Working with Dates (and timezones)

First of all the constructor new Date() will make the same date no matter what timezone you are in..

So if you do new Date() in The Netherlands and at the same time somebody else does new Date() in USA. then that is the exact same date, both represent the time in milliseconds after 1970 in UTC. This is called Unix Time.

The only thing that makes is different is that when you print it or format it. And the toString() of a date already formats it, so yes it shows there the date and timezone that you are currently in. But this does not mean that Dates have a timezone, that is not correct.

The date object has a constructor as:

new Date(year, month[, day[, hour[, minutes[, seconds[, milliseconds]]]]]);

Those arguments are then the hours in your timezone. Just like if you would format the string with "yyyy/MM/dd HH:mm" with your timezone given to the formatter.

The Date object has another function Date#UTC(year,month,date,hrs,min,sec,ms) that one you will give back the milliseconds that is in UTC (you can wrap this number in a new Date(number) again). So if you give that function a "hrs" of 10 and you are in +2, then the date in your timezone will be 12.

In Servoy the problem arises when you are calling Date#getHours() (and other methods like that) because that is in local time. And your server could be in in the timezone +2 but your client could be in +3, and servoy code runs at the server, so the getHours() call will be the hours that the server reports it (the same is the toString())

Example of local or UTC time with getHours()

```
var date1 = new Date(2019,10,1,12);
    var date2 = new Date(Date.UTC(2019,10,1,12));
    application.output(date1)
    application.output(date2)
    application.output(date1.getHours())
    application.output(date2.getHours())
```

The above code will print a different hours depending on the offset you have to UTC. So "date1" will just print "12" in both outputs because you constructed it for your local time, if you are in +1 then "date2" will print "13" in both scenario's, because the "local utc (London)" time of "12" is "13" in your local time zone (Amsterdam)

But the above will do this all with the timezone the server is in, not the actual client which can be different again (so not UTC or +1 but +2)

For getting the hours that the client has (NG and Smart) you can use the utils.dateFormat

get the hours of the client (NGClient and SmartClient)

```
application.output(utils.dateFormat(date1,"HH"));
```

That will use the timezone of the client that we got from the browser. And for example if the server would be in +1 and the NGClient would be in +2 then the above code would print "13". Because "12:00" in the servers timezone is for the clients timezone 1 hour later. (Amsterdam compared to Athens)

Use as LocalDateTime

For displaying stuff in the client we he have for NGClient an option to Use as LocalDateTime in the date format dialog. By default if that is not checked we send the date over in a ISO-8601 dateformat like: 2019-11-01T12:00:00+01:00 if the server is in +1 . So we send over the date in the servers format but we also give the timezone where we formatted this date in. This information is received by a client and if that is in +2 the time that the client will display that same date will be: 2019-11-01 13:00, because it knows that that give date was for +1 and it is in +2 so it must add 1 hour to the displayed time (still the time in milliseconds does not change! this is purely a formatting thing)

This is logical because a calendar datetime of 12:00 in +1 (server timezone) is in +2 (client timezone) 13:00. So the only "conversion" that is done here is that the same milliseconds is formatted differently based on a different timezone (so the toString of the date changes, not the Date itself)

Now if you have not calendar times but more like a birthday (you want to just specify a day in a year not time) then you should check "Use as LocalDateTime" this means that where ever you go the date should be treated the same (formatted). What this means is that instead of sending over the date string as: 2019-11-01T12:00:00+01:00 we send it over without a timezone info: 2019-11-01T12:00:00 this means that the client can only assume this date is in its timezone (so it assumes 2019-11-01T12:00:00+02:00). Then the formatted time that the user is presented to will just be 2019-11-01 12:00 without any "conversion" as it seems.

Problem is that now we do convert, because now if you would ask the getTime() in milliseconds from it, it would be different, it would be 1 hour in milliseconds less. Because we kind of removed 1 hour from it (the difference between the server +1 and the clients +2)

So now the formatted date stays the same (no "conversion" here) but the actual date did change because now it represents that same formatted (server) time in a different timezone so it needed to adjust the milliseconds.

The browser if it changes the date and we send over the date string again it will always just send over the timezone info with it, sending it always in the format: 2019-11-01T13:00:00+02:00, but the server knows that for that property it should use LocalDateTime and will just ignore or not that timezone information based on that boolean.

application.getTimeStamp()

If you are using "Use as LocalDateTime" then it could be that you want to make the date object that must be really in "today" or "now" on the client. So the same thing as doing a new Date() in the browser and then sending that to the server that ignored the timezone.

To do the same thing that you really are in the "now" on the client you can use application.getTimeStamp() this will create a date that will be in the same day as the browser. as an example new Date() on the server that is in +1 would generate 2019-28-11 23:00 then application.getTimeStamp() would generate if the client was in +2 a date string of 2019-28-12 00:00. Which is really the time that that client is in at that time.

Searching

If you want to search between 2 dates, like searching for everything of yesterday, but then you mean everything of what the users means of yesterday. You need to make dates that for that user is yesterday. Which could be dates like: "2019-28-10 00:00" and "2019-28-10 23:59", but those times are client times so in the current example in timezone +2. But the server is in +1 , and the server will do the searching if it does that with the 2 dates above as is then it would be searching for stuff that is in the "yesterday" of the server. So you need to convert or create them first to the times of the user:

Creating dates in the client timezone

```
var startDate = utils.parseDate("2019-11-28 00:00","yyyy-MM-dd HH:mm", null)
var endDate = utils.parseDate("2019-11-28 23:59","yyyy-MM-dd HH:mm", null)
```

This parseDate with 3 arguments is new in Servoy 2019.12, the 3rd argument can also be a TimeZone ID like "GMT+3" but if it is null it will use the clients timezone, the standard 2 arguments will use the servers timezone.

The output of that if the server is in +1 and the client is in +2:

Wed Nov 27 23:00:00 CET 2019
Thu Nov 28 22:59:00 CET 2019

(dates printed/formatted in +1/CET)

If these 2 timestamps would be send to the client (Use as LocalDateTime as false), then the browser will "add" (formats it in its timezone) the time difference (1 hour) back on to it and it will then represent the exactly that day.