Documentation

Invalid Reflections

`consteval bool is_invalid(info reflection)`

Returns true if the reflection is invalid.

`consteval info invalid_reflection(const char* error_message)`

Returns an invalid reflection with the given error message.

Scopes

`consteval bool is_local(info reflection)`

Returns true if the reflected entity is declared in block scope.

Variables

`consteval bool is_variable(info reflection)`

Returns true if the reflected entity is a variable.

`consteval bool has_static_storage_duration(info variable)`

Requires: `is_variable(variable)`

Returns true if the reflected variable has static storage duration.

`consteval bool has_thread_local_storage_duration(info variable)`

Requires: `is_variable(variable)`

Returns true if the reflected variable thread local storage duration.

`consteval bool has_automatic_storage_duration(info variable)`

Requires: `is_variable(variable)`

Returns true if the reflected variable has automatic storage duration.

Functions

`consteval bool is_function(info reflection)`

Returns true if the reflected entity is a function.

`consteval bool is_nothrow(info function)`

Requires: `is_function(function)`

Returns true if the reflected entity is a function which does not throw.

Classes

`consteval bool is_class(info reflection)`

Equivalent to `std::is_class<T>::value`.

Returns true if the reflected entity is a `class` or `struct` where `reflection` is `reflexpr(T)`.

`consteval bool is_union(info reflection)`

Equivalent to `std::is_union<T>::value` where `reflection` is `reflexpr(T)`.

Returns true if the reflected entity is a union.

`consteval bool has_virtual_destructor(info class_type)`

Requires: `is_class(class_type)`

Equivalent to `std::has_virtual_destructor<T>::value` where `class_type` is `reflexpr(T)`.

Returns true if the reflected entity is a class or struct tag declaration ("class type") and has a virtual destructor, either directly or via a parent class.

`consteval bool is_declared_class(info class_type)`

Requires: `is_class(class_type)`

Returns true if the reflected entity is a class, declared via the `class` keyword.

`consteval bool is_declared_struct(info class_type)`

Requires: `is_class(class_type)`

Returns true if the reflected entity is a class, declared via the `struct` keyword.

Class Members

`consteval bool is_class_member(info reflection)`

Returns true if the reflected entity is declared in class scope.

Data Members

`consteval bool is_data_member(info reflection)`

Returns true if the reflected entity is a data member.

`consteval bool is_static_data_member(info reflection)`

Returns true if the reflected entity is a data member declared with the `static` declaration specifier.

`consteval bool is_nonstatic_data_member(info reflection)`

Returns true if the reflected entity is a data member declared without the `static` declaration specifier.

`consteval bool is_bit_field(info data_mem)`

Requires: `is_data_member(data_mem)`

Returns true if the reflected entity is a bit-field.

`consteval bool is_mutable(info data_mem)`

Requires: `is_data_member(data_mem)`

Returns true if the reflected entity is a mutable data member.

Member Functions

`consteval bool is_member_function(info reflection);`

Returns true if the reflected entity is a member function.

`consteval bool is_static_member_function(info reflection)`

Returns true if the reflected entity is a member function declared with the `static` declaration specifier.

`consteval bool is_nonstatic_member_function(info reflection)`

Returns true if the reflected entity is a member function declared without the `static` declaration specifier.

`consteval bool is_normal(info mem_function)`

Requires: `is_member_function(mem_function)`

Returns true if the reflected entity is a normal member function, i.e. a function with no special language rules associated with it.

`consteval bool is_override(info mem_function)`

Requires: `is_member_function(mem_function)`

Returns true if the reflected entity is either implicitly or explicitly a member function override.

`consteval bool is_override_specified(info mem_function)`

Requires: `is_member_function(mem_function)`

Returns true if the reflected entity is a member function override that was explicitly specified via `override` specifier.

`consteval bool is_deleted(info mem_function)`

Requires: `is_member_function(mem_function)`

Returns true if the reflected entity is a member function with a deleted definition.

`consteval bool is_virtual(info mem_function)`

Requires: `is_member_function(mem_function)`

Returns true if the reflected entity is a member function that was declared with the `virtual` specifier.

`consteval bool is_pure_virtual(info mem_function)`

Requires: `is_member_function(mem_function)`

Returns true if the reflected entity is a member function that was declared with the `virtual` specifier and defined as pure virtual.

Special Members

`consteval bool is_special_member_function(info reflection)`

Returns true if the reflected entity is a special member function.

`consteval bool is_constructor(info reflection)`

Returns true if the reflected entity is any kind of constructor.

`consteval bool is_default_constructor(info reflection)`

Returns true if the reflected entity is a default constructor.

`consteval bool is_copy_constructor(info reflection)`

Returns true if the reflected entity is a copy constructor.

`consteval bool is_move_constructor(info reflection)`

Returns true if the reflected entity is a move constructor.

`consteval bool is_copy_assignment_operator(info reflection)`

Returns true if the reflected entity is a copy assignment operator.

`consteval bool is_move_assignment_operator(info reflection)`

Returns true if the reflected entity is a move assignment operator.

`consteval bool is_copy(info reflection)`

Returns true if the reflected entity is a copy constructor, or copy assignment operator.

`consteval bool is_move(info reflection)`

Returns true if the reflected entity is a move constructor, or move assignment operator.

`consteval bool is_destructor(info reflection)`

Returns true if the reflected entity is a destructor.

`consteval bool is_conversion(info reflection)`

Returns true if the reflected entity is type conversion function.

`consteval bool is_defaulted(info spec_mem_function)`

Requires: `is_special_member_function(spec_mem_function)`

Returns true if the reflected entity is a special member function with a defaulted definition.

`consteval bool is_explicit(info spec_mem_function)`

Requires: `is_special_member_function(spec_mem_function)`

Returns true if the reflected entity is a special member function explicitly declared as `explicit`.

Access

`consteval bool is_public(info base_or_mem)`

Requires: `is_base_class(base_or_mem) or is_class_member(base_or_mem)`

Returns true if the reflected entity is a base class or member and has `public` access specification.

`consteval bool is_protected(info base_or_mem)`

Requires: `is_base_class(base_or_mem) or is_class_member(base_or_mem)`

Returns true if the reflected entity is a base class or member and has `protected` access specification.

`consteval bool is_private(info base_or_mem)`

Requires: `is_base_class(base_or_mem) or is_class_member(base_or_mem)`

Returns true if the reflected entity is a base class or member and has `private` access specification.

`consteval bool has_default_access(info mem)`

Requires: `is_class_member(mem)`

Returns true if no access specifier precedes the declaration of the reflected entity.

Note that any preceding access specifier, even one corresponding with the default access of the record type (e.g. `public` for `struct`), will cause this function to return false.

Example:

```
struct T {
  int x;
};

struct U {
public:
```

```
  int y;
};

has_default_access(reflexpr(T::x));  // true
has_default_access(reflexpr(U::y));  // false
```

Linkage

`consteval bool has_linkage(info reflection)`

Returns true if the reflected entity has linkage.

`consteval bool is_externally_linked(info reflection)`

Returns true if the reflected entity is externally linked.

`consteval bool is_internally_linked(info reflection)`

Returns true if the reflected entity is internally linked.

Initializers

`consteval bool has_initializer(info var_or_data_mem)`

Requires: `is_variable(var_or_data_mem) or is_data_member(var_or_data_mem)`

Returns true if the reflected entity has an initializer.

General Purpose

`consteval bool is_extern_specified(info reflection)`

Returns true if the reflected entity has an extern specifier.

`consteval bool is_inline(info reflection)`

Returns true if the reflected entity is either implicitly or explicitly inline.

`consteval bool is_inline_specified(info reflection)`

Returns true if the reflected entity is explicitly inline.

`consteval bool is_constexpr(info reflection)`

Returns true if the reflected entity is constexpr.

`consteval bool is_consteval(info reflection)`

Returns true if the reflected entity is consteval.

`consteval bool is_final(info reflection)`

Returns true if the reflected entity is final.

`consteval bool is_defined(info reflection)`

Returns true if the reflected entity is defined.

`consteval bool is_complete(info reflection)`

Returns true if the reflected entity is complete.

Namespaces

`consteval bool is_namespace(info reflection)`

Returns true if the reflected entity is a namespace.

Aliases

`consteval bool is_alias(info reflection)`

Returns true if the reflected entity is an alias.

`consteval bool is_namespace_alias(info reflection)`

Returns true if the reflected entity is a namespace alias.

`consteval bool is_type_alias(info reflection)`

Returns true if the reflected entity is a type alias.

`consteval bool is_alias_template(info reflection)`

Returns true if the reflected entity is an alias template.

Enums

`consteval bool is_enum(info reflection)`

Equivalent to `std::is_enum<T>` where `reflection` is `reflexpr(T)`.

Returns true if the reflected entity is an enumeration, scoped or unscoped.

`consteval bool is_unscoped_enum(info reflection)`

Returns true if the reflected entity is an unscoped enumeration.

`consteval bool is_scoped_enum(info reflection)`

Returns true if the reflected entity is a scoped enumeration.

Enumerators

`consteval bool is_enumerator(info reflection)`

Returns true if the reflected entity is an enumerator.

Templates

`consteval bool is_template(info reflection)`

Returns true if the reflected entity is any kind of template.

`consteval bool is_class_template(info reflection)`

Returns true if the reflected entity is a class template.

`consteval bool is_function_template(info reflection)`

Returns true if the reflected entity is a function template.

`consteval bool is_variable_template(info reflection)`

Returns true if the reflected entity is a variable template.

`consteval bool is_member_function_template(info reflection)`

Returns true if the reflected entity is a member function template.

`consteval bool is_static_member_function_template(reflection)`

Returns true if the reflected entity is a member function template declared with the `static` declaration specifier.

`consteval bool is_nonstatic_member_function_template(info reflection)`

Returns true if the reflected entity is a member function template declared without the `static` declaration specifier.

`consteval bool is_constructor_template(info reflection)`

Returns true if the reflected entity is a template of a constructor.

`consteval bool is_destructor_template(info reflection)`

Returns true if the reflected entity is a template of a destructor.

`consteval bool is_concept(info reflection)`

Returns true if the reflected entity is a concept.

Specializations

`consteval bool is_specialization(info reflection)`

Returns true if the reflected entity is any kind of template specialization.

`consteval bool is_partial_specialization(info reflection)`

Returns true if the reflected entity is a partial template specialization.

`consteval bool is_explicit_specialization(info reflection)`

Returns true if the reflected entity is an explicit template specialization.

`consteval bool is_implicit_instantiation(info reflection)`

Returns true if the reflected entity is an implicit template instantiation.

`consteval bool is_explicit_instantiation(info reflection)`

Returns true if the reflected entity is an explicit template instantiation.

Base Classes

`consteval bool is_base_class(info reflection)`

Returns true if the reflected entity is a base class specifier.

`consteval bool is_direct_base_class(info reflection)`

Returns true if the reflected entity is a direct base class specifier.

`consteval bool is_virtual_base_class(info reflection)`

Returns true if the reflected entity is a virtual base class specifier.

Parameters

`consteval bool is_function_parameter(info reflection)`

Returns true if the reflected entity is a function parameter declaration.

`consteval bool is_template_parameter(info reflection)`

Returns true if the reflected entity is a template parameter.

`consteval bool is_type_template_parameter(info reflection)`

Returns true if the reflected entity is a type template parameter.

`consteval bool is_nontype_template_parameter(info reflection)`

Returns true if the reflected entity is a nontype template parameter.

`consteval bool is_template_template_parameter(info reflection)`

Returns true if the reflected entity is a template template parameter.

`consteval bool has_default_argument(info parameter)`

Requires: `is_function_parameter(parameter) or is_template_parameter(parameter)`

Returns true if the reflected entity is a function parameter and has a default argument.

Types

`consteval bool is_type(info reflection)`

Returns true if the reflected entity is a type, either builtin or user-defined.

`consteval bool is_fundamental_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_fundamental<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a fundamental type.

`consteval bool has_fundamental_type(info reflection)`

Returns true if the reflected entity has fundamental type.

`consteval bool is_arithmetic_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_arithmetic<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is an arithmetic type.

`consteval bool has_arithmetic_type(info reflection)`

Returns true if the reflected entity has arithmetic type.

`consteval bool is_scalar_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_scalar<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a scalar type.

`consteval bool has_scalar_type(info reflection)`

Returns true if the reflected entity has scalar type.

`consteval bool is_object_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_object<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is an object type.

`consteval bool has_object_type(info reflection)`

Returns true if the reflected entity has object type.

`consteval bool is_compound_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_compound<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a compound type.

`consteval bool has_compound_type(info reflection)`

Returns true if the reflected entity has compound type.

`consteval bool is_function_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_function<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a function type.

`consteval bool has_function_type(info reflection)`

Returns true if the reflected entity has function type.

`consteval bool is_class_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_class<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a class type.

`consteval bool has_class_type(info reflection)`

Returns true if the reflected entity has class type.

`consteval bool is_union_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_union<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a union type.

`consteval bool has_union_type(info reflection)`

Returns true if the reflected entity has union type.

`consteval bool is_enum_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_enum<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is an enum type.

`consteval bool has_enum_type(info type)`

Returns true if the reflected entity has enum type.

`consteval bool is_unscoped_enum_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_unscoped_enum<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is an unscoped enum type.

`consteval bool has_unscoped_enum_type(info reflection)`

Returns true if the reflected entity has unscoped enum type.

`consteval bool is_scoped_enum_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_scoped_enum<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a type that is a scoped enum.

`consteval bool has_scoped_enum_type(info reflection)`

Returns true if the reflected entity has scoped enum type.

`consteval bool is_void_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_void<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a void type.

`consteval bool has_void_type(info reflection)`

Returns true if the reflected entity has void type.

`consteval bool is_null_pointer_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_nullptr<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a null pointer type.

`consteval bool has_null_pointer_type(info reflection)`

Returns true if the reflected entity has null pointer type.

`consteval bool is_integral_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_integral<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is an integral type.

`consteval bool has_integral_type(info reflection)`

Returns true if the reflected entity has integral type.

`consteval bool is_floating_point_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_floating_point<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a floating point type.

`consteval bool has_floating_point_type(info reflection)`

Returns true if the reflected entity has floating point type.

`consteval bool is_array_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_array<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is an array type.

`consteval bool has_array_type(info reflection)`

Returns true if the reflected entity has array type.

`consteval bool is_pointer_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_pointer<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a pointer type.

`consteval bool has_pointer_type(info reflection)`

Returns true if the reflected entity has pointer type.

`consteval bool is_reference_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_reference<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a reference type.

`consteval bool has_reference_type(info reflection)`

Returns true if the reflected entity has reference type.

`consteval bool is_lvalue_reference_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_lvalue_reference<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is an lvalue reference type.

`consteval bool has_lvalue_reference_type(info reflection)`

Returns true if the reflected entity has lvalue reference type.

`consteval bool is_rvalue_reference_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_rvalue_reference<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is an rvalue reference type.

`consteval bool has_rvalue_reference_type(info reflection)`

Returns true if the reflected entity has rvalue reference type.

`consteval bool is_member_pointer_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_member_pointer<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a member pointer type.

`consteval bool has_member_pointer_type(info reflection)`

Returns true if the reflected entity has member pointer type.

`consteval bool is_member_object_pointer_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_member_object_pointer<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a member object pointer type.

`consteval bool has_member_object_pointer_type(info reflection)`

Returns true if the reflected entity has member object pointer type.

`consteval bool is_member_function_pointer_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_member_function_pointer<T>::value` when `type` is `reflexpr(T)`.

Returns true if the reflected entity is a member function pointer type.

`consteval bool has_member_function_pointer_type(info reflection)`

Returns true if the reflected entity has member function pointer type.

`consteval bool is_closure_type(info type)`

Requires: `is_type(type)`

Returns true if the reflected entity is a closure type.

`consteval bool has_closure_type(info reflection)`

Returns true if the reflected entity has closure type.

Type Properties

`consteval bool is_incomplete_type(info type)`

Requires: `is_type(type)`

Returns true if the reflected entity is an incomplete type.

`consteval bool has_incomplete_type(info reflection)`

Returns true if the reflected entity has incomplete type.

`consteval bool is_const_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_const<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a const qualified type.

`consteval bool has_const_type(info reflection)`

Returns true if the reflected entity has a const qualified type.

`consteval bool is_volatile_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_volatile<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a volatile qualified type.

`consteval bool has_volatile_type(info reflection)`

Returns true if the reflected entity has volatile qualified type.

`consteval bool is_trivial_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_trivial<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a trivial type.

`consteval bool has_trivial_type(info reflection)`

Returns true if the reflected entity has trivial type.

`consteval bool is_trivially_copyable_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_trivially_copyable<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a trivially copyable type.

`consteval bool has_trivially_copyable_type(info reflection)`

Returns true if the reflected entity has trivially copyable type.

`consteval bool is_standard_layout_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_standard_layout<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a standard layout type.

`consteval bool has_standard_layout_type(info reflection)`

Returns true if the reflected entity has standard layout type.

`consteval bool is_pod_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_pod<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a pod type.

`consteval bool has_pod_type(info reflection)`

Returns true if the reflected entity has a pod type.

`consteval bool is_literal_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_literal<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a literal type.

`consteval bool has_literal_type(info reflection)`

Returns true if the reflected entity has literal type.

`consteval bool is_empty_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_empty<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is an empty type.

`consteval bool has_empty_type(info reflection)`

Returns true if the reflected entity has empty type.

`consteval bool is_polymorphic_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_polymorphic<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a polymorphic type.

`consteval bool has_polymorphic_type(info reflection)`

Returns true if the reflected entity has polymorphic type.

`consteval bool is_abstract_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_abstract<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is an abstract type.

`consteval bool has_abstract_type(info reflection)`

Returns true if the reflected entity has abstract type.

`consteval bool is_final_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_final<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a final type.

`consteval bool has_final_type(info reflection)`

Returns true if the reflected entity has final type.

`consteval bool is_aggregate_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_aggregate<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is an aggregate type.

`consteval bool has_arggregate_type(info reflection)`

Returns true if the reflected entity has aggregate type.

`consteval bool is_signed_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_signed<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a signed type.

`consteval bool has_signed_type(info reflection)`

Returns true if the reflected entity has signed type.

`consteval bool is_unsigned_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_unsigned<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is an unsigned type.

`consteval bool has_unsigned_type(info reflection)`

Returns true if the reflected entity has unsigned type.

`consteval bool has_unique_object_representations(info type)`

Requires: `is_type(type)`

Equivalent to `std::has_unique_object_representations<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is a type with unique object representations.

`consteval bool has_type_with_unique_object_representations(info reflection)`

Returns true if the reflected entity has a type with unique object representations.

Type Operations

`consteval bool is_default_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_default_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is default constructible.

`consteval bool has_default_constructible_type(info reflection)`

Returns true if the reflected entity has default constructible type.

`consteval bool is_trivially_default_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_trivially_default_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is trivially default constructible.

`consteval bool has_trivially_default_constructible_type(info reflection)`

Returns true if the reflected entity has trivially default constructible type.

`consteval bool is_nothrow_default_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_nothrow_default_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is nothrow default constructible.

`consteval bool has_nothrow_default_constructible_type(info reflection)`

Returns true if the reflected entity has trivially nothrow constructible type.

`consteval bool is_copy_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_copy_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is copy constructible.

`consteval bool has_copy_constructible_type(info reflection)`

Returns true if the reflected entity has copy constructible type.

`consteval bool is_trivially_copy_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_trivially_copy_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is trivially copy constructible.

`consteval bool has_trivially_copy_constructible_type(info reflection)`

Returns true if the reflected entity has trivially copy constructible type.

`consteval bool is_nothrow_copy_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_nothrow_copy_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is nothrow copy constructible.

`consteval bool has_nothrow_copy_constructible_type(info reflection)`

Returns true if the reflected entity has copy constructible type.

`consteval bool is_move_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_move_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is move constructible.

`consteval bool has_move_constructible_type(info reflection)`

Returns true if the reflected entity has move constructible type.

`consteval bool is_trivially_move_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_trivially_move_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is trivially move constructible.

`consteval bool has_trivially_move_constructible_type(info reflection)`

Returns true if the reflected entity has trivially move constructible type.

`consteval bool is_nothrow_move_constructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_nothrow_move_constructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is nothrow move constructible.

`consteval bool has_nothrow_move_constructible_type(info reflection)`

Returns true if the reflected entity has nothrow move constructible type.

`consteval bool is_assignable_type(info type, info assigned_type)`

Requires: `is_type(type) and is_type(assigned_type)`

Equivalent to `std::is_assignable<T, U>::value` where `type` is `reflexpr(T), and assigned_type is reflexpr(U)`.

Returns true if the reflected entity of assigned_type is assignable to the reflected entity of type.

`consteval bool is_trivially_assignable_type(info type, info assigned_type)`

Requires: `is_type(type) and is_type(assigned_type)`

Equivalent to `std::is_trivially_assignable<T, U>::value` where `type` is `reflexpr(T), and assigned_type is reflexpr(U)`.

Returns true if the reflected entity of assigned_type is trivially assignable to the reflected entity of type.

`consteval bool is_nothrow_assignable_type(info type, info assigned_type)`

Requires: `is_type(type) and is_type(assigned_type)`

Equivalent to `std::is_nothrow_assignable<T, U>::value` where `type` is `reflexpr(T), and assigned_type is reflexpr(U)`.

Returns true if the reflected entity of assigned_type is nothrow assignable to the reflected entity of type.

`consteval bool is_copy_assignable_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_copy_assignable<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is copy assignable.

`consteval bool has_copy_assignable_type(info reflection)`

Returns true if the reflected entity has copy assignable type.

`consteval bool is_trivially_copy_assignable_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_trivially_copy_assignable<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is trivially copy assignable.

`consteval bool has_trivially_copy_assignable_type(info reflection)`

Returns true if the reflected entity has trivially copy assignable type.

`consteval bool is_nothrow_copy_assignable_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_nothrow_copy_assignable<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is nothrow copy assignable.

`consteval bool has_nothrow_copy_assignable_type(info reflection)`

Returns true if the reflected entity has nothrow copy assignable type.

`consteval bool is_move_assignable_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_move_assignable<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is move assignable.

`consteval bool has_move_assignable_type(info reflection)`

Returns true if the reflected entity has move assignable type.

`consteval bool is_trivially_move_assignable_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_trivially_move_assignable<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is trivially move assignable.

`consteval bool has_trivially_move_assignable_type(info reflection)`

Returns true if the reflected entity has move assignable type.

`consteval bool is_nothrow_move_assignable_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_nothrow_move_assignable<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is nothrow move assignable.

`consteval bool has_nothrow_move_assignable_type(info reflection)`

Returns true if the reflected entity has nothrow move assignable type.

`consteval bool is_destructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_destructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is destructible.

`consteval bool has_destructible_type(info reflection)`

Returns true if the reflected entity has destructible type.

`consteval bool is_trivially_destructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_trivially_destructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is trivially destructible.

`consteval bool has_trivially_destructible_type(info reflection)`

Returns true if the reflected entity has trivially destructible type.

`consteval bool is_nothrow_destructible_type(info type)`

Requires: `is_type(type)`

Equivalent to `std::is_nothrow_destructible<T>::value` where `type` is `reflexpr(T)`.

Returns true if the reflected entity is nothrow destructible.

`consteval bool has_nothrow_destructible_type(info reflection)`

Returns true if the reflected entity has nothrow destructible type.

Type Transformations

`consteval info remove_const(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::remove_const<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection with the const qualifier removed, if present.

`consteval info remove_volatile(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::remove_volatile<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection with volatile qualifier removed, if present.

`consteval info remove_cv(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::remove_cv<T>::type)` where `type` is `reflexpr(T)`.

Shorthand returning the result of `remove_volatile(remove_const(type))`.

`consteval info add_const(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::add_const<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection with const qualifier added, if not already present.

`consteval info add_volatile(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::add_volatile<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection with volatile qualifier added, if not already present.

`consteval info add_cv(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::add_cv<T>::type)` where `type` is `reflexpr(T)`.

Shorthand returning the result of `add_const(add_volatile(type))`.

`consteval info remove_reference(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::remove_reference<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection which reflects the type referred to by the reference type. If the provided type reflection does not reflect a reference type, returns a new type reflection of equivalent type.

`consteval info add_lvalue_reference(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::remove_lvalue_reference<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection of an lvalue reference type to the type reflected by the provided type reflection. If the provided type reflection does not reflected a referenceable type, returns a new type reflection of equivalent type.

`consteval info add_rvalue_reference(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::remove_rvalue_reference<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection of an rvalue reference type to the type reflected by the provided type reflection. If the provided type reflection does not reflected a referenceable type, returns a new type reflection of equivalent type.

`consteval info remove_pointer(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::remove_pointer<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection of the type pointed to by the type reflected by the provided type reflection. If the provided type reflection does not reflected a pointer type, returns a new type reflection of equivalent type.

`consteval info add_pointer(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::add_pointer<T>::type)` where `type` is `reflexpr(T)`.

Returns a new type reflection of a pointer type pointing to the type reflected by the provided type reflection. If the provided type reflection does not reflect a type which can be pointed to, returns a new type reflection of equivalent type.

`consteval info remove_cvref(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::remove_cvref<T>::type)` where `type` is `reflexpr(T)`.

Shorthand returning the result of `remove_cv(remove_reference(type))`.

`consteval info decay(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::decay<T>::type)` where `type` is `reflexpr(T)`.

If reflected type names the type "array of U" or "reference to array of U", returns a new type reflection of U*.

If T is a function type F or a reference thereto, equivalent to add_pointer(type).

Otherwise, equivalent to remove_cvref(type).

`consteval info make_signed(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::make_signed<T>::type)` where `type` is `reflexpr(T)` without undefined behavior.

If `type` is a type reflection of an integral (except bool) or enumeration type, returns a new type reflection of the signed integer type corresponding said reflected type, with the same cv-qualifiers.

If type is a type reflection of a signed integral type, returns a new type reflection of equivalent type.

Otherwise, returns an invalid reflection.

`consteval info make_unsigned(info type)`

Requires: `is_type(type)`

Equivalent to `reflexpr(std::make_unsigned<T>::type)` where `type` is `reflexpr(T)` without undefined behavior.

If `type` is a type reflection of an integral (except bool) or enumeration type, returns a new type reflection of the unsigned integer type corresponding said reflected type, with the same cv-qualifiers. The unsigned

integer type corresponding to an enumeration type is the unsigned integer type with the smallest rank having the same `sizeof` as the enumeration.

If type is a type reflection of an unsigned integral type, returns a new type reflection of equivalent type.

Otherwise, returns an invalid reflection.

Associated Types

`consteval info this_ref_type_of(info mem_function)`

Requires: `is_member_function(mem_function)`

Given a reflection, `mem_function`, of a member function, `f`, return a reflection of the type of the `this` reference of `f`.

`consteval info underlying_type_of(info reflection)`

Returns a reflection of the underlying type of an enumeration.

Example:

```
enum byte : unsigned char {};
constexpr info r = underlying_type(reflexpr(byte));
typename(r); // unsigned char
```

`consteval info type_of(info reflection)`

Returns a reflection to the type of the reflected entity.

`consteval info return_type_of(info function)`

Requires: `is_function(function)`

Returns a reflection to the return type of the reflected entity.

Associated Reflections

`consteval info entity_of(info reflection)`

Returns a reflection of the entity of `reflection`.

In the case of a reflected type, returns a reflection of the canonical type.

In the case of a reflected declaration, returns a reflection of the canonical declaration.

In the case of a reflected expression, if a canonical declaration is associated, returns a reflection of the associated canonical declaration.

In the case of a base specifier, returns a reflection of the canonical type named by the base specifier.

Otherwise, returns an invalid reflection.

`consteval info parent_of(info reflection)`

Returns a reflection to the lexical context of the declaration reflected by `reflection`.

Example:

```
struct S {
  struct T {};
};

parent_of(reflexpr(S::T)); // reflexpr(S)
```

`consteval info definition_of(info reflection)`

Returns a reflection of the declaration defining the reflected entity.
If this reflected entity has no associated defining declaration, returns an invalid reflection.

Names

`consteval bool is_named(info reflection)`

Returns true if the reflected entity has a name.

`consteval std::string name_of(info named)`

Requires: `is_named(named)`

Returns the name of the reflected entity.

Expressions

`consteval bool is_lvalue(info reflection)`

Returns true if `reflection` reflects an lvalue expression.

`consteval bool is_xvalue(info reflection)`

Returns true if `reflection` reflects an xvalue expression.

`consteval bool is_prvalue(info reflection)`

Returns true if `reflection` reflects a prvalue expression.

`consteval bool is_glvalue(info reflection)`

Returns true if `reflection` reflects a glvalue expression.

`consteval bool is_rvalue(info reflection)`

Returns true if `reflection` reflects an rvalue expression.

Traversal

`constexpr info front_member(info reflection)`

Returns the first member of the reflected entity.

`constexpr info next_member(info reflection)`

Returns the next member in the sequence of declarations of which the reflected entity is a member of.

`constexpr info front_param(info reflection)`

Returns the first function parameter of the reflected entity.

`constexpr info next_param(info reflection)`

Returns the next function parameter in the sequence of function parameters of which the reflected entity is a member of.

`constexpr info front_template_param(info reflection)`

Returns the first template parameter of the reflected entity.

`constexpr info next_template_param(info reflection)`

Returns the next template parameter in the sequence of template parameters of which the reflected entity is a member of.

`constexpr info front_base_spec(info reflection)`

Returns the first base specifier of the reflected entity.

`constexpr info next_base_spec(info reflection)`

Returns the next base specifier in the sequence of base specifiers of which the reflected entity is a member of.

`member_iterator`

An iterator built upon `front_member`, and `next_member` for traversal of members of a class or namespace.

`member_range`

A range built upon `member_iterator`s for range based traversal of members of a class or namespace.

`member_fn_iterator`

An iterator built upon `front_member`, and `next_member`, that filters on the result of `is_member_function`, for traversal of member functions in a class.

`member_fn_range`

A range built upon `member_fn_iterator`s for range based traversal of member functions in a class.

`data_member_iterator`

An iterator built upon `front_member`, and `next_member`, that filters on the result of `is_data_member`, for traversal of data members in a class.

`data_member_range`

A range built upon `data_member_iterator`s for range based traversal of data members in a class.

`param_iterator`

An iterator built upon `front_param`, and `next_param` for traversal of function parameters.

`param_range`

A range built upon `param_iterator`s for range based traversal of function parameters.

`template_param_iterator`

An iterator built upon `front_template_param`, and `next_template_param` for traversal of template parameters of templated entities.

`template_param_range`

A range built upon `template_param_iterator`s for range based traversal of templated parameters of templated entities.

`base_spec_iterator`

An iterator built upon `front_base_spec`, and `next_base_spec` for traversal of base specifiers.

`base_spec_range`

A range built upon `base_spec_iterator`s for range based traversal of templated parameters of templated entities.