# Overview

This document is an attempt to describe the current state of Intl API (formally ECMA-402 specification) implementation in Hermes JavaScript engine, on Android platform. I am still coding, learning, and testing the implementation; hence this document is by no means comprehensive and will be a live document for a few more days.

ECMA-402 underwent multiple iterations and is still evolving. The latest one is 7th edition published in June 2020. Each new iteration is built on top of the last one and only adds new functionalities.

We decided to implement Intl APIs leveraging the libraries and data available in Android Platform (instead of statically linking against an internationalization library at build time and packaging) primarily for space efficiency. Android platform internationalization libraries have always been based on [ICU4j project](https://unicode-org.github.io/icu-docs/#/icu4j). Version of ICU4j and the backing [CLDR data](http://cldr.unicode.org/) varies across Android platform versions. Also, the ICU APIs were never exposed directly, but only through wrappers or aliases. This results in a lot of variance in internationalization API surface across platform versions.

# Internationalization framework in Android Platform

The following table summarizes ICU, CLDR and Unicode versions available on the Android platforms that we intent to support.

**Platform 24+ where ICU4j APIs are available.**

|  |  |
| --- | --- |
| Android 11 (API level 30) | ICU4J 66.1 ([ref](https://android.googlesource.com/platform/external/icu/+/refs/heads/android11-mainline-release/icu4j/readme.html))  Unicode 13 beta  CLDR 36.1 |
| Android 10 (API level 29) | ICU4j 63.2 ([ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 34  Unicode 11.0 |
| Android 9 (API level 28) | ICU4j 60.2 ([ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 32.0.1  Unicode 10.0 |
| Android 8.0 - 8.1 (API levels 26 - 27) | ICU4j 58.2( [ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 30.0.3  Unicode 9.0 |
| Android 7.0 - 7.1 (API levels 24 - 25) | ICU4j 56 ([ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 28  Unicode 8.0 |

**Pre-24 platforms**

|  |  |
| --- | --- |
| Android 6.0 (API level 23) | ICU4j 55.1 ([ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 27.0.1  Unicode 7.0 |
| Android 5.0 (API levels 21–22) | ICU4j 53 ([ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 25  Unicode 6.3 |
| Android 4.4 (API levels 19–20) | ICU4j 51 ([ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 23  Unicode 6.2 |
| Android 4.3 (API level 18) | ICU4j 50 ([ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 22.1  Unicode 6.2 |
| Android 4.1 (API levels 16–17) | ICU4j 4.8 ([ref](https://developer.android.com/guide/topics/resources/internationalization))  CLDR 2.0  Unicode 6.0 |

To Summarize, there are three big partitions in the platform domain,

Platforms >=24 has much better internationalization support than earlier, as many ICU classes are available as is.

Platforms 21-24 still has reasonable internationalization support, by allowing creation of Locale objects and exposing selected ICU services through java.text namespace.

Platforms < 21 doesn’t allow creation of Locale objects from tags, severely limiting general purpose international code.

# ECMA-402 Compliance of our implementation

## Android 11

|  |  |
| --- | --- |
| ECMA-402 Edition  & Corresponding ECMA-262 edition | ECMA-402 public API Summary (Deltas) |
| [1st Edition / December 2012](https://www.ecma-international.org/ecma-402/1.0/index.html)  ES 5.1 | **Service Constructors in Intl Object:**  Intl.Collator  Intl.NumberFormat  Intl.DateTimeFormat  **Non-internal properties of Intl.Collator constructor:**  Length = 0  Intl.Collator.prototype  Intl.Collator.supportedLocalesOf  **intializeCollator options:**  usage: [sort, search]  localeMatcher: [lookup, best fit]  sensitivity: [base, accent, case, variant, undefined]  ignorePunctuation: bool  numeric(kn) and caseFirst (kf) collation accepted through locale extension and options.  collation type(co) only through locale extension  **Non-internal properties of the Intl.Collator Prototype Object:**  Intl.Collator.prototype.compare  Intl.Collator.prototype.resolvedOptions  **Non-internal properties of Intl.NumberFormat constructor:**  Length = 0  Intl.NumberFormat.prototype  Intl.NumberFormat.supportedLocalesOf  **initializeNumberFormat options:**  localeMatcher: [lookup, best fit]  style: [decimal, percent, currency]  I could not find an ICU API to achieve this. The implementation is hacky, hence expect some bugs here.  currency: string  currencyDisplay: [code, symbol, name]  minimumIntegerDigits  minimumFractionDigits & maximumFractionDigits  minimumSignificantDigits & maximumSignificantDigits  useGrouping: bool  numbering system (nu) accepted through locale extension  **Non-internal properties of the Intl.NumberFormat Prototype Object:**  Intl.NumberFormat.prototype.format  Intl.NumberFormat.prototype.resolvedOptions  **Non-internal properties of Intl.DateTimeFormat constructor:**  Length = 0  Intl.DateTimeFormat.prototype  Intl.DateTimeFormat.supportedLocalesOf  **initializeDateTimeFormat options:**  localeMatcher: [lookup, best fit]  timeZone: string (accepts only UTC ?)  weekday: ["narrow", "short", "long”]  era: ["narrow", "short", "long"]  year: ["2-digit", "numeric"]  month: ["2-digit", "numeric", "narrow", "short", "long"]  day: ["2-digit", "numeric"]  We/ICU supports all subsets of fields; hence the option does not have any value. Firefox/Chrome also does not seem to respect this.  hour: ["2-digit", "numeric"]  minute: ["2-digit", "numeric"]  second: ["2-digit", "numeric"]  timeZoneName: ["short", "long"]  formatMatcher: [basic, best fit]  hour12: bool  Calendar (ca) and Numbering system (nu) accepted through locale extension.  **Non-internal properties of the Intl.NumberFormat Prototype Object:**  Intl.DateTimeFormat.prototype.format  Intl.DateTimeFormat.prototype.resolvedOptions  **Locale Sensitive Functions:**  String.prototype.localeCompare  Number.prototype.toLocaleString  Date.prototype.toLocaleString  Date.prototype.toLocaleDateString  Date.prototype.toLocaleTimeString |
| [2nd Edition / June 2015](https://www.ecma-international.org/ecma-402/2.0/index.html)  ECMAScript 2015 (ECMA-262 6th Edition or successor | **Non-internal properties of the Intl.Collator Prototype Object:**  Intl.Collator.prototype[@@toStringTag] =” Object”  **Non-internal properties of the Intl.NumberFormat Prototype Object:**  Intl.NumberFormat.prototype[@@toStringTag] =” Object”  **Non-internal properties of the Intl.DateTimeFormat Prototype Object:**  Intl.DateTimeFormat.prototype[@@toStringTag] =” Object”  **Locale Sensitive Functions:**  String.prototype.toLocaleLowerCase  String.prototype.toLocaleUpperCase  Array.prototype.toLocaleString |
| [3rd Edition / June 2016](https://www.ecma-international.org/ecma-402/3.0/index.html)  ECMAScript 2016 (ECMA-262 7th Edition or successor) | **Function property on Intl Object:**  getCanonicalLocales |
| [4th Edition, June 2017](https://www.ecma-international.org/ecma-402/4.0/index.html)  ECMAScript 2017 (ECMA-262 8th Edition or successor) | **Non-internal properties of the Intl.DateTimeFormat Prototype Object:**  Intl.DateTimeFormat.prototype.formatToParts |
| [5th edition, June 2018](https://www.ecma-international.org/ecma-402/5.0/index.html)  ECMAScript 2018 (ECMA-262 9th Edition or successor) | **Service Constructors in Intl Object:**  Intl.PluralRules  **Non-internal properties of the Intl.NumberFormat Prototype Object:**  Intl.NumberFormat.prototype.formatToParts  **initializeDateTimeFormat options:**  hourCycle: ["h11", "h12", "h23", "h24"] |
| [6th edition, June 2019](https://www.ecma-international.org/ecma-402/6.0/index.html)  ECMAScript 2020 (ECMA-262 10th Edition or successor) | **Locale Sensitive Functions:**  BigInt.prototype.toLocaleString |
| [7th Edition / June 2020 (Latest)](https://www.ecma-international.org/ecma-402/7.0/index.html)  ECMAScript 2020 (ECMA-262 11th Edition or successor) | **Service Constructors in Intl Object:**  Unit format style does not work well with formatToParts.  Intl.RelativeTimeFormat  **initializeNumberFormat options:**  numberingSystem: string  style: [decimal, percent, currency, unit]  CurrencySign: ["standard", "accounting"]  CompactFormatter does not implement formatToParts.  notation: [standard, scientific, engineering, compact]  unit: string  unitDisplay: ["short", "narrow", "long"]  compactDisplay: [short, long]  signDisplay: ["auto", "always", "never", or "exceptZero"]  **initializeDateTimeFormat Options:**  signDisplay implementation is mostly incomplete.  Calendar: string  numberingSystem: string |
| Not yet in spec ...  But is already supported by V8, SpiderMonkey | **initializeDateTimeFormat Options:**  [dateStyle/timeStyle](https://tc39.es/proposal-intl-datetime-style/)  [dayPeriod](https://github.com/tc39/ecma402/issues/29)  [fractionalSecondDigits](https://github.com/tc39/ecma402/pull/347) |

Other known bugs/issues:

1. The keys of the object returned by ‘resolvedOptions’ function in all Intl services are not deterministically ordered as prescribed by spec.
2. DateFormat: ECMAScript [beginning of time](https://www.ecma-international.org/ecma-262/11.0/index.html#sec-time-values-and-time-range) (-8,640,000,000,000,000), is formatted as November 271817, instead of expected April 271822.

## Android 10 (SDK < 30)

1. NumberFormat: Scientific notation formatting has issues on some cases such as Infinity (e.g. Expected SameValue(«-∞E0», «-∞») to be true).
2. NumberFormat: Compact notation formatToParts doesn’t identify unit, hence we report unit as ‘literal’ (Compact short: 987654321: parts[1].type Expected SameValue(«literal», «compact») to be true)
3. NumberFormat: formatToParts doesn’t produce expected result with Scientific/Engineering notation and input Infinity (-Infinity - engineering: length Expected SameValue(«4», «2») to be true).

## Android 9 (SDK < 29)

1. There are some failures likely due to older Unicode and CLDR version, which are hard to generalize. Some examples are,
   1. NumberFormat: “Percent” is not accepted as a unit.
   2. NumberFormat: unit symbols difference, kph vs km/h
   3. Some issue in significant digit precision, which is not yet looked into the details.

## Android 8.0 – 8.1 (SDK < 28)

1. getCanonicalLocales: Some differences in the keyword values due to CLDR/Unicode version difference (Expected SameValue(«und-u-tz-utc», «und-u-tz-gmt») to be true)
2. NumberFormat: CompactFormatter doesn’t respect the precision inputs (Expected SameValue(«9,900만», «9877만») to be true; Expected SameValue(«990M», «988M») to be true).

## Android 7.0 - 7.1 (SDK < 26)

1. getCanonicalLocales: Unicode/CLDR version differences Expected SameValue(«und-u-ms-imperial», «und-u-ms-uksystem») to be true

## Android 7.0 - 7.1 (SDK < 24)

1. Collator: Doesn’t canonically decompose the input strings. Canonically equivalent string with non-identical code points may not match.
2. getCanonicalLocales: Unicode/CLDR version differences (Expected SameValue(«und-u-ca-ethiopic-amete-alem», «und-u-ca-ethioaa») to be true; Expected SameValue(«und-u-ks-primary», «und-u-ks-level1») to be true)
3. Unit style does not work.
4. There are issues in the precision configuration due to lack of APIs.
5. DateFormat: There are issues with the calendar configuration which needs to be dug into.

## SDK < 21

On platforms before 21, Locale.forLanguageTag() is not available, hence we can’t construct java.util.Locale object from locale tag. Hence, we fallback to English for any locale input.

Also, I haven’t got a chance to test them as I couldn’t find a suitable image.

# Impact on Application Size

The following numbers are measured using a test application which takes dependency on the Hermes library to evaluate a JavaScript snippet. Essentially, enabling Intl APIs adds 57-62K per ABI.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Product APK Size** | **NOINTL** | **INTL** | **DIFF** | **PERC** |
| ARM64 | 1,672,235 | 1,729,579 | 57,344 | 3.43% |
| ARM | 1,471,539 | 1,528,883 | 57,344 | 3.90% |
| X86\_64 | 1,844,255 | 1,901,599 | 57,344 | 3.11% |
| X86 | 1,950,739 | 2,012,179 | 61,440 | 3.15% |

# The overhead is contributed by both compiled native C++ and Java bits.

The uncompressed size of the Hermes shared library got bigger as follows,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **libhermes.so Size** | **NOINTL** | **INTL** | **DIFF** | **PERC** |
| ARM64 | 2,473,760 | 2,551,592 | 77,832 | 3.15% |
| ARM | 1,696,672 | 1,754,016 | 57,344 | 3.38% |
| X86\_64 | 2,633,528 | 2,711,368 | 77,840 | 2.96% |
| X86 | 2,859,916 | 2,945,936 | 86,020 | 3.01% |

And the Java bits got bigger as well,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Java Size** | **NOINTL** | **INTL** | **DIFF** | **PERC** |
| classes.jar (in hermes.aar) | 559 | 120975 | 120,416 | 21541.32% |
| classes.dex (intltestapp.apk) | 160708 | 234808 | 74,100 | 46.11% |

*Please note that the application dex file contains non-hermes class files too.*

# And finally, this is the increase in the final npm package,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NPM Package** | **NOINTL** | **INTL** | **DIFF** | **PERC** |
|  | 214447973 | 219291220 | 4,843,247 | 2.26% |

# Future Work

1. Implement Intl.PluralRules and Intl.RelativeTimeFormat services.
2. Most of the known issues with the NumberFormat can be fixed with the [android.icu.number.NumberFormatter](https://developer.android.com/reference/android/icu/number/NumberFormatter) class which is available on API 30+. We need to have a third implementation of IPlatformNumberFormat based on this class.
3. Fix the known fixable issues above.
4. At the start, I spent too much time on the Intl.getCanonicalLocales method to fit the output to the Test262 dataset. There is a lot of code in LocaleIdentifier.java and downstream files, which parses and canonicalizes a locale id. But later I switched to relying on ULocaleBuilder class for canonicalization, and our custom parser code in LocaleIdentifier is being used only on pre-24 platform. Current ULocaleBuilder based code fails a bunch of Test262 getCanonicalLocales tests, but it is a lot lesser code and easier to maintain, and likely performs better. Also, we are now conditionally compiling out the auto-generated tables that we extracted from CLDR, which is used in LocaleIdentier.java and hence pre-24 platforms, to avoid it from impacting the APK size and the memory footprint. In short, our Intl.getCanonicalizedLocales() is not standard compliant. We need to re-evaluate and consolidate the code. Please note that these failing cases are still rare corner cases, and we do well for most common scenarios. (For more details please look at HermesIntlGetCanonicalLocalesTest.java)