Input conversions

Peter Fox author@vulpeculox.net

Status of this document

March 2015: New.

Contents

1 General use

The object is to take a number, string or Date object and convert it into a DAY object.

There will always be some inputs that are incompatible. These should always be handled *without* raising an exception, but return a DAY with a **not valid** signature and status code telling what sort of problem was encountered.

Typical string inputs from user input might be 14 Apr 2012, Apr 2012, 2012, 140412, 04142012, however we also come across various 32bit numbers and strings returned by standard database queries such as 2012-04-14 12:12:12 which clearly need to be interpreted with caution.

The environmental configuration allows for default patterns to be recognised, for example the month-day order in general use, or how to interpret two digit years, if allowed. Month names are easily internationalised too.

Note: These settings are given in simple 'appendices', but for the purpose of this document we will deal with the default language only. When we will refer to T and Today the same logic will apply to A and aujourd'hui and so on.

The design philosophy for the flexible patterns is to make it as rule-free as possible and as user friendly as possible. So for example T is a shortcut for the frequently inputted value of Today and even in a M-Y-D environment inputting 14 May 2012 is acceptable.

2 Intervals

In D/a/y dates and intervals are separate things but in any serious application they are both likely to appear. For example use a date of birth to calculate age to date or vice-versa.

- Intervalsmust start with + or –
- There must one to three groups of at least one digit or digits followed by y, m, or d[1]

There is no need to put spaces in between the groups

The following are legal

+ 1d	zeros can be omited
+ 18m 2y	will automatically convert to + 4m 3y
– 3y 2d 4m	Any group order
+6d4m0y	Spacing can be omitted
+1234d	See below. This is OK so long as no m or y are involved

The following are illegal

1d	No + or –
+18m –2y	Illegal –
+1d 2d	Two d-groups
+1234d 1y	We can't mix excess days and years or excess days and months because we don't know how to convert exactly.[2]

X or Not Valid may appear as an output but are not valid inputs

[1] d, m and y are configurable in the appendix.

[2] There are explicit conversion functions in the API but this is silent-fudging.

3 Shortcuts

To make life easier and standardised for humans we allow special codes for commonly used concepts. These can be set in an appendix. For the most part, we allow the absolute minimum of input to avoid ambiguity. The defaults are:

- B or any starting part or whole of Beg-of-Time[3]
- E or any starting part or whole of End-of-Time
- U or any starting part or whole of Unknown
- ! or any starting part or whole of Not Valid

Month names may normally be abbreviated. The exact details of the minimal input depends on the appendix currently in use.

4 Formal numeric patterns

Forcing

There are some ambiguous numbers which can be used in specific ways by using a prefix to a string version.

- I/ Interval from days
- J/ Fully qualified Cal from Julian day number
- U/ Ordinary Unix timestamp (String version of millisecond number) to fully qualified Cal (H:M:S parts are ignored)
- E/ Extended Unix timestamp (String version of millisecond number) to any DAY (H:M:S parts are interpreted)
- N/ 32-bit representation (String version of number)

So for example if you had a Julian date you could feed it to Day as 'J/2454525' to give 28 Feb 2008. 'I/19' would become + 19d

Stringy dates and times

Databases and many other documents use a YYYY-MM-DD HH:MM:SS style of representation. Day can recognise this but if it is provided with the time part it needs to know if it is to use it or not. (Remember the time can be used to store important Day information. If the time is ignored then it becomes a fully specified calendar date representing a specific day.

---

[3] These concepts are described in detail in the main specification.

5 Flexible patterns

- All punctuation is treated as a space with multiple spaces combined. 1-2/2012 is valid as is 1.2....2012
- For the following assume any BC/AD flag has been dealt with.

Single token

- Shortcut
  - This will always be tested for first even with multiple tokens so for example 'T foo bar' will be interpreted as Today. Shortcuts will always ne the first token.
- Month name
- ddmm or mmdd
  - Affected by DM_ORDER and IMPLIED_YEAR
- ddmmyy or mmddyy
  - Affected by DM_ORDER and TWO_DIGIT_FIX
- ddmmyyyy or mmddyyyy
- yyyy

Two tokens

- Day-num Month-num (or M then D)
  - Affected by DM_ORDER and IMPLIED_YEAR
- Day-num Month-name
- Month-name year-num or Month-name day-num
  - Affected by DM_ORDER

  May 2012 is unambiguous but May 12 could be the 12th of May! Month-name followed by day-num will only be allowed if DM_ORDER is MDY. Developers are warned this is quicksand so a clear user interface will be required. Note that in some applications May 12 for what month something is scheduled for is just the sort of input you're looking for.

Three tokens

- Day-num Month-num Year-num ir Month-num Day-num Year-num
  - Affected by DM_ORDER and TWO_DIGIT_FIX
- Day-num Month-name Year-num ir Month-name Day-num Year-num
  - Affected by TWO_DIGIT_FIX

Key appendix settings

- DM_ORDER can be DMY or MDY. There is always scope for mis-input so we suggest users have clear feedback and strict rules.
- TWO_DIGIT_FIX controls how to handle two-digit years.
  - 0 Disallow
  - 20 convert yy -> 20yy

    - 50 convert yy so that it lies between 50 years ago and 50 years ahead
- IMPLIED_YEAR allows a given year to be implied if it is missing.
    - 0 Do not allow inputs that miss out year part
    - -1 Use the current year as an implied substitute
    - -2 Allow floating date input (eg 2 Mar for a birthday)
    - nnnn Use the specified year if none is given

6 non-string inputs

Javascript date object

When a Date object is created by the .ToDate method it always has hours minutes and seconds set according to the extended timestamp method qv. Unless explicitly forced @@@ the Day constructor will use that h, m, s information to reconstruct a fully functional object.

For example an existing Javascript routine where a database knows a bit about Date objects might look like:

```
   myRecord.startDate = someDateVar;
   UpdateDatabase('myTable',myRecord);
```

then

```
  if(myRecord = ReadDatabase(someParameters)){
    DisplayDate('someScreenElement',myRecord.startDate);
```

Now you want to use Day features without having to change the database schema so here's all you do to retro fit it.

This means that existing infrastructure

```
   myRecord.startDate = shinyNewDayVar.toDate();
   UpdateDatabase('myTable',myRecord);
```

then

```
  if(myRecord = ReadDatabase(someParameters)){
    DisplayDate('someScreenElement',new DAY(myRecord.startDate));
```

If a Date given to say the Day constructor then if the h,m and s are zero it will always be treated as a fully qualified calendar date.