Package 'bigD'

October 12, 2022

Type Package

Title Flexibly Format Dates and Times to a Given Locale

Version 0.2.0

Description Format dates and times flexibly and to whichever locales make sense. Parses dates, times, and date-times in various formats (including string-based ISO 8601 constructions). The formatting syntax gives the user many options for formatting the date and time output in a precise manner. Time zones in the input can be expressed in multiple ways and there are many options for formatting time zones in the output as well. Several of the provided helper functions allow for automatic generation of locale-aware formatting patterns based on date/time skeleton formats and standardized date/time formats with varying specificity.

License MIT + file LICENSE

URL https://github.com/rich-iannone/bigD

BugReports https://github.com/rich-iannone/bigD/issues

Depends R (>= 3.2.0)

Encoding UTF-8

RoxygenNote 7.2.1

Suggests covr, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author Richard Iannone [aut, cre] (<https://orcid.org/0000-0003-3925-190X>)

Maintainer Richard Iannone <riannone@me.com>

Repository CRAN

Date/Publication 2022-09-05 08:30:08 UTC

R topics documented:

`fdt` *Format a datetime with a formatting string*

Description

With `fdt()`, we can format datetime values with the greatest of ease, and, with great power. There is a lot of leniency in what types of input date/time/datetime values can be passed in. The formatting string allows for a huge array of possibilities when formatting. Not only that, we can set a `locale` value and get the formatted values localized in the language/region of choice. There's plenty of ways to represent time zone information, and this goes along with the option to enrich the input values with a precise time zone identifier (like `"America/Los_Angeles"`). The choices are ample here, with the goal being a comprehensiveness and ease-of-use in date/time formatting.

Usage

```
fdt(input, format = NULL, use_tz = NULL, locale = NULL)
```

Arguments

`input`	A vector of date, time, or datetime values. Several representations are acceptable here including strings, `Date` objects, or `POSIXct` objects. Refer to the *Valid Input Values* section for more information.
`format`	The formatting string to apply to all `input` values. If not provided, the inputs will be formatted to ISO 8601 datetime strings. The *Date/Time Format Syntax* section has detailed information on how to create a formatting string.
`use_tz`	A tzid (e.g., `"America/New_York"`) time-zone designation for precise formatting of related outputs. This overrides any time zone information available in `character`-based input values and is applied to all vector components.

`locale` The output locale to use for formatting the input value according to the specified locale's rules. Example locale names include `"en"` for English (United States) and `"es-EC"` for Spanish (Ecuador). If a locale isn't provided the `"en"` locale will be used. The fdt_locales_vec vector contains the valid locales and fdt_locales_lst list provides an easy way to obtain a valid locale.

Value

A character vector of formatted dates, times, or datetimes.

Valid Input Values

The `input` argument of the `fdt()` function allows for some flexibility on what can be passed in. This section describes the kinds of inputs that are understandable by `fdt()`. A vector of strings is allowed, as are vectors of `Date` or `POSIXct` values.

If using strings, a good option is to use those that adhere to the ISO 8601:2004 standard. For a datetime this can be of the form `YYYY-MM-DDThh:mm:ss.s<TZD>`. With this, `YYYY-MM-DD` corresponds to the date, the literal `"T"` is optional, `hh:mm:ss` is the time (where seconds, `ss`, is optional as are `.s` for fractional seconds), and `<TZD>` refers to an optional time-zone designation (more on time zones in the next paragraph). You can provide just the date part, and this assumes midnight as an implicit time. It's also possible to provide just the time part, and this internally assembles a datetime that uses the current date. When formatting standalone dates or times, you'll probably just format the explicit parts but `fdt()` won't error if you format the complementary parts.

The time zone designation on string-based datetimes is completely optional. If not provided then `"UTC"` is assumed. If you do want to supply time zone information, it can be given as an offset value with the following constructions:

- `<time>Z`
- `<time>(+/-)hh:mm`
- `<time>(+/-)hhmm`
- `<time>(+/-)hh`

The first, `<time>Z`, is zone designator for the zero UTC offset; it's equivalent to `"+00:00"`. The next two are formats for providing the time offsets from UTC with hours and minutes fields. Examples are `"-05:00"` (New York, standard time), `"+0200"` (Cairo), and `"+05:30"` (Mumbai). Note that the colon is optional but leading zeros to maintain two-digit widths are essential. The final format, `<time>(+/-)hh`, omits the minutes field and as so many offsets have `"00"` minutes values, this can be convenient.

We can also supply an Olson/IANA-style time zone identifier (tzid) in parentheses within the string, or, as a value supplied to `use_tz` (should a tzid apply to all date/time/datetime values in the `input` vector). By extension, this would use the form: `YYYY-MM-DDThh:mm:ss.s<TZD>(<tzid>)`. Both a `<TZD>` (UTC offset value) and a `<tzid>` shouldn't really be used together but if that occurs the `<tzid>` overrides the UTC offset. Here are some examples:

- `"2018-07-04T22:05(America/Vancouver)"` (preferable)
- `"2018-07-04T22:05-0800(America/Vancouver)"` (redundant, but still okay)

A tzid contains much more information about the time zone than a UTC offset value since it is tied to some geographical location and the timing of Standard Time (STD) and Daylight Saving Time

(DST) is known. In essence we can derive UTC offset values from a tzid and also a host of other identifiers (time zone names, their abbreviations, etc.). Here are some examples of valid tzid values that can be used:

- `"America/Jamaica"` (the official time in Jamaica, or, `"Jamaica Time"`)
- `"Australia/Perth"` (`"+08:00"` year round in Western Australia)
- `"Europe/Dublin"` (IST/GMT time: `"+01:00"`/`"+00:00"`)

The tz database (a compilation of information about the world's time zones) consists of canonical zone names (those that are primary and preferred) and alternative names (less preferred in modern usage, and was either discarded or more commonly replaced by a canonical zone name). The `fdt()` function can handle both types and what occurs is that non-canonical tzid values are internally mapped onto canonical zone names. Here's a few examples:

- `"Africa/Luanda"` (in Angola) maps to `"Africa/Lagos"`
- `"America/Indianapolis"` maps to `"America/Indiana/Indianapolis"`
- `"Asia/Calcutta"` maps to `"Asia/Kolkata"`
- `"Pacific/Midway"` maps to `"Pacific/Pago_Pago"`
- `"Egypt"` maps to `"Africa/Cairo"`

For the most part, the Olson-format tzid follows the form `"{region}/{city}"` where the region is usually a continent, the city is considered an 'exemplar city', and the exemplar city itself belongs in a country.

Date/Time Format Syntax

A formatting pattern as used in **bigD** consists of a string of characters, where certain strings are replaced with date and time data that are derived from the parsed input.

The characters used in patterns are tabulated below to show which specific strings produce which outputs (e.g., `"y"` for the year). A common pattern is characters that are used consecutively to produce variations on a date, time, or timezone output. Say that the year in the input is 2015. If using `"yy"` you'll get `"15"` but with `"yyyy"` the output becomes `"1999"`. There's a whole lot of this, so the following subsections try to illustrate as best as possible what each string will produce. All of the examples will use this string-based datetime input unless otherwise indicated:

`"2018-07-04T22:05:09.2358(America/Vancouver)"`

The Era Designator (big G):

Formatting String	Output
`"G"`, `"GG"`, or `"GGG"`	`"AD"`
`"GGGG"`	`"Anno Domini"`
`"GGGGG"`	`"A"`

Year (little y):

Formatting String	Output
`"y"`	`"2018"`

"yy"	"18"
"yyy"	"2018"
"yyyy"	"2018"
"yyyyy"	"02018"
"yyyyyy"	"002018"
"yyyyyyy"	"0002018"
"yyyyyyyy"	"00002018"
"yyyyyyyyy"	"000002018"

Year in the Week in Year Calendar (big Y):

This is the year in 'Week of Year' based calendars in which the year transition occurs on a week boundary. This may differ from calendar year 'y' near a year transition. This numeric year designation is used in conjunction with pattern character 'w' in the ISO year-week calendar as defined by ISO 8601.

Formatting String	Output
"Y"	"2018"
"YY"	"18"
"YYY"	"2018"
"YYYY"	"2018"
"YYYYY"	"02018"
"YYYYYY"	"002018"
"YYYYYYY"	"0002018"
"YYYYYYYY"	"00002018"
"YYYYYYYYY"	"000002018"

Quarter of the Year: formatting ver. (big Q):

Formatting String	Output
"Q"	"3"
"QQ"	"03"
"QQQ"	"Q3"
"QQQQ"	"3rd quarter"
"QQQQQ"	"3"

Quarter of the Year: standalone ver. (little q):

Formatting String	Output
"q"	"3"
"qq"	"03"
"qqq"	"Q3"
"qqqq"	"3rd quarter"
"qqqqq"	"3"

Month: formatting ver. (big M):

Formatting String	Output
"M"	"7"
"MM"	"07"
"MMM"	"Jul"
"MMMM"	"July"
"MMMMM"	"J"

Month: standalone ver. (big L):

Formatting String	Output
"L"	"7"
"LL"	"07"
"LLL"	"Jul"
"LLLL"	"July"
"LLLLL"	"J"

Week of Year (little w):

Formatting String	Output
"w"	"27" (minimal digits)
"ww"	"27" (two digits, zero padded)

Week of Month (big W):

Formatting String	Output
"W"	"1"

Day of Month (little d):

Formatting String	Output
"d"	"4" (minimal digits)
"dd"	"04" (two digits, zero padded)

Day of Year (big D):

Formatting String	Output
"D"	"185"
"DD"	"185" (zero padded to min-width of 2)
"DDD"	"185" (zero padded to min-width of 3)

Day of Week in Month (big F):

Formatting String	Output
"F"	"1"

Modified Julian Day (little g):

Formatting String	Output
"g"	"58303"
"gg"	"58303"
"ggg"	"58303"
"gggg"	"58303"
"ggggg"	"58303"
"gggggg"	"058303"
"ggggggg"	"0058303"
"gggggggg"	"00058303"
"ggggggggg"	"000058303"

Day of Week Name (big E):

Formatting String	Output
"E"	"Wed"
"EE"	"Wed"
"EEE"	"Wed"
"EEEE"	"Wednesday"
"EEEEE"	"W"
"EEEEEE"	"We"

AM/PM Period of Day (little a):

Formatting String	Output	Note
"a", "aa", or "aaa"	"PM"	Abbreviated
"aaaa"	"PM"	Wide
"aaaaa"	"p"	Narrow

AM/PM Period of Day Plus Noon and Midnight (little b):

(a) input_midnight: "2020-05-05T00:00:00"

(b) input_noon: "2020-05-05T12:00:00"

Formatting String	Output	Note
"b", "bb", or "bbb"	(a) "midnight"	Abbreviated
"" "" ""	(b) "noon"	"" "" ""
"bbbb"	(a) "midnight"	Wide
"" "" ""	(b) "noon"	"" "" ""
"bbbbb"	(a) "mi"	Narrow
"" "" ""	(b) "n"	"" "" ""

Flexible Day Periods (big B):

(a) `input_morning`: `"2020-05-05T00:08:30"`

(b) `input_afternoon`: `"2020-05-05T14:00:00"`

Formatting String	Output	Note
`"B"`, `"BB"`, or `"BBB"`	(a) `"in the morning"`	Abbreviated
`"" "" ""`	(b) `"in the afternoon"`	`"" "" ""`
`"BBBB"`	(a) `"in the morning"`	Wide
`"" "" ""`	(b) `"in the afternoon"`	`"" "" ""`
`"BBBBB"`	(a) `"in the morning"`	Narrow
`"" "" ""`	(b) `"in the afternoon"`	`"" "" ""`

Hour 1-12 (little h):

Using: `"2015-08-01T08:35:09"`

Formatting String	Output	Note
`"h"`	`"8"`	Numeric, minimum digits
`"hh"`	`"08"`	Numeric, 2 digits (zero padded)

Hour 0-23 (big H):

Using: `"2015-08-01T08:35:09"`

Formatting String	Output	Note
`"H"`	`"8"`	Numeric, minimum digits
`"HH"`	`"08"`	Numeric, 2 digits (zero padded)

Hour 0-11 (big K):

Using: `"2015-08-01T08:35:09"`

Formatting String	Output	Note
`"K"`	`"7"`	Numeric, minimum digits
`"KK"`	`"07"`	Numeric, 2 digits (zero padded)

Hour 1-24 (little k):

Using: `"2015-08-01T08:35:09"`

Formatting String	Output	Note
`"k"`	`"9"`	Numeric, minimum digits
`"kk"`	`"09"`	Numeric, 2 digits (zero padded)

Minute (little m):

Formatting String	Output	Note
`"m"`	`"5"`	Numeric, minimum digits
`"mm"`	`"06"`	Numeric, 2 digits (zero padded)

Second (little s):

Formatting String	Output	Note
`"s"`	`"9"`	Numeric, minimum digits
`"ss"`	`"09"`	Numeric, 2 digits (zero padded)

Fractional Second (big S):

Formatting String	Output
`"S"`	`"2"`
`"SS"`	`"23"`
`"SSS"`	`"235"`
`"SSSS"`	`"2350"`
`"SSSSS"`	`"23500"`
`"SSSSSS"`	`"235000"`
`"SSSSSSS"`	`"2350000"`
`"SSSSSSSS"`	`"23500000"`
`"SSSSSSSSS"`	`"235000000"`

Milliseconds Elapsed in Day (big A):
Using: `"2011-07-27T00:07:19.7223"`

Formatting String	Output
`"A"`	`"439722"`
`"AA"`	`"439722"`
`"AAA"`	`"439722"`
`"AAAA"`	`"439722"`
`"AAAAA"`	`"439722"`
`"AAAAAA"`	`"439722"`
`"AAAAAAA"`	`"0439722"`
`"AAAAAAAA"`	`"00439722"`
`"AAAAAAAAA"`	`"000439722"`

TZ // Short and Long Specific non-Location Format (little z):

Formatting String	Output	Note
`"z"`, `"zz"`, or `"zzz"`	`"PDT"`	Short Specific
`"zzzz"`	`"Pacific Daylight Time"`	Long Specific

TZ // Short and Long Specific non-Location Formats (big Z):

Formatting String	Output	Note
`"Z"`, `"ZZ"`, or `"ZZZ"`	`"-0700"`	ISO 8601 basic format
`"ZZZZ"`	`"GMT-7:00"`	Long localized GMT format

`"ZZZZZ"`	`"-07:00"`	ISO 8601 extended format

TZ // Short and Long Localized GMT Formats (big O):

Formatting String	Output	Note
`"O"`	`"GMT-7"`	Short localized GMT format
`"OOOO"`	`"GMT-07:00"`	Long localized GMT format

TZ // Short and Long Localized GMT Formats (little v):

Formatting String	Output	Note
`"v"`	`"PT"`	Short generic non-location format
`"vvvv"`	`"Pacific Time"`	Long generic non-location format

TZ // Short Time Zone IDs and Exemplar City Formats (big V):

Formatting String	Output	Note
`"V"`	`"cavan"`	Short time zone ID
`"VV"`	`"America/Vancouver"`	Long time zone ID
`"VVV"`	`"Vancouver"`	The tz exemplar city
`"VVVV"`	`"Vancouver Time"`	Generic location format

TZ // ISO 8601 Formats with Z for +0000 (big X):

Formatting String	Output	Note
`"X"`	`"-07"`	ISO 8601 basic format (h; optional m)
`"XX"`	`"-0700"`	ISO 8601 basic format (h & m)
`"XXX"`	`"-07:00"`	ISO 8601 extended format (h & m)
`"XXXX"`	`"-0700"`	ISO 8601 basic format (h & m, optional s)
`"XXXXX"`	`"-07:00"`	ISO 8601 extended format (h & m, optional s)

TZ // ISO 8601 Formats (no use of Z for +0000) (little x):

Formatting String	Output	Note
`"x"`	`"-07"`	ISO 8601 basic format (h; optional m)
`"xx"`	`"-0700"`	ISO 8601 basic format (h & m)
`"xxx"`	`"-07:00"`	ISO 8601 extended format (h & m)
`"xxxx"`	`"-0700"`	ISO 8601 basic format (h & m, optional s)
`"xxxxx"`	`"-07:00"`	ISO 8601 extended format (h & m, optional s)

Examples

Basics with `input` datetimes, formatting strings, and localization:

With an input datetime of `"2018-07-04 22:05"` supplied as a string, we can format to get just a date with the full year first, the month abbreviation second, and the day of the month last (separated by hyphens):

```
fdt(
  input = "2018-07-04 22:05",
  format = "y-MMM-dd"
)
```

```
#> [1] "2018-Jul-04"
```

There are sometimes many options for each time part. Instead of using `"y-MMM-dd"`, let's try a variation on that with `"yy-MMMM-d"`:

```
fdt(
  input = "2018-07-04 22:05",
  format = "yy-MMMM-d"
)
```

```
#> [1] "18-July-4"
```

The output is localizable and so elements will be translated when supplying the appropriate locale code. Let's use `locale = es` to get the month written in Spanish:

```
fdt(
  input = "2018-07-04 22:05",
  format = "yy-MMMM-d",
  locale = "es"
)
```

```
#> [1] "18-julio-4"
```

`POSIXct` or `POSIXlt` datetimes can serve as an `input` to `fdt()`. Let's create a single datetime value where the timezone is set as `"Asia/Tokyo"`.

```
fdt(
  input = lubridate::ymd_hms("2020-03-15 19:09:12", tz = "Asia/Tokyo"),
  format = "EEEE, MMMM d, y 'at' h:mm:ss B (VVVV)"
)
```

```
#> [1] "Sunday, March 15, 2020 at 7:09:12 in the evening (Tokyo Time)"
```

If you're going minimal, it's possible to supply an input datetime string without a `format` directive. What this gives us is formatted datetime output that conforms to ISO 8601. Note that the implied time zone is UTC.

```
fdt(input = "2018-07-04 22:05")
```

```
#> [1] "2018-07-04T22:05:00Z"
```

Using locales stored in the fdt_locales_lst list:

The fdt_locales_lst object is provided in **bigD** to make it easier to choose one of supported locales. You can avoid typing errors and every element of the list is meant to work. For example, we can use the `"it"` locale by accessing it from fdt_locales_lst (autocomplete makes this relatively simple).

```
fdt(
  input = "2018-07-04 22:05",
  format = "yy-MMMM-d",
  locale = fdt_locales_lst$it
)
```

```
#> [1] "18-luglio-4"
```

Omission of date or time in `input`:

You don't have to supply a full datetime to `input`. Just supplying the date portion implies midnight (and is just fine if you're only going to present the date anyway).

```
fdt(input = "2018-07-04")
```

```
#> [1] "2018-07-04T00:00:00Z"
```

If you omit the date and just supply a time, `fdt()` will correctly parse this. The current date on the user system will be used because we need to create some sort of datetime value internally. Again, this is alright if you just intend to present a formatted time value.

```
fdt(input = "22:05")
```

```
#> [1] "2022-08-16T22:05:00Z"
```

To see all of the supported locales, we can look at the vector supplied by the `fdt_locales_vec()` function.

Using standardized forms with the `standard_*()` helper functions:

With an input datetime of `"2018-07-04 22:05(America/Vancouver)"`, we can format the date and time in a standardized way with `standard_date_time()` providing the correct formatting string. This function is invoked in the `format` argument of `fdt()`:

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time(type = "full")
)
```

```
#> [1] "Wednesday, July 4, 2018 at 10:05:00 PM Pacific Daylight Time"
```

The locale can be changed and we don't have to worry about the particulars of the formatting string (they are standardized across locales).

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time(type = "full"),
  locale = fdt_locales_lst$nl
)
```

```
#> [1] "woensdag 4 juli 2018 om 22:05:00 Pacific-zomertijd"
```

We can use different `type` values to control the output datetime string. The default is `"short"`.

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time()
)
```

```
#> [1] "7/4/18, 10:05 PM"
```

After that, it's "medium":

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time(type = "medium")
)
```

```
#> [1] "Jul 4, 2018, 10:05:00 PM"
```

The "short" and "medium" types don't display time zone information in the output. Beginning with "long", the tz is shown.

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time(type = "long")
)
```

```
#> [1] "July 4, 2018 at 10:05:00 PM PDT"
```

If you don't include time zone information in the input, the "UTC" time zone will be assumed:

```
fdt(
  input = "2018-07-04 22:05",
  format = standard_date_time(type = "full")
)
```

```
#> [1] "Wednesday, July 4, 2018 at 10:05:00 PM GMT+00:00"
```

Using flexible date and time (12- and 24-hour) formatting:

The **bigD** package supplies a set of lists to allow for flexible date and time formatting (flex_d_lst, flex_t24_lst, and flex_t12_lst). These are useful when you need a particular format that works across all locales. Here's an example that uses the "yMMMEd" flexible date type by accessing it from the flex_d_lst object, yielding a formatted date.

```
fdt(
  input = "2021-01-09 16:32(America/Toronto)",
  format = flex_d_lst$yMMMEd,
)
```

```
#> [1] "Sat, Jan 9, 2021"
```

If we wanted this in a different locale, the locale-specific format pattern behind the flexible date identifier would ensure consistency while moving to that locale.

```
fdt(
  input = "2021-01-09 16:32(America/Toronto)",
  format = flex_d_lst$yMMMEd,
  locale = "fr_CA"
)
```

```
#> [1] "sam. 9 janv. 2021"
```

Formatting as a 12-hour time with the flex_t12_lst list and using the "hms" flexible type:

```
fdt(
  input = "2021-01-09 16:32(America/Toronto)",
  format = flex_t12_lst$hms
)
```

```
#> [1] "4:32:00 PM"
```

The 24-hour variant, flex_t24_lst, has a similar "Hms" flexible type that will give us a 24-hour version of the same clock time:

```
fdt(
  input = "2021-01-09 16:32(America/Toronto)",
  format = flex_t24_lst$Hms
)
```

```
#> [1] "16:32:00"
```

A flexible date and time can be used together by enveloping the two in a list (**bigD** will handle putting the date and time together in a sensible manner).

```
fdt(
  input = "2021-01-09 16:32(America/Toronto)",
  format = list(flex_d_lst$yMMMEd, flex_t24_lst$Hmv)
)
```

```
#> "Sat, Jan 9, 2021, 16:32 ET"
```

`fdt_locales_lst` *A list of all supported locales*

Description

The `fdt_locales_lst` object is a list of all supported locales. This is useful when used with the `fdt()` function as the list can be auto-completed with a locale identifier and this generates valid input for the `locale` argument.

Usage

```
fdt_locales_lst
```

Format

An object of class `list` of length 574.

Value

A list where each element corresponds to a supported locale ID.

Examples

The `fdt_locales_lst` object can be incredibly useful when choosing one of supported locales. You can avoid typing errors and every element of the list is meant to work. In this example, we'll use the `"da"` locale through use of the list.

```
fdt(
  input = "2018-07-04 22:05",
  format = "yy-MMMM-d",
  locale = fdt_locales_lst$da
)
```

```
#> [1] "18-juli-4"
```

`fdt_locales_vec` *Get a vector of all supported locales*

Description

The `fdt_locales_vec()` function produces a vector of all supported locale IDs in the **bigD** package.

Usage

```
fdt_locales_vec()
```

Value

A character vector of supported locale IDs.

Examples

```
# Let's get all the `ar` locales that exist
# in the vector produced by `fdt_locales_vec()`
fdt_locales_vec()[grep("^ar", fdt_locales_vec())]

# Let's get all the locales that pertain to the
# `CH` territory in the vector produced by
# `fdt_locales_vec()`
fdt_locales_vec()[grep("CH", fdt_locales_vec())]
```

first_day_of_week — *Get a named vector of all first-day-of-the-week names for different regions*

Description

The `names_months()` function produces a vector of all short month names used by the **bigD** package.

Usage

```
first_day_of_week()
```

Value

A character vector of short month names.

Examples

```
# Let's get a vector of regions where the
# first day of the week is Saturday
names(first_day_of_week()[first_day_of_week() == "sat"])
```

flex_d_lst — *A list of all flexible date types*

Description

The `flex_d_lst` object is a list of widely supported flexible date types. Flexible date types are classes of date formatting which can be translated across locales. There are 26 flexible date types in `flex_d_lst`.

Usage

```
flex_d_lst
```

Format

An object of class `list` of length 26.

Value

A list where each element corresponds to a classifier for a flexible date type.

Examples

The `flex_d_lst` object can be incredibly useful when you need to get a format for date formatting that works across all locales. You can avoid typing errors by using this list and every flexible date type from this list is guaranteed to work across all supported locales. In this example, we'll use the `"yMMMEd"` flexible date type by accessing it from the `flex_d_lst` object.

```
fdt(
  input = "2018-07-04 22:05",
  format = flex_d_lst$yMMMEd,
  locale = "en"
)
```

```
#> [1] "Wed, Jul 4, 2018"
```

If we wanted this in a different locale, the locale-specific `format` pattern behind the flexible date identifier would ensure consistency while moving to that locale. Let's use the `fdt_locales_lst` object in the same spirit to specify the French (Canada) locale.

```
fdt(
  input = "2018-07-04 22:05",
  format = flex_d_lst$yMMMEd,
  locale = fdt_locales_lst$fr_CA
)
```

```
#> [1] "mer. 4 juill. 2018"
```

---

flex_d_vec *Get a vector of all flexible date types*

Description

The `flex_d_vec()` function produces a vector of all supported flexible date types in the **bigD** package. These types are essentially identifiers for classes of cross-locale date formatting, so, none of these should be used directly in the `format` argument of the `fdt()` function (use the flex_d_lst object for that).

Usage

```
flex_d_vec()
```

Value

A character vector of supported flexible date types.

`flex_t12_lst` *A list of all 12-hour flexible time types*

Description

The `flex_t12_lst` object is a list of the 12-hour flexible time types which are widely supported. Flexible time types are classes of time formatting which can be translated across locales. There are 12 flexible time types of the 12-hour variety in `flex_t12_lst`.

Usage

```
flex_t12_lst
```

Format

An object of class `list` of length 12.

Value

A list where each element corresponds to a classifier for a 12-hour flexible time type.

Examples

The `flex_t12_lst` object can be incredibly useful when you need to get a format for 12-hour time formatting that works across all locales. You can avoid typing errors by using this list and every flexible time type from this list is guaranteed to work across all supported locales. In this example, we'll use the `"Ehms"` flexible time type by accessing it from the `flex_t12_lst` object.

```
fdt(
  input = "2018-07-04 22:05",
  format = flex_t12_lst$Bhms,
  locale = "en"
)
```

```
#> [1] "10:05:00 at night"
```

If we wanted this in a different locale, the locale-specific `format` pattern behind the flexible date identifier would ensure consistency while moving to that locale. Let's use the `fdt_locales_lst` object in the same spirit to specify the German (Austria) locale.

```
fdt(
  input = "2018-07-04 22:05",
  format = flex_t12_lst$Bhms,
  locale = fdt_locales_lst$de_AT
)
```

```
#> [1] "10:05:00 abends"
```

flex_t12_vec *Get a vector of all 12-hour flexible time types*

Description

The `flex_t12_vec()` function produces a vector of all supported flexible 12-hour time types in the **bigD** package. These types are essentially identifiers for classes of cross-locale time formatting, so, none of these should be used directly in the `format` argument of the `fdt()` function (use the flex_t12_lst object for that).

Usage

```
flex_t12_vec()
```

Value

A character vector of supported 12-hour flexible time types.

flex_t24_lst *A list of all 24-hour flexible time types*

Description

The `flex_t24_lst` object is a list of the 24-hour flexible time types which are widely supported. Flexible time types are classes of time formatting which can be translated across locales. There are 8 flexible time types of the 24-hour variety in `flex_t24_lst`.

Usage

```
flex_t24_lst
```

Format

An object of class `list` of length 8.

Value

A list where each element corresponds to a classifier for a 24-hour flexible time type.

Examples

The `flex_t24_lst` object can be incredibly useful when you need to get a format for 24-hour time formatting that works across all locales. You can avoid typing errors by using this list and every flexible time type from this list is guaranteed to work across all supported locales. In this example, we'll use the `"EHms"` flexible time type by accessing it from the `flex_t24_lst` object.

```
fdt(
  input = "2018-07-04 22:05",
  format = flex_t24_lst$EHms,
  locale = "en"
)
```

```
#> [1] "Wed 22:05:00"
```

If we wanted this in a different locale, the locale-specific `format` pattern behind the flexible date identifier would ensure consistency while moving to that locale. Let's use the `fdt_locales_lst` object in the same spirit to specify the German locale.

```
fdt(
  input = "2018-07-04 22:05",
  format = flex_t24_lst$EHms,
  locale = fdt_locales_lst$de
)
```

```
#> [1] "Mi, 22:05:00"
```

`flex_t24_vec` *Get a vector of all 24-hour flexible time types*

Description

The `flex_t24_vec()` function produces a vector of all supported flexible 24-hour time types in the **bigD** package. These types are essentially identifiers for classes of cross-locale time formatting, so, none of these should be used directly in the `format` argument of the `fdt()` function (use the flex_t24_lst object for that).

Usage

```
flex_t24_vec()
```

Value

A character vector of supported 24-hour flexible time types.

names_months *Get a vector of all the short month names*

Description

The `names_months()` function produces a vector of all short month names used by the **bigD** package.

Usage

```
names_months()
```

Value

A character vector of short month names.

Examples

```
# Let's get all the short month names with
# the `names_months()` function
names_months()
```

names_wkdays *Get a vector of all the short weekday names*

Description

The `names_wkdays()` function produces a vector of all short weekday names used by the **bigD** package.

Usage

```
names_wkdays()
```

Value

A character vector of short weekday names.

Examples

```
# Let's get all the short weekday names with
# the `names_wkdays()` function
names_wkdays()
```

`standard_date` *Obtain a standard date format that works across locales*

Description

The `standard_date()` function can be invoked in the `format` argument of the `fdt()` function to help generate a locale-specific formatting string of a certain 'type' of formatted date. The `type` value is a keyword that represents precision and verbosity; the available keywords are `"short"` (the default), `"medium"`, `"long"`, and `"full"`.

Usage

```
standard_date(type = c("short", "medium", "long", "full"))
```

Arguments

`type`	One of four standardized types for the resulting date that range in precision and verbosity. These are `"short"` (the default), `"medium"`, `"long"`, and `"full"`.

Value

A vector of class `date_time_pattern`.

Examples

With an input datetime of `"2018-07-04 22:05(America/Vancouver)"`, we can format as a date in a standardized way with `standard_date()` providing the correct formatting string. This function is invoked in the `format` argument of `fdt()`:

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date(type = "full")
)
```

```
#> [1] "Wednesday, July 4, 2018"
```

The locale can be changed and we don't have to worry about the particulars of the formatting string (they are standardized across locales).

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date(type = "full"),
  locale = fdt_locales_lst$nl
)
```

```
#> [1] "woensdag 4 juli 2018"
```

We can use different type values to control the output date string. The default is "short".

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date()
)
```

```
#> [1] "7/4/18"
```

After that, it's "medium":

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date(type = "medium")
)
```

```
#> [1] "Jul 4, 2018"
```

Then, "long":

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date(type = "long")
)
```

```
#> [1] "July 4, 2018"
```

And finally up to "full", which was demonstrated in the first example.

`standard_date_time` *Obtain a standard datetime format that works across locales*

Description

The standard_date_time() function can be invoked in the format argument of the fdt() function to help generate a locale-specific formatting string of a certain 'type' of formatted datetime. The type value is a keyword that represents precision and verbosity; the available keywords are "short" (the default), "medium", "long", and "full".

Usage

```
standard_date_time(type = c("short", "medium", "long", "full"))
```

Arguments

type	One of four standardized types for the resulting datetime that range in precision and verbosity. These are "short" (the default), "medium", "long", and "full".

Value

A vector of class `date_time_pattern`.

Examples

With an input datetime of `"2018-07-04 22:05(America/Vancouver)"`, we can format the date and time in a standardized way with `standard_date_time()` providing the correct formatting string. This function is invoked in the `format` argument of `fdt()`:

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time(type = "full")
)
```

```
#> [1] "Wednesday, July 4, 2018 at 10:05:00 PM Pacific Daylight Time"
```

The locale can be changed and we don't have to worry about the particulars of the formatting string (they are standardized across locales).

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time(type = "full"),
  locale = fdt_locales_lst$nl
)
```

```
#> [1] "woensdag 4 juli 2018 om 22:05:00 Pacific-zomertijd"
```

We can use different `type` values to control the output datetime string. The default is `"short"`.

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time()
)
```

```
#> [1] "7/4/18, 10:05 PM"
```

After that, it's `"medium"`:

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time(type = "medium")
)
```

```
#> [1] "Jul 4, 2018, 10:05:00 PM"
```

The `"short"` and `"medium"` types don't display time zone information in the output. Beginning with `"long"`, the tz is shown.

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_date_time(type = "long")
)
```

```
#> [1] "July 4, 2018 at 10:05:00 PM PDT"
```

If you don't include time zone information in the input, the "UTC" time zone will be assumed:

```
fdt(
  input = "2018-07-04 22:05",
  format = standard_date_time(type = "full")
)
```

```
#> [1] "Wednesday, July 4, 2018 at 10:05:00 PM GMT+00:00"
```

`standard_time` *Obtain a standard time format that works across locales*

Description

The `standard_time()` function can be invoked in the `format` argument of the `fdt()` function to help generate a locale-specific formatting string of a certain 'type' of formatted time. The `type` value is a keyword that represents precision and verbosity; the available keywords are `"short"` (the default), `"medium"`, `"long"`, and `"full"`.

Usage

```
standard_time(type = c("short", "medium", "long", "full"))
```

Arguments

`type`	One of four standardized types for the resulting time that range in precision and verbosity. These are `"short"` (the default), `"medium"`, `"long"`, and `"full"`.

Value

A vector of class `date_time_pattern`.

Examples

With an input datetime of `"2018-07-04 22:05(America/Vancouver)"`, we can format as a time in a standardized way with `standard_time()` providing the correct formatting string. This function is invoked in the `format` argument of `fdt()`:

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_time(type = "full")
)
```

```
#> [1] "10:05:00 PM Pacific Daylight Time"
```

The locale can be changed and we don't have to worry about the particulars of the formatting string (they are standardized across locales).

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_time(type = "full"),
  locale = fdt_locales_lst$nl
)
```

```
#> [1] "22:05:00 Pacific-zomertijd"
```

We can use different type values to control the output date string. The default is "short".

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_time()
)
```

```
#> [1] "10:05 PM"
```

After that, it's "medium":

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_time(type = "medium")
)
```

```
#> [1] "10:05:00 PM"
```

Then, "long":

```
fdt(
  input = "2018-07-04 22:05(America/Vancouver)",
  format = standard_time(type = "long")
)
```

```
#> [1] "10:05:00 PM PDT"
```

And finally up to "full", which was demonstrated in the first example.

Index