# Enrich Facade Creation Facilities for the Pointer-Semantics-Based Polymorphism Library - Proxy

| | |
|---|---|
| Document number: | P3584R0 |
| Date: | 2024-01-13 |
| Project: | Programming Language C++ |
| Audience: | LEWGI, LEWG |
| Authors: | Mingxin Wang |
| Reply-to: | Mingxin Wang <mingxwa@microsoft.com> |

## Table of Contents

## 1 Introduction

This paper introduces the **`facade_builder`** class template to the standard, as an enhancement to P3086R3: Proxy: A Pointer-Semantics-Based Polymorphism Library. **`facade_builder`** simplifies the creation and configuration of facades for polymorphic types, allowing for more flexible and efficient design of facade-based runtime abstraction spec.

## 2 Motivation and scope

The primary motivation for this proposal is to provide a systematic and user-friendly way to create and manage facades for runtime polymorphism in C++. The existing facilities in the Proxy library are powerful but can be complex and cumbersome to use. The `basic_facade_builder` class template abstracts the intricacies involved in configuring facades, making the process more intuitive and less error prone.

The scope of this proposal includes the introduction of the `basic_facade_builder` class template and its associated member types and alias templates. It aims to standardize the process of creating facades with customizable constraints, conventions and reflections, thus enhancing the usability and flexibility of the Proxy library.

The proposed library is a minimal subset of our open-source implementation hosted on GitHub without support for RTTI, `proxy_view` or `std::format`. If the committee has a consensus in the proposed part, those extensions can be proposed separately without requiring significant changes to the core design.

# 3 Technical specifications

## 3.1 Feature test macro

In ***[version.syn]***, add:

```
#define __cpp_lib_proxy YYYYMML // also in <memory>
```

The placeholder value shall be adjusted to denote this proposal's date of adoption.

## 3.2 Header <memory> synopsis

```
// all freestanding
namespace std {
  template <class Cs, class Rs, proxiable_ptr_constraints C>
    class basic_facade_builder;

  using facade_builder = basic_facade_builder<tuple<>, tuple<>,
      proxiable_ptr_constraints{
          .max_size = default-size,
          .max_align = default-size,
          .copyability = default-cl,
          .relocatability = default-cl,
          .destructibility = default-cl}>;
}
```

The definitions of **`basic_facade_builder`** and **`facade_builder`** make use of the following exposition-only constants:

```
constexpr std::size_t default-size = numeric_limits<size_t>::max();
constexpr constraint_level default-cl = static_cast<constraint_level>(
    numeric_limits<underlying_type_t<constraint_level>>::min());
```

*default-size* and *default-cl* denote that a field in **`proxiable_ptr_constraints`** is not specified in the template parameters of a **`basic_facade_builder`** specialization. In an instantiation of **`proxiable_ptr_constraints`**, any meaningful value of **`max_size`** and **`max_align`** is less than *default-size*; any meaningful value of **`copyability`**, **`relocatability`**, and **`destructibility`** is greater than *default-cl*.

## 3.3 Class template `basic_facade_builder`

```
template <class Cs, class Rs, proxiable_ptr_constraints C>
struct basic_facade_builder {
  template <class D, class... Os> requires(see below)
  using add_indirect_convention = see below;
  template <class D, class... Os> requires(see below)
  using add_direct_convention = see below;
  template <class D, class... Os> requires(see below)
  using add_convention = see below;
  template <class R>
  using add_indirect_reflection = see below;
  template <class R>
  using add_direct_reflection = see below;
  template <class R>
  using add_reflection = add_indirect_reflection<R>;
  template <facade F, bool WithUpwardConversion = false>
  using add_facade = see below;
  template <size_t PtrSize, size_t PtrAlign = see below> requires(see below)
  using restrict_layout = see below;
  template <constraint_level CL>
  using support_copy = see below;
  template <constraint_level CL>
  using support_relocation = see below;
  template <constraint_level CL>
  using support_destruction = see below;
  using build = see below;
  basic_facade_builder() = delete;
};
```

**template <class D, class... Os> requires(***see below***)**
**using add_indirect_convention =** *see below***;**
**template <class D, class... Os> requires(***see below***)**
**using add_direct_convention =** *see below***;**
**template <class D, class... Os> requires(***see below***)**
**using add_convention =** *see below***;**

Adds a convention type to the template parameters. The expression inside **requires** is equivalent to **sizeof...(Os) > 0u** and each type in **Os** meets the *ProOverload* requirements. Let **F** be a facade type,

- **add_convention** is equivalent to **add_indirect_convention**.
- **add_indirect_convention** merges an implementation-defined convention type **IC** into **Cs**, where:
  - **IC::is_direct** is **false**.
  - **typename IC::dispatch_type** is **D**.
  - **typename IC::overload_types** is a tuple-like type of distinct types in **Os**.
  - **typename IC::template accessor<F>** is **typename D::template accessor<F, false, D, Os...>** if applicable.
- **add_direct_convention** merges an implementation-defined convention type **IC** into **Cs**, where:
  - **IC::is_direct** is **true**.
  - **typename IC::dispatch_type** is **D**.
  - **typename IC::overload_types** is a tuple-like type of distinct types in **Os**.
  - **typename IC::template accessor<F>** is **typename D::template accessor<F, true, D, Os...>** if applicable.

When **Cs** already contains a convention type **IC2** where **IC2::is_direct == IC::is_direct && is_same_v<typename IC2::dispatch_type, typename IC::dispatch_type>** is **true**, **Os** merges with **typename IC2::overload_types** and removes duplicates, and **tuple_size_v<Cs>** shall not change.

**template <class R>**
**using add_indirect_reflection =** *see below***;**
**template <class R>**
**using add_direct_reflection =** *see below***;**

Adds a reflection type to the template parameters. Specifically,

- **add_indirect_reflection** merges an implementation-defined reflection type **Refl** into **Rs**, where:
  - **Refl::is_direct** is **false**.
  - **typename Refl::reflector_type** is **R**.
  - **typename Refl::template accessor<F>** is **typename R::template accessor<F, false, R>** if applicable.

- **`add_direct_reflection`** merges an implementation-defined reflection type **`Refl`** into **`Rs`**, where:
    - **`Refl::is_direct`** is **`true`**.
    - **`typename Refl::reflector_type`** is **`R`**.
    - **`typename Refl::template accessor<F>`** is **`typename R::template accessor<F, true, R>`** if applicable.

When **`Rs`** already contains **`Refl`**, the template parameters shall not change.

**`template <facade F, bool WithUpwardConversion = false>`**
**`using add_facade =`** *see below*;

Adds a facade type into the template parameters. It merges **`typename F::convention_types`** into **`Cs`**, **`typename F::reflection_types`** into **`Rs`**, and **`F::constraints`** into **`C`**. Optionally, it adds a convention for implicit upward conversion into **`Cs`** when **`WithUpwardConversion`** is **`true`**.

**`template <size_t PtrSize, size_t PtrAlign =`** *see below*> **`requires(`***see below***`)`**
**`using restrict_layout =`** *see below*;

Adds layout restrictions to the template parameters, specifically **`C::max_size`** and **`C::max_align`**. The expression inside **`requires`** is equivalent to **`has_single_bit(PtrAlign) && PtrSize % PtrAlign == 0u`**. The default value of **`PtrAlign`** is the maximum possible alignment of an object of size **`PtrSize`**, not greater than **`alignof(max_align_t)`**. After applying the restriction, **`C::max_size`** becomes **`min(C::max_size, PtrSize)`**, and **`C::max_align`** becomes **`min(C::max_align, PtrAlign)`**.

**`template <constraint_level CL>`**
**`using support_copy =`** *see below*;

Adds copyability support to the template parameters, specifically **`C::copyability`**. After the operation, **`C::copyability`** becomes **`max(C::copyability, CL)`**.

**`template <constraint_level CL>`**
**`using support_relocation =`** *see below*;

Adds relocatability support to the template parameters, specifically **`C::relocatability`**. After the operation, **`C::relocatability`** becomes **`max(C::relocatability, CL)`**.

**`template <constraint_level CL>`**
**`using support_destruction =`** *see below*;

Adds destruction support to the template parameters, specifically **`C::destructibility`**. After the operation, **`C::destructibility`** becomes **`max(C::destructibility, CL)`**.

**`using build =`** *see below*;

Specifies a facade type deduced from the template parameters of

**`basic_facade_builder<Cs, Rs, C>`**. Specifically,

- **`typename build::convention_types`** is defined as **`Cs`**, and
- **`typename build::reflection_types`** is defined as **`Rs`**, and
- **`build::constraints`** is a core constant expression of type **`proxiable_ptr_constraints`** that defines constraints to the pointer types, and
- **`build::constraints.max_size`** is **`C::max_size`** if defined by **`restrict_layout`**, otherwise **`sizeof(void*) * 2u`** when **`C::max_size`** is *default-size*, and
- **`build::constraints.max_align`** is **`C::max_align`** if defined by **`restrict_layout`**, otherwise **`alignof(void*)`** when **`C::max_align`** is *default-size*, and
- **`build::constraints.copyability`** is **`C::copyability`** if defined by **`support_copy`**, otherwise **`constraint_level::none`** when **`C::copyability`** is *default-cl*, and
- **`build::constraints.relocatability`** is **`C::relocatability`** if defined by **`support_rellocation`**, otherwise **`constraint_level::nothrow`** when **`C::relocatability`** is *default-cl*, and
- **`build::constraints.destructibility`** is **`C::destructibility`** if defined by **`support_destruction`**, otherwise **`constraint_level::nothrow`** when **`C::destructibility`** is *default-cl.*

# 4 Summary

By providing a more intuitive and structured way to create facades, we believe the **`basic_facade_builder`** class template will lower the barrier to entry for developers looking to leverage the Proxy library.