Unicode character properties

Document #: P1628R0
Date: 2019-06-17
Project: Programming Language C++
Audience: SG-16, LEWG
Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

1 Abstract

We propose an API to query the properties of Unicode characters as specified by the Unicode Standard and several Unicode Technical Reports.

2 Motivation

This API can be used as a foundation for various Unicode algorithms and Unicode facilities such as Unicode-aware regular expressions.

Being able to query the properties of Unicode characters is important for any application hoping to correctly handle any textual content, including compilers and parsers, text editors, graphical applications, databases, messaging applications, etc.

```
static_assert(uni::cp_script('C') == uni::script::latin);
static_assert(uni::cp_block(U'🕴') == uni::block::misc_pictographs);
static_assert(!uni::cp_is<uni::property::xid_start>('1'));
static_assert(uni::cp_is<uni::property::xid_continue>('1'));
static_assert(uni::cp_age(U'😍') == uni::version::v10_0);
static_assert(uni::cp_is<uni::property::alphabetic>(U'ß'));
static_assert(uni::cp_category(U'∩') == uni::category::sm);
static_assert(uni::cp_is<uni::category::lowercase_letter>('a'));
static_assert(uni::cp_is<uni::category::letter>('a'));
```

3 Design Consideration

3.1 constexpr

An important design decision of this proposal is that it is fully `constexpr`. Notably, the presented design allows an implementation to only link the Unicode tables that are actually used by a program. This can reduce considerably the size requirements of an Unicode-aware executable as most applications often depend on a small subset of the Unicode properties. While the complete

Unicode database has a substantial memory footprint, developers should not pay for the table they don’t use.

It also ensures that developers can enforce a specific version of the Unicode Database at compile time and get a consistent and predictable run-time behavior.

Given that, and because the API is otherwise non-allocating, there is no reason not to make it `constexpr`. We expect this to be useful in some cases notably to sanitize data at compile time, as well as make possible to use locale-independent Unicode manipulation available on memory constrained devices or devices on which ICU is not available.

The proposed design ultimately strive to fight the incorrect sentiment that Unicode is ”bloated” or hard to deploy. only the CLDR (Unicode Common Locale Data Repository) or the name/alias properties (which we are not proposing) represent a significant amount of data.

3.2 Unicode Versions, Runtime and ABI stability

The Unicode Standard is updated at the current rate of once a year. Each version might add new characters, new scripts, new blocks and change some - but not all properties of existing characters. This might in specific cases translate to an observable change of run-time properties. It might also be valuable to work with a specific Unicode version. At the same time, we want to give implementers the capability to support versions release after the C++ Standard, while making sure new Unicode versions are actually supported by the standard library.

To this effect, we propose the following:

- The C++ Standard mandates a minimum version of Unicode that needs to be supported - That minimum version will change with each new C++ standard.
- The Unicode Character Properties database API proposed here is designed such that it can be extended by implementer wishing to support newer versions **In addition to the minimum mandated version**.
- The Unicode Character Properties database API proposed here is designed such that implementers can support older versions **In addition to the minimum mandated version**.
- Functions templated on Unicode version are designed to be ABI stable.
- The default behavior is to use the version specified by the standard.

3.3 Properties selection

Not all properties are equally useful. We do therefore not propose to standardize deprecated properties, local-specific or algorithm-specific properties. We do not propose the name or alias properties which had a significant amount of data (around 500KB) and is not generally considered useful in most cases. Beyond that, given that individual properties are cheap and the proposed design does not incur any cost for not used properties, we feel reluctant to further select properties as we are not unicode expert. Strictly algorithm-specific properties could be omitted provided the algorithms themselves are provided by the standard library.

3.4 String representation of enumerated property values

While it is useful to be able to manipulate properties as strings, notably to implement Unicode aware regex engines, we do not propose such facility because, the use cases are limited and the desired values could be extracted with the currently discussed reflection proposals.

4 Implementation

This proposal is implemented in [Implementation]. Several strategies are possible, notably:

- Put the implementation in a single-header.
- Put the implementation in a single module-interface-unit.
- Use `is_constant_evaluated` to provide runtime specific implementations.

5 Proposed wording

```
namespace std::uni {

enum class version;
enum class script;
enum class block;
enum class category;
enum class property;
enum class line_break;
enum class word_break;
enum class grapheme_cluster_break;
enum class sentence_break;
enum class canonical_combining_class;


template<uni::version v = uni::version::standard_unicode_version>
struct script_extensions_view {
    constexpr script_extensions_view(uint32_t c);
    struct sentinel;
    struct iterator {
        using value_type = script;
        using iterator_category = std::forward_iterator_tag;

        constexpr iterator(char32_t c);
        constexpr script operator*();
        constexpr iterator& operator++(int);
        constexpr iterator operator++();
        constexpr bool operator==(sentinel) const;
        constexpr bool operator!=(sentinel) const;
        constexpr bool operator==(iterator) const;
        constexpr bool operator!=(iterator) const;
    };

    constexpr iterator begin() const;
    constexpr sentinel end() const;
    //exposition only
    private:
    char32_t c;
};


template<uni::version = uni::version::unassigned>
constexpr category cp_category(uint32_t) noexcept;

template<uni::category, uni::version = uni::version::unassigned>
constexpr bool cp_is(uint32_t) noexcept;

template<uni::version = uni::version::unassigned>
constexpr script cp_script(uint32_t) noexcept;

template<uni::script, uni::version = uni::version::unassigned>
constexpr bool cp_is(uint32_t) noexcept;

template<uni::version Version = uni::version::unassigne>
constexpr auto cp_script_extensions(uint32_t) noexcept
    -> script_extensions_view<Version>;

template<uni::property, uni::version = uni::version::unassigned>
constexpr bool cp_is(uint32_t) noexcept;

constexpr uni::version cp_age(uint32_t) noexcept;

constexpr uni::block cp_block(uint32_t) noexcept;

constexpr bool cp_is_valid(uint32_t) noexcept;
constexpr bool cp_is_assigned(uint32_t) noexcept;
constexpr bool cp_is_ascii(uint32_t) noexcept;

struct numeric_value {
    constexpr double value() const; //not freestanding
    constexpr int_least64_t numerator() const noexcept;
    constexpr int_least64_t denominator() const noexcept;
};

constexpr std::optional<numeric_value>
cp_numeric_value(uint32_t) noexcept;

}

enum class version : uint32_t {
    v1_1        =  0x010100,
    v2_0        =  0x020000,
    v2_1        =  0x020100,
    v3_0        =  0x030000,
    v3_1        =  0x030100,
    v3_2        =  0x030200,
    v4_0        =  0x040000,
    v4_1        =  0x040100,
    v5_0        =  0x050000,
    v5_1        =  0x050100,
    v5_2        =  0x050100,
    v6_0        =  0x060000,
    v6_1        =  0x060100,
    v6_2        =  0x060200,
    v6_3        =  0x060300,
    v7_0        =  0x070000,
    v8_0        =  0x080000,
    v9_0        =  0x090000,
    v10_0       =  0x100000,
    v11_0       =  0x110000,
    v12_0       =  0x120000,
    standard_unicode_version = v12_0,
    minimum_version = v10_0,
    latest_version = v12_0,
    unassigned  =  0xFFFFFFFF
};
```

This represents the various Unicode versions. It is used to encode the age of a code point (see `cp_age`), as well and to provide information as to which versions of Unicode are supported by the implementation.

`uni::standard_unicode_version` represent the version of Unicode that the standard mandates and shall be equal to `v12_0`. This value may change in future version of the standard.

`uni::minimum_version` represents the minimum version of Unicode that can be used as parameter to `cp_category`, `cp_is`, `cp_script` and `cp_script_extensions`. `uni::minimum_version` value is implementation defined and shall be a valid Unicode standard version that is no less that `uni::standard_unicode_version`.

`uni::latest_version` represents the greatest version of Unicode that can be used as parameter to `cp_category`, `cp_is`, `cp_script` and `cp_script_extensions`. `uni::latest_version` value is implementation defined and shall be a valid Unicode standard version that is no less that `uni::minimum_version`.

`uni::unassigned` represents the age of unassigned codepoint.

The identifiers `v1_1` to `v12_0` and their value are derived from [PropertyValueAliases.txt].

An age `<major>.<minor>` is transformed into an identifier `v<major>_<minor>` (which is the lower cased alias of the Unicode age) with a value such that the bits 16-31 are the value of the number `<major>`$_{10}$ and the bits 8-15 are the value of the number `<minor>`$_{10}$.

Given a valid `uni::version Version`, If `Version` is equal to `uni::version::unassigned` then `UNICODE_VERSION(Version)` is equal to `uni::version::latest_version`. Otherwise, `UNICODE_VERSION(Version)` is equal to `Version`.

```
enum class category;
```

The values are listed at the end of the paper.

> Represents the category of a Unicode code-point.
>
> Each enumerator is derived from [PropertyValueAliases.txt].
>
> [*Note:* An implementation may not add new values to `category`. — *end note*]

```
enum class block;
```

The values are listed at the end of the paper.

> Represents a block of Unicode codepoints.
>
> Each enumerator is derived from [PropertyValueAliases.txt], with the following transformation applied:
>
> - Each space, dash, and other non-alphanumeric character is replaced by an underscore.
> - The identifier is lower-cased.
>
> The value of each enumerator is equal to the value of the start of the block it represents.
>
> [*Note:* An implementation may add new values to `block` provided is defined in [PropertyValueAliases.txt] by applying the same transformation as described above. — *end note*]

```
enum class script;
```

> Represents the script of a Unicode code point.
>
> Each enumerator is derived from [PropertyValueAliases.txt], with the following transformation applied:
>
> - Each space, dash, and other non-alphanumeric character is replaced by an underscore.
> - The identifier is lower-cased.
>
> The value of each enumerator is equal to the value of the smallest code point with that script in the version of Unicode in which the script was first introduced.
>
> [*Note:* An implementation may add new values to `script` provided is defined in [PropertyValueAliases.txt] by applying the same transformation as described above. — *end note*]

```
enum class property;
```

Unless otherwise specified, each enumerator refers to a Boolean property specified by the Unicode Standard in [Unicode® Standard Annex #44 - UNICODE CHARACTER DATABASE]. As Unicode properties can be aliased, several enumerators may refer to the same Unicode property.

Only a subset of properties is proposed for standardization. We notably excluded properties used for derivation and deprecated or superseded properties.

`ahex` `ascii_hex_digit`	ASCII_Hex_Digit
`alpha` `alpha`	Alphabetic
`bidi_c` `bidi_control`	BiDi_Control
`bidi_m` `bidi_mirrored`	BiDi_Mirrored
`cased`	Cased
`ci` `case_ignorable`	Case_Ignorable
`dash`	Dash
`dep` `deprecated`	Deprecated
`di` `default_ignorable_code_point`	Default_Ignorable_Codepoint
`dia` `diacritic`	Diatric
`emoji`	Emoji *This property is specified in [TR#51 - Emoji]*
`emoji_component`	Emoji_Component *This property is specified in [TR#51 - Emoji]*
`emoji_modifier`	Emoji_Modifier *This property is specified in [TR#51 - Emoji]*
`emoji_modifier_base`	Emoji_Modifier_Base *This property is specified in [TR#51 - Emoji]*
`ext` `extender`	Extender
`extender_pictographic`	Extender_Pictographic
`gr_base` `grapheme_base`	Grapheme_Base
`gr_ext` `grapheme_extend`	Grapheme_Extend
`hex` `hex_digit`	Hex_Digit
`ideo` `ideographic`	Ideographic

`idsb` `ids_binary_operator`	IDS_Binary_Operator
`idst` `ids_ternary_operator`	IDS_Ternary_Operator
`join_c` `join_control`	Join_Control
`loe` `logical_order_exception`	Logical_Order_Exception
`lower` `lowercase`	Lower
`math`	Math
`nchar` `noncharacter_code_point`	Noncharacter_Code_Point
`pat_syn` `pattern_syntax`	Pattern_Syntax
`pat_ws` `pattern_white_space`	Pattern_White_Space
`pcm` `prepended_concatenation_mark`	Prepended_Concatenation_Mark
`qmark` `quotation_mark`	Quotation_Mark
`radical`	Radical
`ri` `regional_indicator`	Regional_Indicator
`sd` `soft_dotted`	Soft_Dotted
`sterm` `sentence_terminal`	Sentence_Terminal
`term` `terminal_punctuation`	Terminal_Punctuation
`uideo` `unified_ideograph`	Unified_Ideograph
`upper` `uppercase`	Upper
`vs` `variation_selector`	Variation_Selector

`wspace`
`white_space`
`space`
	Space

`xidc` `xid_continue`	XID_Continue
`xid_start` `xid_start`	XID_Start

```
template<uni::version Version = uni::version::unassigned>
constexpr uni::category cp_category(uint32_t cp) noexcept;
```

Mandates: `UNICODE_VERSION(Version) >= uni::version::minimum_version and UNICODE_VERSION(Version) <= uni::version::version`

Returns: The `category` associated with the codepoint `cp` according to the Unicode Standard `UNICODE_VERSION(Version)`, or `uni::category::unassigned` if `cp` is not a valid codepoint or was not assigned in the specified version.

```
template<uni::category Category, uni::version Version = uni::version::unassigned>
constexpr bool cp_is(uint32_t cp) noexcept;
```

Mandates: `UNICODE_VERSION(Version) >= uni::version::minimum_version and UNICODE_VERSION(Version) <= uni::version::version`

Returns: `true` if the `category` associated with the codepoint `cp` according to the `General_Category` property specified by the the Unicode Standard `UNICODE_VERSION(Version)`, is `Category` and `false` otherwise.

```
template<uni::version Version = uni::version::unassigned>
constexpr script cp_script(uint32_t cp) noexcept;
```

Mandates: `UNICODE_VERSION(Version) >= uni::version::minimum_version and UNICODE_VERSION(Version) <= uni::version::version`

Returns: The `uni::script` associated with the codepoint `cp` according to the `Script` property specified by the Unicode Standard `UNICODE_VERSION(Version)`, or `uni::script::unknown` if `cp` is not a valid codepoint or was not assigned in the specified Unicode version.

```
template<uni::script Script, uni::version Version = uni::version::unassigned>
constexpr bool cp_is(uint32_t cp) noexcept;
```

Mandates: `UNICODE_VERSION(Version) >= uni::version::minimum_version and UNICODE_VERSION(Version) <= uni::version::version`

Returns: `true` if the `script` associated with the codepoint `cp` according to the Unicode Standard `UNICODE_VERSION(Version)`, is `Script` and `false` otherwise.

```
template<uni::version Version = uni::version::unassigned>
constexpr auto cp_script_extensions(uint32_t cp) noexcept
-> script_extensions_view<Version>;
```

Mandates: `UNICODE_VERSION(Version) >= uni::version::minimum_version and UNICODE_VERSION(Version) <= uni::version::version`

Returns: A `script_extensions_view<Version>` that can iterate over all the script extensions of `cp` according to the `Script_Extension` property specified by Unicode Standard `UNICODE_VERSION(Version)`.

If `cp` is not a valid codepoint or was not assigned in the specified Unicode version, the return object shall represent an empty range.

[*Note:* In accordance with to the Unicode standard, if a codepoint has no explicit script extension, the return object is a range with a single element denoting the `script` as returned by `cp_script`. — *end note*]

[*Note:* The order of elements is unspecified — *end note*]

```
template<uni::property Prop, uni::version Version = uni::version::unassigned>
constexpr bool cp_is(uint32_t cp) noexcept;
```

Mandates: `UNICODE_VERSION(Version) >= uni::version::minimum_version and UNICODE_VERSION(Version) <= uni::version::version`

Returns: `true` if the codepoint `cp` is valid, assigned in the specified version and has the boolean property associated with `Prop` according to the Unicode Standard `UNICODE_VERSION(Version)`, and `false` otherwise.

```
constexpr uni::version cp_age(uint32_t cp) noexcept;
```

Returns: The `version` corresponding to the `Age` property of this code point or `version::unassigned` if `cp` is not a valid assigned codepoint.

```
constexpr uni::version cp_block(uint32_t cp) noexcept;
```

Returns: The `block` corresponding to the `Block` property of this code point or `block::no_block` if `cp` is not a valid codepoint.

[*Note:* Unassigned codepoints may still have a block value different from `block::no_-block`. — *end note*]

```
constexpr bool cp_is_valid(uint32_t cp) noexcept;
```

Returns: `true` if the codepoint `cp` is a valid Unicode codepoint.

```
template<uni::version Version = uni::version::unassigned>
constexpr bool cp_is_assigned(uint32_t cp) noexcept;
```

Returns: `true` if the codepoint `cp` is assigned in `UNICODE_VERSION(Version)`.

[*Note:* This function is equivalent to `cp_age(cp) <= UNICODE_VERSION(Version)` — *end note*]

```
constexpr bool cp_is_ascii(uint32_t) noexcept;
```

Returns: `true` if `cp <= 0x7F`

```
constexpr std::optional<numeric_value>
cp_numeric_value(uint32_t cp) noexcept;
```

Returns: if `cp` is not a valid assigned codepoint or has a `Numeric_Type` property with the value `None`, a default constructed optional. Otherwise, returns a `numeric_value` such that:

- if `cp` has a ”`Digit`” `Numeric_Type` : `numerator` represents the value of `Numeric_-Value` converted to a `int64_t` in base 10 and `denominator` has the value `1`.

- if `cp` has a "`Numeric`" `Numeric_Type` and `Numeric_Value` does not contain a forward slash: `numerator` represents the value of `Numeric_Value` converted to a `int64_t` in base 10 and `denominator` has the value `1`.
- if `cp` has a "`Numeric`" `Numeric_Type` and `Numeric_Value` contains a forward slash: `numerator` represents the value of the `Numeric_Value` string up to but not including the forward slash converted to a `int64_t` in base 10 and `denominator` represents the value of the `Numeric_Value` string starting from the character following the forward slash converted to a `int64_t` in base 10.

6 Acknowledgments

Special thanks to Hana Dusíková for her valuable feedbacks while integrating this library to CTRE faster than I could implement it and the Men and Women IN BUSINESS SUIT LEVITATING of SG-16.

7 Values of Enumerations

```
enum class category {
    c,
    other = c,
    cc,
    control = cc,
    cf,
    format = cf,
    cn,
    unassigned = cn,
    co,
    private_use = co,
    cs,
    surrogate = cs,
    l,
    letter = l,
    lc,
    cased_letter = lc,
    ll,
    lowercase_letter = ll,
    lm,
    modifier_letter = lm,
    lo,
    other_letter = lo,
    lt,
    titlecase_letter = lt,
    lu,
    uppercase_letter = lu,
    m,
    mark = m,
    mc,
    spacing_mark = mc,
    me,
    enclosing_mark = me,
    mn,
    nonspacing_mark = mn,
    n,
    number = n,
    nd,
    decimal_number = nd,
    nl,
    letter_number = nl,
    no,
    other_number = no,
    p,
    punctuation = p,
    pc,
    connector_punctuation = pc,
    pd,
    dash_punctuation = pd,
    pe,
    close_punctuation = pe,
    pf,
    final_punctuation = pf,
    pi,
    initial_punctuation = pi,
    po,
    other_punctuation = po,
    ps,
    open_punctuation = ps,
    s,
    symbol = s,
    sc,
    currency_symbol = sc,
    sk,
    modifier_symbol = sk,
    sm,
    math_symbol = sm,
    so,
    other_symbol = so,
    z,
    separator = z,
    zl,
    line_separator = zl,
    zp,
    paragraph_separator = zp,
    zs,
    space_separator = zs
};
```

```
enum class block {
    no_block,
    nb = no_block,
    basic_latin,
    ascii = basic_latin,
    latin_1_supplement,
    latin_1_sup = latin_1_supplement,
    latin_extended_a,
    latin_ext_a = latin_extended_a,
    latin_extended_b,
    latin_ext_b = latin_extended_b,
    ipa_extensions,
    ipa_ext = ipa_extensions,
    spacing_modifier_letters,
    modifier_letters = spacing_modifier_letters,
    combining_diacritical_marks,
    diacriticals = combining_diacritical_marks,
    greek_and_coptic,
    greek = greek_and_coptic,
    cyrillic,
    cyrillic_supplement,
    cyrillic_sup = cyrillic_supplement,
    armenian,
    hebrew,
    arabic,
    syriac,
    arabic_supplement,
    arabic_sup = arabic_supplement,
    thaana,
    nko,
    samaritan,
    mandaic,
    syriac_supplement,
    syriac_sup = syriac_supplement,
    arabic_extended_a,
    arabic_ext_a = arabic_extended_a,
    devanagari,
    bengali,
    gurmukhi,
    gujarati,
    oriya,
    tamil,
    telugu,
    kannada,
    malayalam,
    sinhala,
    thai,
    lao,
    tibetan,
    myanmar,
    georgian,
    hangul_jamo,
    jamo = hangul_jamo,
    ethiopic,
    ethiopic_supplement,
    ethiopic_sup = ethiopic_supplement,
    cherokee,
    unified_canadian_aboriginal_syllabics,
    ucas = unified_canadian_aboriginal_syllabics,
    ogham,
    runic,
    tagalog,
    hanunoo,
    buhid,
    tagbanwa,
    khmer,
    mongolian,
    unified_canadian_aboriginal_syllabics_extended,
    ucas_ext = unified_canadian_aboriginal_syllabics_extended,
    limbu,
    tai_le,
    new_tai_lue,
    khmer_symbols,
    buginese,
    tai_tham,
    combining_diacritical_marks_extended,
    diacriticals_ext = combining_diacritical_marks_extended,
    balinese,
    sundanese,
    batak,
    lepcha,
    ol_chiki,
    cyrillic_extended_c,
    cyrillic_ext_c = cyrillic_extended_c,
    georgian_extended,
    georgian_ext = georgian_extended,
    sundanese_supplement,
    sundanese_sup = sundanese_supplement,
    vedic_extensions,
    vedic_ext = vedic_extensions,
    phonetic_extensions,
    phonetic_ext = phonetic_extensions,
    phonetic_extensions_supplement,
    phonetic_ext_sup = phonetic_extensions_supplement,
    combining_diacritical_marks_supplement,
    diacriticals_sup = combining_diacritical_marks_supplement,
    latin_extended_additional,
    latin_ext_additional = latin_extended_additional,
    greek_extended,
    greek_ext = greek_extended,
    general_punctuation,
    punctuation = general_punctuation,
    superscripts_and_subscripts,
    super_and_sub = superscripts_and_subscripts,
    currency_symbols,
    combining_diacritical_marks_for_symbols,
    diacriticals_for_symbols = combining_diacritical_marks_for_symbols,
    letterlike_symbols,
    number_forms,
    arrows,
    mathematical_operators,
    math_operators = mathematical_operators,
    miscellaneous_technical,
    misc_technical = miscellaneous_technical,
    control_pictures,
    optical_character_recognition,
    ocr = optical_character_recognition,
    enclosed_alphanumerics,
    enclosed_alphanum = enclosed_alphanumerics,
    box_drawing,
    block_elements,
    geometric_shapes,
    miscellaneous_symbols,
    misc_symbols = miscellaneous_symbols,
    dingbats,
    miscellaneous_mathematical_symbols_a,
    misc_math_symbols_a = miscellaneous_mathematical_symbols_a,
    supplemental_arrows_a,
    sup_arrows_a = supplemental_arrows_a,
    braille_patterns,
    braille = braille_patterns,
    supplemental_arrows_b,
    sup_arrows_b = supplemental_arrows_b,
    miscellaneous_mathematical_symbols_b,
    misc_math_symbols_b = miscellaneous_mathematical_symbols_b,
    supplemental_mathematical_operators,
    sup_math_operators = supplemental_mathematical_operators,
    miscellaneous_symbols_and_arrows,
    misc_arrows = miscellaneous_symbols_and_arrows,
    glagolitic,
    latin_extended_c,
    latin_ext_c = latin_extended_c,
    coptic,
    georgian_supplement,
    georgian_sup = georgian_supplement,
    tifinagh,
    ethiopic_extended,
    ethiopic_ext = ethiopic_extended,
    cyrillic_extended_a,
    cyrillic_ext_a = cyrillic_extended_a,
    supplemental_punctuation,
    sup_punctuation = supplemental_punctuation,
    cjk_radicals_supplement,
    cjk_radicals_sup = cjk_radicals_supplement,
    kangxi_radicals,
    kangxi = kangxi_radicals,
    ideographic_description_characters,
    idc = ideographic_description_characters,
    cjk_symbols_and_punctuation,
    cjk_symbols = cjk_symbols_and_punctuation,
    hiragana,
    katakana,
    bopomofo,
    hangul_compatibility_jamo,
    compat_jamo = hangul_compatibility_jamo,
    kanbun,
    bopomofo_extended,
    bopomofo_ext = bopomofo_extended,
    cjk_strokes,
    katakana_phonetic_extensions,
    katakana_ext = katakana_phonetic_extensions,
    enclosed_cjk_letters_and_months,
    enclosed_cjk = enclosed_cjk_letters_and_months,
    cjk_compatibility,
    cjk_compat = cjk_compatibility,
    cjk_unified_ideographs_extension_a,
    cjk_ext_a = cjk_unified_ideographs_extension_a,
    yijing_hexagram_symbols,
    yijing = yijing_hexagram_symbols,
    cjk_unified_ideographs,
    cjk = cjk_unified_ideographs,
    yi_syllables,
    yi_radicals,
    lisu,
    vai,
    cyrillic_extended_b,
    cyrillic_ext_b = cyrillic_extended_b,
    bamum,
    modifier_tone_letters,
    latin_extended_d,
    latin_ext_d = latin_extended_d,
    syloti_nagri,
    common_indic_number_forms,
    indic_number_forms = common_indic_number_forms,
    phags_pa,
    saurashtra,
    devanagari_extended,
    devanagari_ext = devanagari_extended,
    kayah_li,
    rejang,
    hangul_jamo_extended_a,
    jamo_ext_a = hangul_jamo_extended_a,
    javanese,
    myanmar_extended_b,
    myanmar_ext_b = myanmar_extended_b,
    cham,
    myanmar_extended_a,
    myanmar_ext_a = myanmar_extended_a,
    tai_viet,
    meetei_mayek_extensions,
    meetei_mayek_ext = meetei_mayek_extensions,
    ethiopic_extended_a,
    ethiopic_ext_a = ethiopic_extended_a,
    latin_extended_e,
    latin_ext_e = latin_extended_e,
    cherokee_supplement,
    cherokee_sup = cherokee_supplement,
    meetei_mayek,
    hangul_syllables,
    hangul = hangul_syllables,
    hangul_jamo_extended_b,
    jamo_ext_b = hangul_jamo_extended_b,
    high_surrogates,
    high_private_use_surrogates,
    high_pu_surrogates = high_private_use_surrogates,
    low_surrogates,
    private_use_area,
    pua = private_use_area,
    cjk_compatibility_ideographs,
    cjk_compat_ideographs = cjk_compatibility_ideographs,
    alphabetic_presentation_forms,
    alphabetic_pf = alphabetic_presentation_forms,
    arabic_presentation_forms_a,
    arabic_pf_a = arabic_presentation_forms_a,
    variation_selectors,
    vs = variation_selectors,
    vertical_forms,
    combining_half_marks,
    half_marks = combining_half_marks,
    cjk_compatibility_forms,
    cjk_compat_forms = cjk_compatibility_forms,
    small_form_variants,
    small_forms = small_form_variants,
    arabic_presentation_forms_b,
    arabic_pf_b = arabic_presentation_forms_b,
    halfwidth_and_fullwidth_forms,
    half_and_full_forms = halfwidth_and_fullwidth_forms,
    specials,
    linear_b_syllabary,
    linear_b_ideograms,
    aegean_numbers,
    ancient_greek_numbers,
    ancient_symbols,
    phaistos_disc,
    phaistos = phaistos_disc,
    lycian,
    carian,
    coptic_epact_numbers,
    old_italic,
    gothic,
    old_permic,
    ugaritic,
    old_persian,
    deseret,
    shavian,
    osmanya,
    osage,
    elbasan,
    caucasian_albanian,
    linear_a,
    cypriot_syllabary,
    imperial_aramaic,
    palmyrene,
    nabataean,
    hatran,
    phoenician,
    lydian,
    meroitic_hieroglyphs,
    meroitic_cursive,
    kharoshthi,
    old_south_arabian,
    old_north_arabian,
    manichaean,
    avestan,
    inscriptional_parthian,
    inscriptional_pahlavi,
    psalter_pahlavi,
    old_turkic,
    old_hungarian,
    hanifi_rohingya,
    rumi_numeral_symbols,
    rumi = rumi_numeral_symbols,
    old_sogdian,
    sogdian,
    elymaic,
    brahmi,
    kaithi,
    sora_sompeng,
    chakma,
    mahajani,
    sharada,
    sinhala_archaic_numbers,
    khojki,
    multani,
    khudawadi,
    grantha,
    newa,
    tirhuta,
    siddham,
    modi,
    mongolian_supplement,
    mongolian_sup = mongolian_supplement,
    takri,
    ahom,
    dogra,
    warang_citi,
    nandinagari,
    zanabazar_square,
    soyombo,
    pau_cin_hau,
    bhaiksuki,
    marchen,
    masaram_gondi,
    gunjala_gondi,
    makasar,
    tamil_supplement,
    tamil_sup = tamil_supplement,
    cuneiform,
    cuneiform_numbers_and_punctuation,
    cuneiform_numbers = cuneiform_numbers_and_punctuation,
    early_dynastic_cuneiform,
    egyptian_hieroglyphs,
    egyptian_hieroglyph_format_controls,
    anatolian_hieroglyphs,
    bamum_supplement,
    bamum_sup = bamum_supplement,
    mro,
    bassa_vah,
    pahawh_hmong,
    medefaidrin,
    miao,
    ideographic_symbols_and_punctuation,
    ideographic_symbols = ideographic_symbols_and_punctuation,
    tangut,
    tangut_components,
    kana_supplement,
    kana_sup = kana_supplement,
    kana_extended_a,
    kana_ext_a = kana_extended_a,
    small_kana_extension,
    small_kana_ext = small_kana_extension,
    nushu,
    duployan,
    shorthand_format_controls,
    byzantine_musical_symbols,
    byzantine_music = byzantine_musical_symbols,
    musical_symbols,
    music = musical_symbols,
    ancient_greek_musical_notation,
    ancient_greek_music = ancient_greek_musical_notation,
    mayan_numerals,
    tai_xuan_jing_symbols,
    tai_xuan_jing = tai_xuan_jing_symbols,
    counting_rod_numerals,
    counting_rod = counting_rod_numerals,
    mathematical_alphanumeric_symbols,
    math_alphanum = mathematical_alphanumeric_symbols,
    sutton_signwriting,
    glagolitic_supplement,
    glagolitic_sup = glagolitic_supplement,
    nyiakeng_puachue_hmong,
    wancho,
    mende_kikakui,
    adlam,
    indic_siyaq_numbers,
    ottoman_siyaq_numbers,
    arabic_mathematical_alphabetic_symbols,
    arabic_math = arabic_mathematical_alphabetic_symbols,
    mahjong_tiles,
    mahjong = mahjong_tiles,
    domino_tiles,
    domino = domino_tiles,
    playing_cards,
    enclosed_alphanumeric_supplement,
    enclosed_alphanum_sup = enclosed_alphanumeric_supplement,
    enclosed_ideographic_supplement,
    enclosed_ideographic_sup = enclosed_ideographic_supplement,
    miscellaneous_symbols_and_pictographs,
    misc_pictographs = miscellaneous_symbols_and_pictographs,
    emoticons,
    ornamental_dingbats,
    transport_and_map_symbols,
    transport_and_map = transport_and_map_symbols,
    alchemical_symbols,
    alchemical = alchemical_symbols,
    geometric_shapes_extended,
    geometric_shapes_ext = geometric_shapes_extended,
    supplemental_arrows_c,
    sup_arrows_c = supplemental_arrows_c,
    supplemental_symbols_and_pictographs,
    sup_symbols_and_pictographs = supplemental_symbols_and_pictographs,
    chess_symbols,
    symbols_and_pictographs_extended_a,
    symbols_and_pictographs_ext_a = symbols_and_pictographs_extended_a,
    cjk_unified_ideographs_extension_b,
    cjk_ext_b = cjk_unified_ideographs_extension_b,
    cjk_unified_ideographs_extension_c,
    cjk_ext_c = cjk_unified_ideographs_extension_c,
    cjk_unified_ideographs_extension_d,
    cjk_ext_d = cjk_unified_ideographs_extension_d,
    cjk_unified_ideographs_extension_e,
    cjk_ext_e = cjk_unified_ideographs_extension_e,
    cjk_unified_ideographs_extension_f,
    cjk_ext_f = cjk_unified_ideographs_extension_f,
    cjk_compatibility_ideographs_supplement,
    cjk_compat_ideographs_sup = cjk_compatibility_ideographs_supplement,
    tags,
    variation_selectors_supplement,
    vs_sup = variation_selectors_supplement,
    supplementary_private_use_area_a,
    sup_pua_a = supplementary_private_use_area_a,
    supplementary_private_use_area_b,
    sup_pua_b = supplementary_private_use_area_b,
    __max
};

enum class script {
    adlm,
    adlam = adlm,
    aghb,
    caucasian_albanian = aghb,
    ahom,
    arab,
    arabic = arab,
    armi,
    imperial_aramaic = armi,
    armn,
    armenian = armn,
    avst,
    avestan = avst,
    bali,
    balinese = bali,
    bamu,
    bamum = bamu,
    bass,
    bassa_vah = bass,
    batk,
    batak = batk,
    beng,
    bengali = beng,
    bhks,
    bhaiksuki = bhks,
    bopo,
    bopomofo = bopo,
    brah,
    brahmi = brah,
    brai,
    braille = brai,
    bugi,
    buginese = bugi,
    buhd,
    buhid = buhd,
    cakm,
    chakma = cakm,
    cans,
    canadian_aboriginal = cans,
    cari,
    carian = cari,
    cham,
    cher,
    cherokee = cher,
    copt,
    coptic = copt,
    cprt,
    cypriot = cprt,
    cyrl,
    cyrillic = cyrl,
    deva,
    devanagari = deva,
    dogr,
    dogra = dogr,
    dsrt,
    deseret = dsrt,
    dupl,
    duployan = dupl,
    egyp,
    egyptian_hieroglyphs = egyp,
    elba,
    elbasan = elba,
    elym,
    elymaic = elym,
    ethi,
    ethiopic = ethi,
    geor,
    georgian = geor,
    glag,
    glagolitic = glag,
    gong,
    gunjala_gondi = gong,
    gonm,
    masaram_gondi = gonm,
    goth,
    gothic = goth,
    gran,
    grantha = gran,
    grek,
    greek = grek,
    gujr,
    gujarati = gujr,
    guru,
    gurmukhi = guru,
    hang,
    hangul = hang,
    hani,
    han = hani,
    hano,
    hanunoo = hano,
    hatr,
    hatran = hatr,
    hebr,
    hebrew = hebr,
    hira,
    hiragana = hira,
    hluw,
    anatolian_hieroglyphs = hluw,
    hmng,
    pahawh_hmong = hmng,
    hmnp,
    nyiakeng_puachue_hmong = hmnp,
    hrkt,
    katakana_or_hiragana = hrkt,
    hung,
    old_hungarian = hung,
    ital,
    old_italic = ital,
    java,
    javanese = java,
    kali,
    kayah_li = kali,
    kana,
    katakana = kana,
    khar,
    kharoshthi = khar,
    khmr,
    khmer = khmr,
    khoj,
    khojki = khoj,
    knda,
    kannada = knda,
    kthi,
    kaithi = kthi,
    lana,
    tai_tham = lana,
    laoo,
    lao = laoo,
    latn,
    latin = latn,
    lepc,
    lepcha = lepc,
    limb,
    limbu = limb,
    lina,
    linear_a = lina,
    linb,
    linear_b = linb,
    lisu,
    lyci,
    lycian = lyci,
    lydi,
    lydian = lydi,
    mahj,
    mahajani = mahj,
    maka,
    makasar = maka,
    mand,
    mandaic = mand,
    mani,
    manichaean = mani,
    marc,
    marchen = marc,
    medf,
    medefaidrin = medf,
    mend,
    mende_kikakui = mend,
    merc,
    meroitic_cursive = merc,
    mero,
    meroitic_hieroglyphs = mero,
    mlym,
    malayalam = mlym,
    modi,
    mong,
    mongolian = mong,
    mroo,
    mro = mroo,
    mtei,
    meetei_mayek = mtei,
    mult,
    multani = mult,
    mymr,
    myanmar = mymr,
    nand,
    nandinagari = nand,
    narb,
    old_north_arabian = narb,
    nbat,
    nabataean = nbat,
    newa,
    nkoo,
    nko = nkoo,
    nshu,
    nushu = nshu,
    ogam,
    ogham = ogam,
    olck,
    ol_chiki = olck,
    orkh,
    old_turkic = orkh,
    orya,
    oriya = orya,
    osge,
    osage = osge,
    osma,
    osmanya = osma,
    palm,
    palmyrene = palm,
    pauc,
    pau_cin_hau = pauc,
    perm,
    old_permic = perm,
    phag,
    phags_pa = phag,
    phli,
    inscriptional_pahlavi = phli,
    phlp,
    psalter_pahlavi = phlp,
    phnx,
    phoenician = phnx,
    plrd,
    miao = plrd,
    prti,
    inscriptional_parthian = prti,
    rjng,
    rejang = rjng,
    rohg,
    hanifi_rohingya = rohg,
    runr,
    runic = runr,
    samr,
    samaritan = samr,
    sarb,
    old_south_arabian = sarb,
    saur,
    saurashtra = saur,
    sgnw,
    signwriting = sgnw,
    shaw,
    shavian = shaw,
    shrd,
    sharada = shrd,
    sidd,
    siddham = sidd,
    sind,
    khudawadi = sind,
    sinh,
    sinhala = sinh,
    sogd,
    sogdian = sogd,
    sogo,
    old_sogdian = sogo,
    sora,
    sora_sompeng = sora,
    soyo,
    soyombo = soyo,
    sund,
    sundanese = sund,
    sylo,
    syloti_nagri = sylo,
    syrc,
    syriac = syrc,
    tagb,
    tagbanwa = tagb,
    takr,
    takri = takr,
    tale,
    tai_le = tale,
    talu,
    new_tai_lue = talu,
    taml,
    tamil = taml,
    tang,
    tangut = tang,
    tavt,
    tai_viet = tavt,
    telu,
    telugu = telu,
    tfng,
    tifinagh = tfng,
    tglg,
    tagalog = tglg,
    thaa,
    thaana = thaa,
    thai,
    tibt,
    tibetan = tibt,
    tirh,
    tirhuta = tirh,
    ugar,
    ugaritic = ugar,
    vaii,
    vai = vaii,
    wara,
    warang_citi = wara,
    wcho,
    wancho = wcho,
    xpeo,
    old_persian = xpeo,
    xsux,
    cuneiform = xsux,
    yiii,
    yi = yiii,
    zanb,
    zanabazar_square = zanb,
    zinh,
    inherited = zinh,
    zyyy,
    common = zyyy,
    zzzz,
    unknown = zzzz
};
```

8 References

[DerivedAge.txt] Unicode® *Unicode Character Database: Derived Property Data - Age Property*
http://www.unicode.org/Public/12.0.0/ucd/DerivedAge.txt

[PropertyValueAliases.txt] Unicode® *Unicode Character Database: Derived Property Data - Age Property*
https://www.unicode.org/Public/12.0.0/ucd/PropertyValueAliases.txt

[TR#51 - Emoji] Unicode® *Technical Standard #51 - UNICODE EMOJI*
http://unicode.org/reports/tr51/

[Unicode® Standard Annex #44 - UNICODE CHARACTER DATABASE] Unicode® *[Unicode® Standard Annex #44 - UNICODE CHARACTER DATABASE*
http://unicode.org/reports/tr44/

[CTRE] Hana Dusíková Unicode Compliant version of CTRE based on this proposal https://github.com/hanickadot/compile-time-regular-expressions/tree/ecma-unicode

[Implementation] Experimental implementation of this proposal https://github.com/cor3ntin/ext-unicode-db/