

ICU Documentation

[Formatting](#) / [Formatting Dates and Times](#)

Formatting Dates and Times

CONTENTS

- 1 [Formatting Dates and Times Overview](#)
- 2 [DateFormat](#)
 - a [Formatting Dates](#)
 - b [Parsing Dates](#)
 - c [Producing Normal Date Formats for a Locale](#)
 - d [Producing Relative Date Formats for a Locale](#)
 - e [Setting Time Zones](#)
 - f [Working with Positions](#)
- 3 [SimpleDateFormat](#)
 - a [Date/Time Format Syntax](#)
 - a [Date Field Symbol Table](#)
 - b [Time Zone Display Names](#)
 - a [Time Zone Display Name Types](#)
 - b [Time Zone Pattern Usage](#)
- 4 [DateTimePatternGenerator](#)
- 5 [DateFormatSymbols](#)
- 6 [Programming Examples](#)

Formatting Dates and Times Overview

Date and time formatters are used to convert dates and times from their internal representations to textual form and back again in a language-independent manner. The date and time formatters use `UDate`, which is the internal representation. Converting from the internal representation (milliseconds since midnight, January 1, 1970) to text is known as “formatting,” and converting from text to milliseconds is known as “parsing.” These processes involve two mappings:

- A mapping between a point in time (`UDate`) and a set of calendar fields, which in turn depends on:
 - The rules of a particular calendar system (e.g. Gregorian, Buddhist, Chinese Lunar)

- The time zone
- A mapping between a set of calendar fields and a formatted textual representation, which depends on the fields selected for display, their display style, and the conventions of a particular locale.

DateFormat

`DateFormat` helps format and parse dates for any locale. Your code can be completely independent of the locale conventions for months, **days** of the week, or calendar format.

Formatting Dates

The `DateFormat` interface in ICU enables you to format a `Date` in milliseconds into a string representation of the date. It also parses the string back to the internal `Date` representation in milliseconds.

```
DateFormat* df = DateFormat::createDateInstance();
UnicodeString myString;
UDate myDateArr[] = { 0.0, 100000000.0, 2000000000.0 };
for (int32_t i = 0; i < 3; ++i) {
    myString.remove();
    cout << df->format( myDateArr[i], myString ) << endl;
}
```

To format a date for a different `Locale`, specify it in the call to:

```
DateFormat* df = DateFormat::createDateInstance
( DateFormat::SHORT, Locale::getFrance());
```

Parsing Dates

Use a `DateFormat` to parse also:

```
UErrorCode status = ZERO_ERROR;
UDate myDate = df->parse(myString, status);
```

When numeric fields abut one another directly, with no intervening delimiter characters, they constitute a run of abutting numeric fields. Such runs are parsed specially. For example, the format `"HHmmss"` parses the input text "123456" to 12:34:56, parses the input text "12345" to 1:23:45, and fails to parse "1234". In other words, the leftmost field of the run is flexible, while the others keep a fixed width. If the parse fails anywhere in the run, then the leftmost field is

shortened by one character, and the entire run is parsed again. This is repeated until either the parse succeeds or the leftmost field is one character in length. If the parse still fails at that point, the parse of the run fails.

Producing Normal Date Formats for a Locale

Use `createDateInstance` to produce the normal date format for that country. There are other static factory methods available. Use `createTimeInstance` to produce the normal time format for that country. Use `createDateTimeInstance` to produce a `DateFormat` that formats both date and time. You can pass different options to these factory methods to control the length of the result; from `SHORT` to `MEDIUM` to `LONG` to `FULL`. The exact result depends on the locale, but generally:

- 1 `SHORT` is numeric, such as 12/13/52 or 3:30pm
- 2 `MEDIUM` is longer, such as Jan. 12, 1952
- 3 `LONG` is longer, such as January 12, 1952 or 3:30:32pm
- 4 `FULL` is completely specified, such as Tuesday, April 12, 1952 AD or 3:30:42pm PST

For more general flexibility, the `DateTimePatternGenerator` can map a custom selection of time and date fields, along with various display styles for those fields, to a locale-appropriate format that can then be set as the format to use by the `DateFormat`.

Producing Relative Date Formats for a Locale

ICU currently provides limited support for formatting dates using a “relative” style, specified using `RELATIVE_SHORT`, `RELATIVE_MEDIUM`, `RELATIVE_LONG` or `RELATIVE_FULL`. As currently implemented, relative date formatting only affects the formatting of dates within a limited range of calendar days before or after the current date, based on the CLDR `<field type="day">/<relative>` data: For example, in English, “Yesterday”, “Today”, and “Tomorrow”. Within this range, the specific relative style currently makes no difference. Outside of this range, relative dates are formatted using the corresponding non-relative style (`SHORT`, `MEDIUM`, etc.). Relative time styles are not currently supported, and behave just like the corresponding non-relative style.

Setting Time Zones

You can set the time zone on the format. If you want more control over the format or parsing, cast the `DateFormat` you get from the factory methods to a `SimpleDateFormat`. This works for the majority of countries.



Note: Remember to check `getDynamicClassID()` before carrying out the cast.

Working with Positions

You can also use forms of the parse and format methods with `ParsePosition` and `FieldPosition` to enable you to:

- 1 Progressively parse through pieces of a string.
- 2 Align any particular field, or find out where it is for selection on the screen.

SimpleDateFormat

`SimpleDateFormat` is a concrete class used for formatting and parsing dates in a language-independent manner. It allows for formatting, parsing, and normalization. It formats or parses a date or time, which is the standard milliseconds since 24:00 GMT, Jan. 1, 1970.

`SimpleDateFormat` is the only built-in implementation of `DateFormat`. It provides a programmable interface that can be used to produce formatted dates and times in a wide variety of formats. The formats include almost all of the most common ones.

Create a date-time formatter using the following methods rather than constructing an instance of `SimpleDateFormat`. In this way, the program is guaranteed to get an appropriate formatting pattern of the locale.

- 1 `DateFormat.getInstance()`
- 2 `getDateInstance()`
- 3 `getDateTimeInstance()`

If you need a more unusual pattern, construct a `SimpleDateFormat` directly and give it an appropriate pattern.

Date/Time Format Syntax

A date pattern is a string of characters, where specific strings of characters are replaced with date and time data from a calendar when formatting or used to generate data for a calendar when parsing.

The Date Field Symbol Table below contains the characters used in patterns to show the appropriate formats for a given locale, such as `yyyy` for the year. Characters may be used multiple times. For example, if `y` is used for the year, `"yy"` might produce "99", whereas `"yyyy"` produces "1999". For most numerical fields, the number of characters specifies the field width. For example, if `h` is the hour, `"h"` might produce "5", but `"hh"` produces "05". For some characters, the count specifies whether an abbreviated or full form should be used, but may have other choices, as given below.

Two single quotes represents a literal single quote, either inside or outside single quotes. Text within single quotes is not interpreted in any way (except for two adjacent single quotes). Otherwise all ASCII letter from **a** to **z** and **A** to **Z** are reserved as syntax characters, and require quoting if they are to represent literal characters. In addition, certain ASCII punctuation characters may become variable in the future (eg `'` being interpreted as the time separator and `'` as a date separator, and replaced by respective locale-sensitive characters in display).

“Stand-alone” values refer to those designed to stand on their own independently, as opposed to being with other formatted values. “2nd quarter” would use the wide stand-alone format `"qqqq"`, whereas “2nd quarter 2007” would use the regular format `"QQQQ yyyy"`. For more information about format and stand-alone forms, see [CLDR Calendar Elements](#).

The pattern characters used in the Date Field Symbol Table are defined by CLDR; for more information see [CLDR Date Field Symbol Table](#).

Note that the examples may not reflect current CLDR data.


DATE FIELD SYMBOL TABLE

Symbol	Meaning	Pattern	Example Output
G	era designator	G, GG, or GGG GGGG GGGGG	AD Anno Domini A
y	year	yy y or yyyy	96 1996
Y	year of “Week of Year”	Y	1997
u	extended year	u	4601
U	cyclic year name, as in Chinese lunar calendar	U	甲子
r	related Gregorian year	r	1996
Q	quarter	Q QQ QQQ QQQQ QQQQQ	2 02 Q2 2nd quarter 2

Symbol	Meaning	Pattern	Example Output
q	stand-alone quarter	q qq qqq qqqq qqqqq	2 02 Q2 2nd quarter 2
M	month in year	M MM MMM MMMM MMMMM	9 09 Sep September S
L	stand-alone month in year	L LL LLL LLLL LLLLL	9 09 Sep September S
w	week of year	w ww	27 27
W	week of month	W	2
d	day in month	d dd	2 02
D	day of year	D	189
F	day of week in month	F	2 (2nd Wed in July)
g	modified julian day	g	2451334
E	day of week	E, EE, or EEE EEEE EEEEE EEEEEE	Tue Tuesday T Tu
e	local day of week example: if Monday is 1st day, Tuesday is 2nd)	e OR ee eee eeee eeeee eeeeee	2 Tue Tuesday T Tu

Symbol	Meaning	Pattern	Example Output
c	stand-alone local day of week	c OR cc ccc cccc ccccc cccccc	2 Tue Tuesday T Tu
a	AM or PM	a, aa, OR aaa aaaa aaaaa	PM [abbrev] PM [wide] p
b	am, pm, noon, midnight	b, bb, OR bbb bbbb bbbbb	mid. midnight md
B	flexible day periods	B, BB, OR BBB BBBB BBBBB	at night [abbrev] at night [wide] at night [narrow]
h	hour in am/pm (1~12)	h hh	7 07
H	hour in day (0~23)	H HH	0 00
k	hour in day (1~24)	k kk	24 24
K	hour in am/pm (0~11)	K KK	0 00
m	minute in hour	m mm	4 04
s	second in minute	s ss	5 05
S	fractional second - truncates (like other time fields) to the count of letters when formatting. Appends zeros if more than 3 letters specified. Truncates at three significant digits when parsing.	S SS SSS SSSS	2 23 235 2350
A	milliseconds in day	A	61201235

Symbol	Meaning	Pattern	Example Output
<code>z</code>	Time Zone: specific non-location	<code>z</code> , <code>zz</code> , or <code>zzz</code> <code>zzzz</code>	PDT Pacific Day light Time
<code>Z</code>	Time Zone: ISO8601 basic hms? / RFC 822 Time Zone: long localized GMT (=OOOO) Time Zone: ISO8601 extended hms? (=XXXXX)	<code>z</code> , <code>zz</code> , or <code>zzz</code> <code>zzzz</code> <code>zzzzz</code>	-0800 GMT-08:00 -08:00, -07:52:58, Z
<code>O</code>	Time Zone: short localized GMT Time Zone: long localized GMT (=ZZZZ)	<code>O</code> <code>OOOO</code>	GMT-8 GMT-08:00
<code>v</code>	Time Zone: generic non-location (falls back first to VVVV)	<code>v</code> <code>vvvv</code>	PT Pacific Time or Los Angeles Time
<code>V</code>	Time Zone: short time zone ID Time Zone: long time zone ID Time Zone: time zone exemplar city Time Zone: generic location (falls back to OOOO)	<code>V</code> <code>VV</code> <code>VVV</code> <code>VVVV</code>	uslax America/Los_Angeles Los Angeles Los Angeles Time
<code>x</code>	Time Zone: ISO8601 basic hm?, with Z for 0 Time Zone: ISO8601 basic hm, with Z Time Zone: ISO8601 extended hm, with Z Time Zone: ISO8601 basic hms?, with Z Time Zone: ISO8601 extended hms?, with Z	<code>x</code> <code>xx</code> <code>xxx</code> <code>xxxx</code> <code>xxxxx</code>	-08, +0530, Z -0800, Z -08:00, Z -0800, -075258, Z -08:00, -07:52:58, Z
<code>x</code>	Time Zone: ISO8601 basic hm?, without Z for 0 Time Zone: ISO8601 basic hm, without Z Time Zone: ISO8601 extended hm, without Z Time Zone: ISO8601 basic hms?, without Z Time Zone: ISO8601 extended hms?, without Z	<code>x</code> <code>xx</code> <code>xxx</code> <code>xxxx</code> <code>xxxxx</code>	-08, +0530 -0800 -08:00 -0800, -075258 -08:00, -07:52:58
<code>'</code>	escape for text	<code>'</code>	(nothing)
<code>' '</code>	two single quotes produce one	<code>' '</code>	,


 **Note:** Any characters in the pattern that are not in the ranges of `[‘a’..‘z’]` and `[‘A’..‘Z’]` will be treated as quoted text. For instance, characters like `‘:’`, `‘.’`, `‘ ’`, `‘#’` and `‘@’` will appear in the resulting time text even they are not enclosed within single quotes. The single quote is used to ‘escape’ letters. Two single quotes in a row, whether inside or outside a quoted sequence, represent a ‘real’ single quote.

 **Note:** A pattern containing any invalid pattern letter results in a failing `UErrorCode` result during formatting or parsing.

Format Pattern	Result
<code>"yyyy.MM.dd G 'at' HH:mm:ss zzz"</code>	"1996.07.10 AD at 15:08:56 PDT"
<code>"EEE, MMM d, 'yy"</code>	"Wed, July 10, '96"
<code>"h:mm a"</code>	"12:08 PM"
<code>"hh 'o''clock' a, zzzz"</code>	"12 o'clock PM, Pacific Daylight Time"
<code>"K:mm a, z"</code>	"0:00 PM, PST"
<code>"yyyyy.MMMM.dd GGG hh:mm aaa"</code>	"01996.July.10 AD 12:08 PM"

Time Zone Display Names

ICU supports time zone display names defined by the LDML ([Unicode Locale Data Markup Language](#)) specification. Since ICU 3.8, the vast majority of localized time zone names are no longer associated with individual time zones. Instead, a set of localized time zone names are associated with a *metazone* and one or more individual time zones are mapped to the same *metazone*. For example, *metazone* "America_Pacific" has its own display name data such as "PST" "PDT" "PT" "Pacific Standard Time" "Pacific Daylight Time" "Pacific Time" and these names are shared by multiple individual time zones "America/Los_Angeles", "America/Vancouver", "America/Tijuana" and so on. The mapping from individual time zone to *metazone* is not a simple 1-to-1 mapping, but it changes time to time. For example, time zone "America/Indiana/Tell_City" uses name data from *metazone* "America_Eastern" until April 2, 2006, but it changes to *metazone* "America_Central" after the date. So the display name used for "America/Indiana/Tell_City" before the date (e.g. "Eastern Time") differs from the one after the date (e.g. "Central Time").

 **Note:** Prior to ICU 3.8, a localized time zone name (except GMT format) and a time zone ID was in 1-to-1 relationship always. Therefore, a time zone name produced by `DateFormat` can be parsed back to the original time zone. This assumption no longer applies to ICU 3.8 and later releases for all time zone format types. If you program requires to roundtrip specific time zone ID, you must use the generic location format `vvv` explained below.

There are several different display name types available in the LDML specification.

TIME_ZONE_DISPLAY_NAME_TYPES

Type	Description	Examples
Generic non-location	Reflects wall time, suited for displaying recurring events, meetings or anywhere people do not want to be overly specific. Available in two length options – long and short.	Pacific Time PT
Generic partial location	Reflects wall time, used as a fallback format when the generic non-location format is not specific enough. A generic partial location name is constructed from a generic non-location name with a location name. For example, “PT” is shared by multiple time zones via metazone “America_Pacific”. When GMT offset in the time zone at the given time differs from the preferred time zone of the metazone for the locale, location name is appended to generic non-location name to distinguish the time zone from the preferred zone. Available in two length options – long and short.	Pacific Time (Canada) PT (Yellowknife)
Generic location	Reflects wall time, suited for populating choice list for time zones. If the time zone is the single time zone available in the region (country), the generic location name is constructed with the region name. Otherwise, the name is constructed from the region name and the city name. Unlike other format types, this name is unique per time zone.	United States (Los Angeles) Time Italy Time
Specific non-location	Reflects a specific standard or daylight time. Available in two length options – long and short.	Pacific Standard Time PDT
Localized GMT	A constant, specific offset from GMT in a localized form.	GMT-08:00
RFC822 GMT	A constant, specific offset from GMT in a locale insensitive format.	-0800

Each format type in the above table is used as a primary type or a fallback in `SimpleDateFormat`. The table below explains how ICU time zone format pattern work and its characteristics.

TIME_ZONE_PATTERN_USAGE

Pattern	Behavior	Round-trip time at daylight transitions(*)	Round-trip Time Zone	Suggested Usage

Pattern	Behavior	Round-trip time at daylight transitions(*)	Round-trip Time Zone	Suggested Usage
<code>z</code> , <code>zz</code> , <code>zzz</code>	Short specific non-location format (e.g. "PST"). If the localized data is not available or the short abbreviation is not commonly used for the locale, localized GMT format is used (e.g. "GMT-08:00").	yes	no	For displaying a time with a user friendly time zone name.
<code>zzzz</code>	Long specific non-location format (e.g. "Pacific Standard Time"). If the localized data is not available, localized GMT format is used (e.g. "GMT-08:00").	yes	no	Same as <code>z</code> , but longer format.
<code>V</code>	Short generic non-location format (e.g. "PT"). If the localized data is not available or the short abbreviation is not commonly used for the locale, generic location format (e.g. "United States(Los Angeles) Time") is used. If the localized data comes from metazone and the GMT offset at the given time in the specified time zone differs from the preferred time zone of the metazone for the locale, generic partial location format (e.g. "PT (Canada)") is used.	no	no	For displaying a recurring wall time (e.g. events, meetings) or anywhere people do not want to be overly specific.

Pattern	Behavior	Round-trip time at daylight transitions(*)	Round-trip Time Zone	Suggested Usage
<code>vvvv</code>	Long generic non-location format (e.g. "Pacific Time"). If the localized data is not available, generic location format (e.g. "United States(Los Angeles) Time") is used.	no	no	Same as <code>v</code> , but longer format.
<code>v</code>	Same as <code>z</code> , except using the short abbreviation even it is not commonly used for the locale.	yes	no	Same as <code>z</code> .
<code>vvv</code>	Generic location format (e.g. "United States (Los Angeles) Time").	no	yes	For populating a choice list for time zones, because it supports 1-to-1 name/zone ID mapping and is more uniform than other text formats. Also, this is only the pattern supporting time zone round-trip. If your program requires to preserve the original time zone information, use this pattern.

Pattern	Behavior	Round-trip time at daylight transitions(*)	Round-trip Time Zone	Suggested Usage
<code>z</code> , <code>zz</code> , <code>zzz</code>	Localized GMT format (e.g. "GMT-08:00").	yes	no	For displaying a time in UI in a uniformed manner.
<code>zzzz</code>	RFC822 GMT format (e.g. "-0800").	yes	no	For formatting a time for non- user-facing data.

* At a transition from **daylight** saving time to standard time, there is a wall time interval occurs twice.

DateTimePatternGenerator

The `DateTimePatternGenerator` class provides a way to map a request for a set of date/time fields, along with their width, to a locale-appropriate format pattern. The request is in the form of a "skeleton" which just contains pattern letters for the desired fields using the representation for the desired width. In a skeleton, anything other than a pattern letter is ignored, field order is insignificant, and there are two special additional pattern letters that may be used: `j` requests the preferred hour-cycle type for the locale (it gets mapped to one of `H`, `h`, `k`, or `K`); `J` is similar but requests no AM/PM marker even if the locale's preferred hour-cycle type is `h` or `k`.

For example, a skeleton of `"MMMMdjmm"` might result in the following format patterns for different locales:

locale	format pattern for skeleton <code>"MMMMdjmm"</code>	example
en_US	<code>"MMMM d 'at' h:mm a"</code>	"April 2 at 5:00 PM"
es_ES	<code>"d 'de' MMMM, H:mm"</code>	"2 de abril, 17:00"
ja_JP	<code>"M月d日 H:mm"</code>	"4月2日 17:00"

The most important `DateTimePatternGenerator` methods are the varieties of `getBestPattern`.

Note that the fields in the format pattern may be adjusted as appropriate for the locale and may not exactly match those in the skeleton. For example:

- In Russian (locale "ru"), the skeleton "yMMMM" will produce the format pattern "LLLL y" (or "LLLL y 'r' .") since a month name without a **day** number must be in nominative form, as indicated by LLLL.
- When using the Japanese calendar in the Japanese locale (locale "ja@calendar=japanese"), the skeleton "yMMMd" will produce the format pattern "Gy年M月d日" since the era must always be shown with the year in the Japanese calendar.

DateFormatSymbols

`DateFormatSymbols` is a public class for encapsulating localizable date-time formatting data, including time zone data. `DateFormatSymbols` is used by `DateFormat` and `SimpleDateFormat`.

`DateFormatSymbols` specifies the exact character strings to use for various parts of a date or time. For example, the names of the months and **days** of the week, the strings for AM and PM and the **day** of the week considered to be the first **day** of the week (used in drawing calendar grids) are controlled by `DateFormatSymbols`.

Create a date-time formatter using the `createTimeInstance`, `createDateInstance`, or `createDateTimeInstance` methods in `DateFormat`. Each of these methods can return a date/time formatter initialized with a default format pattern, along with the date-time formatting data for a given or default locale. After a formatter is created, modify the format pattern using `applyPattern`.

If you want to create a date-time formatter with a particular format pattern and locale, use one of the `SimpleDateFormat` constructors:

```
UnicodeString aPattern("GyyyyMMddHHmmssSSZ", "");
new SimpleDateFormat(aPattern, new DateFormatSymbols(Locale::getUS()))
```

This loads the appropriate date-time formatting data from the locale.

Programming Examples

See [date and time formatting examples](#).

TABLE OF CONTENTS

- [Date and Time Formatting Examples](#)

