P3870R0: Renaming `std::nontype` to `std::tag`

Document number: P3870R0
Date: 2025-10-06
Audience: Library Evolution Working Group (LEWG)
Authors: Andrei Zissu (*andrziss@gmail.com*), Ville Voutilainen (*ville.voutilainen@gmail.com*)
Reply-to: Andrei Zissu (*andrziss@gmail.com*)

---

1 Introduction

`std::nontype` was introduced by P2472R3 as a lightweight wrapper for compile-time constant template parameters. Since P2841R5 renamed "non-type template parameter" to "constant template parameter," the identifier *nontype* is now inconsistent with core-language terminology. A clearer, neutral name— `std::tag` —better communicates the facility's intent and follows existing library idioms.

---

2 Motivation

- **Terminology alignment.** After P2841R5, "non-type" is legacy wording; keeping it in the library creates an avoidable mismatch.
- **Neutrality and generality.** To paraphrase P3794R0: "the motivation here is indeed calling a dog a dog, and therefore calling a type that's meant to be a tag type a `std::tag`". No other meaning is implied beyond that, nor any particular use case.

---

3 Proposed wording (relative to N5008)

Change `nontype` to `tag` in all the relevant places.

3.1 Modify `[utility.syn]` (22.2.1)

Before:

```
// nontype argument tag
template<auto V>
struct nontype_t {
explicit nontype_t() = default;
};
template<auto V> constexpr nontype_t<V> nontype{};
```

After:

```
// tag argument
template<auto V>
struct tag_t {
explicit tag_t() = default;
};
template<auto V> constexpr tag_t<V> tag{};
```

3.2 Modify `[func.wrap.ref.class]` (22.10.17.6.2)

`constructors and assignment operators` (22.10.17.6.3)

Before:

```
template<auto f> constexpr function_ref(nontype_t<f>) noexcept;
template<auto f, class U> constexpr function_ref(nontype_t<f>, U&&) noexcept;
template<auto f, class T> constexpr function_ref(nontype_t<f>, cv T*) noexcept;
```

After:

```
template<auto f> constexpr function_ref(tag_t<f>) noexcept;
template<auto f, class U> constexpr function_ref(tag_t<f>, U&&) noexcept;
template<auto f, class T> constexpr function_ref(tag_t<f>, cv T*) noexcept;
```

`deduction guides` (22.10.17.6.5)

Before:

```
template<auto f>
function_ref(nontype_t<f>) -> function_ref<see below>;
template<auto f, class T>
function_ref(nontype_t<f>, T&&) -> function_ref<see below>;
```

After:

```
template<auto f>
function_ref(tag_t<f>) -> function_ref<see below>;
template<auto f, class T>
function_ref(tag_t<f>, T&&) -> function_ref<see below>;
```

3.3 Modify `[func.wrap.ref.ctor]` (22.10.17.6.3)

`(7)`

Before:

```
template<auto f> constexpr function_ref(nontype_t<f>) noexcept;
```

After:

```
template<auto f> constexpr function_ref(tag_t<f>) noexcept;
```

`(11)`

Before:

```
template<auto f, class U>
constexpr function_ref(nontype_t<f>, U&& obj) noexcept;
```

After:

```
template<auto f, class U>
constexpr function_ref(tag_t<f>, U&& obj) noexcept;
```

`(15)`

Before:

```
template<auto f, class T>
constexpr function_ref(nontype_t<f>, cv T* obj) noexcept;
```

After:

```
template<auto f, class T>
constexpr function_ref(tag_t<f>, cv T* obj) noexcept;
```

`(21.3)`

Before:

```
— T is not a specialization of nontype_t.
```

After:

```
— T is not a specialization of tag_t.
```

3.3 Modify `[func.wrap.ref.deduct]` (22.10.17.6.5)

`(1)`

Before:

```
template<auto f>
function_ref(nontype_t<f>) -> function_ref<see below>;
```

After:

```
template<auto f>
function_ref(tag_t<f>) -> function_ref<see below>;
```

`(4)`

Before:

```
template<auto f, class T>
function_ref(nontype_t<f>, T&&) -> function_ref<see below >;
```

After:

```
template<auto f, class T>
function_ref(tag_t<f>, T&&) -> function_ref<see below >;
```

4 References

1. N5008 — Working Draft, Standard for Programming Language C++, 2025-03-15.

2. P2472R3 — make function_ref more functional, Jarrad J. Waterloo & Zhihao Yuan, 2022-05-12.

3. P2841R5 — Concept and variable-template template-parameters, Corentin Jabot & Gašper Ažman & James Touton, 2024-10-16.

4. P3794R0 — An idea or two on renaming the nontype tag, Zhihao Yuan, 2025-7-13.