `list<int, bool, double>`
    - List transforms -> **`transform<list<int, bool>, add_const>`**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - Metafunctions
  - Template specialization
  - Traits classes
  - Type lists -> `list<int, bool, double>`
    - List transforms -> `transform<list<int, bool>, add_const>`
  - **Compile-time strings**

Assumptions

- Familiarity with reflection
- High level familiarity with serialization
- High level familiarity with search algorithms
  - Binary search
  - Trie data structure (dictionary lookup)
- Familiarity with meta-programming
  - Metafunctions
  - Template specialization
  - Traits classes
  - Type lists -> `list<int, bool, double>`
    - List transforms -> `transform<list<int, bool>, add_const>`
  - Compile-time strings -> `sequence<char, 'h', 'e', 'y'>`

Abbreviations - Scalar list

```
list<
  std::integral_constant<int, 10>,
  std::integral_constant<int, 7>,
  std::integral_constant<int, 15>
>
```

Abbreviations - Scalar list

```
list<
  std::integral_constant<int, 10>,
  std::integral_constant<int, 7>,
  std::integral_constant<int, 15>
>
```

Abbreviated as

```
-> list<10, 7, 15>
```

Abbreviations - String list

```
list<
  sequence<char, 'h', 'e', 'l', 'l', 'o'>,
  sequence<char, 'w', 'o', 'r', 'l', 'd'>
>
```

Abbreviations - String list

```
list<
  sequence<char, 'h', 'e', 'l', 'l', 'o'>,
  sequence<char, 'w', 'o', 'r', 'l', 'd'>
>
```

Abbreviated as

```
-> list<"hello", "world">
```

JSON Ugly Printer

JSON Printer - Public Interface

```
template <typename T>
void print(T const &what);
```

JSON Printer - Implementation

```
template <typename T>
void print(T const &what) {

}
```

JSON Printer - Implementation

```
template <typename T>
void print(T const &what) {
          reflect_type_class<T>
}
```

Type Class

```
reflect_type_class<T>

type_class:

    integral                    list
    floating_point              set
    enumeration                 map
    structure                   variant

    ...
```

JSON Printer - Implementation

```
template <typename T>
void print(T const &what) {
  printer<reflect_type_class<T>>
}
```

JSON Printer - Implementation

```
template <typename T>
void print(T const &what) {
  printer<reflect_type_class<T>>::print(what);
}
```

JSON Printer - General case

```
template <typename TypeClass>
struct printer {








};
```

JSON Printer - General case

```
template <typename TypeClass>
struct printer {
  template <typename T>
  static void print(T const &what) {

  }





};
```

JSON Printer - General case

```
template <typename TypeClass>
struct printer {
  template <typename T>
  static void print(T const &what) {
    std::cout << what;
  }





};
```

JSON Printer - Booleans

```
template <typename TypeClass>
struct printer {
  template <typename T>
  static void print(T const &what) {
    std::cout << what;
  }

                    bool
                          true      false

};
```

JSON Printer - Booleans

```
template <typename TypeClass>
struct printer {
  template <typename T>
  static void print(T const &what) {
    std::cout << what;
  }

  static void print(bool const what) {
    std::cout << (what ? "true" : "false");
  }
};
```

JSON Printer - String

```
"string value"
```

JSON Printer - String

```
template <>
struct printer<type_class::string> {




};
```

JSON Printer - String

```
template <>
struct printer<type_class::string> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '"' << what << '"';
  }
};
```

JSON Printer - List

```
[
  "value 1",
  "value 2",
  "value 3",
  "value 4"
]
```

JSON Printer - List

```
template <                              >
struct printer<type_class::list<                      >> {









};
```

JSON Printer - List

```
template <typename ValueTypeClass>
struct printer<type_class::list<ValueTypeClass>> {









};
```

JSON Printer - List

```
template <typename ValueTypeClass>
struct printer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '[';







    std::cout << ']';
  }
};
```

JSON Printer - List

```
template <typename ValueTypeClass>
struct printer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '[';

    for (auto const &i: what) {




    }
    std::cout << ']';
  }
};
```

JSON Printer - List

```
template <typename ValueTypeClass>
struct printer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '[';
    bool first = true;
    for (auto const &i: what) {
      if (first) { first = false; } else { std::cout << ','; }


    }
    std::cout << ']';
  }
};
```

JSON Printer - List

```
template <typename ValueTypeClass>
struct printer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '[';
    bool first = true;
    for (auto const &i: what) {
      if (first) { first = false; } else { std::cout << ','; }

      printer<ValueTypeClass>::print(i);
    }
    std::cout << ']';
  }
};
```

JSON Printer - Set

```
[
  "value 1",
  "value 2",
  "value 3",
  "value 4"
]
```

JSON Printer - Set

```
template <typename ValueTypeClass>
struct printer<type_class::set<ValueTypeClass>>
```

JSON Printer - Set

```
template <typename ValueTypeClass>
struct printer<type_class::set<ValueTypeClass>>:
  public printer<type_class::list<ValueTypeClass>>
{};
```

JSON Printer - Map

```
{
  "key 1": "value 1",
  "key 2": "value 2",
  "key 3": "value 3",
  "key 4": "value 4"
}
```

JSON Printer - Map

```
template <                                                   >
struct printer<type_class::map<                              >> {
  template <typename T>
  static void print(T const &what) {









  }
};
```

JSON Printer - Map

```
template <typename KeyTypeClass, typename ValueTypeClass>
struct printer<type_class::map<KeyTypeClass, ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {









  }
};
```

JSON Printer - Map

```
template <typename KeyTypeClass, typename ValueTypeClass>
struct printer<type_class::map<KeyTypeClass, ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';









    std::cout << '}';
  }
};
```

JSON Printer - Map

```
template <typename KeyTypeClass, typename ValueTypeClass>
struct printer<type_class::map<KeyTypeClass, ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';

    for (auto const &i: what) {





    }
    std::cout << '}';
  }
};
```

JSON Printer - Map

```
template <typename KeyTypeClass, typename ValueTypeClass>
struct printer<type_class::map<KeyTypeClass, ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    bool first = true;
    for (auto const &i: what) {
      if (first) { first = false; } else { std::cout << ','; }




    }
    std::cout << '}';
  }
};
```

JSON Printer - Map

```
template <typename KeyTypeClass, typename ValueTypeClass>
struct printer<type_class::map<KeyTypeClass, ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    bool first = true;
    for (auto const &i: what) {
      if (first) { first = false; } else { std::cout << ','; }

      std::cout << ':';

    }
    std::cout << '}';
  }
};
```

JSON Printer - Map

```
template <typename KeyTypeClass, typename ValueTypeClass>
struct printer<type_class::map<KeyTypeClass, ValueTypeClass>> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    bool first = true;
    for (auto const &i: what) {
      if (first) { first = false; } else { std::cout << ','; }
      printer<KeyTypeClass>::print(i.first);
      std::cout << ':';
      printer<ValueTypeClass>::print(i.second);
    }
    std::cout << '}';
  }
};
```

JSON Printer - Enum

```
"enum_field_name"
```

JSON Printer - Enum

```
template <>
struct printer<type_class::enumeration> {
  template <typename T>
  static void print(T const &what) {

  }
};
```

JSON Printer - Enum

```
template <>
struct printer<type_class::enumeration> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '"' <<                            << '"';
  }
};
```

JSON Printer - Enum

```
template <>
struct printer<type_class::enumeration> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '"' << fatal::enum_to_string(what) << '"';
  }
};
```

JSON Printer - Struct

```
{
  "member1": "value 1",
  "member2": 2,
  "member3": [ 3, 5 ],
  "member4": "value 4"
}
```

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {








  }
};
```

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';







    std::cout << '}';
  }
};
```

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using struct_info = reflect_struct<T>;







    std::cout << '}';
  }
};
```

Reflected struct

```
struct reflected_struct {
  using name = ...;
  using members = list<...>;
  using annotations = ...;
  ...
};
```

Reflected struct

```
struct reflected_struct {
  using name = …;
  using members = list<…>;
  using annotations = …;
  …
};

struct reflected_struct_data_member {
  using name = …;
  using type = …;
  using getter = …;
  using annotations = …;
};
```

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using struct_info = reflect_struct<T>;







    std::cout << '}';
  }
};
```

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using struct_info = reflect_struct<T>;
    using members_info = typename struct_info::members;






    std::cout << '}';
  }
};
```

Iterating type lists

```
list<int, bool, double, float>
```

Iterating type lists

```
foreach<list<int, bool, double, float>>(


);
```

Iterating type lists

```
foreach<list<int, bool, double, float>>(
  visitor

);
```

Iterating type lists

```
foreach<list<int, bool, double, float>>(
  visitor

);

-> visitor(indexed<int, 0>()                    )
```

Iterating type lists

```
foreach<list<int, bool, double, float>>(
  visitor,
  additional_args...
);

-> visitor(indexed<int, 0>(), additional_args...)
```

Iterating type lists

```
foreach<list<int, bool, double, float>>(
  visitor,
  additional_args...
);

-> visitor(indexed<int, 0>(), additional_args...)
   visitor(indexed<bool, 1>(), additional_args...)
```

Iterating type lists

```
foreach<list<int, bool, double, float>>(
  visitor,
  additional_args...
);

-> visitor(indexed<int, 0>(), additional_args...)
   visitor(indexed<bool, 1>(), additional_args...)
   visitor(indexed<double, 2>(), additional_args...)
```

Iterating type lists

```
foreach<list<int, bool, double, float>>(
  visitor,
  additional_args...
);

-> visitor(indexed<int, 0>(), additional_args...)
   visitor(indexed<bool, 1>(), additional_args...)
   visitor(indexed<double, 2>(), additional_args...)
   visitor(indexed<float, 3>(), additional_args...)
```

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using struct_info = reflect_struct<T>;
    using members_info = typename struct_info::members;




    std::cout << '}';
  }
};
```

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using struct_info = reflect_struct<T>;
    using members_info = typename struct_info::members;

    fatal::foreach<members_info>(


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

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using struct_info = reflect_struct<T>;
    using members_info = typename struct_info::members;

    fatal::foreach<members_info>(
      struct_member_printer()

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

JSON Printer - Struct

```
template <>
struct printer<type_class::structure> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using struct_info = reflect_struct<T>;
    using members_info = typename struct_info::members;

    fatal::foreach<members_info>(
      struct_member_printer(),
      what
    );
    std::cout << '}';
  }
};
```

JSON Printer - Struct member

```
struct struct_member_printer {









};
```

JSON Printer - Struct member

```
struct struct_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {







  }
};
```

JSON Printer - Struct member

```
struct struct_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {
    if (Index) { std::cout << ','; }







  }
};
```

JSON Printer - Struct member

```
struct struct_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {
    if (Index) { std::cout << ','; }

    auto const name =                       Member::name   ;
    std::cout << '"' <<  name << "\":";




  }
};
```

JSON Printer - Struct member

```
struct struct_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {
    if (Index) { std::cout << ','; }

    auto const name = fatal::z_data<typename Member::name>();
    std::cout << '"' <<  name << "\":";



  }
};
```

JSON Printer - Struct member

```
struct struct_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {
    if (Index) { std::cout << ','; }

    auto const name = fatal::z_data<typename Member::name>();
    std::cout << '"' <<  name << "\":";

    auto const &value = Member::getter::ref(what);

  }
};
```

JSON Printer - Struct member

```
struct struct_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {
    if (Index) { std::cout << ','; }

    auto const name = fatal::z_data<typename Member::name>();
    std::cout << '"' <<  name << "\":";

    auto const &value = Member::getter::ref(what);
    printer<typename Member::type_class>::print(value);
  }
};
```

JSON Printer - Variant

```
{
  "which": "the value"
}
```

Variant traits

```
struct variant_traits {
  using name = ...;
  using id = ...;

  ...
};
```

Variant traits

```
struct variant_traits {
  using name = ...;
  using id = ...;

  ...
};
```

```
variant_traits<SomeVariant>
```

Variant traits

```
struct variant_traits {
  using name = ...;
  using id = ...;

  ...
};

using info = variant_traits<SomeVariant>;
                    info::name
```

Variant traits

```
struct variant_traits {
  using name = ...;
  using id = ...;

  ...
};

using info = variant_traits<SomeVariant>;
std::cout << z_data<info::name>();

-> "SomeVariant"
```

Variant traits - Reflecting members

```
struct variant_traits {
  using name = ...;
  using id = ...;
  using descriptors = ...;
  ...
};
```

Variant member descriptors

```
struct variant_member_descriptor {
  using type = ...;
  using id = ...;
  static auto get(T &variant);
  static void set(T &variant, Args &&...args);
  ...
};
```

Variant member descriptors

```
struct variant_member_descriptor {
  using type = ...;
  using id = ...;
  static auto get(T &variant);
  static void set(T &variant, Args &&...args);
  ...
};

auto variant = some_variant;

  std::cout << Descriptor::get(variant);
```

Variant member descriptors

```
struct variant_member_descriptor {
  using type = ...;
  using id = ...;
  static auto get(T &variant);
  static void set(T &variant, Args &&...args);
  ...
};

auto variant = some_variant;
if (variant.getType() == Descriptor::id::value) {
  std::cout << Descriptor::get(variant);
}
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {









  }
};
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';









    std::cout << '}';
  }
};
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';






      what.getType()

    std::cout << '}';
  }
};
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';

      fatal::variant_traits<T>::descriptors




      what.getType()

    std::cout << '}';
  }
};
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };
```

```
  list<some_type<1, 99>, some_type<2, 42>>
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

int needle = 2;


  list<some_type<1, 99>, some_type<2, 42>>
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

int needle = 2;
```

```
  list<some_type<1, 99>, some_type<2, 42>>
  get_type::foo
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

int needle = 2;

sorted_search<
  list<some_type<1, 99>, some_type<2, 42>>,
  get_type::foo
>(                                    )
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

int needle = 2;

sorted_search<
  list<some_type<1, 99>, some_type<2, 42>>,
  get_type::foo
>(needle                              )
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

int needle = 2;

sorted_search<
  list<some_type<1, 99>, some_type<2, 42>>,
  get_type::foo
>(needle, visitor                    )
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

int needle = 2;

sorted_search<
  list<some_type<1, 99>, some_type<2, 42>>,
  get_type::foo
>(needle, visitor                    )

-> visitor(
      tag<some_type<2, 42>>()

   )
```

Indexing types at runtime

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

int needle = 2;

sorted_search<
  list<some_type<1, 99>, some_type<2, 42>>,
  get_type::foo
>(needle, visitor, additional_args...)

-> visitor(
      tag<some_type<2, 42>>(),
      additional_args...
   )
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';

      fatal::variant_traits<T>::descriptors




      what.getType()

    std::cout << '}';
  }
};
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';

      fatal::variant_traits<T>::descriptors


    fatal::sorted_search<                    fatal::get_type::id>(
      what.getType()
    );
    std::cout << '}';
  }
};
```

Sorting lists

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };
```

```
  list<some_type<2, 42>, some_type<1, 99>>
```

Sorting lists

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };


  list<some_type<2, 42>, some_type<1, 99>>

  get_type::foo
```

Sorting lists

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

sort<
  list<some_type<2, 42>, some_type<1, 99>>,

  get_type::foo
>
```

Sorting lists

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

sort<
  list<some_type<2, 42>, some_type<1, 99>>,
  less,
  get_type::foo
>
```

Sorting lists

```
template <typename Foo, typename Bar>
struct some_type { using foo = Foo; using bar = Bar; };

sort<
  list<some_type<2, 42>, some_type<1, 99>>,
  less,
  get_type::foo
>

-> list<some_type<1, 99>, some_type<2, 42>>
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';

      fatal::variant_traits<T>::descriptors


    fatal::sorted_search<                    fatal::get_type::id>(
      what.getType()
    );
    std::cout << '}';
  }
};
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
                                    fatal::sort<
      fatal::variant_traits<T>::descriptors,
                         fatal::get_type::id
    >
    fatal::sorted_search<                   fatal::get_type::id>(
      what.getType()
    );
    std::cout << '}';
  }
};
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using members_by_id = fatal::sort<
      fatal::variant_traits<T>::descriptors,
      fatal::less, fatal::get_type::id
    >;
    fatal::sorted_search<members_by_id, fatal::get_type::id>(
      what.getType()
    );
    std::cout << '}';
  }
};
```

JSON Printer - Variant

```
template <>
struct printer<type_class::variant> {
  template <typename T>
  static void print(T const &what) {
    std::cout << '{';
    using members_by_id = fatal::sort<
      fatal::variant_traits<T>::descriptors,
      fatal::less, fatal::get_type::id
    >;
    fatal::sorted_search<members_by_id, fatal::get_type::id>(
      what.getType(), variant_member_printer(), what
    );
    std::cout << '}';
  }
};
```

JSON Printer - Variant member

```
struct variant_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {







  }
};
```

JSON Printer - Variant member

```
struct variant_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {
    auto const name = fatal::enum_to_string(what.getType());
    std::cout << '"' << name << "\":";







  }
};
```

JSON Printer - Variant member

```
struct variant_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {
    auto const name = fatal::enum_to_string(what.getType());
    std::cout << '"' << name << "\":";

    auto const &value = Member::get(what);



  }
};
```

JSON Printer - Variant member

```
struct variant_member_printer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what
  ) const {
    auto const name = fatal::enum_to_string(what.getType());
    std::cout << '"' << name << "\":";

    auto const &value = Member::get(what);

    using type_class = typename Member::metadata::type_class;
    printer<type_class>::print(value);
  }
};
```

Sample Output - Using an external formatter

```json
{
  "int_field": 98,
  "bool_field": true,
  "floating_point_field": 7.2,
  "string_field": "HELLO, WORLD",
  "struct_list_field": [
    { "the_int": 0, "the_enum": "field0" },
    { "the_int": 1, "the_enum": "field1" },
    { "the_int": 2, "the_enum": "field2" }
  ],
  "map_field": {
    "hard": false,
    "works": true,
    "worth it": true
  },
  "set_field": [],
  "variant_field": { "floating_point_data": 0.5 }
}
```

Serialization

Serialization - Public Interface

```
template <typename T>
void serialize(T const &what, data_writer &writer);



template <typename T>
void deserialize(T &out, data_reader &reader);
```

Serialization - Data Writer

```
struct data_writer {
  template <typename T>
  void write_raw(T const &value);

  template <typename T>
  void write_string(T const *data, std::size_t size);
};
```

Serialization - Data Reader

```
struct data_reader {
  template <typename T>
  T read_raw();

  template <typename T>
  void read_string(std::basic_string<T> &out);
};
```

Serialization - Implementation

```
template <typename T>
void serialize(T const &what, data_writer &writer) {

}

template <typename T>
void deserialize(T &out, data_reader &reader) {

}
```

Serialization - Implementation

```
template <typename T>
void serialize(T const &what, data_writer &writer) {
            reflect_type_class<T>            what
}

template <typename T>
void deserialize(T &out, data_reader &reader) {
            reflect_type_class<T>              out
}
```

Serialization - Implementation

```
template <typename T>
void serialize(T const &what, data_writer &writer) {
  serializer<reflect_type_class<T>>                what
}

template <typename T>
void deserialize(T &out, data_reader &reader) {
  serializer<reflect_type_class<T>>                  out
}
```

Serialization - Implementation

```
template <typename T>
void serialize(T const &what, data_writer &writer) {
  serializer<reflect_type_class<T>>::serialize(what, writer);
}

template <typename T>
void deserialize(T &out, data_reader &reader) {
  serializer<reflect_type_class<T>>::deserialize(out, reader);
}
```

Serializer - General case serialize

```
template <typename TypeClass>
struct serializer {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {

  }





};
```

Serializer - General case serialize

```
template <typename TypeClass>
struct serializer {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what);
  }







};
```

Serializer - General case deserialize

```
template <typename TypeClass>
struct serializer {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what);
  }

  template <typename T>
  static void deserialize(T &out, data_reader &reader) {

  }
};
```

Serializer - General case deserialize

```
template <typename TypeClass>
struct serializer {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what);
  }

  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    out = reader.read_raw<T>();
  }
};
```

Serializer - String serialize

```
template <>
struct serializer<type_class::string> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {

  }






};
```

Serializer - String serialize

```
template <>
struct serializer<type_class::string> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_string(what.data(), what.size());
  }






};
```

Serializer - String deserialize

```
template <>
struct serializer<type_class::string> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_string(what.data(), what.size());
  }

  template <typename T>
  static void deserialize(T &out, data_reader &reader) {

  }
};
```

Serializer - String deserialize

```
template <>
struct serializer<type_class::string> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_string(what.data(), what.size());
  }

  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    reader.read_string(out);
  }
};
```

Serializer - Enum serialize

```
template <>
struct serializer<type_class::enumeration> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {


  }






};
```

Serializer - Enum serialize

```
template <>
struct serializer<type_class::enumeration> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    auto const name = fatal::enum_to_string(what);

  }







};
```

Serializer - Enum serialize

```
template <>
struct serializer<type_class::enumeration> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    auto const name = fatal::enum_to_string(what);
    writer.write_string(name, std::strlen(name));
  }









};
```

Serializer - Enum deserialize

```
template <>
struct serializer<type_class::enumeration> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    auto const name = fatal::enum_to_string(what);
    writer.write_string(name, std::strlen(name));
  }

  template <typename T>
  static void deserialize(T &out, data_reader &reader) {



  }
};
```

Serializer - Enum deserialize

```
template <>
struct serializer<type_class::enumeration> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    auto const name = fatal::enum_to_string(what);
    writer.write_string(name, std::strlen(name));
  }

  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string name;
    reader.read_string(name);

  }
};
```

Serializer - Enum deserialize

```
template <>
struct serializer<type_class::enumeration> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    auto const name = fatal::enum_to_string(what);
    writer.write_string(name, std::strlen(name));
  }

  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string name;
    reader.read_string(name);
          fatal::enum_traits<T>::parse(name);
  }
};
```

Serializer - Enum deserialize

```
template <>
struct serializer<type_class::enumeration> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    auto const name = fatal::enum_to_string(what);
    writer.write_string(name, std::strlen(name));
  }

  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string name;
    reader.read_string(name);
    out = fatal::enum_traits<T>::parse(name);
  }
};
```

Serializer - List serialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {






  }
};
```

Serializer - List serialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());





  }
};
```

Serializer - List serialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());

    for (auto const &i: what) {

    }
  }
};
```

Serializer - List serialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());

    for (auto const &i: what) {
      serializer<ValueTypeClass>::serialize(i, writer);
    }
  }
};
```

Serializer - List deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {









  }
};
```

Serializer - List deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();









  }
};
```

Serializer - List deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {






    }
  }
};
```

Serializer - List deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      out.emplace_back();





    }
  }
};
```

Serializer - List deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::list<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      out.emplace_back();
      serializer<ValueTypeClass>::deserialize(
        out.back(),
        reader
      );
    }
  }
};
```

Serializer - Set serialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {





  }
};
```

Serializer - Set serialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());





  }
};
```

Serializer - Set serialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());

    for (auto const &i: what) {

    }
  }
};
```

Serializer - Set serialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());

    for (auto const &i: what) {
      serializer<ValueTypeClass>::serialize(i, writer);
    }
  }
};
```

Serializer - Set deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {







  }
};
```

Serializer - Set deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();







  }
};
```

Serializer - Set deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {




    }
  }
};
```

Serializer - Set deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      typename T::value_type value;


    }
  }
};
```

Serializer - Set deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      typename T::value_type value;
      serializer<ValueTypeClass>::deserialize(value, reader);

    }
  }
};
```

Serializer - Set deserialize

```
template <typename ValueTypeClass>
struct serializer<type_class::set<ValueTypeClass>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      typename T::value_type value;
      serializer<ValueTypeClass>::deserialize(value, reader);
      out.emplace(std::move(value));
    }
  }
};
```

Serializer - Map serialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {








  }
};
```

Serializer - Map serialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());







  }
};
```

Serializer - Map serialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());

    for (auto const &i: what) {


    }
  }
};
```

Serializer - Map serialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());

    for (auto const &i: what) {
      serializer<KeyTC>::serialize(i.first, writer);

    }
  }
};
```

Serializer - Map serialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    writer.write_raw(what.size());

    for (auto const &i: what) {
      serializer<KeyTC>::serialize(i.first, writer);
      serializer<ValueTC>::serialize(i.second, writer);
    }
  }
};
```

Serializer - Map deserialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {







  }
};
```

Serializer - Map deserialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();









  }
};
```

Serializer - Map deserialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {






    }
  }
};
```

Serializer - Map deserialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      typename T::key_type key;





    }
  }
};
```

Serializer - Map deserialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      typename T::key_type key;
      serializer<KeyTC>::deserialize(key, reader);



    }
  }
};
```

Serializer - Map deserialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      typename T::key_type key;
      serializer<KeyTC>::deserialize(key, reader);

              out[std::move(key)];

    }
  }
};
```

Serializer - Map deserialize

```
template <typename KeyTC, typename ValueTC>
struct serializer<type_class::map<KeyTC, ValueTC>> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    auto count = reader.read_raw<typename T::size_type>();

    while (count--) {
      typename T::key_type key;
      serializer<KeyTC>::deserialize(key, reader);

      auto &value = out[std::move(key)];
      serializer<ValueTC>::deserialize(value, reader);
    }
  }
};
```

Serializer - Struct serialize

```
template <>
struct serializer<type_class::structure> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {



  }







};
```

Serializer - Struct serialize

```
template <>
struct serializer<type_class::structure> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
                                          reflect_struct<T>::members



  }







};
```

Serializer - Struct serialize

```
template <>
struct serializer<type_class::structure> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    fatal::foreach<typename reflect_struct<T>::members>(

    );
  }







};
```

Serializer - Struct serialize

```
template <>
struct serializer<type_class::structure> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    fatal::foreach<typename reflect_struct<T>::members>(
      struct_member_serializer(), what, writer
    );
  }






};
```

Serializer - Struct serialize member

```
struct struct_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what,
    data_writer &writer
  ) const {




  }
};
```

Serializer - Struct serialize member

```
struct struct_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what,
    data_writer &writer
  ) const {
                        Member::getter::ref(what);


  }
};
```

Serializer - Struct serialize member

```
struct struct_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what,
    data_writer &writer
  ) const {
    auto const &value = Member::getter::ref(what);

    serializer<          >::serialize(value, writer);
  }
};
```

Serializer - Struct serialize member

```
struct struct_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what,
    data_writer &writer
  ) const {
    auto const &value = Member::getter::ref(what);
                                   Member::type_class
    serializer<          >::serialize(value, writer);
  }
};
```

Serializer - Struct serialize member

```
struct struct_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &what,
    data_writer &writer
  ) const {
    auto const &value = Member::getter::ref(what);
    using type_class = typename Member::type_class;
    serializer<type_class>::serialize(value, writer);
  }
};
```

Serializer - Struct deserialize

```
template <>
struct serializer<type_class::structure> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    fatal::foreach<typename reflect_struct<T>::members>(
      struct_member_serializer(), what, writer
    );
  }
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {




  }
};
```

Serializer - Struct deserialize

```
template <>
struct serializer<type_class::structure> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    fatal::foreach<typename reflect_struct<T>::members>(
      struct_member_serializer(), what, writer
    );
  }
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
                         reflect_struct<T>::members



  }
};
```

Serializer - Struct deserialize

```
template <>
struct serializer<type_class::structure> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    fatal::foreach<typename reflect_struct<T>::members>(
      struct_member_serializer(), what, writer
    );
  }
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    fatal::foreach<typename reflect_struct<T>::members>(

    );
  }
};
```

Serializer - Struct deserialize

```
template <>
struct serializer<type_class::structure> {
  template <typename T>
  static void serialize(T const &what, data_writer &writer) {
    fatal::foreach<typename reflect_struct<T>::members>(
      struct_member_serializer(), what, writer
    );
  }
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    fatal::foreach<typename reflect_struct<T>::members>(
      struct_member_deserializer(), out, reader
    );
  }
};
```

Serializer - Struct deserialize member

```
struct struct_member_deserializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T &out,
    data_reader &reader
  ) const {




  }
};
```

Serializer - Struct deserialize member

```
struct struct_member_deserializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T &out,
    data_reader &reader
  ) const {
                    Member::getter::ref(out);



  }
};
```

Serializer - Struct deserialize member

```
struct struct_member_deserializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T &out,
    data_reader &reader
  ) const {
    auto &member_ref = Member::getter::ref(out);

    serializer<          >::deserialize(member_ref, reader);
  }
};
```

Serializer - Struct deserialize member

```
struct struct_member_deserializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T &out,
    data_reader &reader
  ) const {
    auto &member_ref = Member::getter::ref(out);
                               Member::type_class
    serializer<          >::deserialize(member_ref, reader);
  }
};
```

Serializer - Struct deserialize member

```
struct struct_member_deserializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T &out,
    data_reader &reader
  ) const {
    auto &member_ref = Member::getter::ref(out);
    using type_class = typename Member::type_class;
    serializer<type_class>::deserialize(member_ref, reader);
  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {









  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {






      v.getType()

  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {

      fatal::variant_traits<T>::descriptors





      v.getType()

  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {
                                   fatal::sort<
      fatal::variant_traits<T>::descriptors,
      fatal::less, fatal::get_type::id
    >



      v.getType()

  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {
                                                fatal::sort<
      fatal::variant_traits<T>::descriptors,
      fatal::less, fatal::get_type::id
    >
                      fatal::sorted_search<
                            fatal::get_type::id
    >(v.getType()                                                  )

  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {
    using members_by_id = fatal::sort<
      fatal::variant_traits<T>::descriptors,
      fatal::less, fatal::get_type::id
    >;
                fatal::sorted_search<
      members_by_id, fatal::get_type::id
    >(v.getType()                                         )

  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {
    using members_by_id = fatal::sort<
      fatal::variant_traits<T>::descriptors,
      fatal::less, fatal::get_type::id
    >;
              fatal::sorted_search<
      members_by_id, fatal::get_type::id
    >(v.getType(), variant_member_serializer(), v, writer);

  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {
    using members_by_id = fatal::sort<
      fatal::variant_traits<T>::descriptors,
      fatal::less, fatal::get_type::id
    >;
    bool found = fatal::sorted_search<
      members_by_id, fatal::get_type::id
    >(v.getType(), variant_member_serializer(), v, writer);
    if (!found) {                              }
  }
};
```

Serializer - Variant serialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void serialize(T const &v, data_writer &writer) {
    using members_by_id = fatal::sort<
      fatal::variant_traits<T>::descriptors,
      fatal::less, fatal::get_type::id
    >;
    bool found = fatal::sorted_search<
      members_by_id, fatal::get_type::id
    >(v.getType(), variant_member_serializer(), v, writer);
    if (!found) { writer.write_string("", 0); }
  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {







  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
                        Member::metadata::name





  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
                         Member::metadata::name;
    writer.write_string(

    );



  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
                            Member::metadata::name;
    writer.write_string(
      fatal::z_data<    >()
    );




  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
                          Member::metadata::name;
    writer.write_string(
      fatal::z_data<    >(), fatal::size<    >::value
    );




  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
    using name = typename Member::metadata::name;
    writer.write_string(
      fatal::z_data<name>(), fatal::size<name>::value
    );




  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
    using name = typename Member::metadata::name;
    writer.write_string(
      fatal::z_data<name>(), fatal::size<name>::value
    );
                            Member::get(variant);


  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
    using name = typename Member::metadata::name;
    writer.write_string(
      fatal::z_data<name>(), fatal::size<name>::value
    );
    auto const &value = Member::get(variant);

    serializer<          >::serialize(value, writer);
  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
    using name = typename Member::metadata::name;
    writer.write_string(
      fatal::z_data<name>(), fatal::size<name>::value
    );
    auto const &value = Member::get(variant);
                                   Member::metadata::type_class
    serializer<         >::serialize(value, writer);
  }
};
```

Serializer - Variant serialize member

```
struct variant_member_serializer {
  template <typename Member, std::size_t Index, typename T>
  void operator ()(
    fatal::indexed<Member, Index>,
    T const &variant, data_writer &writer
  ) const {
    using name = typename Member::metadata::name;
    writer.write_string(
      fatal::z_data<name>(), fatal::size<name>::value
    );
    auto const &value = Member::get(variant);
    using type_class = typename Member::metadata::type_class;
    serializer<type_class>::serialize(value, writer);
  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {









  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;









  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);







  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);
                          fatal::variant_traits<T>::id







  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);
    using id_type = typename fatal::variant_traits<T>::id;
                          fatal::enum_traits<id_type>::names





  }
};
```

String -> Compile-time String

```cpp
std::string needle = "hello";
```

String -> Compile-time String

```
std::string needle = "hello";

          list<"hello", "world">
```

String -> Compile-time String

```
std::string needle = "hello";

trie_find<list<"hello", "world">>(



)
```

String -> Compile-time String

```
std::string needle = "hello";

trie_find<list<"hello", "world">>(
  needle.begin(), needle.end()


)
```

String -> Compile-time String

```
std::string needle = "hello";

trie_find<list<"hello", "world">>(
  needle.begin(), needle.end(),
  visitor

)
```

String -> Compile-time String

```
std::string needle = "hello";

trie_find<list<"hello", "world">>(
  needle.begin(), needle.end(),
  visitor

)

-> visitor(
      tag<"hello">()

   )
```

String -> Compile-time String

```
std::string needle = "hello";

trie_find<list<"hello", "world">>(
  needle.begin(), needle.end(),
  visitor,
  additional_args...
)

-> visitor(
      tag<"hello">(),
      additional_args...
   )
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);
    using id_type = typename fatal::variant_traits<T>::id;
                           fatal::enum_traits<id_type>::names




  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);
    using id_type = typename fatal::variant_traits<T>::id;
    using names = typename fatal::enum_traits<id_type>::names;
                 fatal::trie_find<names>(



    );

  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);
    using id_type = typename fatal::variant_traits<T>::id;
    using names = typename fatal::enum_traits<id_type>::names;
                 fatal::trie_find<names>(
      which.begin(), which.end()

    );

  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);
    using id_type = typename fatal::variant_traits<T>::id;
    using names = typename fatal::enum_traits<id_type>::names;
                 fatal::trie_find<names>(
      which.begin(), which.end(),
      variant_member_deserializer(), out, reader
    );

  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);
    using id_type = typename fatal::variant_traits<T>::id;
    using names = typename fatal::enum_traits<id_type>::names;
    bool found = fatal::trie_find<names>(
      which.begin(), which.end(),
      variant_member_deserializer(), out, reader
    );
    if (!found) {                                           }
  }
};
```

Serializer - Variant deserialize

```
template <>
struct serializer<type_class::variant> {
  template <typename T>
  static void deserialize(T &out, data_reader &reader) {
    std::string which;
    reader.read_string(which);
    using id_type = typename fatal::variant_traits<T>::id;
    using names = typename fatal::enum_traits<id_type>::names;
    bool found = fatal::trie_find<names>(
      which.begin(), which.end(),
      variant_member_deserializer(), out, reader
    );
    if (!found) { fatal::variant_traits<T>::clear(out); }
  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {







  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {

          member




  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {

          member                              by_name<Name>




  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {
    using info = reflect_variant<T>;
          member            info::template by_name<Name>





  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {
    using info = reflect_variant<T>;
    using member = typename info::template by_name<Name>;
    member::set(out);




  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {
    using info = reflect_variant<T>;
    using member = typename info::template by_name<Name>;
    member::set(out);


    serializer<          >::deserialize(       reader);
  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {
    using info = reflect_variant<T>;
    using member = typename info::template by_name<Name>;
    member::set(out);
                  member::get(out)

    serializer<            >::deserialize(       reader);
  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {
    using info = reflect_variant<T>;
    using member = typename info::template by_name<Name>;
    member::set(out);
    auto &value = member::get(out);

    serializer<          >::deserialize(value, reader);
  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {
    using info = reflect_variant<T>;
    using member = typename info::template by_name<Name>;
    member::set(out);
    auto &value = member::get(out);
                                   member::metadata::type_class
    serializer<          >::deserialize(value, reader);
  }
};
```

Serializer - Variant deserialize member

```
struct variant_member_deserializer {
  template <typename Name, typename T>
  void operator ()(
    fatal::tag<Name>,
    T &out, data_reader &reader
  ) const {
    using info = reflect_variant<T>;
    using member = typename info::template by_name<Name>;
    member::set(out);
    auto &value = member::get(out);
    using type_class = typename member::metadata::type_class;
    serializer<type_class>::deserialize(value, reader);
  }
};
```

Untyped data translation

Untyped data - Thrift IDL

```
typedef map<string, string> legacy_config
```

Untyped data - Thrift IDL

```
typedef map<string, string> legacy_config

const legacy_config example = {
  "host-name": "localhost",
  "host-port": "80",
  "client-name": "my_client",
  "socket-send-timeout": "100",
  "socket-receive-timeout": "120",
  "transport-frame-size": "1024",
  "apply-compression": "1",
  "log-sampling-rate": ".01"
}
```

Typed data - Thrift IDL

```
typedef map<string, string> legacy_config

struct flat_config {
  1: string host_name
  2: i16 host_port
  3: string client_name
  4: i32 send_timeout
  5: i32 receive_timeout
  6: i32 frame_size
  7: bool compress
  8: double log_rate
}
```

Typed data - Desired mapping

```
typedef map<string, string> legacy_config

struct flat_config {
  1: string host_name ->            "host-name"
  2: i16 host_port ->            "host-port"
  3: string client_name ->            "client-name"
  4: i32 send_timeout ->            "socket-send-timeout"
  5: i32 receive_timeout ->            "socket-receive-timeout"
  6: i32 frame_size ->            "transport-frame-size"
  7: bool compress ->            "apply-compression"
  8: double log_rate ->            "log-sampling-rate"
}
```

Typed data - Annotated IDL

```
typedef map<string, string> legacy_config

struct flat_config {
  1: string host_name (property = "host-name")
  2: i16 host_port (property = "host-port")
  3: string client_name (property = "client-name")
  4: i32 send_timeout (property = "socket-send-timeout")
  5: i32 receive_timeout (property = "socket-receive-timeout")
  6: i32 frame_size (property = "transport-frame-size")
  7: bool compress (property = "apply-compression")
  8: double log_rate (property = "log-sampling-rate")
}
```

Untyped data translation - Public Interface

```
void translate(legacy_config const &from, flat_config &to);

void translate(flat_config const &from, legacy_config &to);
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {










}
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {




   for (auto const &i: from) {





   }
}
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {





  for (auto const &i: from) {
    fatal::trie_find<          >(
      i.first.begin(), i.first.end()




    );
  }
}
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {

    reflect_struct<flat_config>::members




  for (auto const &i: from) {
    fatal::trie_find<          >(
      i.first.begin(), i.first.end()




    );
  }
}
```

Untyped data translation - Get property

annotations::values

```
struct flat_config {
  1: string host_name (property = "host-name")
  2: i16 host_port (property = "host-port")
  3: string client_name (property = "client-name")
  4: i32 send_timeout (property = "socket-send-timeout")
  5: i32 receive_timeout (property = "socket-receive-timeout")
  6: i32 frame_size (property = "transport-frame-size")
  7: bool compress (property = "apply-compression")
  8: double log_rate (property = "log-sampling-rate")
}
```

Untyped data translation - Get property

annotations::values::**property**

```
struct flat_config {
  1: string host_name (property = "host-name")
  2: i16 host_port (property = "host-port")
  3: string client_name (property = "client-name")
  4: i32 send_timeout (property = "socket-send-timeout")
  5: i32 receive_timeout (property = "socket-receive-timeout")
  6: i32 frame_size (property = "transport-frame-size")
  7: bool compress (property = "apply-compression")
  8: double log_rate (property = "log-sampling-rate")
}
```

Untyped data translation - Get property

Member::annotations::values::property

```
struct flat_config {
  1: string host_name (property = "host-name")
  2: i16 host_port (property = "host-port")
  3: string client_name (property = "client-name")
  4: i32 send_timeout (property = "socket-send-timeout")
  5: i32 receive_timeout (property = "socket-receive-timeout")
  6: i32 frame_size (property = "transport-frame-size")
  7: bool compress (property = "apply-compression")
  8: double log_rate (property = "log-sampling-rate")
}
```

Untyped data translation - Get property

```
template <typename Member>
using get_property =
  typename Member::annotations::values::property;

struct flat_config {
  1: string host_name (property = "host-name")
  2: i16 host_port (property = "host-port")
  3: string client_name (property = "client-name")
  4: i32 send_timeout (property = "socket-send-timeout")
  5: i32 receive_timeout (property = "socket-receive-timeout")
  6: i32 frame_size (property = "transport-frame-size")
  7: bool compress (property = "apply-compression")
  8: double log_rate (property = "log-sampling-rate")
}
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {

    reflect_struct<flat_config>::members




  for (auto const &i: from) {
    fatal::trie_find<          >(
      i.first.begin(), i.first.end()




    );
  }
}
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {
                      fatal::transform<
    reflect_struct<flat_config>::members,
    get_property
  >

  for (auto const &i: from) {
    fatal::trie_find<          >(
      i.first.begin(), i.first.end()




    );
  }
}
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {
  using properties = fatal::transform<
    reflect_struct<flat_config>::members,
    get_property
  >;

  for (auto const &i: from) {
    fatal::trie_find<properties>(
      i.first.begin(), i.first.end()




    );
  }
}
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {
  using properties = fatal::transform<
    reflect_struct<flat_config>::members,
    get_property
  >;

  for (auto const &i: from) {
    fatal::trie_find<properties>(
      i.first.begin(), i.first.end(),
      legacy_to_flat_translator()


    );
  }
}
```

Untyped data translation - From untyped

```
void translate(legacy_config const &from, flat_config &to) {
  using properties = fatal::transform<
    reflect_struct<flat_config>::members,
    get_property
  >;

  for (auto const &i: from) {
    fatal::trie_find<properties>(
      i.first.begin(), i.first.end(),
      legacy_to_flat_translator(),
      i.second,
      to
    );
  }
}
```

Untyped data translation - From untyped: member

```
struct legacy_to_flat_translator {
  template <typename Property>
  void operator ()(
    fatal::tag<Property>,
    std::string const &from,
    flat_config &to
  ) const {








  }
};
```

Untyped data translation - From untyped: member

```
struct legacy_to_flat_translator {
  template <typename Property>
  void operator ()(
    fatal::tag<Property>,
    std::string const &from,
    flat_config &to
  ) const {

      reflect_struct<flat_config>::members
      Property




  }
};
```

Untyped data translation - From untyped: member

```
struct legacy_to_flat_translator {
  template <typename Property>
  void operator ()(
    fatal::tag<Property>,
    std::string const &from,
    flat_config &to
  ) const {
                fatal::get<
      reflect_struct<flat_config>::members,
      Property, get_property
    >



  }
};
```

Untyped data translation - From untyped: member

```
struct legacy_to_flat_translator {
  template <typename Property>
  void operator ()(
    fatal::tag<Property>,
    std::string const &from,
    flat_config &to
  ) const {
    using member = fatal::get<
      reflect_struct<flat_config>::members,
      Property, get_property
    >;
                 member::getter::ref(to);

  }
};
```

Untyped data translation - From untyped: member

```
struct legacy_to_flat_translator {
  template <typename Property>
  void operator ()(
    fatal::tag<Property>,
    std::string const &from,
    flat_config &to
  ) const {
    using member = fatal::get<
      reflect_struct<flat_config>::members,
      Property, get_property
    >;
                  member::getter::ref(to);
          folly::to<typename member::type>(from)
  }
};
```

Untyped data translation - From untyped: member

```
struct legacy_to_flat_translator {
  template <typename Property>
  void operator ()(
    fatal::tag<Property>,
    std::string const &from,
    flat_config &to
  ) const {
    using member = fatal::get<
      reflect_struct<flat_config>::members,
      Property, get_property
    >;
    auto &value = member::getter::ref(to);
    value = folly::to<typename member::type>(from);
  }
};
```

Untyped data translation - Halfway there

```
void translate(legacy_config const &from, flat_config &to);
```

```
void translate(flat_config const &from, legacy_config &to);
```

Untyped data translation - From typed

```
void translate(flat_config const &from, legacy_config &to) {



}
```

Untyped data translation - From typed

```
void translate(flat_config const &from, legacy_config &to) {
                 reflect_struct<flat_config>::members

  fatal::foreach<       >(




  );
}
```

Untyped data translation - From typed

```
void translate(flat_config const &from, legacy_config &to) {
  using members = reflect_struct<flat_config>::members;

  fatal::foreach<members>(




  );
}
```

Untyped data translation - From typed

```
void translate(flat_config const &from, legacy_config &to) {
  using members = reflect_struct<flat_config>::members;

  fatal::foreach<members>(
    flat_to_legacy_translator()


  );
}
```

Untyped data translation - From typed

```
void translate(flat_config const &from, legacy_config &to) {
  using members = reflect_struct<flat_config>::members;

  fatal::foreach<members>(
    flat_to_legacy_translator(),
    from,
    to
  );
}
```

Untyped data translation - From typed: member

```
struct flat_to_legacy_translator {
  template <typename Member, std::size_t Index>
  void operator ()(
    fatal::indexed<Member, Index>,
    flat_config const &from,
    legacy_config &to
  ) {




  }
};
```

Untyped data translation - From typed: member

```
struct flat_to_legacy_translator {
  template <typename Member, std::size_t Index>
  void operator ()(
    fatal::indexed<Member, Index>,
    flat_config const &from,
    legacy_config &to
  ) {
    auto const &value = Member::getter::ref(from);


  }
};
```

Untyped data translation - From typed: member

```
struct flat_to_legacy_translator {
  template <typename Member, std::size_t Index>
  void operator ()(
    fatal::indexed<Member, Index>,
    flat_config const &from,
    legacy_config &to
  ) {
    auto const &value = Member::getter::ref(from);
    auto const key =               get_property<Member>   ;

  }
};
```

Untyped data translation - From typed: member

```
struct flat_to_legacy_translator {
  template <typename Member, std::size_t Index>
  void operator ()(
    fatal::indexed<Member, Index>,
    flat_config const &from,
    legacy_config &to
  ) {
    auto const &value = Member::getter::ref(from);
    auto const key = fatal::z_data<get_property<Member>>();

  }
};
```

Untyped data translation - From typed: member

```
struct flat_to_legacy_translator {
  template <typename Member, std::size_t Index>
  void operator ()(
    fatal::indexed<Member, Index>,
    flat_config const &from,
    legacy_config &to
  ) {
    auto const &value = Member::getter::ref(from);
    auto const key = fatal::z_data<get_property<Member>>();
         folly::to<std::string>(value)
  }
};
```

Untyped data translation - From typed: member

```
struct flat_to_legacy_translator {
  template <typename Member, std::size_t Index>
  void operator ()(
    fatal::indexed<Member, Index>,
    flat_config const &from,
    legacy_config &to
  ) {
    auto const &value = Member::getter::ref(from);
    auto const key = fatal::z_data<get_property<Member>>();
    to[key] = folly::to<std::string>(value);
  }
};
```

Nested data translation

Nested data - Thrift IDL

```
struct nested_config {
  1: host_address address
  2: string client_name
  3: network_timeout timeout
  4: transport_config transport
  5: double log_rate
}
```

Nested data - Thrift IDL

```
struct nested_config {
  1: host_address address
  2: string client_name
  3: network_timeout timeout
  4: transport_config transport
  5: double log_rate
}
```

Nested data - Thrift IDL

```
struct host_address {
  1: string name
  2: i16 port
}

struct network_timeout {
  1: i32 send
  2: i32 receive
}

struct transport_config {
  1: i32 frame_size
  2: bool compress
}
```

Nested data - Thrift IDL

```
struct host_address {
  1: string name
  2: i16 port
}

struct network_timeout {
  1: i32 send
  2: i32 receive
}

struct transport_config {
  1: i32 frame_size
  2: bool compress
}
```

Nested data - Thrift IDL

```
struct nested_config {
  1: host_address address
  2: string client_name
  3: network_timeout timeout
  4: transport_config transport
  5: double log_rate
}
```

Nested data - Thrift IDL

```
struct nested_config {
  1: host_address address
  2: string client_name ->            client_name
  3: network_timeout timeout
  4: transport_config transport
  5: double log_rate ->            log_rate
}
```

Nested data - Thrift IDL

```
struct nested_config {
  1: host_address address
  2: string client_name (from_flat = "client_name")
  3: network_timeout timeout
  4: transport_config transport
  5: double log_rate (from_flat = "log_rate")
}
```

Nested data - Thrift IDL

```
struct host_address {
  1: string name
  2: i16 port
}

struct network_timeout {
  1: i32 send
  2: i32 receive
}

struct transport_config {
  1: i32 frame_size
  2: bool compress
}
```

Nested data - Thrift IDL

```
struct host_address {
  1: string name (from_flat = "host_name")
  2: i16 port (from_flat = "host_port")
}

struct network_timeout {
  1: i32 send (from_flat = "send_timeout")
  2: i32 receive (from_flat = "receive_timeout")
}

struct transport_config {
  1: i32 frame_size (from_flat = "frame_size")
  2: bool compress (from_flat = "compress")
}
```

Nested data translation - Public Interface

```
void translate(flat_config const &from, nested_config &to);
```

```
void translate(nested_config const &from, flat_config &to);
```

Nested data translation - From flat

```
void translate(flat_config const &from, nested_config &to) {




}

void translate(nested_config const &from, flat_config &to);
```

Nested data translation - From flat

```
void translate(flat_config const &from, nested_config &to) {

               /* Can we avoid writing the
                    nested nastiness
                  template recursion? */

}

void translate(nested_config const &from, flat_config &to);
```

Nested data

```
foo a = some_foo;
```

Nested data

```
foo a = some_foo;
struct foo { bar b; };
```

Nested data

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
```

Nested data

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };
```

Nested data members

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x =        ;
```

Nested data members

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a      ;
```

Nested data members

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b    ;
```

Nested data members

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c  ;
```

Nested data members

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;
```

Nested reflection getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

int y =

                                         a

          ;
```

Nested reflection getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

int y =

           foo::member::b::getter::ref(a)

         ;
```

Nested reflection getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

int y =
          bar::member::c::getter::ref(
           foo::member::b::getter::ref(a)
          )
         ;
```

Nested reflection getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

int y = baz::member::d::getter::ref(
          bar::member::c::getter::ref(
           foo::member::b::getter::ref(a)
          )
        );
```

Single getter for nested data

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;




int z =                     a ;
```

Flattening nested getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

using flattened =




  ;

int z = flattened::ref(a);
```

Flattening nested getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

using flattened =
  foo::member::b



 ;

int z = flattened::ref(a);
```

Flattening nested getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

using flattened =
  foo::member::b,
  bar::member::c

 ;

int z = flattened::ref(a);
```

Flattening nested getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

using flattened =
  foo::member::b,
  bar::member::c,
  baz::member::d
 ;

int z = flattened::ref(a);
```

Chained getters

```
foo a = some_foo;
struct foo { bar b; };
struct bar { baz c; };
struct baz { int d; };

int x = a.b.c.d;

using flattened = fatal::chained_data_member_getter<
  foo::member::b,
  bar::member::c,
  baz::member::d
>;

int z = flattened::ref(a);
```

Nested data - Chained getters

```
nested_config x;

x.address.name
x.address.port
x.client_name
x.timeout.send
x.timeout.receive
x.transport.frame_size
x.transport.compress
x.log_rate
```

Nested data translation - From flat

```
void translate(flat_config const &from, nested_config &to) {
                         flatten_getters<nested_config>




}

void translate(nested_config const &from, flat_config &to);
```

Nested data translation - From flat

```
void translate(flat_config const &from, nested_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(


  );
}

void translate(nested_config const &from, flat_config &to);
```

Nested data translation - From flat

```
void translate(flat_config const &from, nested_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(
    flat_to_nested_translator()

  );
}

void translate(nested_config const &from, flat_config &to);
```

Nested data translation - From flat

```
void translate(flat_config const &from, nested_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(
    flat_to_nested_translator(),
    from, to
  );
}

void translate(nested_config const &from, flat_config &to);
```

Nested data translation - From flat: member

```
struct flat_to_nested_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    flat_config const &from, nested_config &to
  ) const {









  }
};
```

Nested data translation - From flat: member

```
struct flat_to_nested_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    flat_config const &from, nested_config &to
  ) const {

      reflect_struct<flat_config>::members,
               Leaf          annotations::values::from_flat




  }
};
```

Nested data translation - From flat: member

```
struct flat_to_nested_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    flat_config const &from, nested_config &to
  ) const {
    using from_member = fatal::get<
      reflect_struct<flat_config>::members,
      typename Leaf::member::annotations::values::from_flat,
      fatal::get_type::name
    >;


  }
};
```

Nested data translation - From flat: member

```
struct flat_to_nested_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    flat_config const &from, nested_config &to
  ) const {
    using from_member = fatal::get<
      reflect_struct<flat_config>::members,
      typename Leaf::member::annotations::values::from_flat,
      fatal::get_type::name
    >;

                from_member::getter::ref(from)
  }
};
```

Nested data translation - From flat: member

```
struct flat_to_nested_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    flat_config const &from, nested_config &to
  ) const {
    using from_member = fatal::get<
      reflect_struct<flat_config>::members,
      typename Leaf::member::annotations::values::from_flat,
      fatal::get_type::name
    >;
                    Leaf::getter::ref(to)
              from_member::getter::ref(from)
  }
};
```

Nested data translation - From flat: member

```
struct flat_to_nested_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    flat_config const &from, nested_config &to
  ) const {
    using from_member = fatal::get<
      reflect_struct<flat_config>::members,
      typename Leaf::member::annotations::values::from_flat,
      fatal::get_type::name
    >;
    auto &to_member = Leaf::getter::ref(to);
    to_member = from_member::getter::ref(from);
  }
};
```

Nested data translation - ~~Halfway~~ there?

```
void translate(flat_config const &from, nested_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(
    flat_to_nested_translator(),
    from, to
  );
}

void translate(nested_config const &from, flat_config &to) {









}
```

Nested data translation - From nested

```
~~void translate(flat_config const &from, nested_config &to) {~~
~~  using nested_getters = flatten_getters<nested_config>;~~
~~  fatal::foreach<nested_getters>(~~
~~    flat_to_nested_translator(),~~
~~    from, to~~
~~  );~~
~~}~~

void translate(nested_config const &from, flat_config &to) {
                                  flatten_getters<nested_config>






}
```

Nested data translation - From nested

```
void translate(flat_config const &from, nested_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(
    flat_to_nested_translator(),
    from, to
  );
}
```

```
void translate(nested_config const &from, flat_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(


  );
}
```

Nested data translation - From nested

```
void translate(flat_config const &from, nested_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(
    flat_to_nested_translator(),
    from, to
  );
}

void translate(nested_config const &from, flat_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(
    nested_to_flat_translator()

  );
}
```

Nested data translation - From nested

```
void translate(flat_config const &from, nested_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(
    flat_to_nested_translator(),
    from, to
  );
}
```

```
void translate(nested_config const &from, flat_config &to) {
  using nested_getters = flatten_getters<nested_config>;
  fatal::foreach<nested_getters>(
    nested_to_flat_translator(),
    from, to
  );
}
```

Nested data translation - From nested: member

```
struct nested_to_flat_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    nested_config const &from, flat_config &to
  ) const {







  }
};
```

Nested data translation - From nested: member

```
struct nested_to_flat_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    nested_config const &from, flat_config &to
  ) const {



              Leaf             annotations::values::from_flat




  }
};
```

Nested data translation - From nested: member

```
struct nested_to_flat_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    nested_config const &from, flat_config &to
  ) const {

      reflect_struct<flat_config>::members,
              Leaf::member::annotations::values::from_flat,
      fatal::get_type::name



  }
};
```

Nested data translation - From nested: member

```
struct nested_to_flat_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    nested_config const &from, flat_config &to
  ) const {
                  fatal::get<
      reflect_struct<flat_config>::members,
      typename Leaf::member::annotations::values::from_flat,
      fatal::get_type::name
    >



  }
};
```

Nested data translation - From nested: member

```
struct nested_to_flat_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    nested_config const &from, flat_config &to
  ) const {
    using to_member = fatal::get<
      reflect_struct<flat_config>::members,
      typename Leaf::member::annotations::values::from_flat,
      fatal::get_type::name
    >;
                        to_member::getter::ref(to)

  }
};
```

Nested data translation - From nested: member

```
struct nested_to_flat_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    nested_config const &from, flat_config &to
  ) const {
    using to_member = fatal::get<
      reflect_struct<flat_config>::members,
      typename Leaf::member::annotations::values::from_flat,
      fatal::get_type::name
    >;
                    to_member::getter::ref(to)
                Leaf::getter::ref(from)
  }
};
```

Nested data translation - From nested: member

```
struct nested_to_flat_translator {
  template <typename Leaf, std::size_t Index>
  void operator ()(
    fatal::indexed<Leaf, Index>,
    nested_config const &from, flat_config &to
  ) const {
    using to_member = fatal::get<
      reflect_struct<flat_config>::members,
      typename Leaf::member::annotations::values::from_flat,
      fatal::get_type::name
    >;
    auto &member_ref = to_member::getter::ref(to);
    member_ref = Leaf::getter::ref(from);
  }
};
```

Enabling reflection in Thrift

Generating reflection metadata

`module.thrift`

Generating reflection metadata

```
$ thrift --gen cpp2         module.thrift
```

Generating reflection metadata

```
$ thrift --gen cpp2:fatal module.thrift
```

-> **`fatal`** flag and file names will soon be renamed to **`reflection`**

Generating reflection metadata

```
$ thrift --gen cpp2:fatal module.thrift
```

gen-cpp2/

...

-> `fatal` flag and file names will soon be renamed to `reflection`

Generating reflection metadata

```
$ thrift --gen cpp2:fatal module.thrift

gen-cpp2/

    module_fatal_struct.h
    module_fatal_enum.h
    module_fatal_union.h
    module_fatal_types.h
    ...
```

-> `fatal` flag and file names will soon be renamed to `reflection`

Importing reflection metadata

```
#include <project_dir/gen-cpp2/module_fatal_struct.h>
```

Importing reflection metadata

```
#include <project_dir/gen-cpp2/module_fatal_enum.h>
```

Importing reflection metadata

```
#include <project_dir/gen-cpp2/module_fatal_union.h>
```

Importing reflection metadata

```
#include <project_dir/gen-cpp2/module_fatal_struct.h>
#include <project_dir/gen-cpp2/module_fatal_enum.h>
#include <project_dir/gen-cpp2/module_fatal_union.h>
```

Importing reflection metadata

```
#include <project_dir/gen-cpp2/module_fatal_struct.h>
#include <project_dir/gen-cpp2/module_fatal_enum.h>
#include <project_dir/gen-cpp2/module_fatal_union.h>

#include <project_dir/gen-cpp2/module_fatal_types.h>
```

Importing reflection metadata

```
#include <project_dir/gen-cpp2/module_fatal_struct.h>
#include <project_dir/gen-cpp2/module_fatal_enum.h>
#include <project_dir/gen-cpp2/module_fatal_union.h>

#include <project_dir/gen-cpp2/module_fatal_types.h>
```

-> more information in **`thrift/lib/cpp2/fatal/reflection.h`**

Closing words

Closing words

In the examples...

- Not a single SFINAE
- Not a single `std::enable_if`
- Dirty template trick
- Only a single custom meta-function
- Well defined, well understood primitives
    - `sort`, `sorted_search`, `trie_find`, `get`...
    - Abstracted by a library

Closing words

- Replacing code generation
    - Build times (parsing, explicit instantiation...)
    - Runtime performance
    - Symbol size reduction

Closing words

- Replacing code generation
  - Build times (parsing, explicit instantiation...)
  - Runtime performance
  - Symbol size reduction
  - Native cross-language extensions

Closing words

- Replacing code generation
  - Build times (parsing, explicit instantiation...)
  - Runtime performance
  - Symbol size reduction
  - Native cross-language extensions
  - Cheaper experimentation

Closing words

- Replacing code generation
  - Build times (parsing, explicit instantiation...)
  - Runtime performance
  - Symbol size reduction
  - Native cross-language extensions
  - Cheaper experimentation

- Runtime reflection

Closing words

- Replacing code generation
  - Build times (parsing, explicit instantiation...)
  - Runtime performance
  - Symbol size reduction
  - Native cross-language extensions
  - Cheaper experimentation

- Runtime reflection

- C++ standard support
  - Not a take at standardization

Closing words

- Replacing code generation
  - Build times (parsing, explicit instantiation...)
  - Runtime performance
  - Symbol size reduction
  - Native cross-language extensions
  - Cheaper experimentation
- Runtime reflection
- C++ standard support
  - Not a take at standardization
- Why not a C++ compiler patch?

Before we go

Questions?

- Thrift: https://github.com/facebook/fbthrift
  - Reflection under `thrift/lib/cpp2/fatal`
    - Along with reflection based utility library
  - Demo code and slides will be uploaded soon
  - More demos are coming

- Fatal: https://github.com/facebook/fatal

- Watch out for changes
  - Commit messages containing *`[break]`* label
  - Demo code will be updated accordingly

- Thanks for watching
  - By the way, we're hiring: https://www.facebook.com/careers

We’re done!