Base Language - Bug #6537

problems with ADD-INTERVAL

06/21/2022 03:26 PM - Constantin Asofiei

Status:	WIP	**Start date:**	
Priority:	High	**Due date:**	
Assignee:		**% Done:**	80%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

History

#2 - 06/21/2022 03:32 PM - Constantin Asofiei

There is a bug in ADD-INTERVAL: it overrides the timezone, when a datetimetz is used. See this:

```
def var d1 as datetime-tz.
def var d2 as datetime-tz.
def var d3 as datetime-tz.

def var i as int.

d1 = datetime-tz("01-01-1970T00:00:00.000+00:00").
d2 = datetime-tz("01-01-1970T00:00:10.000+00:00").

i = interval(d2, d1, "seconds").
d3 = add-interval(d1, i, "seconds").
message i d3. // 10 01/01/1970 00:00:10.000+00:00
```

The patch I think is this:

```
### Eclipse Workspace Patch 1.0
#P p2j6129a
Index: src/com/goldencode/p2j/util/DateOps.java
===================================================================
--- workspace.java.open.client/p2j6129a/src/com/goldencode/p2j/util/DateOps.java    (revision 3694)
+++ workspace.java.open.client/p2j6129a/src/com/goldencode/p2j/util/DateOps.java    (working copy)
@@ -132,7 +132,8 @@
       long delta = getIntervalInMillis(amount, unit);
       if (delta != 0)
       {
-         datetime inter = datetime.plusMillis(new datetime(dat.duplicate()), delta);
+         datetime inter = dat instanceof datetime ? (datetime) ((datetime) dat).duplicate() : new datetime(da
t);
+         inter = datetime.plusMillis(inter, delta);
          T ret = (T) dat.duplicate();
          ret.assign(inter);
          return ret;
```

The point is to preserve in inter the dat's type - if is a datetime or datetimetz, then use its type, otherwise create another datetime.

Ovidiu: please review and do some more tests, I don't like that DateOps.interval has this code:

```
      if (d2 instanceof datetimetz)
```

```
      {
         // adjust to GMT (just to have a common denominator)
         d2 = ((datetimetz)d2).toLocalDatetime();
      }
```

We need some checks with different timezones at the interval datetime-tz arguments, and if interval still provides the same 'seconds'.

#3 - 06/21/2022 09:35 PM - Ovidiu Maxiniuc

- % Done changed from 0 to 50

- Status changed from New to WIP

Constantin,
I checked the code and did some tests. I think that those adjustments are fine. But there other issues which are incorrect. Please see this patch:

```
--- old/src/com/goldencode/p2j/util/DateOps.java
+++ new/src/com/goldencode/p2j/util/DateOps.java
@@ -255,7 +258,6 @@
     */
    public static int64 interval(date d1, date d2, String unit)
    {
-
       int calField = getGregorianField(unit);
       if (calField < 0)
       {
@@ -269,15 +271,15 @@
       {
          return new int64();
       }
-
+
       if (d1 instanceof datetimetz)
       {
-         // adjust to GMT (just to have a common denominator)
+         // adjust to local TZ (just to have a common denominator)
          d1 = ((datetimetz)d1).toLocalDatetime();
       }
       if (d2 instanceof datetimetz)
       {
-         // adjust to GMT (just to have a common denominator)
+         // adjust to local TZ (just to have a common denominator)
          d2 = ((datetimetz)d2).toLocalDatetime();
       }

@@ -287,9 +289,12 @@

       // compute time difference between dates:
       long dt = 0;
-      if (d1 instanceof datetime && d2 instanceof datetime)
+      if (d1 instanceof datetime)
       {
          dt += ((datetime) d1).getTime();
+      }
+      if (d2 instanceof datetime)
+      {
          dt -= ((datetime) d2).getTime();
       }
```

Notes:

- if the time-zone is not specified (in case of date and datetime), the local TZ is assumed (not GMT as written in old javadoc).
- the time part was taken into consideration for both operands only if both had time component. This is incorrect and is fixed in my patch: if any of the operands have a time component, it is used in computation, regardless of the structure of the other operand.

#4 - 06/22/2022 05:05 AM - Constantin Asofiei

Ovidiu, the fix is in 6129a/13948 - please review. The point with the interval issue is to convert both instances to GMT - it is not correct to convert it to local time, as this will mess up the timezone.

There still are some issues:

- datetimetz.toLocalDatetime is used by DateOps.compare, which has a single usage in p2j.oo.net.http.CookieJar._isExpired(). DateOps.compare I think can be removed and replaced with datetimetz.compareTo.
- datetimetz.toLocalDatetime is used by datetime.assign(date):

```
      if (value instanceof datetimetz)
      {
         // convert value to local time before assigning
         deepAssign(((datetimetz)value).toLocalDatetime());
      }
```

  This doesn't look correct to me - if you have a datetimetz like 01-01-1970T00:00:00.000+00:00, why convert it to local time? At most, I would convert it to UTC or just remove the timezone and get the 'raw' time without it.

If the two cases above are fixed, then toLocalDateTime can be removed completely.

#5 - 06/22/2022 10:16 AM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> Ovidiu, the fix is in 6129a/13948 - please review. The point with the interval issue is to convert both instances to GMT - it is not correct to convert it to local time, as this will mess up the timezone.

The OE use the local time to compare all date datatypes. For example, the following code is TZ-dependent. On Bucharest/Athens (+1 summer time, +2 TZ offset), the results are in comments:

```
MESSAGE INTERVAL(DATE("06-22-2022"), DATETIME-TZ("06-22-2022T00:00:00+00:00"), "seconds")  // -10800 (i.e. 3h)
        INTERVAL(DATE("06-22-2022"), DATETIME-TZ("06-22-2022T00:00:00+03:00"), "seconds"). // 0
```

There still are some issues:

- datetimetz.toLocalDatetime is used by DateOps.compare, which has a single usage in p2j.oo.net.http.CookieJar._isExpired(). DateOps.compare I think can be removed and replaced with datetimetz.compareTo.

OK, if you do not need the CookieJar dependency on DateOps. But it shares the package with datetimetz so no real benefit except for a flatter call stack.

- datetimetz.toLocalDatetime is used by datetime.assign(date):
  [...]
  This doesn't look correct to me - if you have a datetimetz like 01-01-1970T00:00:00.000+00:00, why convert it to local time? At most, I would convert it to UTC or just remove the timezone and get the 'raw' time without it.

Actually this implementation is correct. The implementation of date.assign() which truncate the time-offset is wrong. Here is an example:

```
DEFINE VARIABLE d4 AS DATE.
DEFINE VARIABLE d5 AS DATETIME.

d2 = DATETIME-TZ("06-22-2022T02:00:00.000+07:00").
d4 = d2.
d5 = d2.
MESSAGE d2 "*" d4 "*" d5. // 06/22/2022 02:00:00.000+07:00 * 06/21/22 * 06/21/2022 22:00:00.000
```

Notice d4 (date) is **wrong on FWD** for the same TZ as above, but the d5 (datetime) is correct, including the hours.

If the two cases above are fixed, then toLocalDateTime can be removed completely.

No, in conclusion.

#6 - 06/22/2022 10:36 AM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> Constantin Asofiei wrote:
>
> > Ovidiu, the fix is in 6129a/13948 - please review. The point with the interval issue is to convert both instances to GMT - it is not correct to convert it to local time, as this will mess up the timezone.
>
> The OE use the local time to compare all date datatypes. For example, the following code is TZ-dependent. On Bucharest/Athens (+1 summer time, +2 TZ offset), the results are in comments:
> [...]

Yes, but my point is that if you have a datetimetz instance, then you can't alter its timezone when comparing or otherwise calculating.

> There still are some issues:
>
> - datetimetz.toLocalDatetime is used by DateOps.compare, which has a single usage in p2j.oo.net.http.CookieJar._isExpired(). DateOps.compare I think can be removed and replaced with datetimetz.compareTo.
>
> OK, if you do not need the CookieJar dependency on DateOps. But it shares the package with datetimetz so no real benefit except for a flatter call stack.

The point here is that DateOps.compare duplicates datetimetz.compare and others. And datetime dt1 = ((datetime)d1).toLocalDatetime(); is wrong when working with a datetimetz.

> Actually this implementation is correct. The implementation of date.assign() which truncate the time-offset is wrong. Here is an example:

Yes, I noticed that too, the assignment sets the time part in a datetime to the local time, thus you need this conversion.

I think only the datetime.assign requires toLocalDateTime.

#7 - 06/22/2022 08:06 PM - Ovidiu Maxiniuc

- % Done changed from 50 to 80

Constantin Asofiei wrote:

> Ovidiu, the fix is in 6129a/13948 - please review.

I reviewed the changes and inserted my additional changeset. I also added implementation for the stubs in datetimetz and addressed the incorrect result from note-5. Latest revision is 13949.

> The point with the interval issue is to convert both instances to GMT - it is not correct to convert it to local time, as this will mess up the timezone.

Yes, that is the actual intent: to be able to 'subtract' one time instance from another they must be brought to a common denominator. 4GL uses the local time-zone (possibly the one specified in Session:timezone attribute ?). FWD does the same.

Please run your testcases and let me know which are now incorrect.

#8 - 06/23/2022 02:25 AM - Constantin Asofiei

Ovidiu, toLocalDateTime is wrong - you can't just adjust a datetimetz using the local timezone - what about DST?

Please see the tests bellow - 'a' and 'd' fail:

```
def var d1 as datetime-tz.
def var d2 as datetime-tz.
def var d3 as datetime-tz.
def var i as int.

d1 = datetime-tz("01-01-1970T00:00:00.000+00:00").
d2 = datetime-tz("06-23-2022T08:55:22.123+03:00").
i = interval(d2, d1, "seconds").
d3 = d1.
d3 = add-interval(d3, i, "seconds").

if string(d1) <> "01/01/1970 00:00:00.000+00:00" then message "a. d1 failed: " d1.
if string(d2) <> "06/23/2022 08:55:22.123+03:00" then message "a. d2 failed: " d2.
if string(d3) <> "06/23/2022 05:55:22.000+00:00" then message "a. d3 failed: " d3.

d1 = datetime-tz("01-01-1970T00:00:00.000+00:00").
d2 = datetime-tz("02-23-2022T08:55:22.123+02:00").
i = interval(d2, d1, "seconds").
d3 = d1.
d3 = add-interval(d3, i, "seconds").

if string(d1) <> "01/01/1970 00:00:00.000+00:00" then message "b. d1 failed: " d1.
if string(d2) <> "02/23/2022 08:55:22.123+02:00" then message "b. d2 failed: " d2.
if string(d3) <> "02/23/2022 06:55:22.000+00:00" then message "b. d3 failed: " d3.
```

```
d1 = datetime-tz("01-01-1970T00:00:00.000+00:00").
d2 = datetime-tz("02-23-2022T08:55:22.123-07:00").
i = interval(d2, d1, "seconds").
d3 = d1.
d3 = add-interval(d3, i, "seconds").

if string(d1) <> "01/01/1970 00:00:00.000+00:00" then message "c. d1 failed: " d1.
if string(d2) <> "02/23/2022 08:55:22.123-07:00" then message "c. d2 failed: " d2.
if string(d3) <> "02/23/2022 15:55:22.000+00:00" then message "c. d3 failed: " d3.

d1 = datetime-tz("01-01-1970T00:00:00.000+00:00").
d2 = datetime-tz("06-23-2022T08:55:22.123-07:00").
i = interval(d2, d1, "seconds").
d3 = d1.
d3 = add-interval(d3, i, "seconds").

if string(d1) <> "01/01/1970 00:00:00.000+00:00" then message "d. d1 failed: " d1.
if string(d2) <> "06/23/2022 08:55:22.123-07:00" then message "d. d2 failed: " d2.
if string(d3) <> "06/23/2022 15:55:22.000+00:00" then message "d. d3 failed: " d3.
```

Lets consider the 'epoch time', which is 01-01-1970T00:00:00.000+00:00. If you adjust this to local time, you will get a 01/01/1970 02:00:00.000 - see how local time (EEST) is +3, and it adds only 2 hours - DST is not considered at all.

I agree we need a common denominator - but this should be UTC, so both instances are in the +00:00 timezone, and you can work with them easily, without having to consider DST or the local server's timezone. Because the difference of two datetimetz instances should be the same, regardless of the current server's timezone.

A client had problems when their machine was in PDT (-07:00) timezone. Using the 'local date' approach in FWD, this would mean 01-01-1970T00:00:00.000+00:00 becomes 12/31/1969 16:00:00.000, because getDefaultTimezoneOffset doesn't account for DST.

So getDefaultTimezoneOffset either needs to account for DST - which can bring other problems, what about SESSION:TIMEZONE and -useSessionTZ 1? - or just use UTC time in interval.

If you disagree, please show why INTERVAL function requires local time conversion and not UTC.

#9 - 06/23/2022 05:23 PM - Ovidiu Maxiniuc

The a and d tests were failing because the computed interval i was incorrect. The add-interval seemed to work fine.

The toLocalDateTime() is still in use because all date and datetime instances are assumed to be on local timezone, and any assignment between them and datetime-tz require this operation.

I added a new method for converting any kind of date instance to its UTC datetime counterparts (equivalent to a datetime-tz with a +00:00 offset. Note that these instances are to be used only locally, internally in FWD because they are not fully compatible with standard instances which are assumed to be on local time-zone. As result, both Constantin's testcases and mines passed. More than that, these special datetime s allows for a more compact code.

Committed revision 13950.