ICU 79 API proposal status

Published on the web
This doc is public for viewing & comments

- New items should be added to the top (as proposals are sent), not the bottom; the most-recently-sent proposal should be at the top, the oldest at the bottom.
- New items should be Heading 1. Approved/withdrawn items become Heading 2.
- As we modify API proposals, this document should be updated, and changes also sent to the icu-design list.
- Once approved, the approved API signatures must be here, so that we can compare it later with the actual implementation.

Contents: use menu View > Show document outline

API Proposal Email Template

[proposed:2026-07-xx] <proposal subject>

xx sent the following to icu-design on julXX

[notice:2026-07-03] ICU API notice: MF2 allow function overrides

Steven sent the following to icu-design on jul03

Dear ICU team & users,

This is new API for: **ICU 79**

Ticket: https://unicode-org.atlassian.net/browse/ICU-23424
PR: https://github.com/unicode-org/icu/pull/4041

As part of improving MessageFormat2, this is a semantic change which allows a user-specified function to override a standard function (such as `date`). There are no API signature changes, however, the apidoc has been updated to reflect this update.

This is C++ only, at this point (and tech preview), but we will work soon on harmonizing the C and J APIs and implementations.

messageformat2_function_registry.h:

class MFFunctionRegistry::Builder

```
/**
 * Registers a function to a given name.
 *
 * @param functionName Name of the formatter being registered.
 * @param function A pointer to a Function object.
 *                 This argument is adopted.
 *                 Functions with the same name as standard functions will
 *                 override the standard functions.
 * @param errorCode Input/output error code
 * @return A reference to the builder.
 *
 * @internal ICU 79 technology preview
 * @deprecated This API is for technology preview only.
 */
Builder& adoptFunction(const data_model::FunctionName& functionName,
                       Function* function,
                       UErrorCode& errorCode);
```

messageformat2.h

class MessageFormat::Builder

```
/**
 * Sets a custom function registry.
 *
 * @param functionRegistry Reference to the function registry to use.
 *        `functionRegistry` is not copied,
 *        and the caller must ensure its lifetime contains
 *        the lifetime of the `MessageFormatter` object built by this
 *        builder.
```

```
     *        Functions with the same name as standard functions will
     *        override the standard functions.
     * @return       A reference to the builder.
     *
     * @internal ICU 75 technology preview
     * @deprecated This API is for technology preview only.
     */
    U_I18N_API Builder& setFunctionRegistry(const MFFunctionRegistry& functionRegistry);
```

Thank you,
Steven

[notice:2026-06-26] ICU API notice: new property values for Unicode 18

Markus sent the following to icu-design on jul02

Update: Like in 2025, the UTC is on track to remove the Chisoi script from Unicode 18, in order to stay in sync with ISO 10646.

As a result, we will remove the API constants from ICU 79 for the Chisoi script and block.

Sincerely,
markus

Markus sent the following to icu-design on jun26

Dear ICU team & users,

This is new API for: **ICU 79**

Ticket: **ICU-23343**

Almost every Unicode release defines some new property values. ICU adds corresponding API constants which are "born @stable" (they are not @draft first for a while) so that they are immediately usable without problems.

See https://blog.unicode.org/2026/05/unicode-180-beta-review-opens-for.html and https://www.unicode.org/review/pri548/ for Unicode 18 beta.

We are adding the following property value constants:

C/C++

unicode/uchar.h

```
enum UBlockCode {
    // New blocks in Unicode 18.0.0

    /** @stable ICU 79 */
    UBLOCK_ARCHAIC_CUNEIFORM_NUMERALS = 347, /*[12550]*/
    /** @stable ICU 79 */
    UBLOCK_BENGALI_SUPPLEMENT = 348, /*[11DF0]*/
    /** @stable ICU 79 */
    UBLOCK_CHISOI = 349, /*[16D80]*/
    /** @stable ICU 79 */
    UBLOCK_JURCHEN = 350, /*[18E00]*/
    /** @stable ICU 79 */
    UBLOCK_JURCHEN_RADICALS = 351, /*[191A0]*/
    /** @stable ICU 79 */
    UBLOCK_MISCELLANEOUS_SYMBOLS_AND_ARROWS_EXTENDED = 352, /*[1DB00]*/
    /** @stable ICU 79 */
    UBLOCK_MUSICAL_SYMBOLS_SUPPLEMENT = 353, /*[1D250]*/
    /** @stable ICU 79 */
    UBLOCK_SEAL = 354, /*[3D000]*/

typedef enum UJoiningGroup {
    U_JG_CROWN_AIN,  /**< @stable ICU 79 */
    U_JG_CROWN_BEH,  /**< @stable ICU 79 */
    U_JG_CROWN_FEH,  /**< @stable ICU 79 */
    U_JG_CROWN_HAH,  /**< @stable ICU 79 */
    U_JG_CROWN_HEH,  /**< @stable ICU 79 */
    U_JG_CROWN_KAF,  /**< @stable ICU 79 */
    U_JG_CROWN_MEEM,  /**< @stable ICU 79 */
    U_JG_CROWN_SAD,  /**< @stable ICU 79 */
    U_JG_CROWN_SEEN,  /**< @stable ICU 79 */
    U_JG_CROWN_TAH,  /**< @stable ICU 79 */
```

unicode/uscript.h

```
typedef enum UScriptCode {
    /** @stable ICU 79 */
    USCRIPT_CHISOI                  = 213, /* Chis */
    /** @stable ICU 79 */
    USCRIPT_PROTO_CUNEIFORM         = 214, /* Pcun */
    /** @stable ICU 79 */
    USCRIPT_SEAL                    = 215, /* Seal */
```

Java

```
public final class UCharacter {
   public static final class UnicodeBlock extends Character.Subset {
        // New blocks in Unicode 18.0.0
        public static final int ARCHAIC_CUNEIFORM_NUMERALS_ID = 347; /*[12550]*/
        public static final int BENGALI_SUPPLEMENT_ID = 348; /*[11DF0]*/
        public static final int CHISOI_ID = 349; /*[16D80]*/
        public static final int JURCHEN_ID = 350; /*[18E00]*/
        public static final int JURCHEN_RADICALS_ID = 351; /*[191A0]*/
        public static final int MISCELLANEOUS_SYMBOLS_AND_ARROWS_EXTENDED_ID = 352; /*[1DB00]*/
        public static final int MUSICAL_SYMBOLS_SUPPLEMENT_ID = 353; /*[1D250]*/
        public static final int SEAL_ID = 354; /*[3D000]*/

     ...

        // New blocks in Unicode 18.0.0
        public static final UnicodeBlock ARCHAIC_CUNEIFORM_NUMERALS =
        public static final UnicodeBlock BENGALI_SUPPLEMENT =
        public static final UnicodeBlock CHISOI =
        public static final UnicodeBlock JURCHEN =
        public static final UnicodeBlock JURCHEN_RADICALS =
        public static final UnicodeBlock MISCELLANEOUS_SYMBOLS_AND_ARROWS_EXTENDED =
        public static final UnicodeBlock MUSICAL_SYMBOLS_SUPPLEMENT =
        public static final UnicodeBlock SEAL =

   public static interface JoiningGroup
        public static final int CROWN_AIN = 106;
        public static final int CROWN_BEH = 107;
        public static final int CROWN_FEH = 108;
        public static final int CROWN_HAH = 109;
```

```
        public static final int CROWN_HEH = 110;
        public static final int CROWN_KAF = 111;
        public static final int CROWN_MEEM = 112;
        public static final int CROWN_SAD = 113;
        public static final int CROWN_SEEN = 114;
        public static final int CROWN_TAH = 115;
```

public final class **UScript** {

```
    public static final int CHISOI = 213; /* Chis */
    public static final int PROTO_CUNEIFORM = 214; /* Pcun */
    public static final int SEAL = 215; /* Seal */
```

public final class **VersionInfo** {

```
    /**
     * Unicode 18.0 version
     *
     * @stable ICU 79
     */
    public static final VersionInfo UNICODE_18_0;
```

[proposed:2026-06-24] MF2 function composition

Discussion jul02

- getStandardFunction() should return a **const** Function *
    - Functions are mutable, cannot be const in order to be usable.
    - However, Functions being mutable seems troubling. Needs to be revisited: ICU-23447
- Proposal competes with setAlternateLocale()? need both? harmless?

Steven Loomis sent the following to icu-design on jun24

subject: ICU API proposal: **MF2 Function Composition**

Dear ICU team & users,

I would like to propose the following C++ Tech Preview APIs for: **ICU 79**

Please provide feedback by: **next Wednesday, 2024-07-01**

Designated API reviewer: **Mihai ⟪U⟫ Nițǎ**

Ticket: https://unicode-org.atlassian.net/browse/ICU-23424
PR: https://github.com/unicode-org/icu/pull/4054

These functions are needed for being able to compose functions, that is, to create new functions that are based on the existing, standard functions. This example shows changing the locale in the context. Another example could be modifying or transforming the output of one function (such as a script transliteration).

```
class FunctionContext {

    /**
     * Returns the original called function name from this context.
     *
     * @return The function name, as registered, used for this function
     *
     * @internal ICU 79 technology preview
     * @deprecated This API is for technology preview only.
     */
    U_I18N_API const FunctionName& getCalledFunctionName() const;

    /**
     * Returns a new context with the specified locale.
     *
     * @return a new locale
     *
     * @internal ICU 79 technology preview
     * @deprecated This API is for technology preview only.
     */
    U_I18N_API FunctionContext withLocale(const Locale& loc) const;

    /**
     * Returns a pointer to a standard function.
     * This function may only be used within the implementation of call()
```

```
     * The returned pointer is invalid once this FunctionContext goes out of scope.
     * A U_INVALID_PARAMETER is set if the requested function is not one of
     * the standard functions.
     *
     * @param name the name of the standard function, such as "datetime" or "number"
     * @return pointer to the standard function, or nullptr
     *
     * @internal ICU 79 technology preview
     * @deprecated This API is for technology preview only.
     */
    U_I18N_API Function *getStandardFunction(const FunctionName &name,
                                             UErrorCode &errorCode);
```

Example Use:

In this example, the function given below is registered as `_date` and `_number`
The `call()` Implementation retrieves the name, removes the underscore, and then delegates to the standard function (such as `date` or `number`).

If overriding the standard functions are allowed, then the underscore would not be needed.

A user of these functions could opt to set the locale via a function argument or option, that is beyond the scope of this example.

```
LocalPointer<FunctionValue> LocaleOverrideFunction::call(const FunctionContext &context,
                                                         const FunctionValue &arg,
                                                         const FunctionOptions &opts,
                                                         UErrorCode &errorCode) {
   const FunctionName& myFunction = context.getCalledFunctionName();
   const FunctionName& stdFunction = myFunction.tempSubString(1); // _date -> date
   Function *delegateFunction =
       context.getStandardFunction(stdFunction, errorCode);
   if (U_FAILURE(errorCode)) {
       return LocalPointer<FunctionValue>();
   }
   // create a new context, with our updated locale
```

```
    FunctionContext myContext = context.withLocale(overrideLocale);
    // call the delegated function
    return delegateFunction->call(myContext, arg, opts, errorCode);
}
```

[proposed:2026-06-18] MF2::Builder::setFormattingLocale

Steven Loomis sent the following to icu-design on jun17

Please provide feedback by: next Wednesday, 2026-06-24
Designated API reviewer: Markus
Ticket: https://unicode-org.atlassian.net/browse/ICU-23424

Alternate Proposal 7.b (below): FunctionContext alternate locales

Discussion jul02

- Where is this builder?
  - MessageFormatter builder
- What is the key string?
  - Allows for multiple alternate locales
  - Might be better as an enum rather than a free-form string (Mihai wanted this open)
  - Use case / needed?
  - If we don’t need the key, then back to a more specific name for the getter & setter?
- This proposal depends on the built-in date function to look at the alternate locale.
  - Rich is uncomfortable with that. Prefers to allow overrides, and use them for this.
  - Seemed ok to Markus & Matt; makes API much easier to use, too
- Markus: At this point, I think I prefer the earlier set/getFormattingLocale(), no key
- Yoshito: In Java, should we look at the JDK formatting locale category?
  - Markus: Don’t want to require users to have to modify global state to achieve an outcome
  - MessageFormatter should default its formatting to its main locale, not to the JDK formatting locale

- On the MessageFormatter builder: a setter that sets an alternate locale

```
class Builder {
    /**
     * @param key An alternate key for this locale.
     * @param locale The desired locale
     * @return       A reference to the builder.
     *
     * @internal ICU 79 technology preview
     * @deprecated This API is for technology preview only.
     */
    U_I18N_API Builder& setAlternateLocale(const UnicodeString& key, const Locale& locale);
```

- On FunctionContext: a getter for the alternate locale

```
    /**
     * Accesses the formatting locale that this `MessageFormatter` object
     * was created with.  If no formatting locale was explicitly set,
     * returns the same locale as getLocale().
     *
     * @param key the identifier matching that passed to Builder::setAlternateLocale
     * @return A reference to the locale, or, the normal locale (the same as getLocale()) if this 'key' was not
registered with Builder::setAlternateLocale(key, …)
     *
     * @internal ICU 79 technology preview
     * @deprecated This API is for technology preview only.
     */
    U_I18N_API const Locale& getAlternateLocale(const UnicodeString& key, UErrorCode &errorCode);
```

PR: https://github.com/unicode-org/icu/pull/4021
Jira: ICU-23424

This proposes a new setter to the MF2 Builder, `setFormattingLocale()` .
The use case is a user of MF2 where the pluralization should be based on the specified locale, however, all formatting (date/time/numbers) should be according to a *fixed* locale, such as `en`.

The user here is an implementation layer on top of ICU4C, so the messages and inputs are coming in from third parties.

More Information

We had originally discussed in the CLDR MF2WG <message-format-wg/issues/1107> the idea of overriding existing MF2 functions (such as `date` or `select`) to have the desired behavior. There was no consensus at MF2WG for this approach.

The “current locale” per spec is part of the “Formatting Context” (TR35), which states:

> *Implementations MAY include additional fields in their formatting context.*

This API proposal is spec-compliant and adds field(s) to the formatting context.

Alternatives

- A0: Instead of (or in addition to) adding `setFormattingLocale()`, add `setSelectorLocale()` which will only be used for plural selection. This can have the advantage of being more specific. Again, if not called, `setLocale()` would be used.
- A1: **Allow function overrides**, then a client can override standard functions at will.

  ```
  ¡Ahora es {$today :datetime}!
  ```

  -> *¡Ahora es Jul 1, 2024 at 7:00 AM!*

  Bring this back to MF2WG, to clarify the spec.

  See note below.
- A2: From Mihai: mutate the message to instead use `{$foo :custom_date ...}` for all formats (instead of `:date` etc.)

  ~~Note that ICU4C does not expose the MF2 data model, so clients can’t do this kind of mutation just using ICU APIs. They would need an independent MF2 implementation that is capable of this mutation.~~
- A3: Mutate the message so that instead of `.input {$count :number} .match $count one {{…}}`

  The message is changed by the user to `.input {$count_PLURAL :string} .match $count_PLURAL one {{…}}`

  And then the string input named `count_PLURAL` is *injected* by the user with the pluralized bucket in the desired locale (one, other, many, etc.)

  This also requires mutation, and adding a new input.

jun18 discussion notes:

1. Translate into some language, but format dates/times in English
2. Why not just format externally and pass in strings?
3. Desire for stable formatting
   a. Selecting English does not provide that; e.g., recent change of space before AM/PM to thin space
   b. Application already using ICU with English, so don’t need absolute stability
4. Robin: YouTube had this general use case backward for short date formatting (never en-US short dates, even in en-US, too many non-US en-US users)
5. Alternative: Main locale for formatting, new setter for plural locale – but it would be weird for the main locale to not match the language of the main message text
6. Markus to check for consistency of where we have getters for build inputs
7. Mihai: Architecturally, the top-level MessageFormatter does not know what the placeholder functions do. How should it know which formatting function should get which locale?
8. Mihai: Instead, could write a custom function
9. Mihai: Could customize locale data to do English formatting in all locales
10. Mihai: MF2 WG considered setting a locale in each placeholder, but rejected the idea

11. Mihai: Want to see a design doc with requirements and different approaches
12. Tried to set a custom function to override a default function, but ICU4C ignored it; Steven: the spec forbids overriding default functions, but allows adding data attributes
13. Mark: Overriding default functions seems reasonable
14. Mihai: I don't think the spec forbids that; if TC agrees, then we could allow it
15. Steven will bring the override discussion back to the MF2 WG
16. Mark: With the current spec, this is implementation-defined, and ICU could just decide to do it

Markus jun18: Consistency of getters

- Java MessageFormatter has getLocale() etc.
- Java MessageFormatter.Builder has only setters
- C++ MessageFormatter has getLocale() etc.
- C++ MessageFormatter::Builder has only setters
- ✔️

jun25 discussion notes:

1. George: Would prefer setFormattingLocale(), fits well with Apple systems
2. Mihai: Architecturally, this is a problem, see discussion last week – the MessageFormatter does not know (should not know) which formatter should be given which locale
3. Mihai: In ICU4C/J, we could consider a locale parameter in the MF2 syntax, on a function. ICU4X does not like it (bad for static data slicing), but in C/J we are less constrained.
4. Steven: C++ MF2 has a "formatting context". Add another locale there. – Mihai: Java does not have such a "context".
5. Mihai: Instead of setting a "formatting locale", set a locale for a specific function. setFunctionLocale("datetime", Locale("en")) – resolves the architectural issue
6. Mihai: Why not have users customize their locale data. – Markus: Then what they do with MF2 constrains everything else they can do.
7. Things to consider
    a. Ability to override built-in functions
        i. Pro: Adds flexibility; easy; localized to a MessageFormatter object
        ii. Con: Can't tell from the message what it does
        iii. Worth pursuing
        iv. Steven R. Loomis to send api proposal for api doc change
    b. Ability to pass multiple locales / formatting context
        i. Pro: easy, each function can pick;
        ii. Mihai: Make the formatting context public, with typed key-value pairs or record-like that we can expand over time
            1. Pro: Don't have to have both public setters and internal context fields
        iii. Steven: The context is a spec concept, and the builder provides it to the formatter
        iv. General agreement that a second locale in the context would be ok
    c. Ability to set per-function locales
        i. setFunctionLocale("datetime", Locale("en"))

ii. May be ok, but a formatting locale in the context seems better

d. Try to move forward with (a) & (b)

API proposal

For C++ for 79

- MessageFormat2 :: Builder :: setFormattingLocale()

```
/**
 * Sets the locale used by formatter factories (:number, :date,
 * etc.) for rendering output. Selector factories (plural rules)
 * continue to use the locale set via setLocale().
 * The formatting locale defaults to the same one set by setLocale().
 * @param locale The desired locale
 * @return       A reference to the builder.
 *
 * @internal ICU 79 technology preview
 * @deprecated This API is for technology preview only.
 */
U_I18N_API Builder& setFormattingLocale(const Locale& locale);
```

- MessageFormat2 :: getFormattingLocale()

```
/**
 * Acceses the formatting locale that this `MessageFormatter` object
 * was created with.  If no formatting locale was explicitly set,
 * returns the same locale as getLocale().
 *
 * @return A reference to the formatting locale.
 *
 * @internal ICU 79 technology preview
 * @deprecated This API is for technology preview only.
 */
U_I18N_API const Locale& getFormattingLocale() { return formattingLocale; }
```

MF2 Function Override

- See experimental PR at https://github.com/unicode-org/icu/pull/4038
    - Allowing override is straightforward, actually doing the override not so muhc.

From the PR, I identified these API gaps: (See proposal *earlier in this document*)

1. ~~Problem: was not able to call the built-in functions.~~
    a. ~~Solution: Rather than expose them directly, it would be sufficient to have some kind of `const MFFunctionRegistry& getDefaultFunctionRegistry()` where I could call `getFunction("date")`... etc~~
2. ~~Problem: was not able to mutate or construct the `FunctionContext`.~~
    a. ~~Solution: Add a builder for it, or make its constructor public. Perhaps it could be only public to Functions.?~~

[approved:2026-06-25] link boundary detection & link formatting

Discussion jun25

- changes and approval see below
- We also discussed: "consider taking & returning a CharSequence"
    - Mark: I could go either way. Unlike for scanning, it is unlikely that the input wouldn't be a String. For scanning, you want to allow for implementations that have their own string structure.
    - Markus: did you check in your updated implementation whether CharSequence would be overly cumbersome?
    - Mark: I looked into it, and is doable but fairly cumbersome. Probably best to leave this as Strings. As I said, it's a different situation than scanning.

jun18 version

Mark Davis sent the following to icu-design on jun18

This includes changes after feedback:

- escapePathQueryFragment() made static
- renamed from LinkUtilities2 (note suffix “2”) to LinkUtilities – typo fix
- removed getSafeCharacters() & getDomainCharacters()
- TC approved jun25 with changes

```
/**
* Utility class for assisting with detecting links (URLs or emails) in text, and formatting them
* for display, implementing the algorithms in https://www.unicode.org/reports/tr58/ to handle
* Unicode characters properly. It supplies lower level APIs for use in augmenting existing scanners
* and formatters.
*/
public class LinkUtilities {
  /**
   * Lower level utility for finding the end of a PathQueryFragment (PQF) in text. It assumes that
```

```
     * the start position is immediately after an identified domain name. The purpose of this
     * routine is for fitting into algorithms that are already in use, just taking over for the PQF
     * scanning. For more information, see https://www.unicode.org/reports/tr58/.
     *
     * @param source the text to be scanned
     * @param start the position in the text to be scanned from. It should be immediately after a
     *     domain name.
     * @param limit the offset after the last character in the text to be scanned.
     * @return the end position of the PQF, or the start value if there is none.
     */
    public static int scanPathQueryFragment(CharSequence source, int start, int limit) {
…
    }
    /**
     * Lower level utility for finding the start of an email address in text. It assumes that the
     * limit position is immediately before an '@' + identified domain name. The purpose of this
     * routine is for fitting into algorithms that are already in use, just handling for the email
     * `local-part`.
     *
     * @param source the text to be scanned
     * @param start the position that is the earliest that should be considered in a backwards scan
     * @return the position to start scanning backwards from — should be just before @ +
     *     domain_name.
     */
    public static int scanBackEmailLocalPart(CharSequence s, int start, int limit) {
…
    }
    /** Enum for determining whether any percent-escaping is minimal or maximal, for use
     * in escapePathQueryFragment.
     */
    public enum Extent {
        /** Minimal percent-escaping only percent-escapes non-ASCII where necessary. */
        MINIMAL,
        /** Maximal percent-escaping percent-escapes all non-ASCII. */
        MAXIMAL
    }
    /**
     * Escapes a URL according to the Extent parameter.
     *
```

```
   * @param source In the source, it is assumed that ASCII syntax characters requiring escaping
   *     have already been escaped. For example, a literal / in a path segment would already be
   *     percent-escaped. For more information, see https://www.unicode.org/reports/tr58/.
   * @param extent either MINIMAL or MAXIMAL
   * @return an escaped string according to the extent parameter.
   */
  public static String escapePathQueryFragment(String source, Extent extent) {
…
  }

  // NOTE for reviewers: These are only temporary, until ICU supplies the following:
  // \p{Link_Term=hard}
  // \p{Idn_Status≠disallowed}
  /**
   * Returns a frozen set of Unicode characters that are guaranteed to never be part of a URL or
   * email address. This allows implementations to make various optimizations because URLs and
   * email addresses can never span these characters. For example, a span of characters between
   * safe characters that doesn't have a sequence of domain character + . + domain character can
   * be skipped in processing.
   */
  public static UnicodeSet getSafeCharacters() {
…
  }
  /**
   * Returns a frozen set of Unicode characters that are possible characters in a domain name
   * (pre-mapping) This allows implementations to make various optimizations because URLs and
   * email addresses must contain a sequence of domain character + . + domain character. It is the
   * same as the set of IDNA Mapping Table character with values ≠ disallowed
   */
  public static UnicodeSet getDomainCharacters() {
…
  }
}
```

jun10 version

Markus jun18

First, sorry for not getting to this earlier.

Second, sorry, but to me it doesn't feel appropriate for ICU to try to implement the whole end-to-end link detection.

Other people / library implementers have spent years figuring out how to detect links in text. UTS #58 covers the piece of the problem where those other libraries are bad or inconsistent.

Therefore, scanPathQueryFragment() feels right -- that's the value that we can reasonably provide -- but the rest should be out of scope for ICU.
And that could be a static method. Especially in Java where lazy init is easy.

I also don't want to be in the business of maintaining a set of TLDs. It's yet one more thing that would complicate a release, but also, link detectors should be free to ignore whether TLDs are valid, and also free to validate below-top-level domain labels, etc.

Even considering the TextHandler callback, that feels like ignoring where the language has gone -- a modern Java iteration API would return a Stream.

Plus a constructor that may well sprout more arguments later... would probably want to have a Builder.

On Wed, Jun 17, 2026 at 6:44 PM Mark Davis Ⓤ <mark@unicode.org> wrote:
> One question for Markus is whether his update to Unicode 18.0 properties includes the [UTS #58] properties

Not yet, but I can try to work on that for ICU 79. Worst case, ICU 80.

Not sure where getDomainCharacters() should live. Is it derived from the IdnaMappingTable.txt type values? We could consider a regular property API for that.

For the UrlEscaper:

Is it likely that enum Extent will ever have more than two values?

If not, then we could just have two separate functions.

Why not make the input a CharSequence like elsewhere?

Discussion jun18

- Markus: see above
- Mark: Also missing is a backward scan for email user names – another value-add from UTS #58: https://www.unicode.org/reports/tr58/#email-algorithm
  - Agreed to add
  - static int **scanBackEmailLocalPart**(CharSequence s, int start, int limit) – scans back from limit where s.charAt(limit) == '@' logically
- scanPathQueryFragment(): drop limit arg, assuming that CharSequence.subSequence() is cheap
  - Mark: limit is useful for parallel processing ~~stopping at LinkTerm=Hard~~
  - Mihai: String.subSequence() makes a contents-copy of the sub-sequence
  - Agreed: keep the limit arg
- escapeUrl()
  - similar, reduce to static escapePathQueryFragment()
  - consider taking & returning a CharSequence
- Static functions on one class
  - LinkUtilities?
- Agreed in principle
- Mark to send an updated proposal

Mark Davis sent the following to icu-design on jun10 (updated since)

```
/**
* Class for detecting links (URLs or emails) in text, and formatting them for display, implementing
* the algorithms in https://www.unicode.org/reports/tr58/ to handle Unicode characters properly. It
* supplies not only higher-level APIs that work directly, but also lower level APIs that can be
* used to augment existing scanners or formatters.
*
* <p>The high-level API handles text with and without schemes (mailto:, https://, etc.). It does a
* fast scan for valid TLDs, then backs up for other components (full host, email user-part,
* scheme). For ambiguous cases, the first valid case is chosen. For example,
* "john.smith@foo.com/somepath?somequery" would detect "john.smith@foo.com", and treat
* "/somepath?somequery" as plain text.
*/
```

```java
public class LinkScanner {
  /**
   * An interface for handling text being scanned. For example, in the text<br>
   * "At abc.net/foo#def can you find joe@somemail.com?", the following handler methods will be
   * called in sequence:
   *
   * <ul>
   *   <li>handlePlainText 0,3 — eg "At "
   *   <li>handleUrl 3,18 — eg "abc.net/foo#def"
   *   <li>handlePlainText 18,32 — eg " can you find "
   *   <li>handleMail 32,48 — eg "joe@somemail.com"
   *   <li>handlePlainText 48,49 — eg "?"
   * </ul>
   */
  public interface TextHandler {
    /**
     * Process a segment of plain text.
     *
     * @param plainText A string containing the plain text to process
     * @param start The start index
     * @param end The end offset (exclusive)
     */
    default void handlePlainText(CharSequence plainText, int start, int end) {}
    /**
     * Process a detected URL.
     *
     * @param urlText A string containing the URL to process. This exactly matches the source
     *     string: no normalization is performed.
     * @param start The start index
     * @param end The end offset (exclusive)
     */
    default void handleUrl(CharSequence urlText, int start, int end) {}
    /**
     * Process a detected email address.
     *
     * @param emailAddressText A string containing the email address to process. This exactly
     *     matches the source string: no normalization is performed.
     * @param start The start index
     * @param end The end offset (exclusive)
     */
```

```
    default void handleMailAddress(CharSequence emailAddressText, int start, int end) {}
}

/**
 * Constructor
 *
 * @param validTLDs list of valid TLDs. If the list is empty or null, then a default list is
 *    used. That list is derived from https://data.iana.org/TLD/tlds-alpha-by-domain.txt just
 *    before the last release of ICU, so for best results a more current list can be used.
 */
public LinkScanner(Set<String> validTLDs) {
        ...
}
/**
 * Processes the given input text using the given handler to process the chunks of text
 * detected. This method is not thread-safe.
 *
 * @param input The plain text to process.
 * @param start where to start the scanning from.
 * @param end where to stop the scanning.
 * @param handler The object which formats the detected chunks of text.
 */
public void process(CharSequence input, int start, int end, TextHandler handler) {
        ...
}
// NOTE to Reviewers. The use of a TextHandler allows the client to find the segments and do any further processing it wants.
// For example, it might copy the plaintext, and copy (but inside of links) the URL or email address text.
// It could also do further processing, such as deciding that the text shouldn't be linked (eg, because something is invalid or confusable),
// or changing the text (eg, for confusables) before adding links.
 /**
  * Lower level utility for finding the end of a PathQueryFragment (PQF) in text. It assumes that
  * the start position is immediately after an identified domain name. The purpose of this
  * routine is for fitting into algorithms that are already in use, just taking over for the PQF
  * scanning.
  *
  * @param source the text to be scanned
  * @param start the position in the text to be scanned from. It should be immediately after a
  *    domain name.
  * @return the end position of the PQF, or the start value if there is none.
  */
```

```
  public static int scanPathQueryFragment(CharSequence source, int start, int limit) {
            ...
  }
  // NOTE for reviewers: if ICU supplies \p{Link_Term=hard}, the following is not necessary
  /**
   * Returns a frozen set of Unicode characters that are guaranteed to never be part of a URL or
   * email address. This allows implementations to make various optimizations because URLs and
   * email addresses can never span these characters. For example, a span of characters between
   * safe characters that doesn't have a sequence of domain-character + . + domain-character can
   * be skipped in processing.
   */
  public static UnicodeSet getSafeCharacters() {
            ...
  }
  // NOTE for reviewers: if ICU supplies \p{Idn_Status≠disallowed}, the following is not necessary
  /**
   * Returns a frozen set of Unicode characters that are possible characters in a domain name
   * (pre-mapping) This allows implementations to make various optimizations because URLs and
   * email addresses must contain a sequence of domain-character + . + domain-character. It is the
   * same as the set of IDNA Mapping Table character with values ≠ disallowed
   */
  public static UnicodeSet getDomainCharacters() {
            ...
  }
}

/**
* Provides for either minimal or maximal escaping of URLs, implementing the algorithms in
* https://www.unicode.org/reports/tr58/.
*/
public class UrlEscaper {
  /** Enum for determining whether any percent-escaping is minimal or maximal */
  public enum Extent {
    /** Minimal percent-escaping only percent-escapes non-ASCII where necessary. */
    MINIMAL,
    /** Maximal percent-escaping percent-escapes all non-ASCII. */
    MAXIMAL
  }
```

```
  /**
   * Escapes a URL according to the Extent parameter
   * @param source In the source, it is assumed that ASCII syntax characters requiring escaping have already been escaped.
   * For example, a literal / in a path segment would already be percent-escaped. [TBD give example]
   * @param extent either MINIMAL or MAXIMAL
   * @return an escaped string according to the extent parameter.
   */
  public String escapeUrl(String source, Extent extent) {
...
  }
}
// NOTE to Reviewers. It would be possible to provide a “higher level” escaper that would take structured data (a division by Parts)
// that would not require the ASCII syntax characters to be quoted. However, it may be more onerous to convert whatever structure the
// implementation has into that structure than it is for the implementation to just quote the ASCII syntax characters itself.
```

[partially approved:2026-06-04] ICU4C/ICU4J API proposal: Add dayOfMonthName support to DateFormatSymbols

- StandardPlural is not public API, public API has so far worked with char * plural category strings
- Type names: Don’t guarantee always 33
- Think about “type name” as a name
- C++ enum DtContextType etc. – we have been trying to consolidate on C enums for sharing with the C API
- DateFormatSymbols memory: Returning a const pointer to an array of UnicodeString requires permanent storage of that array.
    - Rich & Dragan working on DFS change so that normally it only loads the data that its DateFormat needs
    - Ok to keep going with this kind of API for now
    - We expect a relatively small number of locales to have relevant data
    - Request to add UErrorCode to both getters and setters
- Alternative: Keep the symbols private, only support “ddd” in DateFormat
    - We already don’t expose all symbols involved in formatting
- **Agreed**:
    - yes to “ddd” patterns
    - keep the symbols private

Rich sent the following to icu-design on jun03

I would like to propose the following API for: **ICU 79**
Please provide feedback by: **next Wednesday, 2026-jun-10**
Designated API reviewer: **Markus**
Ticket: https://unicode-org.atlassian.net/browse/ICU-23364
Pull request: https://github.com/unicode-org/icu/pull/4010

https://unicode-org.atlassian.net/browse/CLDR-7408 added new dayOfMonths resources to CLDR, along with a spec change (that may not have landed yet) to treat “ddd” in a date formatting pattern as asking to use the dayOfMonths resource for the date’s day field instead of using a number.

The formatting API doesn’t change, other than the change in meaning of “ddd” in a pattern or skeleton, but we also need to add new API to DateFormatSymbols to access the new day-of-month names. In C++, we would add the following to the DateFormatSymbols definition in i18n/unicode/dtfmtsym.h::

```
/**
 * Gets day-of-month ordinal name format patterns by width and context.
 * The returned array is indexed by StandardPlural::Form (see standardplural.h);
 * its length is always StandardPlural::COUNT.  An empty entry means no data for that
 * plural category.  Currently only FORMAT context and ABBREVIATED width are
 * supported; other combinations return nullptr.
 * @param context  The formatting context.  Currently only FORMAT is supported.
 * @param width    The width.  Currently only ABBREVIATED is supported.
 * @return the day-of-month ordinal name strings, or nullptr if not available.
 *         (DateFormatSymbols retains ownership.)
 * @draft ICU 79
 */
U_I18N_API const UnicodeString* getDayOfMonthOrdinalNames(DtContextType context,
                                                          DtWidthType width) const;
/**
 * Sets day-of-month ordinal name format patterns by width and context.
 * Currently only FORMAT context and ABBREVIATED width are supported;
 * other combinations are silently ignored.
 * @param names    The new strings, indexed by StandardPlural::Form.
 *                 (not adopted; caller retains ownership)
 * @param count    The length of the array.
 * @param context  The formatting context.  Currently only FORMAT is supported.
 * @param width    The width.  Currently only ABBREVIATED is supported.
 * @draft ICU 79
 */
```

```
    U_I18N_API void setDayOfMonthOrdinalNames(const UnicodeString* names,
                                              int32_t count,
                                              DtContextType context,
                                              DtWidthType width);
    /**
     * Gets day-of-month type-specific name strings by width and context.
     * The returned array has length 33 and is indexed by day-of-month (1-32);
     * index 0 is unused.  An empty entry means no type-specific override for
     * that day number.  Currently only FORMAT context and ABBREVIATED width are
     * supported; other combinations return nullptr.
     * @param count    Filled in with the length of the array (33).
     * @param context  The formatting context.  Currently only FORMAT is supported.
     * @param width    The width.  Currently only ABBREVIATED is supported.
     * @return the day-of-month type-specific strings, or nullptr if not available.
     *         (DateFormatSymbols retains ownership.)
     * @draft ICU 79
     */
    U_I18N_API const UnicodeString* getDayOfMonthTypeNames(int32_t& count,
                                                           DtContextType context,
                                                           DtWidthType width) const;
    /**
     * Sets day-of-month type-specific name strings by width and context.
     * Currently only FORMAT context and ABBREVIATED width are supported;
     * other combinations are silently ignored.
     * @param names    The new strings (length 33, indexed by day-of-month 1-32;
     *                 use an empty UnicodeString for days without a specific override).
     *                 (not adopted; caller retains ownership)
     * @param count    The length of the array.
     * @param context  The formatting context.  Currently only FORMAT is supported.
     * @param width    The width.  Currently only ABBREVIATED is supported.
     * @draft ICU 79
     */
    U_I18N_API void setDayOfMonthTypeNames(const UnicodeString* names,
                                           int32_t count,
                                           DtContextType context,
                                           DtWidthType width);
```

In C, we add two new enum constants to UDateFormatSymbolType:

```
    /**
     * Day-of-month ordinal name patterns (e.g. "{0}st", "{0}nd"),
     * indexed by StandardPlural::Form (0 = ZERO ... 7 = EQ_1).
     * @draft ICU 79
     */
    UDAT_DAY_OF_MONTH_ORDINAL_NAMES,
    /**
     * Day-of-month type name strings, indexed by day number
     * (entries 1-32; entry 0 is unused/empty).
     * @draft ICU 79
     */
    UDAT_DAY_OF_MONTH_TYPE_NAMES,
```

In Java, we add the same new methods to DateFormatSymbols:

```
/**
* Gets day-of-month ordinal name format patterns by width and context.
* The returned array is indexed by {@link com.ibm.icu.impl.StandardPlural} ordinal;
* its length is {@link com.ibm.icu.impl.StandardPlural#COUNT}. An empty entry means
* no data for that plural category. Currently only FORMAT context and ABBREVIATED
* width are supported; other combinations return null.
*
* @param context The formatting context. Currently only FORMAT is supported.
* @param width   The width. Currently only ABBREVIATED is supported.
* @return the day-of-month ordinal name strings, or null if not available.
* @draft ICU 79
*/
public String[] getDayOfMonthOrdinalNames(int context, int width) {
}

/**
* Sets day-of-month ordinal name format patterns by width and context.
* Currently only FORMAT context and ABBREVIATED width are supported;
* other combinations are silently ignored.
*
* @param names   The new strings, indexed by {@link com.ibm.icu.impl.StandardPlural} ordinal.
* @param context The formatting context. Currently only FORMAT is supported.
* @param width   The width. Currently only ABBREVIATED is supported.
* @draft ICU 79
*/
```

```
public void setDayOfMonthOrdinalNames(String[] names, int context, int width) {
}

/**
* Gets day-of-month type-specific name strings by width and context.
* The returned array has length 33 and is indexed by day-of-month (1-32);
* index 0 is unused. An empty entry means no type-specific override for that
* day number. Currently only FORMAT context and ABBREVIATED width are
* supported; other combinations return null.
*
* @param context The formatting context. Currently only FORMAT is supported.
* @param width   The width. Currently only ABBREVIATED is supported.
* @return the day-of-month type-specific strings, or null if not available.
* @draft ICU 79
*/
public String[] getDayOfMonthTypeNames(int context, int width) {
}

/**
* Sets day-of-month type-specific name strings by width and context.
* Currently only FORMAT context and ABBREVIATED width are supported;
* other combinations are silently ignored.
*
* @param names   The new strings (length 33, indexed by day-of-month 1-32;
*                use null or empty for days without a specific override).
* @param context The formatting context. Currently only FORMAT is supported.
* @param width   The width. Currently only ABBREVIATED is supported.
* @draft ICU 79
*/
public void setDayOfMonthTypeNames(String[] names, int context, int width) {
}
```

[approved:2026-06-04] ICU4J API proposal: deprecate ICUUncheckedIOException

Note: ICU-TC agreed to do this on 2026-04-02.

Markus sent the following to icu-design on jun03

I would like to propose the following API for: **ICU 79**

Please provide feedback by: **next Wednesday, 2026-jun-10**
Designated API reviewer: **Mihai**
Ticket: https://unicode-org.atlassian.net/browse/**ICU-22148**
Pull request: https://github.com/unicode-org/**icu/pull/4016**

In Java, checked exceptions are inconvenient. For example, widening APIs that take StringBuilder to working with Appendable requires either declaring that an IOException may be thrown, or wrapping one into an unchecked exception.

In ICU 53, we created ICUUncheckedIOException for this, many years before ICU4J could rely on Java 8 which introduced a similar UncheckedIOException. At this point, our own class is redundant.

I propose that we

- deprecate ICUUncheckedIOException
- keep throwing it from code that already does so
- change it into a subclass of UncheckedIOException

Callers can then choose to catch the standard exception, and new ICU code can throw that.

[approved:2026-06-04] ICU4C API proposal: code point iterators & ranges: add base(), front(), back()

Markus sent the following to icu-design on jun02

I would like to propose the following API for: **ICU 79**
Please provide feedback by: **next Wednesday, 2026-jun-10**
Designated API reviewer: **Robin**
Ticket: https://unicode-org.atlassian.net/browse/**ICU-23421**
Pull request: https://github.com/unicode-org/**icu/pull/4014**

This is a small addition to the C++ code point iterator APIs. These are actually common functions in C++ standard iterators and ranges that we had overlooked.

From the ticket:
C++ “ranges” or “containers” usually have a front() function for the first element, and a back() function for the last one. This has turned out to be something that would be a useful addition to [Unsafe]UTFStringCodePoints. Simple example:

- https://en.cppreference.com/cpp/string/basic_string_view
- https://en.cppreference.com/cpp/string/basic_string_view/front
- https://en.cppreference.com/cpp/string/basic_string_view/back

A C++ iterator often wraps another, “more basic” iterator, and commonly provides a base() function to access that. [Unsafe]UTFIterator wraps a code unit iterator; base() would return a code unit iterator at the current logical position. It would be like fetching the current CodeUnits and calling their begin(), except that those cannot be fetched from an exclusive-end() iterator; base() would be much easier to use.

Real example: use std::advance() to move a code point iterator forward n code points, then call its base() to find the position in the input string/string_view, which will usually have moved by a larger number of code units.

- std::ranges::advance - cppreference.com

Similarly, a reverse_iterator usually has a base() function to return an original-direction iterator at the same logical position. The reverse_iterator wrappers over [Unsafe]UTFIterator should have that, too.

- https://en.cppreference.com/cpp/iterator/reverse_iterator/base

New API signatures in common/unicode/utfiterator.h. Note that the reverse_iterator classes are template specializations of C++ reverse_iterator and do not have the usual API docs.

```
class UTFIterator {
    /**
     * Returns the current position as a code unit iterator.
     * Similar to iter->begin() but also works at the exclusive end().
     *
     * @return current position as a code unit iterator
     * @draft ICU 79
     */
    U_FORCE_INLINE UnitIter base() const

class std::reverse_iterator<...UTFIterator...> {
    U_FORCE_INLINE U_HEADER_ONLY_NAMESPACE::UTFIterator<CP32, behavior, UnitIter> base() const

class UTFStringCodePoints {
    /**
     * Returns the CodeUnits for the first character/code point.
     * Requires that the range is not empty.
     *
     * @return the CodeUnits for the first character/code point.
     * @draft ICU 79
```

```
     */
    auto front() const {
        return *begin();
    }

    /**
     * Returns the CodeUnits for the last character/code point.
     * Requires that the range is not empty.
     *
     * @return the CodeUnits for the last character/code point.
     * @draft ICU 79
     */
    auto back() const {
        return *(--end());
    }
```

Same additions in the “unsafe” classes, API docs omitted here:

```
class UnsafeUTFIterator {
    U_FORCE_INLINE UnitIter base() const

class std::reverse_iterator<...UnsafeUTFIterator...> {
    U_FORCE_INLINE U_HEADER_ONLY_NAMESPACE::UnsafeUTFIterator<CP32, UnitIter> base() const

class UnsafeUTFStringCodePoints {
    auto front() const
    auto back() const
```

[proposed:2026-05-15] API proposal: C++ API for Segmenter

... more complete proposal ...

ICU-TC 2026-05-21: agreed to use modern C++ types

Elango sent the following to icu-design on may15.

Hi everyone,

As I begin to take a look at the C++ implementation of the Segmenter API, a few important high level design decision points have already come up. So I want to get your opinions and carry that into the discussion in our next TC meeting.

I've added the C++ specific API design and questions in the **C++ Design Details tab** of the design doc. Specifically, I've added the following high level questions:

1. Input string type
    - UTF-16: `std::u16string_view` or `UnicodeString` ?
    - UTF-8: `std::string_view` or `StringPiece` ?
2. Return type: Classic ICU or modern C++?
    - Classic ICU: return a pointer, and by convention the caller takes ownership.
    - Modern C++: Return a smart pointer -> clearly indicated ownership
3. If modern C++ return type: which style?
    - ICU style: return `LocalPointer<Segments>`
    - Standard C++ style: `std::unique_ptr<Segments>`

Here is an example of one way the answers to the above questions might look like for `segmenter.h`:

```
class U_COMMON_API_CLASS Segmenter : public UObject {

public:

   ~Segmenter() override;

   virtual std::unique_ptr<Segments> segment(std::u16string_view s, UErrorCode &errorCode);

   virtual std::unique_ptr<SegmentsUTF8> segment(StringPiece s, UErrorCode &errorCode);
```

```
};
```

Your answers to the questions will help inform the rest of the design & impl details of the API, and could be useful for how we approach future API design.

Thanks,

Elango

[approved:2026-05-14] changing MessageFormat.format args from Map<String,Object> to Map<String,?>

Mihai sent the following to icu-design on may13.
Vlad Ciorica from Android confirms that this change is binary compatible.
Approved: Option C: Directly modify the existing methods

Dear icu-design@unicode.org,

The ticket is ICU-23339 and there was already some discussion on it.

But I have also created and shared a design document, summarizing the options and the proposal, with pros and cons:
Changing the MessageFormat.format arguments type

Please review when you have some time, if interested.

[approved:2026-01-29] Adding 'lifetimebound' annotations to ICU4C

Fredrik sent the following to icu-design on jan15

Dear ICU Team & Users,

There's a pretty new C++ feature called *lifetimebound*, currently supported by newer versions of the Clang and MSVC compilers (and possibly others):

https://clang.llvm.org/docs/AttributeReference.html#lifetimebound
https://learn.microsoft.com/en-us/cpp/code-quality/c26816?view=msvc-170

While it doesn't turn C++ into Rust quite yet, it makes it possible for the compiler to emit warnings or errors for a bunch of easily made memory bugs that are otherwise hard to detect.

I therefore propose adding a new ICU4C macro for the feature:

```
#ifndef __cplusplus
// Not for C.
#elif defined(U_LIFETIME_BOUND)
// Use the predefined value.
#elif  UPRV_HAS_CPP_ATTRIBUTE(clang::lifetimebound)
#   define U_LIFETIME_BOUND [[clang::lifetimebound]]
#elif UPRV_HAS_CPP_ATTRIBUTE(msvc::lifetimebound)
#   define U_LIFETIME_BOUND [[msvc::lifetimebound]]
#elif UPRV_HAS_ATTRIBUTE(lifetimebound)
#   define U_LIFETIME_BOUND __attribute__((lifetimebound))
#endif
```

With that in place, I'd then proceed to annotate ICU4C functions like this:

```
U_COMMON_API const char* getName() const U_LIFETIME_BOUND;

static const Locale& getLocale(
    const Locale& valid U_LIFETIME_BOUND,
    const Locale& actual U_LIFETIME_BOUND,
    ULocDataLocaleType type, UErrorCode& status);
```

That will make it possible for the compiler to detect the memory bugs in code like this:

```
const char* p = Locale("hi").getName();

const Locale& l = LocaleBased::getLocale(Locale("aa"), Locale("zu"),
                                         ULOC_VALID_LOCALE, status);
```

Such code would then result in compiler errors like these:

```
error: temporary whose address is used as value of local variable 'p' will be destroyed at the end of the
full-expression [-Werror,-Wdangling]
     const char* p = Locale("hi").getName();
                     ^~~~~~~~~~~~

error: temporary bound to local reference 'l' will be destroyed at the end of the full-expression [-Werror,-Wdangling]
     const Locale& l = LocaleBased::getLocale(Locale("aa"), Locale("zu"),
                                              ^~~~~~~~~~~~
error: temporary bound to local reference 'l' will be destroyed at the end of the full-expression [-Werror,-Wdangling]
     const Locale& l = LocaleBased::getLocale(Locale("aa"), Locale("zu"),
                                                            ^~~~~~~~~~~~
```

So when all the above is in place, I furthermore propose adding -Wdangling to the default compiler options.

[approved:2026-01-08] more UnicodeSet parsing changes

Robin sent the following to icu-design on dec30

Dear ICU team & users,

Following up on the changes approved by the TC on 2025-12-11, and continuing the effort to rigorously specify UnicodeSet syntax and have our implementation conform to that specification, I would like to propose the following changes to UnicodeSet behaviour
for: **ICU 79**
Please provide feedback by: **Wednesday, 2026-01-07**
Designated API reviewer: **Markus**

Some changes here are breaking changes, although I think they are in such weird undocumented corners of the syntax that they are unlikely to affect too many people; one of of them is even the worst kind of breaking, changing legal UnicodeSet expressions to legal UnicodeSet expressions with a different meaning.
Note that we have already approved such a change on 2025-12-11, with [\N{LATIN SMALL LETTER A}-\N{LATIN SMALL LETTER Z}] (formerly a 1-element set, now a 26-element set).
The expressions whose interpretation was altered by the \N change were egregiously misleading; I think the same applies to the space-insensitive string literals, and the Properties and Algorithms Group of the UTC agreed, and so that the breaking change is likewise likely to help users more than it harms.

The breaking changes are first in the list below, clearly marked (and the legal-to-legal is first among those), with the list moving on to pure extensions of ICU's supported UnicodeSet syntax towards the bottom. The final item is actually about UnicodeSet parsing, but about the output of toPattern.

Best regards,

Robin Leroy

Breaking, legal-to-legal: Space-sensitive string literals ICU-23307

Currently, `[{T h i s s e t}]` contains the 7-character string "`Thisset`": spaces are ignored in string literals. This came as a surprise to both Mark Davis, who designed the support for string literals, and to Markus Scherer, who is probably the longest continuous maintainer of ICU's UnicodeSet implementation.

The question was brought to the Properties and Algorithms group of the UTC, which was similarly surprised by the idea of a space-insensitive string literal, and pointed out that string literals don't do that in any other language.

Proposal: Stop ignoring spaces in UnicodeSet string literals. `[{T h i s s e t}]` will now contain the 13-character string "`T h i s s e t`".

Breaking, legal-to-illegal: No spaces in [:^ ICU-23306

Currently,

```
[:
XID continue:]
```

and

```
[:
^XID continue:]
```

and

```
[:   ^XID continue:]
```

are valid, but

```
[:XID continue
:]
```

and

```
[:^
XID continue
:]
```

are ill-formed.

```
\p{
XID continue}
```

is also ill-formed. PD UTS61 treats [:^ as atomic. As in current ICU, spaces are not otherwise allowed within [: :] or \p{ }, except that U+0020 is ignored as part of name alias comparison.

Proposal: Disallow spaces (horizontal or vertical) within [:^; disallow vertical space or tabulations after [:. U+0020 SPACE remains allowed in the likes of `[ : X ID CONTINUE :]` by ICU's implementation of UAX44-LM3 (actually an earlier version of that rule; we don't do "is").

Breaking, legal-to-illegal: No unary property queries with a trailing equals sign ICU-23308

Currently, \p{XID_Continue=} is equivalent to \p{XID_Continue} or \p{XID_Continue=Yes}. Even weirder, \p{Uppercase_Letter=} is equivalent to \p{Uppercase_Letter} or \p{General_Category=Uppercase_Letter}.

Proposal: disallow an equals sign on unary queries.

Breaking, legal-to-illegal: Regularize escapes in strings ICU-23311

In ICU 78, \p, \P, and \N are disallowed in a set unless they are property queries or named characters, but they are allowed in string literals, with the same meaning as unescaped p, P, and N.

\N{} escape sequences now being treated as characters rather than sets, they should be allowed in string literals; this means disallowing \N for N. For consistency and ease of lexing, \p, and \P should be disallowed too.

Proposal:

- **Allow \N{} escapes in string literals, and disallow \N as an escaped N.** [This part is already done in #3828, but was not mentioned in the proposal to treat \N as a character rather than a set.]
- **Disallow \p and \P in string literals.**

In UTS61 terms, this means the following change:

string-element ::=

bracketed-literal-element | escaped-element | named-element ~~| \p | \P | \N~~

Breaking only in Java, legal-to-illegal: bracketed ranges but no string ranges ICU-23312

In ICU4J, [{a}]=[a], [{a}-{z}]=[a-z], and [{aa}-{cz}] is the set of all 78 strings starting with [a-c] and ending with [a-z]. The string ranges used to be specified in UTS #35, but have since been retracted.

In ICU4C, [{a}]=[a], and both [{a}-{z}]=[a-z] and [{aa}-{cz}] are ill-formed.

Proposal: Allow ranges of bracketed code points, such as [{a}-{z}] or [{a}-z]; disallow string ranges. Note that this is what ICU4X did.

Binary query negation with NOT EQUAL TO ICU-23313

PD UTS #61 (like UTS #35) allows ≠ as part of a property query, so that \p{Property≠Value} and [:Property≠Value:] are equivalent to \P{Property=Value} and [:^Property=Value:]. This is also supported by ICU4X.

Proposal: Allow ≠, but reject any doubly negated property-query, e.g., [:^Property≠Value:] or \P{Property≠Value}.

Extended name escapes ICU-23314

PD UTS61 proposes adding support for escapes that have both the hex code point and the name, *e.g.*, \xN{0061:LATIN SMALL LETTER A}, or the hex code point, the literal character, and the name, \xlN{0061:a:LATIN SMALL LETTER A}. The need for that has become apparent in testing of properties for new characters, where we make heavy usage of UnicodeSet. It parallels the well-established practice of citing characters by code point and name, or by code point and name while showing the glyph.

The use of the colon as a delimiter is inspired by its use as a delimiter for the version-qualifier in the more advanced corners of property-query syntax (which are out of scope for ICU).

Proposal: Accept hex:name and hex:literal:name escapes, both with the prefix \xN (no need for a separate \xlN prefix as currently proposed in PD UTS #61).

Rationale: \N{} has taken a life of its own, with C++23; extending it might lead to confusion, so let's use \xN. But separate prefixes for \xN and \xlN don't seem to be solving any problem and don't improve readability.

Name aliases and UAX44-LM2 in \N and \p{Name=…}

\N and \p{Name=...} (which are backed by the same implementation) currently ignore spaces and case, but not medial hyphens; they do not take name aliases into account. This means that \N{PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRACKET} does not work (but \N{PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRAKCET} works), and that \N{Latin small ligature o-e} does not work (but \N{Latin small ligature o e} works).

Proposal: Take name aliases into account, implement UAX44-LM2 as specified, as described in PD UTS#61 (#Named-Elements-Semantics) and PD UTS #61 (#Valid-Values-and-Resolved-Sets)

This has actually already been accepted as ICU-8963 and ICU-3736.

More consistent toPattern

When a sufficiently simple UnicodeSet expression is parsed, its toPattern is normalized, e.g.,

UnicodeSet(`[ a-b d- qp-z ]`).toPattern() is `[abd-z]`, where for readability we have written UnicodeSet(*s*) for C++ `UnicodeSet("s", errorCode)`.

However, when the expression contains inner UnicodeSets, including property-queries, the entire syntactic structure of higher levels (but not of the bottommost level) is preserved, although some pretty-printing is performed:

UnicodeSet(`[ a-b [ccc] d- qp-z ]`).toPattern()
is `[a-b[c]d-qp-z]`,
UnicodeSet(`[[ a-b d- qp-z ]  & [: Let ter:]]`).toPattern()
is `[[a-bd-qp-z]&[: Let ter:]]`.

In addition, some escaping is performed even when unnecessary:
UnicodeSet(`[{Baden-Württemberg$}]`).toPattern()
is `[{Baden\-Württemberg\$}]`.

The intent here is that dependencies on properties should be preserved: calling toPattern on a set created from \p{XID_Continue} should yield a versionless reference to XID_Continue, not a set frozen at the current version of Unicode. However, there is no reason not to otherwise simplify the expression, and computing a string matching the exact input syntax while parsing just in case we need to preserve it requires considerable bookkeeping.

Proposal: Change the behaviour of toPattern to something more consistent, while retaining the property that for a string s, with s′ := UnicodeSet(s).toPattern(), UnicodeSet(s) == UnicodeSet(s′) independently of property values. For this purpose no properties are to be considered immutable: \p{ASCII} must not turn into [\x00-\x7F].

This is a bit of a blank cheque, because pretty-printing lies outside of the scope of the formal specification in UTS #61, and because it is not quite clear what will be easy and useful (for instance, converting the set arithmetic on property queries to disjunctive or conjunctive normal form might be counterproductive, but some application of De Morgan's laws might be useful).

Amendment approved 2026-01-15

Extended name escapes: use the prefix \N for all forms (\N{name}, \N{hex:name}, \N{hex:literal:name}, no \xN.
Space-sensitive string literals: Transitionally disallow unescaped Pattern_White_Space in string literals for ICU 79 & 80: ICU ..78: {n g} means {ng}; ICU 79..80: {n g} is disallowed; ICU 81..: {n g} means {n\u0020g}.

[pre-approved:2025-12-11] ICU API proposal: upcoming changes to corner cases of UnicodeSet behaviour

Robin sent the following to icu-design on dec19

Dear ICU Team & Users,

[Note: If you are on the ICU-TC mailing list, you have already read this proposal.]

Since April this year, work has been going on to prepare a new Unicode Technical Standard #61, “Unicode Set Notation”, to rigorously specify UnicodeSet syntax and ensure that it corresponds to what is implemented.
In order to achieve alignment between the proposed draft standard and the ICU implementation, in the draft standard I proposed changes to the existing behaviour of UnicodeSet in ICU4C and ICU4J.
The following changes have already been accepted by the Technical Committee at its 2025-12-11 meeting. Terms like “property-query” and “NamedSingleton” refer to elements of the grammar in draft UTS #61.

1. **Force variables to be grammatical instead of arbitrary text replacement. ICU-23301**
   This only affects users who supply a custom SymbolTable when parsing UnicodeSet pattern strings.
   Currently, variables $like_this are implemented by text substitution in the UnicodeSet expression (with some strange edge cases: a property-query cannot span a variable boundary, and in fact a variable cannot even start with a property-query, so that with $letters=[:L:], [$letters-[z]] is an error).
   This means that it is impossible to parse a UnicodeSet expression without knowing the expansion of the variables (that is a problem for implementers, who often want to pre-parse set-valued variables) and it allows for misleading usage. For instance, with $x=a, **$y=-**, $z=z, [$xyz] is the 26-element set [a-z], [xz$y] is the 3-element set [az-].
   The Technical Committee decided to require that variables correspond to characters, strings, or sets. $a-$b would therefore be a range for two characters, a set difference for two sets, or ill-formed with any other combination of types.
   Note that ICU4X needed to pre-parse its UnicodeSets in transliterator rules, and so has implemented this already; we therefore know that CLDR transliterators are compatible with this change.
2. **Stop using lookupMatcher in UnicodeSet parsing. ICU-23297**
   This only affects users who supply a custom SymbolTable when parsing UnicodeSet pattern strings.
   UnicodeSet currently has a (poorly-documented) alternative way of using variables, by implementing lookupMatcher in the SymbolTable. Characters can be mapped to UnicodeSets.
   This is not used directly, but is instead used by the ICU transliterator and RBBI implementations to get pre-parsed set-valued variables: Transliterator rule variables map to a PUA range (U+E000..U+F8FF), which in turn maps to sets via lookupMatcher. (RBBI uses a single noncharacter instead and relies on the order of calls).
   This is obscure, brittle, and complicates parsing; what it achieves is made redundant by the decision to make variables grammatical above (if a variable can directly represent a set, there is no need to jump through a PUA code point to do so).
   In addition, syntax characters can be remapped by this mechanism, to varying effect; this further complicates parsing and can only lead to absurd behaviour.
   The Technical Committee decided to get rid of this mechanism entirely: UnicodeSet parsing will no longer call SymbolTable.lookupMatcher().
   Note that this will entail changes to RBBI and transliterator rule parsing, so that they have some other way of using their pre-parsed sets.

3. **Treat \N as a character, not a set, with no exception for backward compatibility. ICU-22851**
   Currently [\N{LATIN LETTER SMALL A}-\N{LATIN LETTER SMALL Z}] is the 1-element set [a]. This is highly misleading (and I have been bitten by it).
   The PD UTS #61, Revision 1, draft 2 (archival version: L2/25-265) proposes allowing \N (named-element) where a character is (in RangeElement), but also in some places where a set is (as a NamedSingleton), for backward compatibility with users who might have written \N{LATIN LETTER SMALL A} for [a], or [[a-z]-\N{LATIN LETTER SMALL A}] for [abd-z].
   The Technical Committee discussed this at length, and eventually **decided to allow named-element as a character** (in the RangeElement production), but not to allow named-element to occur as a set. That is, **the TC rejected the changes highlighted in cyan in UTS #61, Revision 1, draft 2**.
   The reasons are twofold:
   1. This complicates the grammar and thus the understandability of UnicodeSet syntax going forward, for a relatively small advantage in backward compatibility: \N is not very widely used, and adding [] in a few places is a small migration cost.
   2. C++23 allows for \N escapes in strings. Allowing \N to stand for a set would mean that, while UnicodeSet("[\\N{LATIN LETTER SMALL A}-\\N{LATIN LETTER SMALL Z}]") and UnicodeSet("[\N{LATIN LETTER SMALL A}-\N{LATIN LETTER SMALL Z}]") would be equivalent, UnicodeSet("\\N{LATIN LETTER SMALL C}") would be OK, but UnicodeSet("\N{LATIN LETTER SMALL C}") would be an error; or that UnicodeSet(R"([[a-z]-\N{LATIN LETTER SMALL C}])") would be OK, but that UnicodeSet("[[a-z]-\N{LATIN LETTER SMALL C}]") would be an error.
   Since UnicodeSet syntax is otherwise largely consistent with C++ escape sequences, the confusion this would induce was deemed to outweigh the backward incompatibility.

Best regards,

Robin Leroy

[proposed:2024-10-07] ICU4C MessageFormat 2.0: Function composition

Picking up from ICU 77 [proposed:2024-10-07] ICU4C MessageFormat 2.0: Function composition

Refreshed 2025-jun-16, TC discussion jun26.

Tim sent the following to icu-design on oct07

Dear ICU team & users,

I would like to propose the following changes to the MF2 API in the
MessageFormat 2.0 tech preview API in ICU 77.

Some people may remember the discussion of function composition in MF2

and the lack of definition of the concept of a "resolved value" in the spec. Recently, the spec has changed to define this concept much more precisely and provide clearer guidance for how to implement function composition.

I've created a design doc at
https://docs.google.com/document/d/1nIYDyaTqB6nChhvoSVxBkRfBAiPchlN4anvaAma9WRc/edit

There is a fair amount of context needed from both the MF2 spec and implementation in order to follow this proposal, so please don't hesitate to comment on the doc with questions and I'll try to clarify things as much as I can.

I would appreciate feedback before Thursday, October 17.

Backlog

1. [approved:2025-06-12] Introduction of ICU 78 Units Converter API
    a. Approved but not landed yet
2. [proposed:2024-10-07] ICU4C MessageFormat 2.0: Function composition
    a. Tim proposed for ICU 77
3. [proposed:2024-10-02] ICUC MessageFormat 2.0: Matching on variables in the data model
    a. Tim proposed for ICU 77
4. [proposed:2023-11-01] ICU4J API Proposal to register an External Break Engine
    a. Frank proposed for ICU 75, got feedback, need to iterate
5. [approved:2023-12-07] Add C APIs to support Collation Folding
    a. Approved for ICU 75 but implementation needs review & iteration