# Modernizing ICU for You

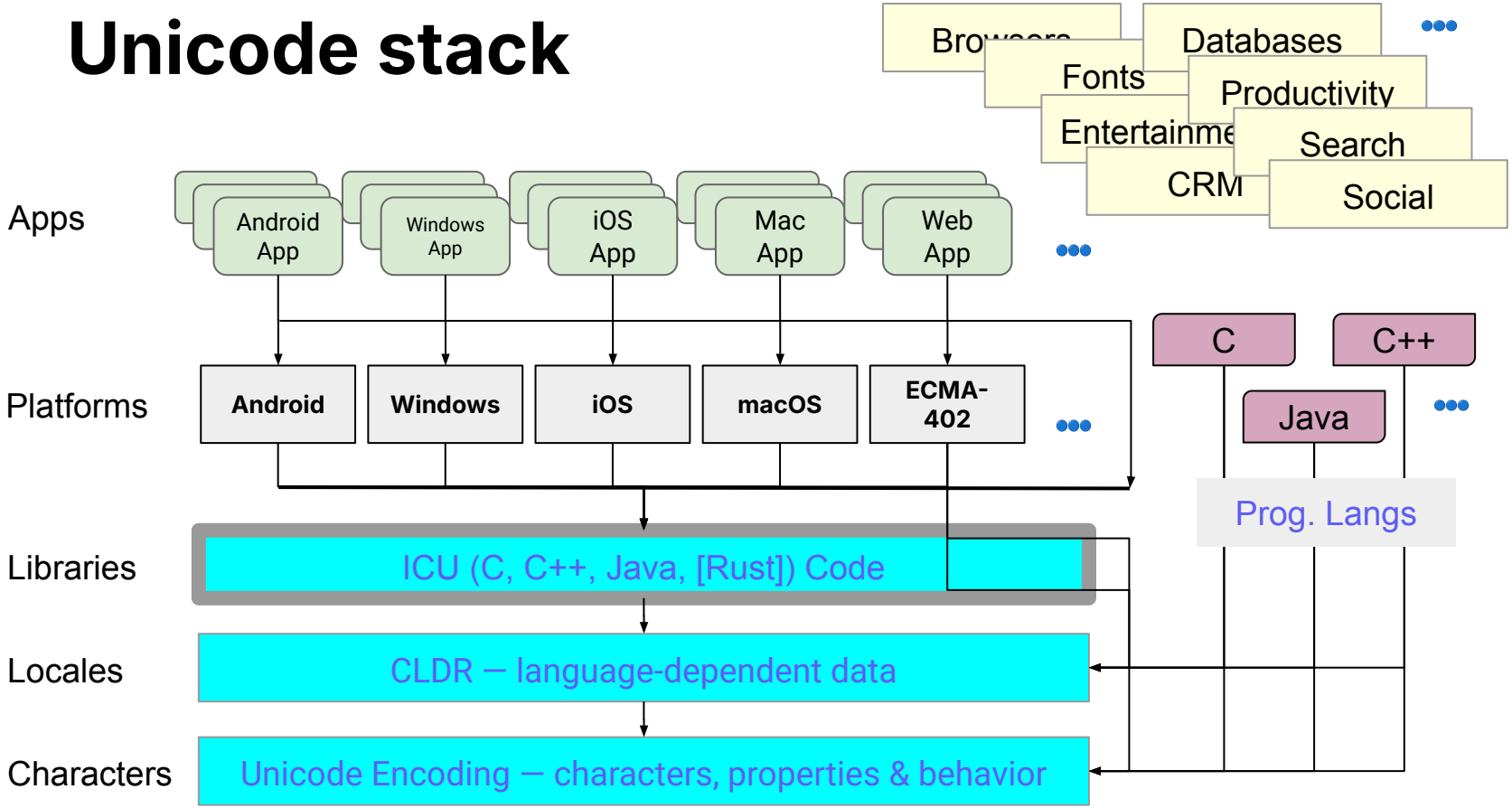
**Unicode Technology Workshop 2025**

Hosted at Microsoft Silicon Valley Campus

Markus Scherer, Google & Unicode

November 11-13

# Unicode stack



# Timeline

1990s


- Apple “Pink” + IBM → Taligent
- i18n → Java 1.1.4 Locale, formatting, collation, text boundaries
- Taligent → IBM — “IBM Classes for Unicode”

1999: open source Java, C++, C

2000: “International Components for Unicode”

2016: Moved to Unicode

# ICU evolution: not just features

- Java 1.1 → Java 8..25
- Early C++ → C++17..23 / early C → C11
- Java CharSequence & Appendable
- Covariant return types
- C++ namespaces
- C++ StringPiece & ByteSink
- LocalPointer ≈ unique_ptr
- Some direct-UTF-8 implementations
- Locale: BCP 47
- NumberFormatter

# Segmentation

# BreakIterator

```
BreakIterator bi = BreakIterator.getWordInstance(locale);
bi.setText(text);
int prev = bi.first();
int b;
while ((b = bi.next()) != DONE) {
    // use segment: text[prev, b[
    prev = b;
}
```

current()
following()
previous()
...

# Ease of use?

```
BreakIterator bi = ...;

bi.setText(text);  // necessary?

String result = CaseMap.toTitle().apply(locale, bi, text);

int b = bi.current();  // moved?
```

# Modern Java

Iterable

- Iterator
  - hasNext(), next()
  - for (T el : container) ...

Stream

- forEach(action), filter(f), map(m), collect(c), ...

# Segmenter

```
Segmenter seg = LocalizedSegmenter.builder().
    setLocale(locale).setSegmentationType(WORD).
    build();  // immutable: “how to segment”
seg.segment(text).  // Segments
    subSequences()...  // Stream<CharSequence>
```

segments()
boundariesAfter(i)
segmentsBefore(i)
...

# Next

CaseMap.Title

StringSearch

C++

# UnicodeSet

... and inspecting its contents

# UnicodeSet

UnicodeSet set(u"[:Basic_Emoji:]", errorCode);

assertTrue(set.contains(U'🚴'));

assertTrue(set.contains(u"\u2602\uFE0F")); // ☂️

- Set of strings
- But mostly code points
  - Stored as [start, end] ranges

# UnicodeSet code points

```
UChar32 c;
for (int32_t i = 0; (c = set.charAt(i)) >= 0; ++i) ...  // slow

int32_t count = set.getRangeCount();
for (int32_t i = 0; i < count; ++i) {
    UChar32 start = set.getRangeStart();
    UChar32 end = set.getRangeEnd();
    // use [start, end]
}
```

*(same in C)*

# UnicodeSetIterator

```
UnicodeSetIterator iter(set);
while (iter.next()) {
    // use iter.getCodepoint() / isString() / getString()
}
iter.reset();
while (iter.nextRange() && !iter.isString()) {
    // use [iter.getCodepoint(), iter.getCodepointEnd()]
}
```

# C++11+ range-based for loops

C++ “range”
begin(), end()
→ iterator
* == !=
++ ++(int)
-- --(int)
…

Wanted:

```
for (auto c : codePointsOf(set)) {
    // use c
}
for (auto range : rangesOf(set)) {  // ..., strings, all elements
    // use range
}
```

# UnicodeSet C++ “ranges”

```
UnicodeSet set(u"[abcç力🚴]", errorCode);
for (UChar32 c : set.codePoints()) {
    printf("set.codePoint U+%04lx\n", (long)c);
}
for (auto [start, end] : set.ranges()) {
    printf("set.range U+%04lx..U+%04lx\n", (long)start, (long)end);
}
```

*strings()*

*begin()+end()*
*→ elements as strings*

# C++ header-only APIs

Modern C++ APIs

on top of

binary stable

C APIs

... or fully inline

```
LocalUSetPointer uset(uset_openPattern(u"[abcç力🚴]", -1, &errorCode));
for (auto [start, end] : USetRanges(uset.getAlias())) {
    printf("uset.range U+%04lx..U+%04lx\n", (long)start, (long)end);
}
for (auto range : USetRanges(uset.getAlias())) {
    for (UChar32 c : range) {
        printf("uset.range.c U+%04lx\n", (long)c);
    }
}
```

UNICODE

# UnicodeSet.java

- ... is-an Iterable<String>  // 2009
- Iterable<Integer> codePoints()  // ICU 76
- Iterable<EntryRange> ranges()  // ICU 54
- Collection<String> strings()  // ICU 4.4

- Stream<String> stream()  // ICU 76
- IntStream codePointStream()
- Stream<EntryRange> rangeStream()
- Stream<String> stringStream()

# Collator

# Collator

```
Collator coll = Collator.getInstance(locale);  // Comparator
TreeSet<String> set = new TreeSet<>(coll);
set.add(string);   // etc.
```

C++: Collator::compare(left, right, errorCode) → -1/0/+1

compareUTF8(left, right, errorCode)

C: ucol_strcoll(...) → -1/0/+1

# Collator: modern C++

```
Collator *coll = Collator::createInstance(locale, errorCode);

std::sort(vec.begin(), vec.end(), coll→less());  // C++17

std::ranges::sort(vec, coll→greater());  // C++20
```

works for UTF-8 & UTF-16 strings

```
UCollator *ucol = ucol_open(localeID, &errorCode);

std::ranges::sort(vec, icu::collator::less(ucol));
```

UNICODE

# Input & output strings

# Input strings

const Unit * + length  // maybe NUL-terminated

const UnicodeString  // can be an alias

UCharIterator, UText

Java: String, char[], CharacterIterator (UCS-2), ...

# Output strings

Unit * + capacity

Return length, buffer overflow, write NUL if possible, ...

UnicodeString  // can be an alias

# Convergence

ICU4C StringPiece ≈ later std::string_view

Java 1.4: CharSequence

ICU4C ByteSink, Appendable

Java 5: Appendable

# C++17 strings

std::string_view, u16string_view, ...

std::string, u16string, ...

C++ UnicodeString ↔ std::u16string_view / wstring_view

- Works seamlessly with u"string literals"

# Challenge: int32_t ⚡ uint64_t

ICU4C lengths & indexes & capacities: int32_t (=int in Java)

- Negative input length = NUL-terminated

C++ on modern CPU: size_t = uint64_t

# maybe more APIs with string_view input

# Unicode strings → code points

# UTF-8/16/32

"bç力🚴"

| | | | |
|---|---|---|---|
| 62 | C3 A7 | E3 82 AB | F0 9F 9A B4 |
| 0062 | 00E7 | 30AB | D83D DEB4 |
| 0062 | 00E7 | 30AB | 1F6B4 |

# C macros

```
const char16_t *s = ...;
int32_t length = ...;
for (int32_t i = 0; i < length;) {  // macro increments i
    UChar32 c;
    U16_NEXT(s, i, length, c);  // c < 0 if ill-formed
    // use c
}
```

# Many macros

U16_NEXT_OR_FFFD()

U16_NEXT_UNSAFE()

U16_PREV()

U16_FWD_1()

**U8**_NEXT()

...

(no UTF-32)

# Macros 👎

C++ devs don’t like macros

- Not type-safe
- Debugging…
- Duplicate APIs for UTF-8/16
- Want:  for (auto c : codePointsOf(s))
- Want STL algorithms

# UTFIterator

```
for (auto units : utfStringCodePoints<UChar32, UTF_BEHAVIOR_NEGATIVE>(s)) {
    sum += units.codePoint();  // < 0 if ill-formed
}

auto range = utfStringCodePoints<char32_t, UTF_BEHAVIOR_FFFD>(s);
for (auto iter = range.rbegin(), limit = range.rend(); iter != limit; ++iter) {
    sum += iter->codePoint();  // U+FFFD if ill-formed
}
```

# UTFIterator++

```
int32_t countCodePoints16(std::u16string_view s) {
    auto range = utfStringCodePoints<UChar32, UTF_BEHAVIOR_SURROGATE>(s);
    return std::distance(range.begin(), range.end());
}
char32_t firstCodePointOrFFFD(std::u16string_view s) {
    if (s.empty()) { return 0xfffd; }
    auto range = utfStringCodePoints<char32_t, UTF_BEHAVIOR_FFFD>(s);
    return range.begin()->codePoint();
}
```

# Iterator features & options

- Validating + “behaviors” when ill-formed
- Non-validating requires well-formed — fast & small
- → Code point, length, begin()/end(), string_view, wellFormed()
- Output code point type
- Auto-UTF-8/16/32
- Compiler removes all unused code

# C++17 iterators / C++20 ranges

Beyond const charXX_t * / XXstring_view

- Flexible code unit type (char, char8_t, uint8_t, char16_t, ...)
- Any code unit iterator
  - UnicodeString is-a code unit “range”
- C++ iterator types: input, forward, bidirectional
  - std::cin, std::wcin
- Efficient reverse_iterator

Satisfy C++17 iterator traits

Satisfy C++20 range & iterator concepts

# Fun with Cuneiform

```
auto codePoint = [](const auto &codeUnits) { return codeUnits.codePoint(); };
const std::u16string text = u"□□□□□ □□□□\n"
                            u"□□□□ □□□□ □□\n"
                            u"□□ □□ □□□□ □□\n";
auto lines3sqq = text | std::ranges::views::lazy_split(u'\n') | std::views::drop(1);
auto codeUnits = *lines3sqq.begin();
assertTrue(std::ranges::equal(
        utfStringCodePoints<char32_t, UTF_BEHAVIOR_FFFD>(codeUnits) |
                std::ranges::views::transform(codePoint),
        std::u32string_view(U"□□□□ □□□□ □□")));
```

# Other new code point helpers

# Code point → string

```
using icu::header::utfstring::appendOrFFFD;
std::string s;  // u8string, u16, u32
appendOrFFFD(s, U'ç');
appendOrFFFD(s, 0xd800);
appendOrFFFD(s, U'🚴');
assertTrue(s =="ç\uFFFD🚴"));

assertTrue(encodeOrFFFD<std::string>(U'カ') =="カ");
```

*appendUnsafe()*
*encodeUnsafe()*

# Test & builder helpers

```
for (UChar32 c : icu::header::AllCodePoints<UChar32>()) ...
```

```
for (char32_t c : icu::header::AllScalarValues<char32_t>()) ...
```

# Maybe next

- Convert code points to code units via {input, output} iterators?
- Prefer const_iterator from range if available?
- Adjust to C++23/26/...

# Java code points

Mostly Java 5 — collaboration with ICU team

- "ç🚴".codePointAt(1)
- StringBuilder.appendCodePoint(c)
- CharSequence.codePoints() → IntStream  // Java 8

ICU 78 UCharacter

- allCodePoints() / allCodePointsStream()
- allScalarValues() / allScalarValuesStream()

# Other news

# Locale: under the hood

- Widely used in & around ICU
- Internal: raw buffers → string_view+ByteSink
- 224+ bytes → 40 for most common locales

# maybe: sym exports / internal

improved symbol exports so
that we can now use
std::variant & similar in private
members of public classes

# Calendar??

mutable

raw input/output types (int/double)

0-based vs. 1-based

Joda Time → java.time → Temporal??

# Java: format date & time

DateFormat can now format java.time (Java 8) types

# wishes?

# The end

# ICU on the web

- icu.unicode.org/ —
- icu.unicode.org/bugs (Jira)
- github.com/unicode-org/icu — repo
- unicode-org.github.io/icu/ — docs & downloads
- unicode-org.github.io/icu/contacts
- unicode-org.github.io/icu-docs/ — API docs

# Thank you!

【 U+3010
シ U+30B7
メ U+30E1
ð U+10DB

☎️ U+260E
デ U+30C7
👽 U+1F47D
ق U+0642

☂️ U+2602
个 U+4E2A
 U+1F60C
Ψ U+03A8
ع U+063A
ミ U+30DF
鏡 U+93E1

ε U+03B5
○ U+3007
♺ U+267A
ζ U+03B6
く U+304F
ß U+0D26
ৰ U+09F0

# Thank you to our Organizational Members

# Thank you to our Industry and Media Partners