Ogonek Documentation

Release 0.6.0

R. Martinho Fernandes

Sep 29, 2019

Contents:

1 About

Ogonek is a C++ library for Unicode support. The goal is to support all the algorithms described in the Unicode Standard[1], with a focus on correctness and on the use of modern C++.

1.1 Design goals

Ogonek's design is driven by the following principles.

validity and correctness Ogonek values validity and correctness above speed and other concerns. The semantics of operations are defined as much as possible by the Unicode Standard and its standard annexes (UAX), technical standards (UTS), and technical reports (UTR).

Ideally it should be impossible to obtain invalid Unicode data after any operation provided by Ogonek.

modern C++ Ogonek uses modern C++ techniques as much as possible. At any given point, the minimum required level of standard C++ support is the C++ standard version before the latest, starting with C++14.

explicit is better than implicit Ogonek performs as little as possible implicitly. If the users don't want to care about certain details, they don't care about correctness, and thus they probably don't need ogonek.

Specifically, Ogonek does not silently perform encoding conversions, and does not assume any encoding except for `char16_t` and `char32_t` (which are clearly intended by the standard as UTF-16 and UTF-32 code units); anything else needs to be made explicit.

fail fast Ogonek does not let errors go away silently. When appropriate, the API accepts error handling callbacks; in all other scenarios exceptions are thrown.

be a good citizen Ogonek works well with the standard library, by providing and using models of the existing standard concepts, like iterators and containers.

[1] http://www.unicode.org/standard/

1.2 Dependencies

Ogonek depends on the following third-party libraries.

range-v3[2] Eric Niebler's range-v3 library provides machinery for implementing and using ranges.

Boost[3] Boost.Optional, Boost.Rational and Boost.Tribool are used for certain interface types.

Catch[4] Phil Nash's Catch library provides unit test support.

1.3 Versioning

Ogonek version numbers use the semantic versioning 2.0.0[5] rules.

Each Ogonek release is tied to a specific Unicode version. This is unavoidable because Unicode updates may require changes not only in the Unicode character data, but also in the algorithms. Because of this, arbitrarily mixing Ogonek versions and Unicode versions is not supported.

At any given time, Ogonek supports the two latest released Unicode major versions, starting with Unicode 9.0.0.

1.4 Licensing

The source code of Ogonek is dedicated to the public domain. Where not applicable, source code is licensed under the terms of the CC0 1.0 Universal[6] public domain dedication.

Note: The person who associated a work with this deed has dedicated the work to the public domain by waiving all of his or her rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent allowed by law.

You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission.

In no way are the patent or trademark rights of any person affected by CC0, nor are the rights that other persons may have in the work or in how the work is used, such as publicity or privacy rights.

Unless expressly stated otherwise, the person who associated a work with this deed makes no warranties about the work, and disclaims liability for all uses of the work, to the fullest extent permitted by applicable law.

When using or citing the work, you should not imply endorsement by the author or the affirmer.

---

[2] https://github.com/ericniebler/range-v3/
[3] https://boost.org/
[4] https://github.com/philsquared/Catch/
[5] http://semver.org/spec/v2.0.0.html
[6] https://creativecommons.org/publicdomain/zero/1.0/

1.5 Trivia

Ogonek means “little tail” in Polish. It is the name of a diacritic used in several European and Native American languages. It exists in the Unicode repertoire as a combining character (U+0328 COMBINING OGONEK), as an isolated character (U+02DB OGONEK), or precomposed with Latin alphabet letters (like U+01EA LATIN CAPITAL LETTER O WITH OGONEK).

The name was picked randomly and it has absolutely no special meaning as the name of this project. It is just a label the author appropriated for it. In English it is pronounced /ˈoʊgənɛk/.

2 How to use this manual

This manual follows certain conventions in the description of Ogonek.

2.1 Other documents

This manual often makes reference to the Unicode Standard[7] and the terms described within. Familiarity with those terms is recommended before reading this manual.

2.2 Namespaces

Unless otherwise mentioned, all entities described are in the namespace ogonek.

2.3 Reference sections

In reference pages, entities are described using following sections.

Invariants conditions that hold after any operation on an object of that class.

Requires preconditions that must hold before calling that function; the behaviour is undefined if the function is called and preconditions are not met.

Effects the actions performed by the function

Postconditions the observable results established by the function

Returns the value returned by that function.

Validation what kind of data validation is performed by that function, and how invalid data is treated.

Complexity the time and/or space complexity of that function.

Remarks additional semantics of that function.

[7] http://www.unicode.org/standard/

Notes additional information about that entity.

Sections that are not applicable are simply omitted.

3 Ranges

A large portion of ogonek works with sequences of values, notably of code points, of code units and of bytes. Throughout the library, the range-v3[8] concepts are used to specify the requirements of input ranges and the capabilities of returned ranges.

Note: This is under consideration and may change in the future as the range-v3 interfaces change, or as the standardization process of ranges proceeds.

Note that most ranges returned from ogonek algorithms have deferred-evaluation. This means that nothing happens until the values of the range are actually required.

[8] https://github.com/ericniebler/range-v3/

4 Character properties

Ogonek exposes the character properties described in the Unicode Standard[9] and in the Unicode Standard Annex #44[10].

The types of these properties are mapped to C++ types as follows.

Unicode type	C++ type
Catalog	specific scoped enum type
Enumeration	specific scoped enum type
Binary	`bool`
String	`std::u32string`
Numeric	`boost::rational<long>`
Miscellaneous	varies

In general, property names and values match the Unicode standard names, except for being written in lowercase and with underscores instead of spaces (i.e. in the style of the C++ standard library).

Todo: Add examples

---

[9] http://www.unicode.org/standard/
[10] http://unicode.org/reports/tr44/

5 Reference

5.1 Concepts

template<typename E>
`concept EncodingForm`
: An *encoding form* type. It describes an encoding form's mapping between *code points* and *code units*, as well as some extra properties which are used for checking constraints and for performing optimizations.

Todo: Document requirements

---

template<typename E>
`concept StatelessEncodingForm`
: An *encoding form* that requires no state for encoding/decoding operations.

Todo: Document requirements

---

template<typename O>
`concept Optional`
: A type of objects that contain an optional value.

Todo: Document requirements

---

template<typename H, typename E>
`concept EncodeErrorHandler`
: H is an error handler that can handle errors when encoding with E.

Todo: Document requirements

---

template<typename H, typename E>
`concept DecodeErrorHandler`
: H is an error handler that can handle errors when encoding with E.

Todo: Document requirements

template<typename N>
concept NormalizationForm
N is a Unicode normalization form.

Todo: Document requirements

5.2 Encoding

template<*EncodingForm* Encoding>
using code_unit_t
The type of the *code units* used by Encoding.

template<*EncodingForm* Encoding>
constexpr auto max_width_v
The maximum number of *code units* needed by Encoding to represent one *code point*.

template<*EncodingForm* Encoding>
constexpr auto replacement_character_v
The character used to replace invalid input when replace_errors is used for an encoding operation.

Note: Decoding always uses U+FFFD REPLACEMENT CHARACTER to replace errors.

template<*EncodingForm* Encoding>
using coding_state_t
The type of the state used by Encoding. For stateless encoding forms this is an empty type.

template<*EncodingForm* Encoding>
using is_stateless
Derives from std::true_type if Encoding is stateless; from std::false_type otherwise.

template<*EncodingForm* Encoding>
constexpr bool is_stateless_v
True if Encoding is stateless; false otherwise.

template<*EncodingForm* Encoding>
coded_character_t<*Encoding*> encode_one(*code_point u*, *coding_state_t*<*Encoding*> &*state*)

Warning: This part of the interface is still being worked out; it is likely to change signficantly or to be removed before the next release.

Encodes u according to Encoding.

Parameters

- `u` – the *code point* to be encoded
- `state` – the current encoding state; it is modified according to the encoding performed

Returns a range of the *code units* that encode `u`

Throws `encode_error` when the input cannot be encoded by `Encoding`

template<*EncodingForm* `Encoding`>
`using coded_character_t`
Range of the *code units* that encode one *code point*.

template<*EncodingForm* `Encoding`, ForwardRange `Rng`, *EncodeErrorHandler* `Handler`>
auto `encode`(*Rng* *rng*, *Handler* &&*handler*)
Encodes a range of *code points*, according to `Encoding`.

Parameters

- `rng` – The range of *code points* to encode
- `handler` – The strategy for error handling

Returns a range of the *code units* that encode the *code points* in `rng`

Validation as performed by `Encoding`; errors are handled by `handler`

template<*EncodingForm* `Encoding`, ForwardIterator `It`, Sentinel `St`>
std::pair<*code_point*, *It*> `decode_one`(*It* *first*, *St* *last*, *coding_state_t*<*Encoding*> &*state*)

> **Warning:** This part of the interface is still being worked out; it is likely to change signficantly or to be removed before the next release.

Decodes the first *code point* from the range [`first`, `last`), according to `Encoding`.

Parameters

- `first` – an iterator to the first *code unit* to be decoded
- `last` – an iterator/sentinel to the end of the range of *code units*
- `state` – the current decoding state; it is modified according to the decoding performed

Returns a pair of the decoded *code point* and an iterator to first *code unit* of the next encoded *code point*

Throws `decode_error` when the input cannot be decoded by `Encoding`

template<*EncodingForm* `Encoding`, ForwardRange `Rng`, *EncodeErrorHandler* `Handler`>
auto `decode`(*Rng* *rng*, *Handler* &&*handler*)
Decodes a range of *code points*, according to `Encoding`.

Parameters

- `rng` – The range of *code points* to encode
- `handler` – The strategy for error handling

Returns a range of the *code points* that the *code units* in `rng` represent

Validation as performed by `Encoding`; errors are handled by `handler`

template<*EncodingForm* `Encoding`>

`using coded_character`
: A container type for the *code units* that encode a single *code point* according to `Encoding`.

5.3 Error handling

`class unicode_error`
: The base class for all Unicode-related errors.

template<*EncodingForm* `Encoding`>
`class encode_error : virtual` *unicode_error*

: **Thrown** when an error occurs during an encoding operation.

template<*EncodingForm* `Encoding`>
`class decode_error : virtual` *unicode_error*

: **Thrown** when an error occurs during a decoding operation.

auto `assume_valid`
: A tag used to request that encoding/decoding functions assume the input has been validated before.

> **Warning:** Using this tag with input that isn't actually valid yields undefined behavior.

auto `discard_errors`
: An error handler for encoding/decoding functions that simply discards the portions of the input that have errors.

auto `replace_errors`
: An error handler for encoding/decoding functions that replaces portions of the input that have errors with a replacement character. When decoding, this is U+FFFD REPLACEMENT CHARACTER, but when encoding and the target doesn't support it, some encoding-specific character is used instead.

auto `throw_error`
: An error handler for encoding/decoding functions that throws when an error is found in the input.

5.4 Normalization

template<*NormalizationForm* `Form`, OutputIterator `Out`>
auto `decompose_into(`*code_point* *u*, *Out* *out*`)`

> **Warning:** This part of the interface is still being worked out; it is likely to change signficantly or to be removed before the next release.

Writes the full decomposition of a *code point* into an output iterator, according to `Form`.

Parameters

- u – The *code points* to decompose
- out – The iterator to write to

template<*NormalizationForm* Form, ForwardRange Rng>
auto compose(*Rng rng*)

> **Warning:** This part of the interface is still being worked out; it is likely to change signficantly or to be removed before the next release.

Applies the Canonical Composition Algorithm to range.

Parameters rng – A range of *code points*

Returns A canonically-composed subrange of rng

template<*NormalizationForm* Form, ForwardRange Rng>
auto normalize(*Rng rng*)

Normalizes a range of *code points* into Form.

Parameters rng – The range of *code points* to normalize

Returns a range of the *code points* that satisfies the normalization form Form

5.5 Text container

Todo: Flesh this out

5.6 Core types

using code_point = char32_t

Many algorithms in ogonek operate on sequences of code points. The sequences themselves can have any type, but their elements must be code points. code_point is the type used to represent code points. It is an alias for the standard C++ char32_t type.

All code_point arguments in Ogonek are required to be valid, i.e. in the range $[0..10FFFF_{16}]$; otherwise the behaviour is undefined. The results of Ogonek functions never include code_point values outside this range.

using byte = std::uint8_t

Encoding schemes and some encoding forms take input or produce output in the form of bytes or sequences of bytes. The type used to represent bytes in ogonek is called byte, an alias for the standard C++ uint8_t type.

5.7 Fraction type

class fraction

A fraction type to support the Numeric_Value Unicode property. This is merely a numer-

ator/denominator pair and does not perform any reduction nor provides any arithmetic functionality.

`constexpr fraction() noexcept`
Creates a new fraction representing 0/1.

`constexpr fraction(long` *numerator*, `long` *denominator*`) noexcept`
Creates a new fraction representing `numerator/denominator`.

Parameters

- `numerator` – the fraction's numerator
- `denominator` – the fraction's denominator

Requires `denominator` is not zero

`constexpr long numerator() const noexcept`

Returns the numerator of this fraction

`constexpr long denominator() const noexcept`

Returns the denominator of this fraction

`constexpr double as_double() const noexcept`

Returns the result of dividing the numerator by the denominator of this fraction, as a `double`.

Note: This feature is only available if the `OGONEK_USE_BOOST` macro is set.

`constexpr fraction(boost::rational<long>` *r*`) noexcept`
Creates a new fraction from a `boost::rational<long>`.

`constexpr operator boost::rational<long>() const noexcept`

Returns a `boost::rational<long>` equivalent to this fraction.

`constexpr bool operator==(`*fraction lhs*, *fraction rhs*`) noexcept`
Compares two fractions for equality.

Returns `true` if both fractions have the same numerator and the same denominator; `false` otherwise.

`constexpr bool operator!=(`*fraction lhs*, *fraction rhs*`) noexcept`
Compares two fractions for inequality.

Returns `true` if both fractions have the different numerators or different denominators; `false` otherwise.

5.8 Optional type

`class none_t`
The unit type of the none constant.

`constexpr` *none_t* `none`
A constant used to create empty optional objects.

template<typename T>

`class optional`
An optional object that may or may not contain a value.

`constexpr optional() noexcept`
Creates a new empty optional object.

`constexpr optional(`*none_t*`) noexcept`
Creates a new empty optional object.

`constexpr optional(`*T t*`) noexcept`
Creates a new optional object.

Parameters `t` – the value of the new optional object

`constexpr` *T* `operator*() const noexcept`

Requires this optional isn't empty.

Returns the value of this optional.

`explicit constexpr operator bool() const noexcept`

Returns `true` if this optional has a value; `false` if it's empty.

Note: This feature is only available when compiling with C++17 support.

`constexpr optional(`std::optional<*T*> *o*`) noexcept`
Creates a new optional from a `std::optional`.

Parameters `o` – the source

Returns an optional with the same value as `o`

`constexpr operator std::optional<T>() const noexcept`
Converts an optional to `std::optional`.

Returns a `std::optional` with the same value as this optional

Note: This feature is only available if the `OGONEK_USE_BOOST` macro is set.

`constexpr optional(`boost::optional<*T*> *o*`) noexcept`
Creates a new optional from a `boost::optional`.

Parameters `o` – the source

Returns an optional with the same value as `o`

`constexpr operator boost::optional<T>() const noexcept`
Converts an optional to `boost::optional`.

Returns a `boost::optional` with the same value as this optional

---

template<typename T, typename U>
`constexpr bool operator==(`*optional*<*T*> `const &`*lhs*, *optional*<*U*> `const &`*rhs*`) noexcept`

template<typename T>
`constexpr bool operator==(`*optional*<*T*> `const &`*lhs*, *T* `const &`*rhs*`) noexcept`

template<typename T>
`constexpr bool operator==(`*T* `const &`*lhs*, *optional*<*T*> `const &`*rhs*`) noexcept`

template<typename T>

`constexpr bool operator==(`*optional*<*T*>` const &`*lhs*, *none_t*`) noexcept`

template<typename T>
`constexpr bool operator==(`*none_t*, *optional*<*T*>` const &`*rhs*`) noexcept`
Compares optional for equality.

Returns `true` if both sides have the same value, or if neither side has a value; `false` otherwise.

template<typename T, typename U>
`constexpr bool operator!=(`*optional*<*T*>` const &`*lhs*, *optional*<*U*>` const &`*rhs*`) noexcept`

template<typename T>
`constexpr bool operator!=(`*optional*<*T*>` const &`*lhs*, *T* `const &`*rhs*`) noexcept`

template<typename T>
`constexpr bool operator!=(`*T* `const &`*lhs*, *optional*<*T*>` const &`*rhs*`) noexcept`

template<typename T>
`constexpr bool operator!=(`*optional*<*T*>` const &`*lhs*, *none_t*`) noexcept`

template<typename T>
`constexpr bool operator!=(`*none_t*, *optional*<*T*>` const &`*rhs*`) noexcept`
Compares optional for inequality.

Returns `!(lhs == rhs)`

5.9 3-state boolean type

`class maybe_t`
The unit type of the `maybe` constant.

`constexpr bool operator()(`*tribool t*`) const noexcept`

Parameters `t` – a `tribool` to check for a maybe state

Returns `true` if `t` is maybe; `false` otherwise

`constexpr` *maybe_t* `maybe`
A constant used to represent the third boolean state.

`class tribool`
A 3-state boolean type. The third state represents a superposition of the `true` and `false` states.

`constexpr tribool() noexcept`
Creates a new tribool, with the `false` value.

`constexpr tribool(bool` *v*`) noexcept`
Creates a new tribool, with the given value.

Parameters `v` – the value of the new tribool

`constexpr tribool(`*maybe_t*`) noexcept`
Creates a new tribool, with the maybe state.

`explicit constexpr operator bool() const noexcept`

Returns `true` if the state is true; `false` otherwise.

Note: This feature is only available if the `OGONEK_USE_BOOST` macro is set.

`constexpr tribool(boost::tribool` *t*`) noexcept`
Creates a new tribool with the value of a `boost::tribool`.

`constexpr operator boost::tribool() const noexcept`

Returns a `boost::tribool` equivalent to this tribool.

`constexpr` *tribool* `operator!(`*tribool* *t*`) noexcept`
Negates a tribool.

Returns `maybe` if *t* is `maybe`; otherwise `!bool(t)`

`constexpr` *tribool* `operator&&(`*tribool* *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator&&(`*tribool* *lhs*, bool *rhs*`) noexcept`

`constexpr` *tribool* `operator&&(`bool *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator&&(`*maybe_t* *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator&&(`*tribool* *lhs*, *maybe_t* *rhs*`) noexcept`
Logical conjunction of tribools.

Returns `maybe` if either side is `maybe`; otherwise `bool(lhs) && bool(rhs)`

`constexpr` *tribool* `operator||(`*tribool* *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator||(`*tribool* *lhs*, bool *rhs*`) noexcept`

`constexpr` *tribool* `operator||(`bool *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator||(`*maybe_t* *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator||(`*tribool* *lhs*, *maybe_t* *rhs*`) noexcept`
Logical disjunction of tribools.

Returns `true` if either side is `true`; otherwise `maybe` if either side is `maybe`; otherwise `false`

`constexpr` *tribool* `operator==(`*tribool* *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator==(`*tribool* *lhs*, bool *rhs*`) noexcept`

`constexpr` *tribool* `operator==(`bool *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator==(`*maybe_t* *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator==(`*tribool* *lhs*, *maybe_t* *rhs*`) noexcept`
Compares tribools for equality.

Returns `maybe` if either side is `maybe`; otherwise `bool(lhs) == bool(rhs)`

`constexpr` *tribool* `operator!=(`*tribool* *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator!=(`*tribool* *lhs*, bool *rhs*`) noexcept`

`constexpr` *tribool* `operator!=(`bool *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator!=(`*maybe_t* *lhs*, *tribool* *rhs*`) noexcept`

`constexpr` *tribool* `operator!=(`*tribool* *lhs*, *maybe_t* *rhs*`) noexcept`
Compares tribools for inequality.

Returns `maybe` if either side is `maybe`; otherwise `bool(lhs) != bool(rhs)`

5.10 Unicode character database

Note: The items in this section are in the namespace ogonek::ucd.

version `get_age`(*code_point* *u*)

Returns the *Age* property of u

std::string `get_name`(*code_point* *u*)

Returns the *Name* property of u

block `get_block`(*code_point* *u*)

Returns the *Block* property of u

block `get_general_category`(*code_point* *u*)

Returns the *General_Category* property of u

combining_class `get_canonical_combining_class`(*code_point* *u*)

Returns the *Canonical_Combining_Class* property of u

bidi_class `get_bidi_class`(*code_point* *u*)

Returns the *Bidi_Class* property of u

bool `is_bidi_mirrored`(*code_point* *u*)

Returns true if u has the *Bidi_Mirrored* property; false otherwise

code_point `get_bidi_mirroring_glyph`(*code_point* *u*)

Returns the *Bidi_Mirroring_Glyph* property of u

bool `is_bidi_control`(*code_point* *u*)

Returns true if u has the *Bidi_Control* property; false otherwise

code_point `get_bidi_paired_bracket`(*code_point* *u*)

Returns the *Bidi_Paired_Bracket* property of u

bracket_type `get_bidi_paired_bracket_type`(*code_point* *u*)

Returns the *Bidi_Paired_Bracket_Type* property of u

decomposition_type `get_decomposition_type`(*code_point* *u*)

Returns the *Decomposition_Type* property of u

std::u32string `get_decomposition_mapping`(*code_point* *u*)

Returns the *Decomposition_Mapping* property of u

bool `is_excluded_from_composition`(*code_point* *u*)

Returns true if u has the *Full_Composition_Exclusion* property; false otherwise

tribool `get_nfc_quick_check`(*code_point* *u*)

Returns the *NFC_Quick_Check* property of u

bool `get_nfd_quick_check`(*code_point* *u*)

Returns the *NFD_Quick_Check* property of u

tribool `get_nfkc_quick_check`(*code_point* *u*)

Returns the *NFKC_Quick_Check* property of u

bool `get_nfkd_quick_check`(*code_point* *u*)

Returns the *NFKD_Quick_Check* property of u

numeric_type `get_numeric_type`(*code_point* *u*)

Returns the *Numeric_Type* property of u

optional<*fraction*> `get_numeric_value`(*code_point* *u*)

Returns the *Numeric_Value* property of u, if present; none otherwise

joining_type `get_joining_type`(*code_point* *u*)

Returns the *Joining_Type* property of u

joining_group `get_joining_group`(*code_point* *u*)

Returns the *Joining_Group* property of u

bool `is_join_control`(*code_point* *u*)

Returns true if u has the *Join_Control* property; false otherwise

line_break `get_line_break`(*code_point* *u*)

Returns the *Line_Break* property of u

east_asian_width `get_east_asian_width`(*code_point* *u*)

Returns the *East_Asian_Width* property of u

bool `is_uppercase`(*code_point* *u*)

Returns true if u has the *Uppercase* property; false otherwise

bool `is_lowercase`(*code_point* *u*)

Returns true if u has the *Lowercase* property; false otherwise

code_point `get_simple_uppercase_mapping`(*code_point* *u*)

Returns the *Simple_Uppercase_Mapping* property of u

code_point `get_simple_lowercase_mapping`(*code_point* *u*)

Returns the *Simple_Lowercase_Mapping* property of u

code_point `get_simple_titlecase_mapping`(*code_point* *u*)

Returns the *Simple_Titlecase_Mapping* property of u

std::u32string `get_uppercase_mapping`(*code_point* *u*)

Returns the *Uppercase_Mapping* property of u

std::u32string `get_lowercase_mapping`(*code_point* *u*)

Returns the *Lowercase_Mapping* property of u

std::u32string `get_titlecase_mapping`(*code_point* *u*)

Returns the *Titlecase_Mapping* property of u

code_point `get_simple_case_folding`(*code_point* *u*)

Returns the *Simple_Case_Folding* property of u

std::u32string `get_case_folding`(*code_point* *u*)

Returns the *Case_Folding* property of u

bool `is_case_ignorable`(*code_point* *u*)

Returns true if u has the *Case_Ignorable* property; false otherwise

bool `is_cased`(*code_point* *u*)

Returns true if u has the *Cased* property; false otherwise

bool `changes_when_lowercased`(*code_point* *u*)

Returns true if u has the *Changes_When_Lowercased* property; false otherwise

bool `changes_when_uppercased`(*code_point* *u*)

Returns true if u has the *Changes_When_Uppercased* property; false otherwise

bool `changes_when_titlecased`(*code_point* *u*)

Returns true if u has the *Changes_When_Titlecased* property; false otherwise

bool `changes_when_casefolded`(*code_point* *u*)

Returns true if u has the *Changes_When_Casefolded* property; false otherwise

bool `changes_when_casemapped`(*code_point* *u*)

Returns true if u has the *Changes_When_Casemapped* property; false otherwise

bool `changes_when_nfkc_casefolded`(*code_point* *u*)

Returns true if u has the *Changes_When_NFKC_Casefolded* property; false otherwise

std::u32string `get_nfkc_casefold`(*code_point* *u*)

Returns the *NFKC_Casefold* property of u

script `get_script`(*code_point* *u*)

Returns the *Script* property of u

hangul_syllable_type `get_hangul_syllable_type`(*code_point* *u*)

Returns the *Hangul_Syllable_Type* property of u

std::string `get_jamo_short_name`(*code_point* *u*)

Returns the *Jamo_Short_Name* property of u

indic_positional_category `get_indic_positional_category`(*code_point* *u*)

Returns the *Indic_Positional_Category* property of u

indic_syllabic_category `get_indic_syllabic_category`(*code_point* *u*)

Returns the *Indic_Syllabic_Category* property of u

bool `is_id_start`(*code_point* *u*)

Returns true if u has the *ID_Start* property; false otherwise

bool `is_id_continue`(*code_point* *u*)

Returns true if u has the *ID_Continue* property; false otherwise

bool `is_xid_start`(*code_point* *u*)

Returns true if u has the *XID_Start* property; false otherwise

bool `is_xid_continue`(*code_point* *u*)

Returns true if u has the *XID_Continue* property; false otherwise

bool `is_pattern_syntax`(*code_point* *u*)

Returns true if u has the *Pattern_Syntax* property; false otherwise

bool `is_pattern_white_space`(*code_point* *u*)

Returns true if u has the *Pattern_White_Space* property; false otherwise

bool `is_dash`(*code_point* *u*)

Returns true if u has the *Dash* property; false otherwise

bool `is_quotation_mark`(*code_point* *u*)

Returns true if u has the *Quotation_Mark* property; false otherwise

bool `is_terminal_punctuation`(*code_point* *u*)

Returns true if u has the *Terminal_Punctuation* property; false otherwise

bool `is_sterm`(*code_point* *u*)

Returns true if u has the *STerm* property; false otherwise

bool `is_diacritic`(*code_point* *u*)

Returns true if u has the *Diacritic* property; false otherwise

bool `is_extender`(*code_point* *u*)

Returns true if u has the *Extender* property; false otherwise

bool `is_soft_dotted`(*code_point* *u*)

Returns true if u has the *Soft_Dotted* property; false otherwise

bool `is_hex_digit`(*code_point* *u*)

Returns true if u has the *Hex_Digit* property; false otherwise

bool `is_ascii_hex_digit`(*code_point* *u*)

Returns true if u has the *ASCII_Hex_Digit* property; false otherwise

bool `is_logical_order_exception`(*code_point* *u*)

Returns true if u has the *Logical_Order_Exception* property; false otherwise

bool `is_white_space`(*code_point* *u*)

Returns true if u has the *White_Space* property; false otherwise

bool `is_variation_selector`(*code_point* *u*)

Returns true if u has the *Variation_Selector* property; false otherwise

bool `is_alphabetic`(*code_point* *u*)

Returns true if u has the *Alphabetic* property; false otherwise

bool `is_math`(*code_point* *u*)

Returns true if u has the *Math* property; false otherwise

bool `is_default_ignorable_code_point`(*code_point* *u*)

Returns true if u has the *Default_Ignorable_Code_Point* property; false otherwise

bool `is_grapheme_base(`*code_point* *u*`)`

Returns true if u has the *Grapheme_Base* property; false otherwise

bool `is_grapheme_extend(`*code_point* *u*`)`

Returns true if u has the *Grapheme_Extend* property; false otherwise

grapheme_cluster_break `get_grapheme_cluster_break(`*code_point* *u*`)`

Returns the *Grapheme_Cluster_Break* property of u

word_break `get_word_break(`*code_point* *u*`)`

Returns the *Word_Break* property of u

sentence_break `get_sentence_break(`*code_point* *u*`)`

Returns the *Sentence_Break* property of u

bool `is_ideographic(`*code_point* *u*`)`

Returns true if u has the *Ideographic* property; false otherwise

bool `is_unified_ideograph(`*code_point* *u*`)`

Returns true if u has the *Unified_Ideograph* property; false otherwise

bool `is_ids_binary_operator(`*code_point* *u*`)`

Returns true if u has the *IDS_Binary_Operator* property; false otherwise

bool `is_ids_trinary_operator(`*code_point* *u*`)`

Returns true if u has the *IDS_Trinary_Operator* property; false otherwise

bool `is_radical(`*code_point* *u*`)`

Returns true if u has the *Radical* property; false otherwise

bool `is_deprecated(`*code_point* *u*`)`

Returns true if u has the *Deprecated* property; false otherwise

bool `is_noncharacter_code_point(`*code_point* *u*`)`

Returns true if u has the *Noncharacter_Code_Point* property; false otherwise

6 Glossary

code point A value in the Unicode *codespace*.

code unit The minimal bit combination that can represent a unit of encoded text for processing or interchange. The Unicode Standard uses 8-bit code units in the UTF-8 encoding form, 16-bit code units in the UTF-16 encoding form, and 32-bit code units in the UTF-32 encoding form.

codespace A range of numerical values available for encoding characters. For the Unicode Standard, this is the range of integers from 0 to $10FFFF_{16}$.

encoding form Mapping that determines how each *code point* for a Unicode character is to be expressed as a sequence of one or more *code units*.

7 Indices and tables

- genindex
- *Glossary*

Index