Periods

A library for manipulating time

by John Wiegley <johnw@newartisans.com>

http://www.newartisans.com/software/periods.html

Table of contents goes here.

1 Introduction

Welcome to the PERIODS library. The intention of this code is to provide a convenient set of utilities for manipulating times, distances between times, and both contiguous and discontiguous ranges of time. By combining these facilities in various ways, almost any type of time expression is possible.

Consider you are writing a calendaring application which must support the idea of recurring tasks. Often, people using a calendar have very certain ideas of what type of recurrence they want — but it is not always easy to calculate. Say they are entering their pay days; this could either fall bi-weekly, the second Friday of every month, or every 15 days, but the following Monday if that day falls on a weekend. How is your code to calculate these moments in time, without resorting to contortionist uses of ENCODE-UNIVERSAL-TIME and DECODE-UNIVERSAL-TIME?

By way of a brief introduction to the following sections, here is how each of the above time expressions would be calculated. The first three occurrences of each are shown, starting from Sun, 18 Nov 2007:

Example 1: Bi-weekly (every 14 days)

```
(scan-times @2007-11-18 (duration :days 14))
  ⇒ #Z(@2007-12-02 @2007-12-16 @2007-12-30 …)
```

Example 2: Second Friday of every month

```
(mapping ((time (scan-times (previous-time @2007-11-18
                                           (relative-time :day 1))
                            (duration :months 1))))
  (next-time (next-time time (relative-time :day-of-week 5)
                            :accept-anchor t)
             (relative-time :day-of-week 5)))
  ⇒ #Z(@2007-12-14 @2008-01-11 @2008-02-08 …)
```

To read this snippet briefly: Beginning with the first day of the current month, advance forward one month *ad infinitum*. For each new month, scan forward to the first Friday (accepting the first day if it is Friday, with :ACCEPT-ANCHOR). Then scan to the Friday after that.

Example 3: Every 15 days, or the following Monday if that day is a weekend

```
(mapping ((time (scan-times @2007-11-03 (duration :days 15))))
  (if (falls-on-weekend-p time)
      (next-time time (relative-time :day-of-week 1))
      time))
  ⇒ #Z(@2007-11-19 @2007-12-03 @2007-12-18 …)
```

The starting time for this example was set to 3 Nov 2007, so that the Monday-shifting could be demonstrated. Note how the shift does not disrupt the regular 15-day cycle, it merely causes the paycheck to be delivered one day late in that instance.

Hopefully this gives an idea of the power and flexibility of the PERIODS library, especially if used in combination with the SERIES library[1], since times are naturally expressable as unbounded series supporting lazy calculation.

2 Fixed time

The most basic element of time used by the PERIODS library is the FIXED-TIME structure. At present, this is just a type alias for LOCAL-TIME structures[2], so all of the usually operations possible on LOCAL-TIME's are applicable to a FIXED-TIME. In addition, FIXED-TIME's may be constructed using a function of the same name, which allows for quickly creating time structures anchored in the current year.

fixed-time **&key :year :month :day :hour :minute :second :millisecond** [*Function*]

Create a FIXED-TIME structure using details taken from the current moment. Elements of the current time with finer resolution than the finest specified are set to zero.

Example 4: April 1st of the current year

```
(fixed-time :month 4) ⇒ @2007-04-01T00:00:00.000
```

If an exact time is desired, the usage LOCAL-TIME constructors should be used to construct the FIXED-TIME. The most common is ENCODE-LOCAL-TIME. Please see the docu-

1. See http://series.sourceforge.net/. This library is an optional dependency of the PERIODS, if the feature *PERIODS-USE-SERIES* was added to the *FEATURES* list before compilation.

2. See http://common-lisp.net/project/local-time/. This library is a dependency of PERIODS.

mentation to LOCAL-TIME for more information.

2.1 Helper functions

There are a few helper functions for performing common operations on fixed-time's:

day-of-week *fixed-time* [*Function*]

Return the day of the week associated with a given FIXED-TIME. The result is a FIXNUM with 0 representing Sunday, through 6 on Saturday.

falls-on-weekend-p *fixed-time* [*Function*]

Return T if the given FIXED-TIME occurs on Saturday or Sunday.

year-of *fixed-time* [*Function*]
month-of *fixed-time* [*Function*]
day-of *fixed-time* [*Function*]
hour-of *fixed-time* [*Function*]
minute-of *fixed-time* [*Function*]
second-of *fixed-time* [*Function*]
millisecond-of *fixed-time* [*Function*]

Return the corresponding detail associated with a given FIXED-TIME. All results are of type FIXNUM.

3 Time durations

4 Relative time

5 Time ranges

6 Time periods