ISO8601

Abstract
This collection of classes attempts to implement the ISO 8601 time standard. Here you'll find representations of time, dates, time with dates, spans of time and methods of manipulating those concepts.

Time.ISO8601

Inherits from: Class, Time.Date
See Also: Time.Clock, Time.Date, http://wikipedia.org/wiki/ISO_8601
Notes: Load this class like so: `ISO8601 = Import("Time.ISO8601").`

Abstract
A collection of classes that implement some of the ISO8601 dates and times standard. This implementation is not compete but hopefully useful.

Contained Classes

- **`Date`**
- **`DateTime`**
- **`Duration`**
- **`Interval`**
- **`Repeater`**
- **`Time`**
- **`TimeZ`**

Functions

- **`init()`**: Returns a **`DateTime`** for the current date and time.
- **`init("<date time>")`**: Returns **`DateTime`**`(text)` for a date and time.
- **`init("P*")`**: Returns **`Duration(`**`text`**`)`**
- **`init("R*")`**: Returns **`Repeater(`**`text`**`)`**

Discussion

Time.ISO8601.Date

Inherits from: Class
Notes: Load this class like so: `Date = Import("Time.ISO8601").Date`

Abstract
This class implements ISO 8601 dates.

Class Variables

- **`day`**: 1 … 12
- **`dayInWeek`**: 1 … 7, 1 is Monday
- **`dayInYear`**: 1 … 366
- **`month`**: 1 … 12
- **`year`**

Functions

- **`add(`**`Date`**`)`**: Add two Dates.
  Throws: TypeError
  Returns: self
- **`addDays(`**`numDays`**`)`**: Add some days to Date. If days are less than zero, subtract those days.
  Returns: self
- **`addMonths(`**`numMonths`**`)`**: Add some months to Date. If less than zero, subtract.
  Returns: self
- **`addYears(`**`numYears`**`)`**: Add some years to Date. If less than zero, subtract.
  Returns: self
- **`addYMD(`**`years,months,days`**`)`**: Additional addition.
  Returns: self
- **`init()`**: Creates a Date instance for the current date.
- `init(year, month=1, day=1)`: Create a Date for YMD.
- **`init("`***`date_string`***`")`**: Parse the date string and create a Date.
  The following formats are recognized:
  - YYYY-MM-DD, YYYY-MM, YYYY, where MM & DD default to 1 and MM and DD can be one digit.
    - `Date("2007")` → `Date(2007,1,1)`
    - `Date("2007-2-7")` → `Date(2007,2,7)`

    YYYY is technically illegal (per the 8601 specification).
  - YYYYMMDD, YYYYMM, YYYY where MM && DD default to 1. You should use all specified digits.
    - `Date("200702")` → `Date(2007,2,1)`
    - `Date("20070207")` → `Date(2007,2,7)`

    YYYY and YYYYMM are technically illegal (per the 8601 specification).
  - YYYY-Www, YYYY-Www-D, D defaults to Monday
    Week format. ww is the nth week of the year, D is a weekday, 1...7
    - `Date("2009-W01-1")` → `Date(2008,12,29)`
  - YYYYWww, YYYYWwwD, D defaults to Monday
    - `Date("2009W537")` → `Date(2010,1,3)`

  You are probably going to need to study the 8601 specification if you want to understand Week format.
  The error checking isn't very strong for strings.
  Throws: IndexError, ValueError
  Returns: Date
- **`isLeapYear()`**: Returns True if the date is a leap year
  Returns: Bool
- **`nthDay()`**: Which day in the year this is.
  Returns: Int

- **`pretty()`**: Pretty print the date.
  For example: “Friday, the 16th of March 2007”
  Returns: String
- **`subDays(`**`numDays`**`)`**: Subtract some days from Date. If days are less than zero, add the days.
  Returns: self
- **`subMonths(`**`numMonths`**`)`**: Subtract some months to Date. If less than zero, add.
  Returns: self
- **`subYMD(`**`years,months,days`**`)`**: Additional subtraction.
  Returns: self
- **`text()`**: Returns “YYYY-MM-DD”.
  Returns: String
- **`toString()`**: Returns “Date(YYYY-MM-DD)”
  If another class inherits from Date, the name of that class replaces “Date”.
  Returns: String
- **`weekText()`**: Returns “YYYY-Www-D” where ww is the nth week and D is the weekday (1 … 7).
- **`whichWeek()`**: Returns the nth week and year Date is in. Note that the week may not be in the current year.
  Examples:
  - 2008-12-29 is written "2009-W01-1" so the week is 1 even though the year is different. L(2009,1) is returned.
  - 2010-01-03 is written "2009-W53-7" so the week is 53 and L(2009,53) is returned.
  - Tuesday, January 23, 2007 is in week 4 resulting in L(2007,4)

  Returns: L(*year*,*week#*)

Time.ISO8601.DateTime

Inherits from: Date, TimeZ
Notes:

- Load this class like so: `DateTime = Import("Time.ISO8601").DateTime` or `DateTime = Import("Time.ISO8601")(`*`date_time_string`*`)`

Abstract
This class implements an ISO 8601 date and time with time zone.

Functions

- **`add(`**`Time`**`)`**:
- **`add(`**`Date`**`)`**:
- **`add(`**`DateTime`**`)`**:
- **`add(`**`Duration`**`)`**:
  Add the some time to self.
  Throws: TypeError
  Returns: self
- **`ctime()`**: Format the time and date to match C's ctime function.

For example: “Fri Mar 16 01:37:42 2007”
Returns: String

- **`init()`**:
- **`init("`***`date_time`***`")`**:
- **`pretty(`**`ampm=False`**`)`**: Pretty print the time and date.
  Examples `pretty()` → “01:36:07 Friday, the 16th of March 2007”
  `DateTime("T13:52").pretty(True)` →
  “01:52:00PM Friday, the 16th of March 2007”
  Returns: String
- **`sub(`**`DateTime`**`)`**:
  Throws: TypeError
  Returns: self
- **`text()`**: Returns “YYYY-MM-DD T HH:MM:SSZ” or “YYYY-MM-DD T HH:MM:SS±HH[:MM[:SS]]” depending on the time zone setting.
  Returns: String
- **`toString()`**: Returns “DateTime(`text()`)
  Returns: String

Discussion

Time.ISO8601.Duration

Inherits from: Class
Notes:

- Load this class like so: `Duration = Import("Time.ISO8601").Duration` or `Duration = Import("Time.ISO8601")("P`*`duration`*`")`

Abstract
This class represents an arbitrary span of time. There is no starting or ending point.

Class Variables

- **`Y`**,**`M`**,**`D`**, **`HH`**,**`MM`**,**`SS`**: The years, months, days, hours, minutes and seconds that define the duration of time.

Functions

- **`format()`**: Returns **`text() - " "`**. This makes text a strictly conforming Duration string. Use text if you want to be able to read it.
  Returns: String
- **`init()`**:
- **`init(`***`text`***`)`**:
  Throws: ValueError
  Returns: Duration
- **`text()`**: “P nY nM nD T nH nM nS”
  Returns: String

Discussion
This is a helper class for Intervals. By itself, it is a just a measure of time that is pretty awkward to read or write.

Time.ISO8601.Interval

Inherits from: Class
Notes: Load this class like so: `Interval = Import("Time.ISO8601").Interval`

Abstract

Class Variables
- **`begin`:**
- **`duration`:**

Functions
- **`format()`**:
- **`init()`**:
- **`next()`**:
- **`text()`**:

Discussion

Time.ISO8601.Repeater

Inherits from: Interval
Notes:
- Load this class like so: `Repeater = Import("Time.ISO8601").Repeater` or `Repeater = Import("Time.ISO8601")("R...")`

Abstract

Class Variables
- **`N`**:

Functions
- **`init(text)`**:
- **`format()`**:
- **`read()`:**
- **`text()`**:
- **`walker()`**:

Discussion

Time.ISO8601.Time

Inherits from: Class
Notes: Load this class like so: `Time = Import("Time.ISO8601").Time`

Abstract
This class encapsulates a twenty four hour clock.

Class Variables

- **`hour`**: 0 … 23
- **`min`**: 0 … 59
- **`sec`**: 0 … 61 (leap second)
- **`format`**: Initially set to function text, you can set this to a function that formats the time to your liking.

Functions

- **`add(`**`Time | TimeZ`**`)`**: Add a Time or TimeZ to self. Self changes.
  Returns: Overflow (Int)
- **`addHMS(`**`hh,mm,ss`**`)`**: Update self. If more than one day is added (ie hh > 24), those extra days are returned (otherwise, zero is). Self changes. Time, in this case, is a arrow into the future, so don't pass in negative numbers. The numbers can be big; if you want to add five days, you could do it with `addHMS(0,0,5*24*60*60)` → 5 (and self doesn't actually move).
  Returns: Overflow (Int). For example, `addHMS(26,0,0)` adds two hours and returns 1 (day).
- **`init()`**: Set the time to the current time.
- **`init(`***`hour`***`, `***`minute`***`=0, `***`seconds`***`=0)`**: Set the time.
- **`init(`***`timeString`***`)`**: Parse the time string.
  Throws: Yes
  Returns: Time
- **`now()`**: Sets self to the current time.
  Returns: self
- **`parse()`**: Set self to the current time.
  `parse(`*`timeString`***`)`**: Set self to `textToHMS(`***`timeString`***`)`.
  Throws: yes
  Returns: String (`text()`)
- **`read()`**: Set self to the current time and return the time as a string.
  Changing the variable "format" can be handy here:

```
tz = ISO8601.TimeZ()      → TimeZ(17:53:40-8)
tz.read()                 → "17:55:14-8"
tz.format=tz.toAMPM
tz.read()                 → "05:56:06PM"
```

Returns: String(`topdog.format()`)

- **`secondsToHMS(seconds)`**: Convert seconds into hours, minutes and seconds.
  Returns: L(h,m,s)
- **`sub(Time | TimeZ)`**: Subtract from, and change, self.
  Returns: The new hour
- **`subHMS(h,m,s)`**: Subtract time from self. If the new time is, like, so yesterday, hour is negative. Time marches ever forward, so keep the numbers positive. Also, keep the fields in their respective ranges, multiple underflows are not handled. Self changes.
  Returns: The new hour
- **`text()`**: Return the time as "HH:MM:SS". Each field is two digits. HH is base twenty four.
  Returns: String
- **`textToHMS()`**: Set self to the current time.
  `textToHMS(text)`: Parse text and update self. The format is "HH:MM:SS" with lots of looseness; as long as there are digits between colons and the digits are in the proper range, you should be OK. Twenty four hour clock and junk after SS is ignored (unless that junk is in the time fields). Digits are optional: ":::" == "00:00:00", "1" == "01:00:00", ":2:" == "00:02:00", etc.
  Throws: ValueError
  Returns: L(h,m,s)
- **`toAMPM()`**: Like text, but appends "AM" or "PM" using a twelve hour clock.
  Returns: String
- **`toFloat()`**: Convert self to HH.*fractionalTime*.
  Returns: Float
- **`toSeconds()`**: Convert self into seconds. Basically the number of seconds since midnight.
  Returns: Int
- **`toString()`**: Returns "Time(HH:MM:SS)". "Time" is the name of the topdog class.
  Returns: String

Discussion
Your basic twenty four hour clock.

Time.ISO8601.TimeZ

Inherits from: Time
Notes: Load this class like so: `TimeZ = Import("Time.ISO8601").TimeZ`

Abstract
This class builds on Time to add the time zone.

Class Variables

- **`format`**: Initially set to function text, you can set this to a function that formats the time to your liking.

- **`DST`**: True if Daylight Savings Time is in effect.
- **`TZ`**: Time zone. The number of seconds from UTC[1]/GMT[2]. For example, Pacific Standard Time is eight hours before UTC so TZ would be -8*60*60 == -28800.

Functions

- **`init()`**: Create a TimeZ form the current time.
  `init(`*`timeString`***`)`**: Parse *timeString* and set self.
  Throws: Yes.
  Returns: self
- **`parse()`**: Set self to the current time and zone (set by the OS).
  `parse(`*`timeString`***`)`**: Parse *timeString* and set self. The following formats are recognized (*time* is something recognized by Time.parse):
  time The current time zone is used
  *time*Z Time zone is UTC (*time*+00)
  time±hh
  time±hh:mm or *time*±hhmm
  time±hh:mm:ss or *time*±hhmmss
  Throws: VaueError
  Returns: String (`text()`)
- **`setTZ(h,m=0,s=0)`**: Set the current time zone.
  Returns: self
- **`shiftTZ(h,m=0,s=0)`**: Move the time zone relative to the current time zone.
  Returns: self
- **`text()`**: Returns "HH:MM:SSZ" or "HH:MM:SS±HH[:MM[:SS]]" depending on the time zone setting. For example, in Pacific Standard Time, the time zone offset is eight hours, so the minutes and seconds are trucated: "21:58:55-8". If the time zone is UTC, "Z" is appended: "11:22:33Z"
  Returns: String
- **`tzToHMS(asString=False)`**: Convert the time zone to a list. For example, `PST.tzToHMS()` → L(-8,0,0). For printing, it usually preferable to have the time zone formated: `PST.tzToHMS(True)` → "-08:00:00". If you would prefer a "crunched" HMS string, use `tzToHMS(True) - ":"` (→ "-080000").
  Returns: L(h,m,s) or String (`"%+03d:%02d:%02d"`)
- **`toString()`**: text()
  Returns: String(**`text()`**)
- **`toUTC()`**: What time is it in Greenwich?
  Returns: String

Discussion

A Time tweaked to become your basic twenty four hour world clock. Strap it on your wrist, head for the airport and hop on a Concorde.

1 Coordinated Universal Time (French: Temps Universel Coordonné). Since both English speakers and French speakers wanted "their" abbreviation for the time standard, "UTC" was the compromise.

2 Greenwich Mean Time is basically the same thing for our use here.