# Adventures in Groovy – Part 36: Manipulating Dates

Manipulating dates is not something you may think is that useful, but there are a number of things that we have done in the past that are slow performing in Essbase, and others that were not possible or you may not have ever thought of.  This is going to walk visitors through ways to manipulate dates for all kinds of uses.  Hopefully it inspires some ideas for you to make your application a little more polished.

## This Is Nice, But When Would It Be Used?

First, if logic is done in Groovy for things like WFP, comparing dates is frequently used to calculate benefits.  How many months has the person been employed?  Did they get a promotion in the last 12 months?  Calculating differences in dates is extremely simple in Groovy and applying more logic around it is often quicker to write and faster to process in Groovy than it is in Essbase.

Capex is also a place that is date dependent to calculate depreciation and other asset related accounts.  There are also some things we can do to make the user experience better, like giving the option to just enter a number of months rather than get the end date based on the start date.

I can’t say you will use it in every build you are involved in, but when it is required, these techniques will undoubtedly make your life a little easier.

## Creating And Formatting Dates

Most of the methods require a date object.  It can be created by getting the current date and time or setting the date and time from a string or concatenation of variables.

```
// This will create a date object with the current time
def date = new Date()
// To create the object with a specific date/time, follow this logic
def date = new Date().parse('yyyy/MM/dd', '1973/07/09')
```

A real-world example might be using the members in the POV to create a date.  Since getting the member values from RTPs or looping through cells has already been covered, this will assume the method you use returns the following values.  The parse method can hold just a year, month, or year and month, or year and moth and day, for example.  The two parameters must match.

```
// drop the first two characters
def sYear = "FY18".drop(2)
def sMonth = "Jan"
def sDay = "1" // The day is likely irrelevant, so 1 will be used as a default
def date = new Date().parse('yyyy/MM/dd', "$sYear/$sMonth/$sDay")
// we are using yy because the date is only two years
// we are using MMM because we have a month abbreviation.  If we had a number, it would be MM
println date
def date1 = new Date().parse('yy/MMM', "$sYear/$sMonth")
println date1

// Both return 'Mon Jan 01 00:00:00 EST 2018'

println date1.format('M/d/yy')
// Returns 1/1/18
```

Here are some more formatting examples.  Depending on the need, some of these may be useful.  There are more, but I think these are the formats you might use most often.

| Pattern | Output | |
|---|---|---|
| dd.MM.yy | 30.06.09 | |
| yyyy.MM.dd G ‘at’ hh:mm:ss z | 2009.06.30 AD at 08:29:36 PDT | |

| EEE, MMM d,yy | Tue, Jun 30, ’09 |
|---|---|
| h:mm a | 8:29 PM |
| H:mm | 8:29 |
| H:mm:ss:SSS | 8:28:36:249 |
| K:mm a,z | 8:29 AM,PDT |
| yyyy.MMMMM.dd GGG hh:mm aaa | 2009.June.30 AD 08:29 AM |

Often times we need the month, day, or year of a date object. This is another simple thing to do.

```
date = new Date().parse('yyyy/MM/dd', '1973/07/09')
println date[Calendar.YEAR]
// Returns 1973
println date[Calendar.MONTH]
// Returns 7
println date.getAt(Calendar.DATE)
// Returns 9
```

## Manipulating Dates

The first thing that is very common is the need to do is some basic changes to the year and month.

Often time the year we have in Planning is prefixed with FY. A simple thing to remove the FY is to drop the first two characters.

```
def sYear = "FY18"
println sYear.drop(2)
// Returns 18
```

Next is months.  There are a couple of things that are typical requests.  One is to pad the number with a 0.  The other is to convert numbers to the month name, and visa versa

```
date = new Date()
println.date[Calendar.MONTH].padLeft(2,'0')
println date1.format('MM')
// These will return the numeric month
// It will also pad it with zeros and return 2 characters
```

```
// Often times we need a two digit string rather than an integer

//If we have the number and need the name
$sMonth = 1
def date = new Date().parse('M', $sMonth)
println date.format('MMM')
// Returns Jan
println date.format('MMMM')
// Returns January
```

Increasing or decreasing a date by a predefined number of days is also simple and often used.

```
date = new Date()
date.plus(1)  // Returns tomorrow
date.minus(1)  // Returns yesterday
```

Increasing months and years is a little more complicated, but still easy to accomplish.

```
d = new GregorianCalendar()
d.setTime(new Date().parse('yyyy/MM/dd', '2018/07/09'))  // sets to the current date/time
d.add(Calendar.MONTH,5)  // Increases the date by 5 months
println d.getTime().format('MM/dd/yyyy')
// Returns 12/9/2018
```

Lastly, getting the difference between two days might be something required.

In days

```
date = new Date().parse('yyyy/MM', '2018/07')
date1 = new Date().parse('yyyy/MM', '2019/08')
println (date..<date1).size()  // Returns 31
```

In Months

```
date = new Date().parse('yyyy/MM', '2018/07')
date1 = new Date().parse('yyyy/MM', '2019/08')
println ((date1[Calendar.MONTH] - date[Calendar.MONTH]) + ((date1[Calendar.YEAR] - date[Calendar.YEAR])*12))
```

```
// Returns 13
```

## Calling It A Day

I can see prompting a user when adding a new asset to just request a start date and how many months the useful life would be, rather than asking for two dates.  When adding seasonal resources, it would be nice to ask for a start date and the number of days they will be needed, in some cases.

The date manipulation is something we all do, and I can tell you, doing it in Groovy is way simpler than doing in in Essbase with all the concatenations required.

If you have a good example of use case, please share!