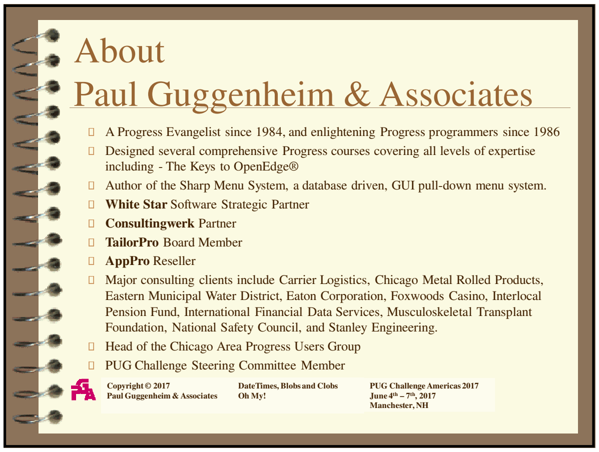
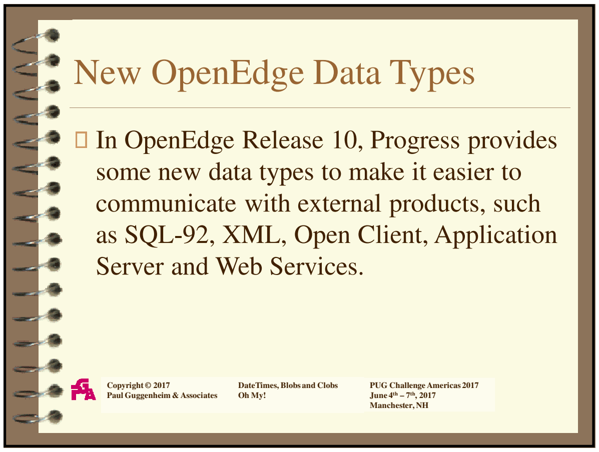
Datetimes, Blobs and Clobs Oh My!

Paul Guggenheim
Paul Guggenheim & Associates, Inc.



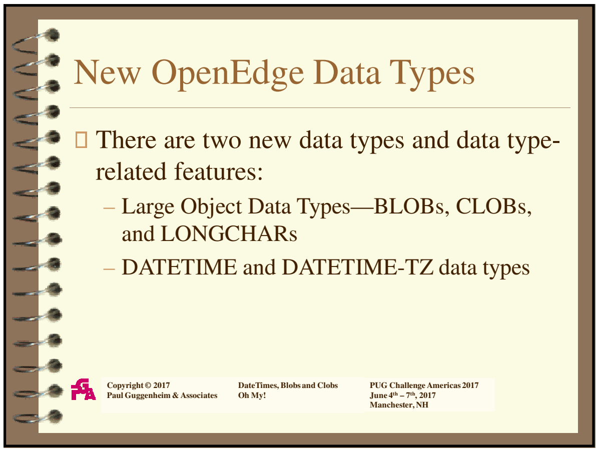
About Paul Guggenheim & Associates

- A Progress Evangelist since 1984, and enlightening Progress programmers since 1986
- Designed several comprehensive Progress courses covering all levels of expertise including - The Keys to OpenEdge®
- Author of the Sharp Menu System, a database driven, GUI pull-down menu system.
- **White Star** Software Strategic Partner
- **Consultingwerk** Partner
- **TailorPro** Board Member
- **AppPro** Reseller
- Major consulting clients include Carrier Logistics, Chicago Metal Rolled Products, Eastern Municipal Water District, Eaton Corporation, Foxwoods Casino, Interlocal Pension Fund, International Financial Data Services, Musculoskeletal Transplant Foundation, National Safety Council, and Stanley Engineering.
- Head of the Chicago Area Progress Users Group
- PUG Challenge Steering Committee Member



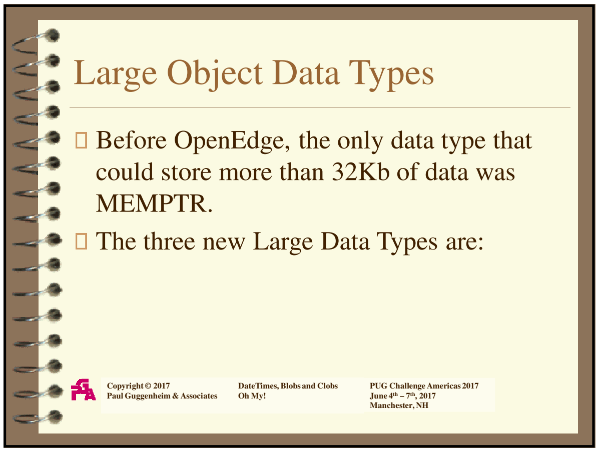
New OpenEdge Data Types

- In OpenEdge Release 10, Progress provides some new data types to make it easier to communicate with external products, such as SQL-92, XML, Open Client, Application Server and Web Services.



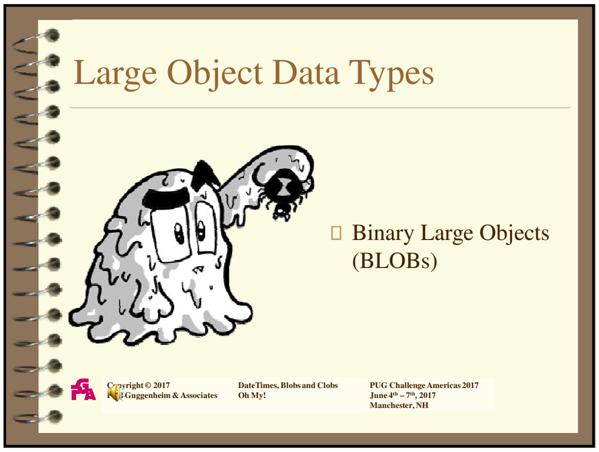
New OpenEdge Data Types

- There are two new data types and data type-related features:
  - Large Object Data Types—BLOBs, CLOBs, and LONGCHARs
  - DATETIME and DATETIME-TZ data types



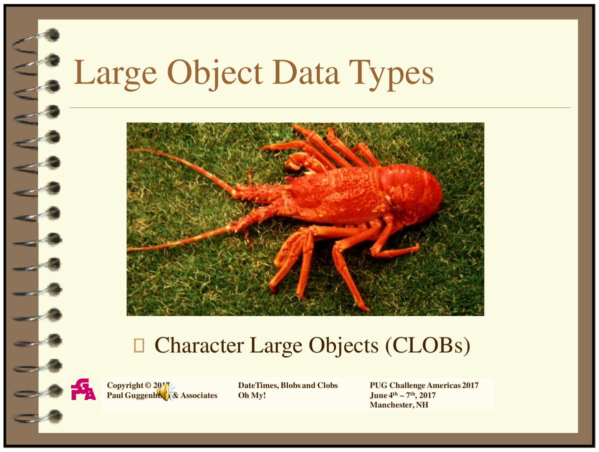
Large Object Data Types

- Before OpenEdge, the only data type that could store more than 32Kb of data was MEMPTR.
- The three new Large Data Types are:



Large Object Data Types

- Binary Large Objects (BLOBs)



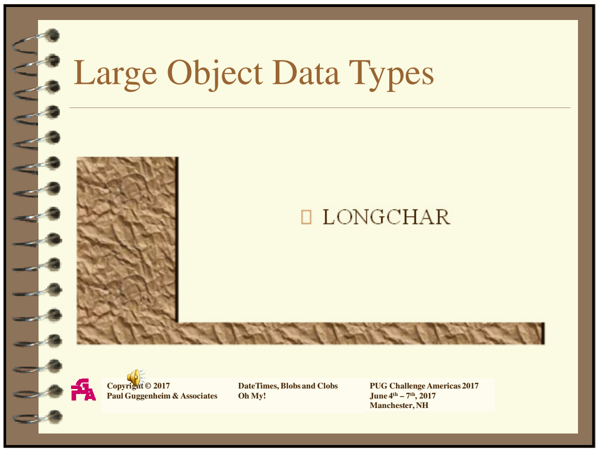
Large Object Data Types

- Character Large Objects (CLOBs)

Copyright © 2017
Paul Guggenheim & Associates

DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



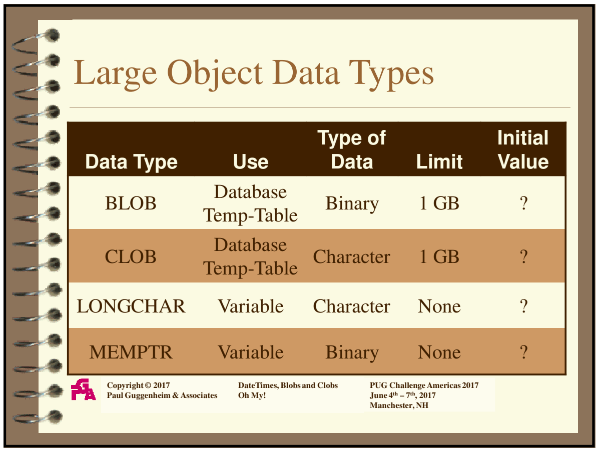
Large Object Data Types

- LONGCHAR

Copyright © 2017
Paul Guggenheim & Associates

DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



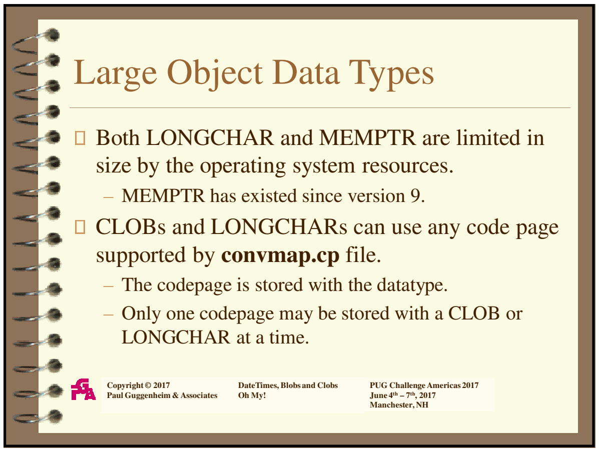
Large Object Data Types

Data Type	Use	Type of Data	Limit	Initial Value
BLOB	Database Temp-Table	Binary	1 GB	?
CLOB	Database Temp-Table	Character	1 GB	?
LONGCHAR	Variable	Character	None	?
MEMPTR	Variable	Binary	None	?

Copyright © 2017
Paul Guggenheim & Associates

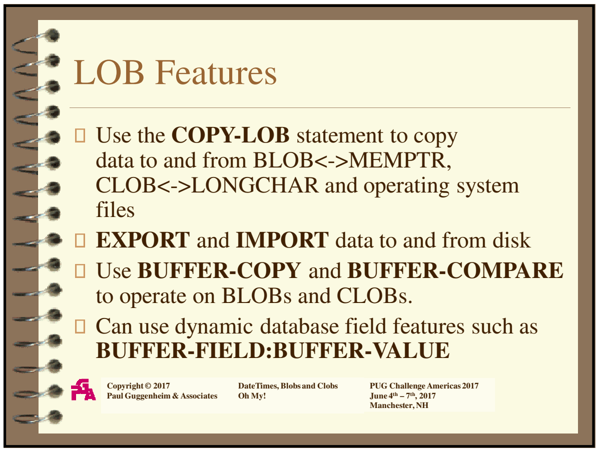
DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



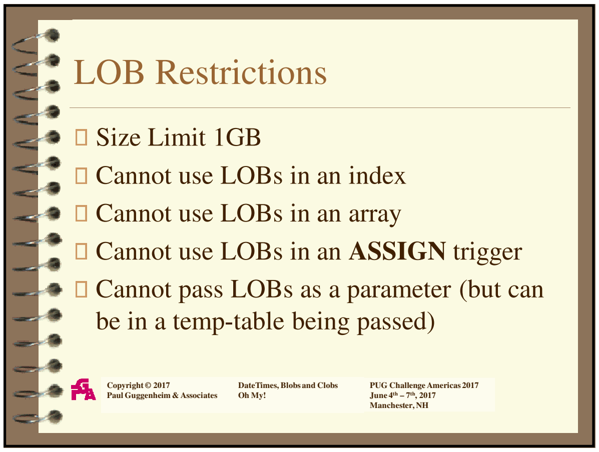
Large Object Data Types

- Both LONGCHAR and MEMPTR are limited in size by the operating system resources.
  - MEMPTR has existed since version 9.
- CLOBs and LONGCHARs can use any code page supported by **convmap.cp** file.
  - The codepage is stored with the datatype.
  - Only one codepage may be stored with a CLOB or LONGCHAR at a time.



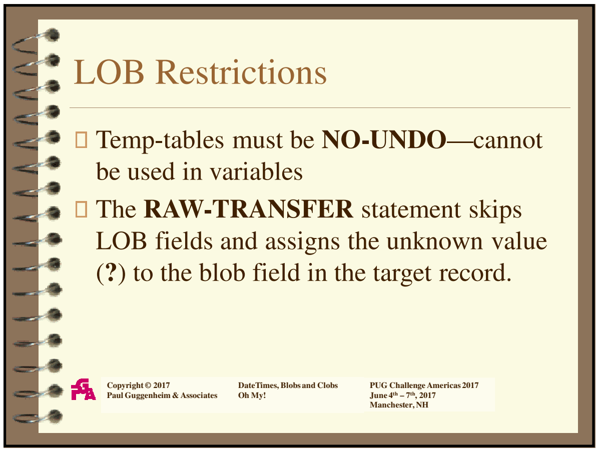
LOB Features

- Use the **COPY-LOB** statement to copy data to and from BLOB<->MEMPTR, CLOB<->LONGCHAR and operating system files
- **EXPORT** and **IMPORT** data to and from disk
- Use **BUFFER-COPY** and **BUFFER-COMPARE** to operate on BLOBs and CLOBs.
- Can use dynamic database field features such as **BUFFER-FIELD:BUFFER-VALUE**



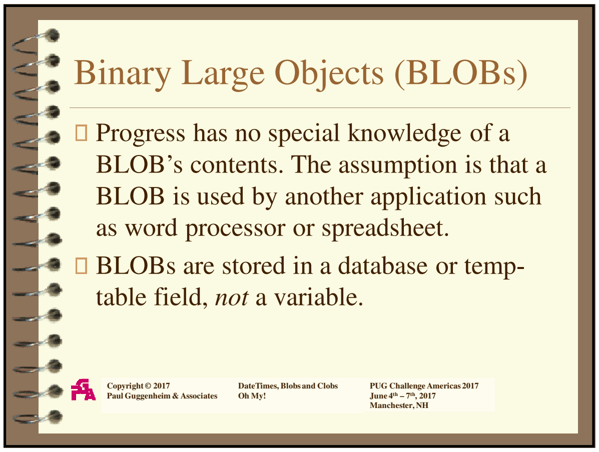
LOB Restrictions

- Size Limit 1GB
- Cannot use LOBs in an index
- Cannot use LOBs in an array
- Cannot use LOBs in an **ASSIGN** trigger
- Cannot pass LOBs as a parameter (but can be in a temp-table being passed)



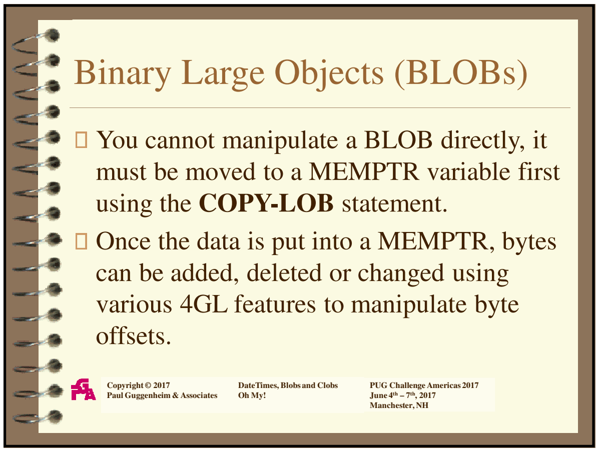
LOB Restrictions

- Temp-tables must be **NO-UNDO**—cannot be used in variables
- The **RAW-TRANSFER** statement skips LOB fields and assigns the unknown value (**?**) to the blob field in the target record.



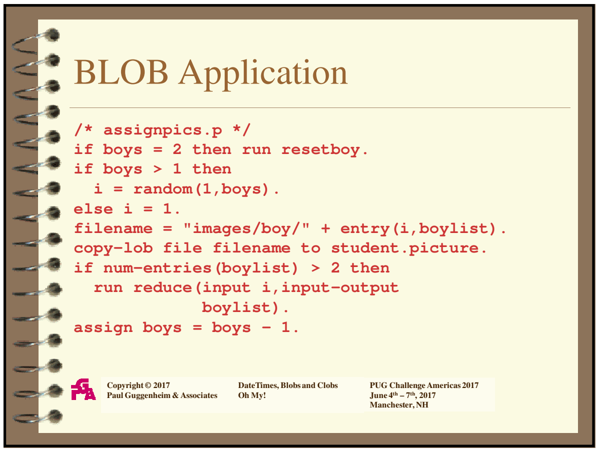
Binary Large Objects (BLOBs)

- Progress has no special knowledge of a BLOB’s contents. The assumption is that a BLOB is used by another application such as word processor or spreadsheet.
- BLOBs are stored in a database or temp-table field, *not* a variable.



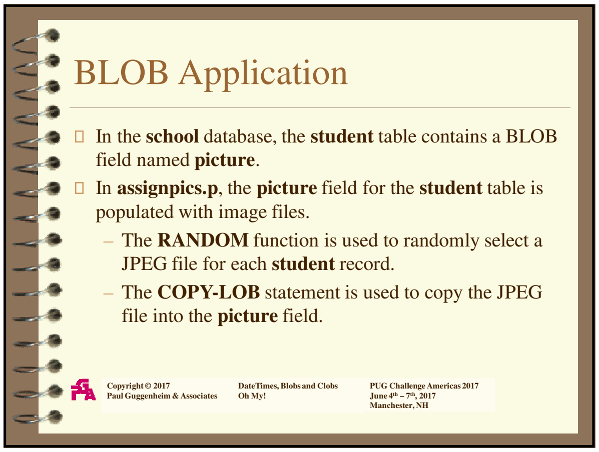
Binary Large Objects (BLOBs)

- You cannot manipulate a BLOB directly, it must be moved to a MEMPTR variable first using the **COPY-LOB** statement.
- Once the data is put into a MEMPTR, bytes can be added, deleted or changed using various 4GL features to manipulate byte offsets.



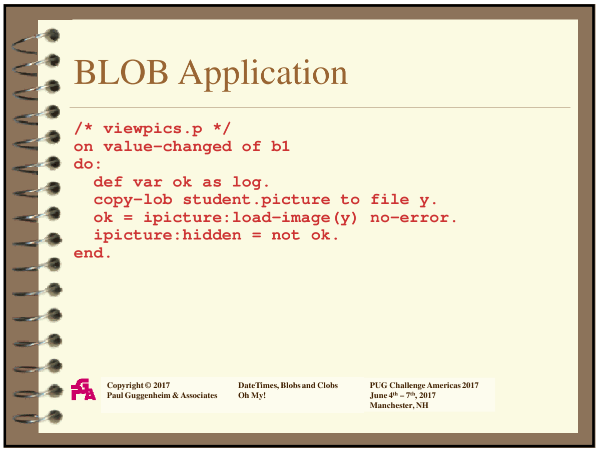
BLOB Application

```
/* assignpics.p */
if boys = 2 then run resetboy.
if boys > 1 then
  i = random(1,boys).
else i = 1.
filename = "images/boy/" + entry(i,boylist).
copy-lob file filename to student.picture.
if num-entries(boylist) > 2 then
  run reduce(input i,input-output
             boylist).
assign boys = boys - 1.
```



BLOB Application

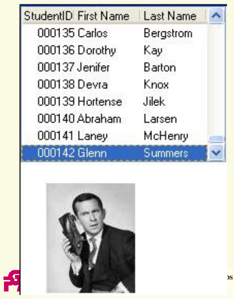
- In the **school** database, the **student** table contains a BLOB field named **picture**.
- In **assignpics.p**, the **picture** field for the **student** table is populated with image files.
  - The **RANDOM** function is used to randomly select a JPEG file for each **student** record.
  - The **COPY-LOB** statement is used to copy the JPEG file into the **picture** field.



BLOB Application

```
/* viewpics.p */
on value-changed of b1
do:
  def var ok as log.
  copy-lob student.picture to file y.
  ok = ipicture:load-image(y) no-error.
  ipicture:hidden = not ok.
end.
```

BLOB Application



StudentID	First Name	Last Name
000135	Carlos	Bergstrom
000136	Dorothy	Kay
000137	Jenifer	Barton
000138	Devra	Knox
000139	Hortense	Jilek
000140	Abraham	Larsen
000141	Laney	McHenry
000142	Glenn	Summers

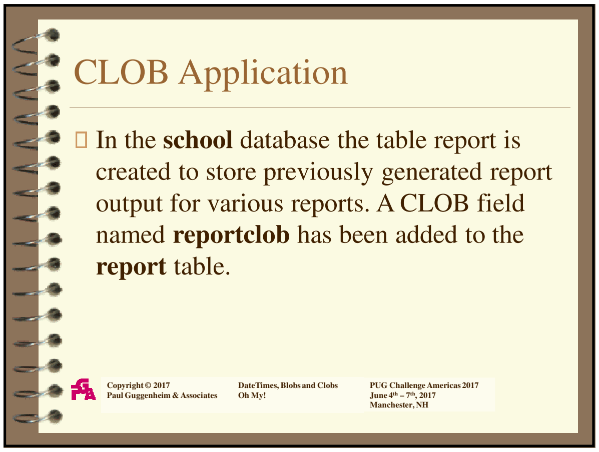
- In **viewpics.p**, the **picture** field for the **student** table is displayed.
  - The BLOB field cannot be loaded into an image widget directly.

BLOB Application

- In **viewpics.p**, the **COPY-LOB** statement copies the image data from the **picture** field to a temporary file named ***y***. The file ***y*** is then loaded into the image widget with the **LOAD-IMAGE** method.

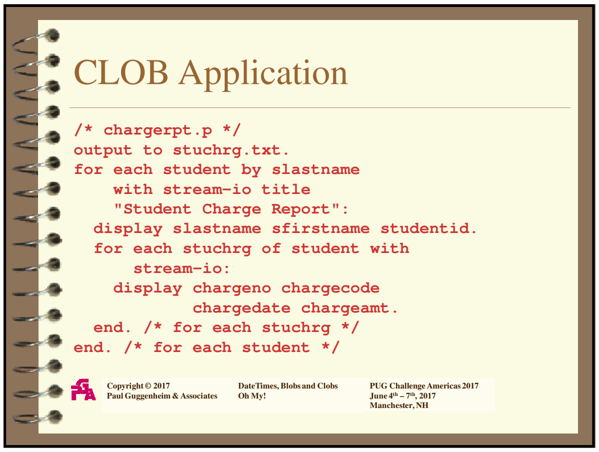
Character Large Objects (CLOBs)

- CLOBs contain only character data and can only be used on database and temp-table fields.
- CLOBs can be displayed in a large editor and the data may be manipulated in a LONGCHAR.



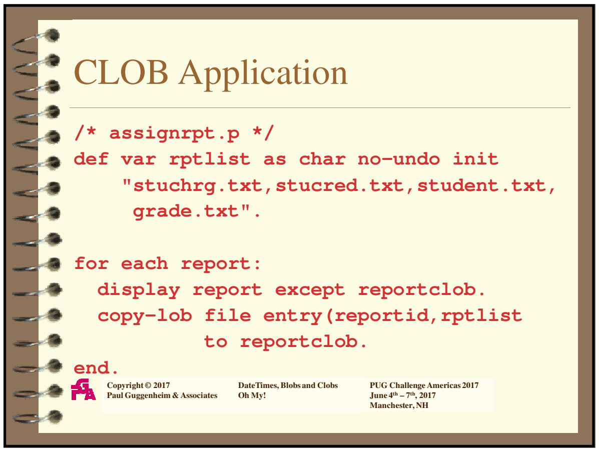
CLOB Application

- In the **school** database the table report is created to store previously generated report output for various reports. A CLOB field named **reportclob** has been added to the **report** table.



CLOB Application

```
/* chargerpt.p */
output to stuchrg.txt.
for each student by slastname
    with stream-io title
    "Student Charge Report":
  display slastname sfirstname studentid.
  for each stuchrg of student with
      stream-io:
    display chargeno chargecode
            chargedate chargeamt.
  end. /* for each stuchrg */
end. /* for each student */
```



CLOB Application

```
/* assignrpt.p */
def var rptlist as char no-undo init
    "stuchrg.txt,stucred.txt,student.txt,
     grade.txt".

for each report:
  display report except reportclob.
  copy-lob file entry(reportid,rptlist
           to reportclob.
end.
```

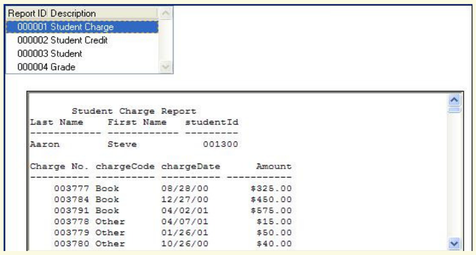
CLOB Application

- In **assignrpt.p**, the **reportclob** field for the **report** table is populated with the report output files.
  - The **reportlist** variable lists the report files that are to be loaded in the report records.
  - The **COPY-LOB** statement is used to copy the large text file into the report field.

CLOB Application

```
/* viewrpt.p */
def var ereport as char view-as
    editor large scrollbar-vertical
    inner-chars 70 inner-lines 14 font 0.
def var y as char init "tmp.txt".

on value-changed of b1
do:
  os-delete value(y).
  copy-lob report.reportclob to file y.
  ereport:read-file(y).
end.
```

CLOB Application



Report ID Description
000001 Student Charge
000002 Student Credit
000003 Student
000004 Grade

```
       Student Charge Report
Last Name    First Name   studentId
------------ ------------ ---------
Aaron        Steve           001300

Charge No. chargeCode chargeDate      Amount
---------- ---------- ---------- -----------
    003777 Book       08/28/00       $325.00
    003784 Book       12/27/00       $450.00
    003791 Book       04/02/01       $575.00
    003778 Other      04/07/01        $15.00
    003779 Other      01/26/01        $50.00
    003780 Other      10/26/00        $40.00
```

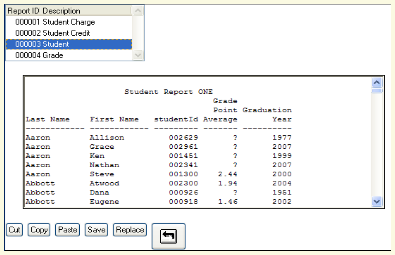
CLOB Application

- In **viewrpt.p**, the **reportclob** field for the report table is loaded into a large editor to be displayed.
  - The **COPY-LOB** statement is used to copy the contents of the **reportclob** record to a temporary disk file, and then the disk file is loaded into a large editor.

CLOB Application

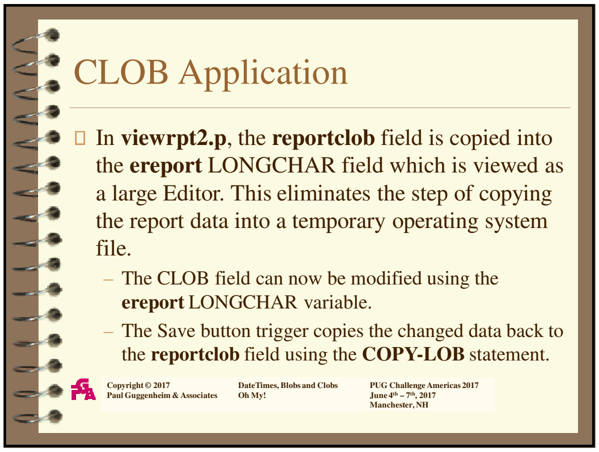
```
/* viewrpt2.p */
def var ereport as longchar view-as
    editor large scrollbar-vertical
    inner-chars 70 inner-lines 14 font 0.

on choose of bsave do:
  assign ereport.
  find current report exclusive-lock.
  copy-lob ereport to report.reportclob.
  find current report no-lock.
end. /* choose of bsave */
```

CLOB Application



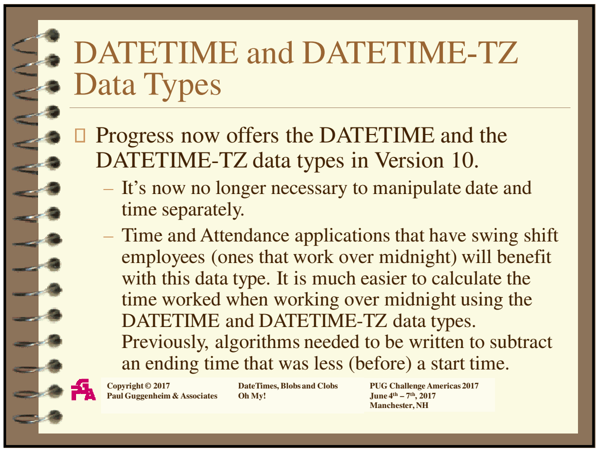
Report ID	Description
000001	Student Charge
000002	Student Credit
000003	Student
000004	Grade

```
                    Student Report ONE
                                        Grade
                                        Point Graduation
Last Name    First Name   studentId Average       Year
------------ ------------ --------- ------- ----------
Aaron        Allison         002629       ?       1977
Aaron        Grace           002961       ?       2007
Aaron        Ken             001451       ?       1999
Aaron        Nathan          002341       ?       2007
Aaron        Steve           001300    2.44       2000
Abbott       Atwood          002300    1.94       2004
Abbott       Dana            000926       ?       1951
Abbott       Eugene          000918    1.46       2002
```



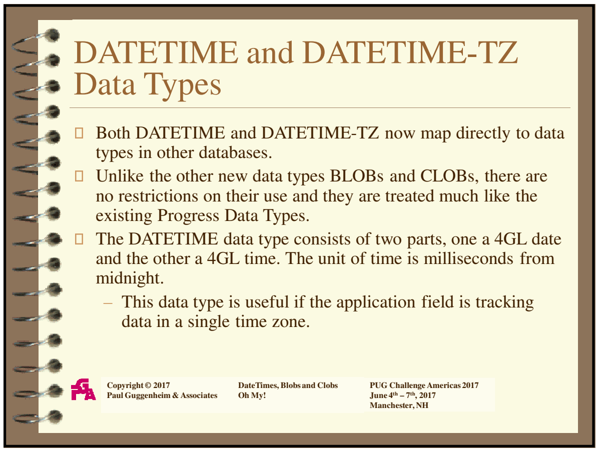
CLOB Application

- In **viewrpt2.p**, the **reportclob** field is copied into the **ereport** LONGCHAR field which is viewed as a large Editor. This eliminates the step of copying the report data into a temporary operating system file.
  - The CLOB field can now be modified using the **ereport** LONGCHAR variable.
  - The Save button trigger copies the changed data back to the **reportclob** field using the **COPY-LOB** statement.



DATETIME and DATETIME-TZ Data Types

- Progress now offers the DATETIME and the DATETIME-TZ data types in Version 10.
  - It's now no longer necessary to manipulate date and time separately.
  - Time and Attendance applications that have swing shift employees (ones that work over midnight) will benefit with this data type. It is much easier to calculate the time worked when working over midnight using the DATETIME and DATETIME-TZ data types. Previously, algorithms needed to be written to subtract an ending time that was less (before) a start time.



DATETIME and DATETIME-TZ Data Types

- Both DATETIME and DATETIME-TZ now map directly to data types in other databases.
- Unlike the other new data types BLOBs and CLOBs, there are no restrictions on their use and they are treated much like the existing Progress Data Types.
- The DATETIME data type consists of two parts, one a 4GL date and the other a 4GL time. The unit of time is milliseconds from midnight.
  - This data type is useful if the application field is tracking data in a single time zone.

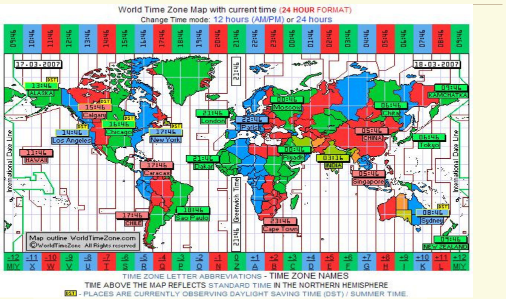
DATETIME and DATETIME-TZ Data Types

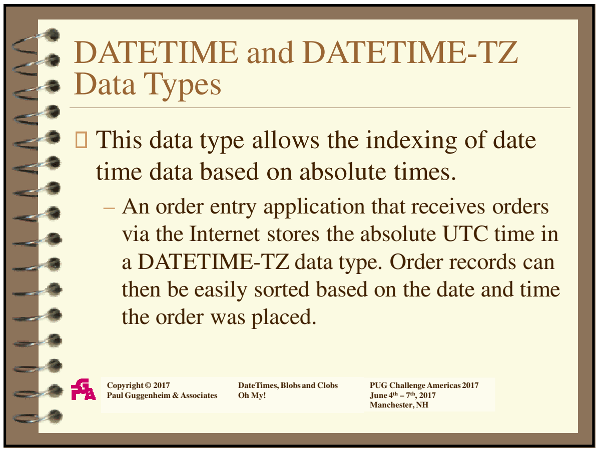
- The DATETIME-TZ data type consists of three parts, the 4GL date and time as with the DATETIME, plus an integer representing the time zone offset from Coordinated Universal Time (UTC) in minutes.
  - ***Coordinated Universal Time*** is a high precision atomic time standard. It was formerly known as Greenwich Mean Time (GMT).
  - Time zones around the world are expressed as positive or negative offsets from UTC.

DATETIME and DATETIME-TZ Data Types

- The 0 UTC time zone is known as Zulu time zone. London, United Kingdom is on the Zulu time zone. The “clock” in Greenwich, England is used as the standard clock for international reference of time in communications, military, aviation, maritime and other activities that cross time zones.
  - For example, in the United States, the Central Time Zone (S) is represented as UTC-06:00 or -360. It is 360 minutes earlier than UTC or GMT.

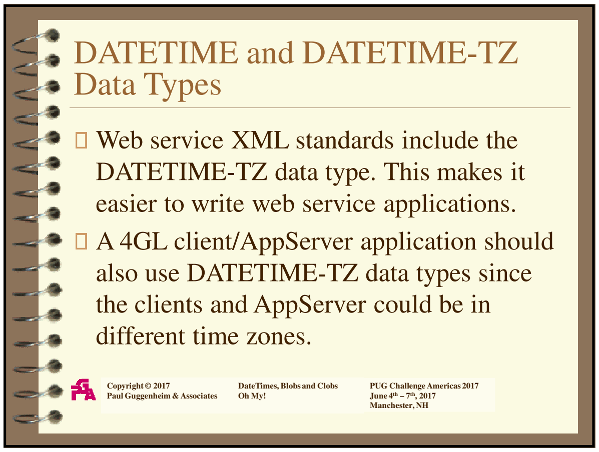
World Time Zones


World Time Zone Map with current time (24 HOUR FORMAT)
Change Time mode: 12 hours (AM/PM) or 24 hours
17-03-2007
18-03-2007
International Date Line
Greenwich Time
Los Angeles
Calgary
Chicago
New York
HAWAII
ALASKA
London
Paris
Moscow
Dakar
Riyadh
Cape Town
Sao Paulo
CHILE
CHINA
Tokyo
Singapore
Sydney
NEW ZEALAND
KAMCHATKA
Map outline WorldTimeZone.com
©WorldTimeZone All Rights reserved
-12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12
M/Y X W V U T S R Q P O N Z A B C D E F G H I K L M/Y
TIME ZONE LETTER ABBREVIATIONS - TIME ZONE NAMES
TIME ABOVE THE MAP REFLECTS STANDARD TIME IN THE NORTHERN HEMISPHERE
DST - PLACES ARE CURRENTLY OBSERVING DAYLIGHT SAVING TIME (DST) / SUMMER TIME.



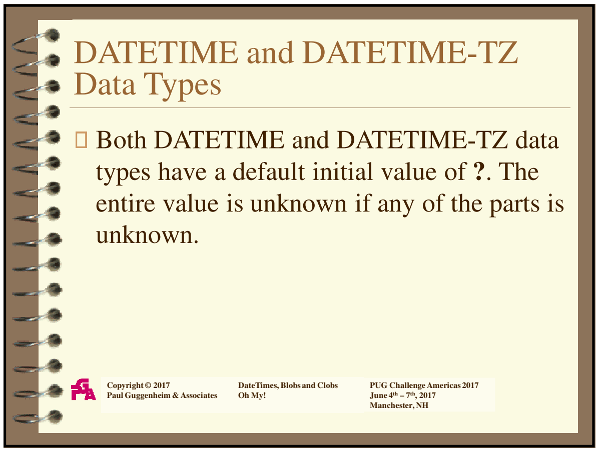
DATETIME and DATETIME-TZ Data Types

- This data type allows the indexing of date time data based on absolute times.
  - An order entry application that receives orders via the Internet stores the absolute UTC time in a DATETIME-TZ data type. Order records can then be easily sorted based on the date and time the order was placed.



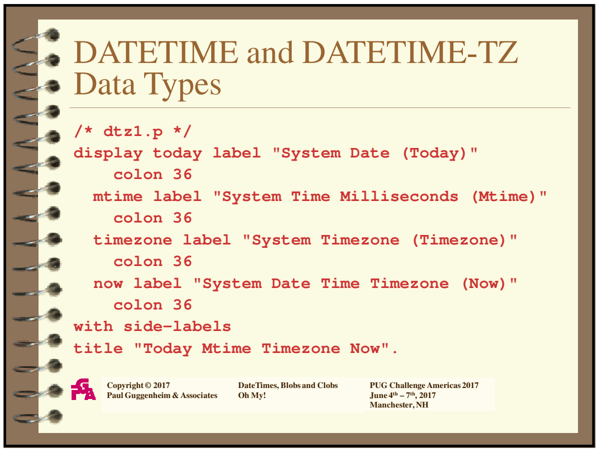
DATETIME and DATETIME-TZ Data Types

- Web service XML standards include the DATETIME-TZ data type. This makes it easier to write web service applications.
- A 4GL client/AppServer application should also use DATETIME-TZ data types since the clients and AppServer could be in different time zones.



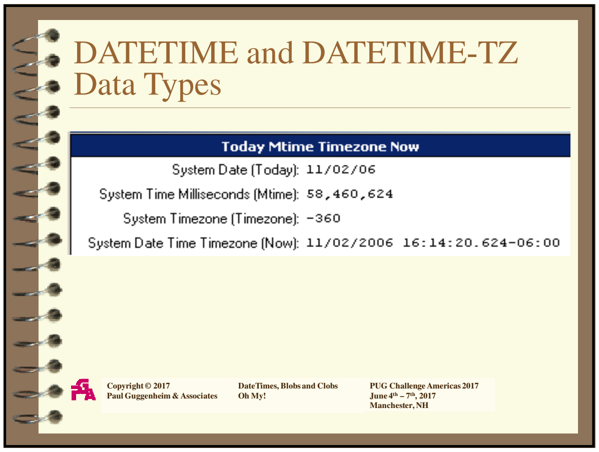
DATETIME and DATETIME-TZ Data Types

- Both DATETIME and DATETIME-TZ data types have a default initial value of **?**. The entire value is unknown if any of the parts is unknown.



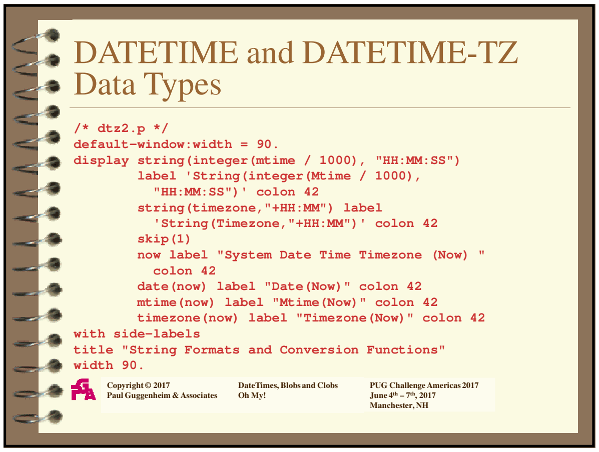
DATETIME and DATETIME-TZ Data Types

```
/* dtz1.p */
display today label "System Date (Today)"
    colon 36
  mtime label "System Time Milliseconds (Mtime)"
    colon 36
  timezone label "System Timezone (Timezone)"
    colon 36
  now label "System Date Time Timezone (Now)"
    colon 36
with side-labels
title "Today Mtime Timezone Now".
```



DATETIME and DATETIME-TZ Data Types

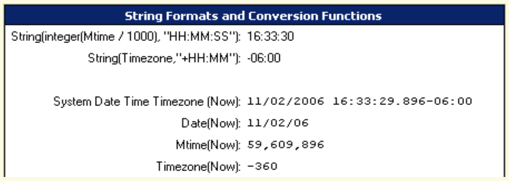
```
             System Date (Today): 11/02/06
  System Time Milliseconds (Mtime): 58,460,624
        System Timezone (Timezone): -360
 System Date Time Timezone (Now): 11/02/2006 16:14:20.624-06:00
```



DATETIME and DATETIME-TZ Data Types

```
/* dtz2.p */
default-window:width = 90.
display string(integer(mtime / 1000), "HH:MM:SS")
        label 'String(integer(Mtime / 1000),
          "HH:MM:SS")' colon 42
        string(timezone,"+HH:MM") label
          'String(Timezone,"+HH:MM")' colon 42
        skip(1)
        now label "System Date Time Timezone (Now) "
          colon 42
        date(now) label "Date(Now)" colon 42
        mtime(now) label "Mtime(Now)" colon 42
        timezone(now) label "Timezone(Now)" colon 42
with side-labels
title "String Formats and Conversion Functions"
width 90.
```

DATETIME and DATETIME-TZ Data Types



String Formats and Conversion Functions

```
String(integer(Mtime / 1000), "HH:MM:SS"):  16:33:30
           String(Timezone,"+HH:MM"):  -06:00

     System Date Time Timezone (Now):  11/02/2006 16:33:29.896-06:00
                          Date(Now):  11/02/06
                         Mtime(Now):  59,609,896
                      Timezone(Now):  -360
```

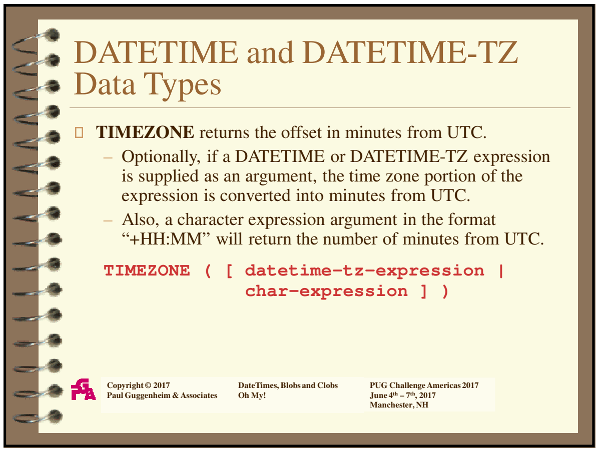
DATETIME and DATETIME-TZ Data Types

- In **dtz1.p** and **dtz2.p**, 3 new functions are introduced, **MTIME**, **TIMEZONE** and **NOW**.

DATETIME and DATETIME-TZ Data Types

- **MTIME** returns the number of milliseconds from midnight.
  - Optionally, if a DATETIME or DATETIME-TZ expression is supplied as an argument, the time portion of the expression is converted into milliseconds from midnight.

```
MTIME ( [ datetime-expression ] )
```



DATETIME and DATETIME-TZ Data Types

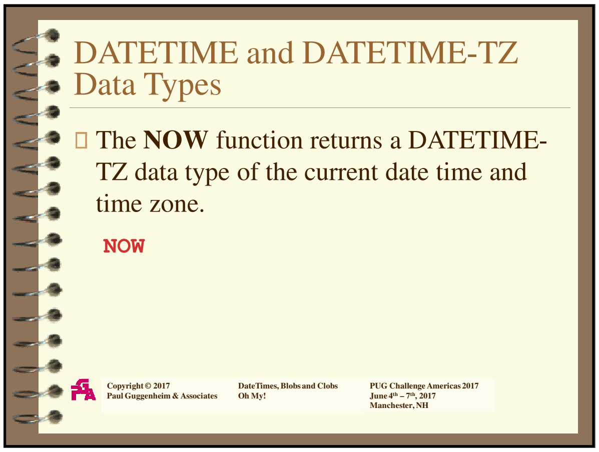
- **TIMEZONE** returns the offset in minutes from UTC.
  - Optionally, if a DATETIME or DATETIME-TZ expression is supplied as an argument, the time zone portion of the expression is converted into minutes from UTC.
  - Also, a character expression argument in the format “+HH:MM” will return the number of minutes from UTC.

```
TIMEZONE ( [ datetime-tz-expression |
             char-expression ] )
```

Copyright © 2017
Paul Guggenheim & Associates

DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



DATETIME and DATETIME-TZ Data Types

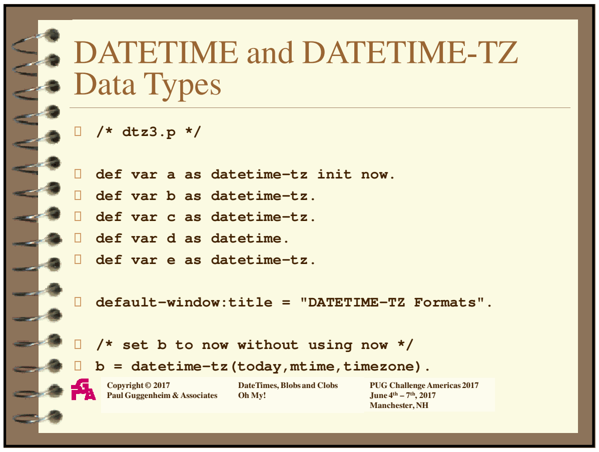
- The **NOW** function returns a DATETIME-TZ data type of the current date time and time zone.

```
NOW
```

Copyright © 2017
Paul Guggenheim & Associates

DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



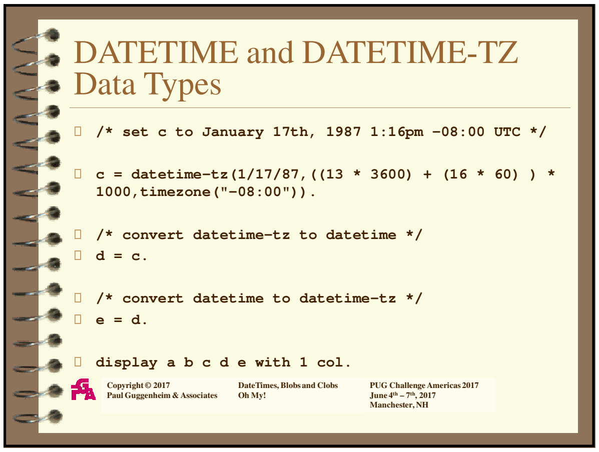
DATETIME and DATETIME-TZ Data Types

```
/* dtz3.p */

def var a as datetime-tz init now.
def var b as datetime-tz.
def var c as datetime-tz.
def var d as datetime.
def var e as datetime-tz.

default-window:title = "DATETIME-TZ Formats".

/* set b to now without using now */
b = datetime-tz(today,mtime,timezone).
```

Copyright © 2017
Paul Guggenheim & Associates

DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



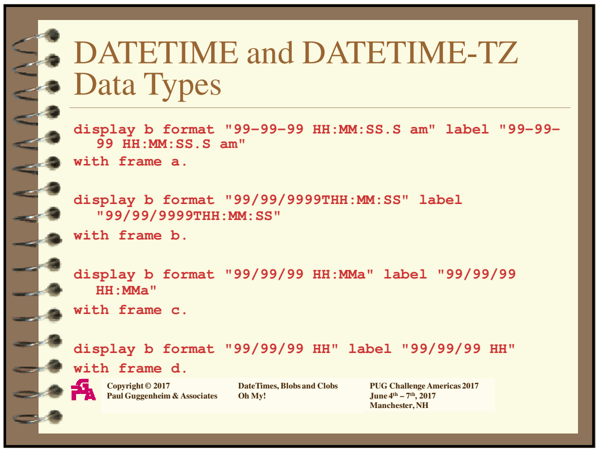
DATETIME and DATETIME-TZ Data Types

```
/* set c to January 17th, 1987 1:16pm -08:00 UTC */

c = datetime-tz(1/17/87,((13 * 3600) + (16 * 60) ) *
  1000,timezone("-08:00")).

/* convert datetime-tz to datetime */
d = c.

/* convert datetime to datetime-tz */
e = d.

display a b c d e with 1 col.
```

Copyright © 2017
Paul Guggenheim & Associates

DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



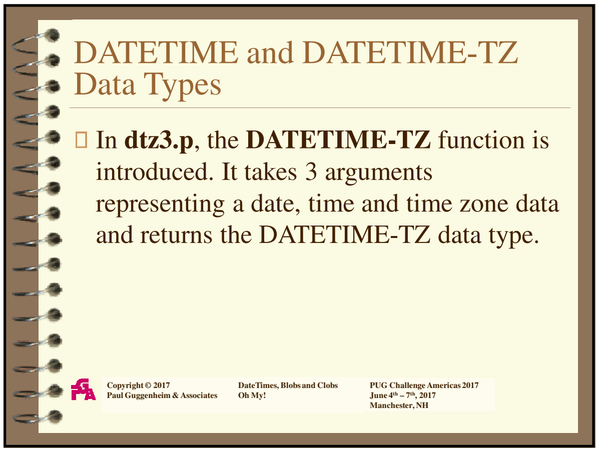
DATETIME and DATETIME-TZ Data Types

```
display b format "99-99-99 HH:MM:SS.S am" label "99-99-
  99 HH:MM:SS.S am"
with frame a.

display b format "99/99/9999THH:MM:SS" label
  "99/99/9999THH:MM:SS"
with frame b.

display b format "99/99/99 HH:MMa" label "99/99/99
  HH:MMa"
with frame c.

display b format "99/99/99 HH" label "99/99/99 HH"
with frame d.
```

Copyright © 2017
Paul Guggenheim & Associates

DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



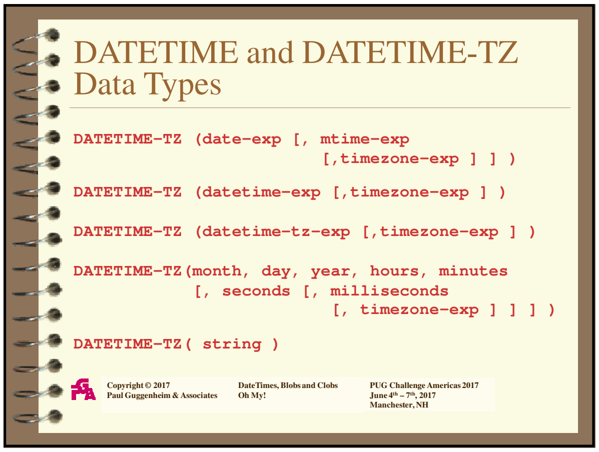
DATETIME and DATETIME-TZ Data Types

- In **dtz3.p**, the **DATETIME-TZ** function is introduced. It takes 3 arguments representing a date, time and time zone data and returns the DATETIME-TZ data type.

Copyright © 2017
Paul Guggenheim & Associates

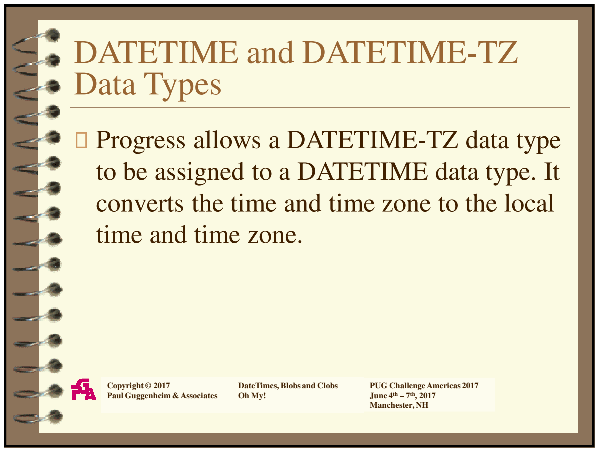
DateTimes, Blobs and Clobs
Oh My!

PUG Challenge Americas 2017
June 4th – 7th, 2017
Manchester, NH



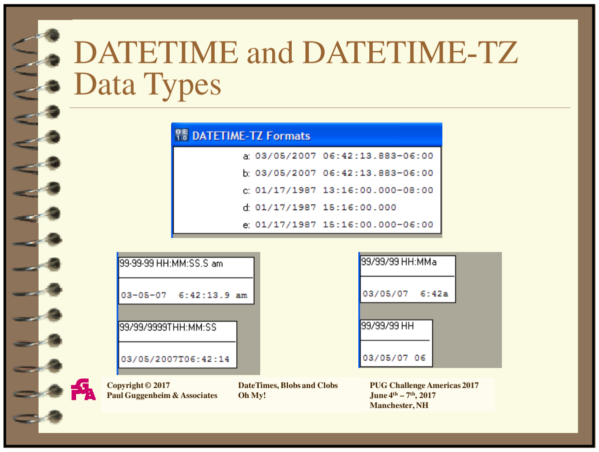
DATETIME and DATETIME-TZ Data Types

```
DATETIME-TZ (date-exp [, mtime-exp
                         [,timezone-exp ] ] )

DATETIME-TZ (datetime-exp [,timezone-exp ] )

DATETIME-TZ (datetime-tz-exp [,timezone-exp ] )

DATETIME-TZ(month, day, year, hours, minutes
            [, seconds [, milliseconds
                          [, timezone-exp ] ] ] )

DATETIME-TZ( string )
```



DATETIME and DATETIME-TZ Data Types

- Progress allows a DATETIME-TZ data type to be assigned to a DATETIME data type. It converts the time and time zone to the local time and time zone.



DATETIME and DATETIME-TZ Data Types

```
a: 03/05/2007 06:42:13.883-06:00
b: 03/05/2007 06:42:13.883-06:00
c: 01/17/1987 13:16:00.000-08:00
d: 01/17/1987 15:16:00.000
e: 01/17/1987 15:16:00.000-06:00
```

```
99-99-99 HH:MM:SS.S am
03-05-07  6:42:13.9 am

99/99/9999THH:MM:SS
03/05/2007T06:42:14

99/99/99 HH:MMa
03/05/07  6:42a

99/99/99 HH
03/05/07 06
```

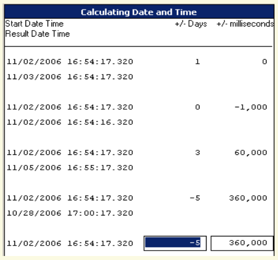
DATETIME and DATETIME-TZ Data Types

- The bottom half of the display shows the various display formats that you can use with DATETIME or DATETIME-TZ data types.

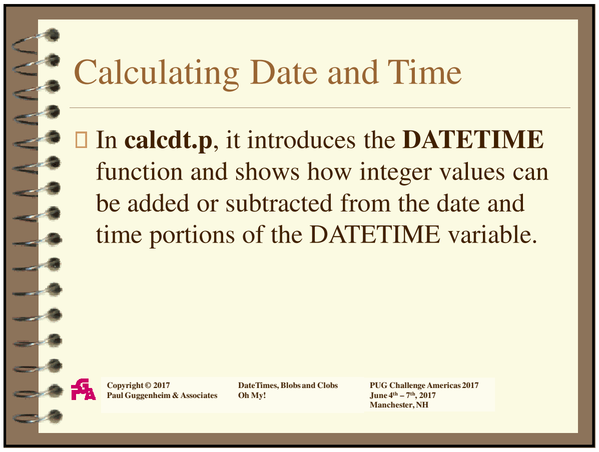
Calculating Date and Time

```
/* calcdt.p */
def var a as datetime label "Start Date Time”
    init now.
def var b as int      label "+/- Days".
def var c as int      label "+/- milliseconds".
def var d as datetime label "Result Date Time".

repeat with title "Calculating Date and Time":
  display a.
  update b c.
  d = datetime(date(a) + b,mtime(a) + c).
  display d skip(1).
end.
```

Calculating Date and Time

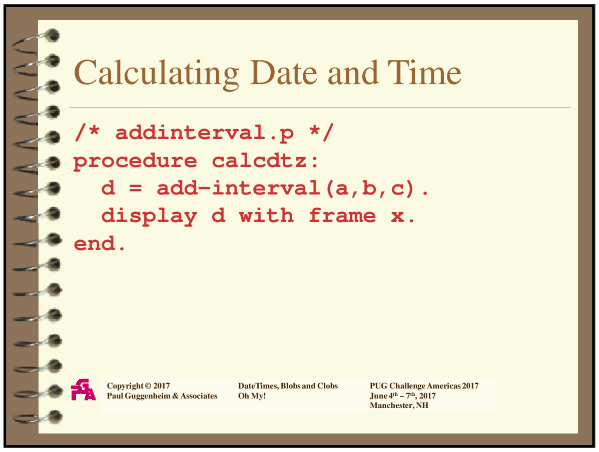


```
Calculating Date and Time
Start Date Time                       +/- Days  +/- milliseconds
Result Date Time

11/02/2006 16:54:17.320                      1                 0
11/03/2006 16:54:17.320

11/02/2006 16:54:17.320                      0            -1,000
11/02/2006 16:54:16.320

11/02/2006 16:54:17.320                      3            60,000
11/05/2006 16:55:17.320

11/02/2006 16:54:17.320                     -5           360,000
10/28/2006 17:00:17.320

11/02/2006 16:54:17.320                     -5           360,000
```



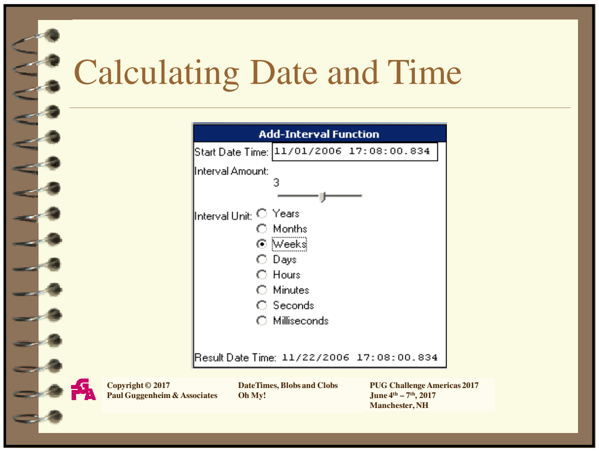
Calculating Date and Time

- In **calcdt.p**, it introduces the **DATETIME** function and shows how integer values can be added or subtracted from the date and time portions of the DATETIME variable.



Calculating Date and Time

```
/* addinterval.p */
procedure calcdtz:
  d = add-interval(a,b,c).
  display d with frame x.
end.
```



Calculating Date and Time

Add-Interval Function

Start Date Time: 11/01/2006 17:08:00.834

Interval Amount: 3

Interval Unit: ○ Years ○ Months ● Weeks ○ Days ○ Hours ○ Minutes ○ Seconds ○ Milliseconds

Result Date Time: 11/22/2006 17:08:00.834

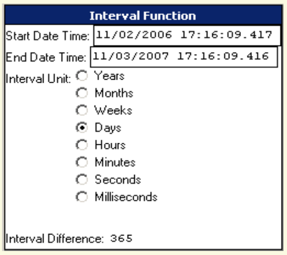
Calculating Date and Time

- The **addinterval.p** program showcases the **ADD-INTERVAL** function. The function allows the specification of a date time unit of either Years, Months, Weeks, Days, Hours, Minutes, Seconds, Milliseconds.
  - The amount in the specified units will be added to the start DATETIME to produce the result DATETIME .
  - Any change in the DATETIME time, interval amount or interval unit will instantly change the result DATETIME .

Calculating Date and Time

```
/* interval.p */
procedure calcdtz:
  b = interval(d,a,c).
  display b with frame x.
end.
```

Calculating Date and Time



Interval Function

Start Date Time: 11/02/2006 17:16:09.417

End Date Time: 11/03/2007 17:16:09.416

Interval Unit:
○ Years
○ Months
○ Weeks
● Days
○ Hours
○ Minutes
○ Seconds
○ Milliseconds

Interval Difference: 365

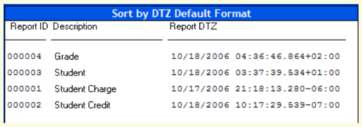
Calculating Date and Time

- The **interval.p** program showcases the interval function. The function allows the specification of a DATETIME unit of either Years, Months, Weeks, Days, Hours, Minutes, Seconds, Milliseconds.
  - The difference between the end DATETIME and the start DATETIME will be shown in the specified units.
  - Any change in the start DATETIME, end DATETIME or interval unit will instantly change the interval difference.

Overriding DATETIME-TZ Default Format

```
/* reportdtz1.p */

default-window:width = 90.
for each report by reportdtz
    with width 85
    title "Sort by DTZ Default Format":
  display reportid reportdescription
          reportdtz.
end.
```

Overriding DATETIME-TZ Default Format



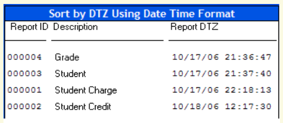
```
                Sort by DTZ Default Format
Report ID Description          Report DTZ
--------- -------------------- ------------------------------
000004    Grade                10/18/2006 04:36:46.864+02:00
000003    Student              10/18/2006 03:37:39.534+01:00
000001    Student Charge       10/17/2006 21:18:13.280-06:00
000002    Student Credit       10/18/2006 10:17:29.539-07:00
```

- In **reportdtz1.p**, the report is sorted by **reportdtz**. However, it is difficult to tell that it is sorted this way because of the default DATETIME-TZ format.

Overriding DATETIME-TZ Default Format

```
/* reportdtz2.p */

for each report by reportdtz with title
    "Sort by DTZ Using Date Time Format":
  display reportid reportdescription
          reportdtz format "99/99/99 hh:mm:ss".
end.
```

Overriding DATETIME-TZ Default Format



```
          Sort by DTZ Using Date Time Format
Report ID Description              Report DTZ
--------- ------------------------ -----------------

000004    Grade                    10/17/06 21:36:47
000003    Student                  10/17/06 21:37:40
000001    Student Charge           10/17/06 22:18:13
000002    Student Credit           10/18/06 12:17:30
```

- In **reportdtz2.p**, the report is sorted by **reportdtz**. But this time, the **reportdtz** field is using a format without the time zone. Whenever you display a DATETIME-TZ field with a DATETIME format, Progress converts the time to the local time zone. As you can see, it is much easier to understand that the report is sorted by **reportdtz**.

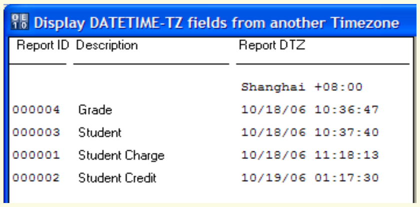
Displaying DATETIME-TZ Field from Another Timezone

- When an application has clients located in different time zones, and the time-source is the server, how does the application display the DATETIME-TZ field in the time zone of each client?
  - Progress provides a settable SESSION attribute called the DISPLAY-TIMEZONE attribute.
  - Set the attribute with the number of minutes from UTC.

Displaying DATETIME-TZ Field from Another Timezone

```
/* reportdtz3.p */
do with frame a:
form reportid reportdescription reportdtz
format "99/99/99 hh:mm:ss” with frame a down.

session:display-timezone = 8 * 60.

display "Shanghai "
        + string(session:display-timezone,"+HH:MM")
        format "x(20)" @ reportdtz.

down 1.

for each report by reportdtz with frame a:
  display reportid reportdescription reportdtz.
  down.
end.
```

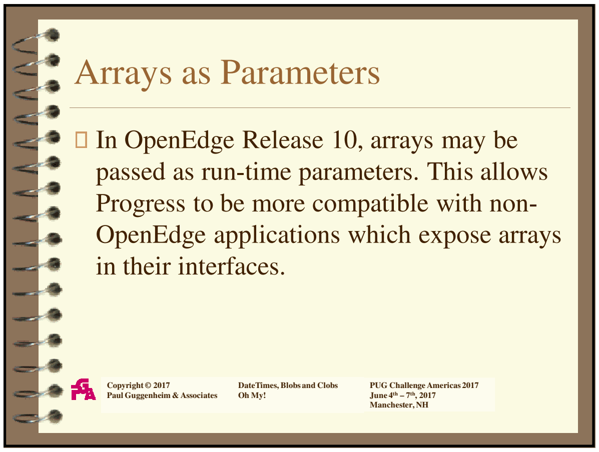
Displaying DATETIME-TZ Field from Another Timezone



```
Report ID  Description          Report DTZ

                                Shanghai +08:00
000004     Grade                10/18/06 10:36:47
000003     Student              10/18/06 10:37:40
000001     Student Charge       10/18/06 11:18:13
000002     Student Credit       10/19/06 01:17:30
```

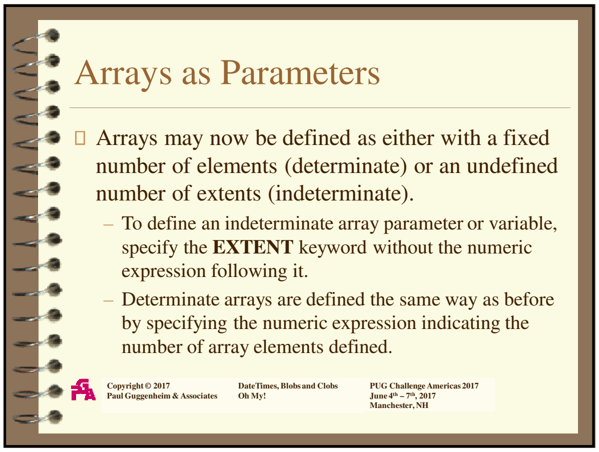
Displaying DATETIME-TZ Field from Another Timezone

- In **reportdtz3.p**, the DISPLAY-TIMEZONE is set to 480 minutes for Shanghai, China, since it is 8 time zones east of UTC.
  - Second, the DISPLAY-TIMEZONE is set to -480 minutes for San Francisco, California, since it is 8 time zones west of UTC.
  - Third, the DISPLAY-TIMEZONE is set to -300 minutes for Chicago, Illinois Daylight Savings Time, since it is 5 time zones west of UTC.
  - Lastly, the DISPLAY-TIMEZONE is set to -360 minutes for Chicago Central Standard Time, since it is 6 time zones west of UTC.



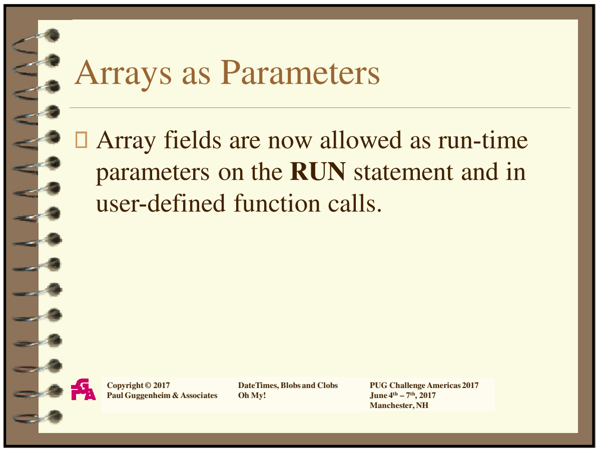
Arrays as Parameters

- In OpenEdge Release 10, arrays may be passed as run-time parameters. This allows Progress to be more compatible with non-OpenEdge applications which expose arrays in their interfaces.



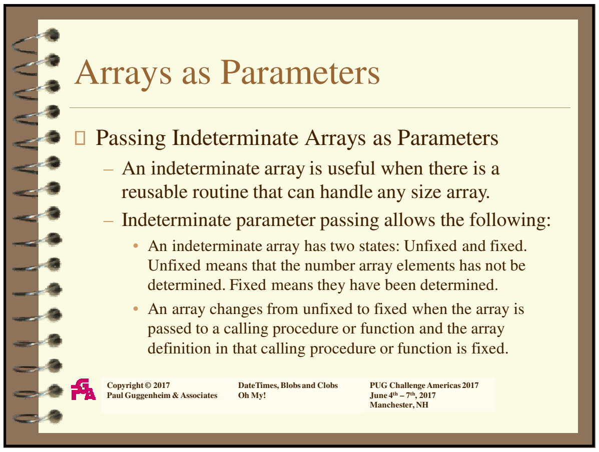
Arrays as Parameters

- Arrays may now be defined as either with a fixed number of elements (determinate) or an undefined number of extents (indeterminate).
  - To define an indeterminate array parameter or variable, specify the **EXTENT** keyword without the numeric expression following it.
  - Determinate arrays are defined the same way as before by specifying the numeric expression indicating the number of array elements defined.



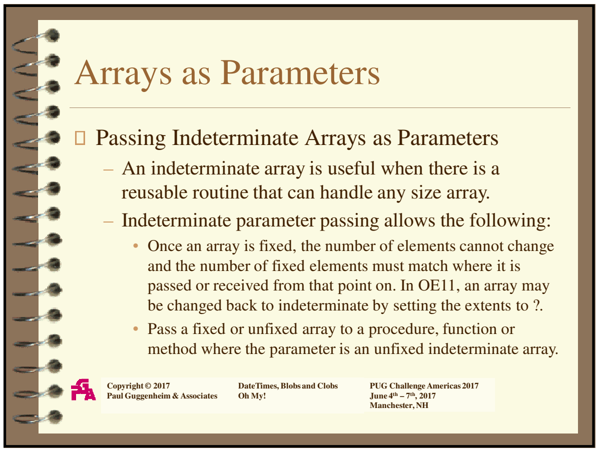
Arrays as Parameters

- Array fields are now allowed as run-time parameters on the **RUN** statement and in user-defined function calls.



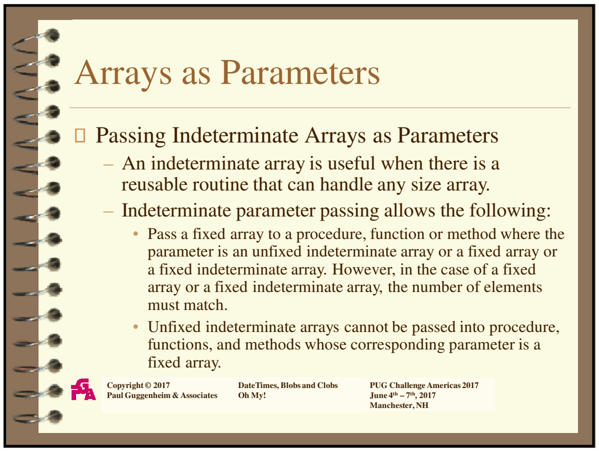
Arrays as Parameters

- Passing Indeterminate Arrays as Parameters
  - An indeterminate array is useful when there is a reusable routine that can handle any size array.
  - Indeterminate parameter passing allows the following:
    - An indeterminate array has two states: Unfixed and fixed. Unfixed means that the number array elements has not be determined. Fixed means they have been determined.
    - An array changes from unfixed to fixed when the array is passed to a calling procedure or function and the array definition in that calling procedure or function is fixed.



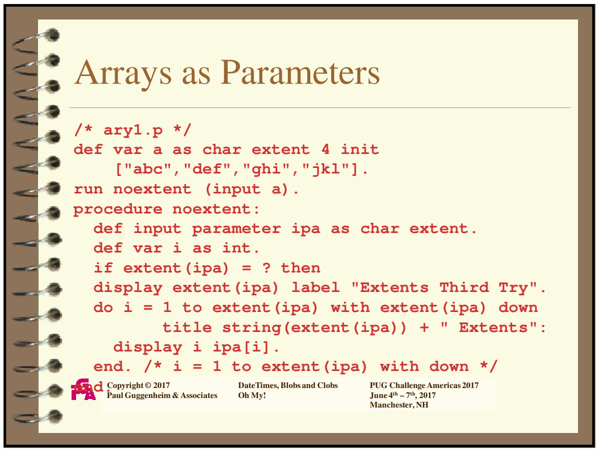
Arrays as Parameters

- Passing Indeterminate Arrays as Parameters
  - An indeterminate array is useful when there is a reusable routine that can handle any size array.
  - Indeterminate parameter passing allows the following:
    - Once an array is fixed, the number of elements cannot change and the number of fixed elements must match where it is passed or received from that point on. In OE11, an array may be changed back to indeterminate by setting the extents to ?.
    - Pass a fixed or unfixed array to a procedure, function or method where the parameter is an unfixed indeterminate array.



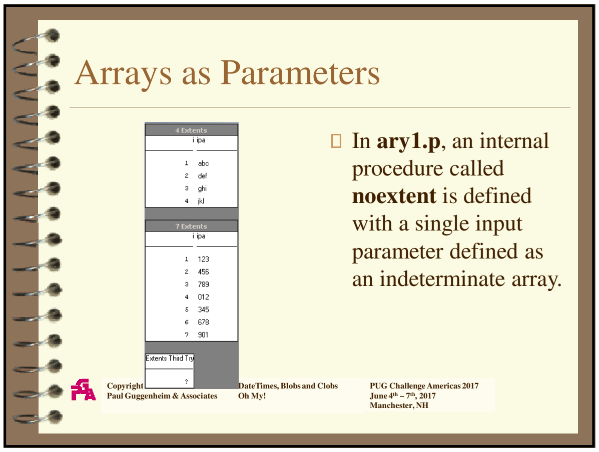
Arrays as Parameters

- Passing Indeterminate Arrays as Parameters
  - An indeterminate array is useful when there is a reusable routine that can handle any size array.
  - Indeterminate parameter passing allows the following:
    - Pass a fixed array to a procedure, function or method where the parameter is an unfixed indeterminate array or a fixed array or a fixed indeterminate array. However, in the case of a fixed array or a fixed indeterminate array, the number of elements must match.
    - Unfixed indeterminate arrays cannot be passed into procedure, functions, and methods whose corresponding parameter is a fixed array.



Arrays as Parameters

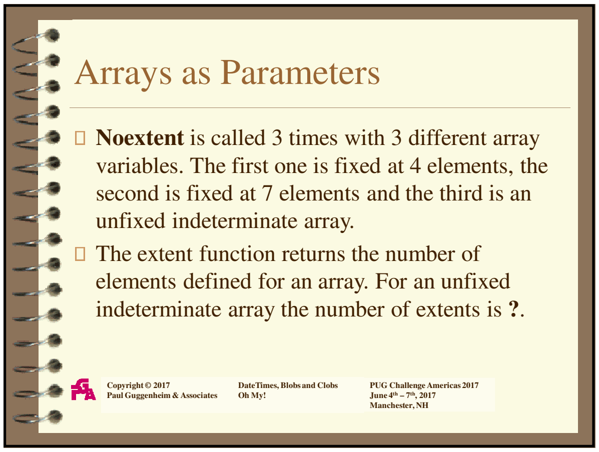
```
/* ary1.p */
def var a as char extent 4 init
    ["abc","def","ghi","jkl"].
run noextent (input a).
procedure noextent:
  def input parameter ipa as char extent.
  def var i as int.
  if extent(ipa) = ? then
  display extent(ipa) label "Extents Third Try".
  do i = 1 to extent(ipa) with extent(ipa) down
         title string(extent(ipa)) + " Extents":
    display i ipa[i].
  end. /* i = 1 to extent(ipa) with down */
end.
```



Arrays as Parameters

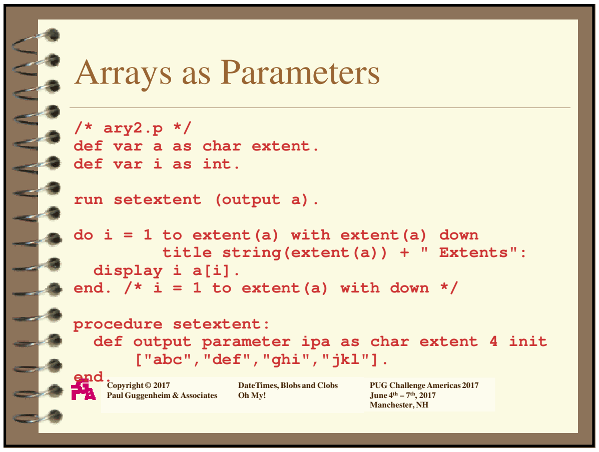
```
         4 Extents
         i ipa
        ---- ----
         1 abc
         2 def
         3 ghi
         4 jkl

         7 Extents
         i ipa
        ---- ----
         1 123
         2 456
         3 789
         4 012
         5 345
         6 678
         7 901

Extents Third Try
-----------------
                ?
```

- In **ary1.p**, an internal procedure called **noextent** is defined with a single input parameter defined as an indeterminate array.



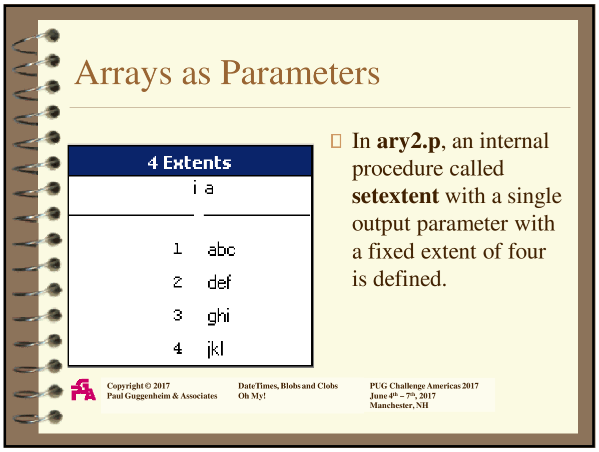
Arrays as Parameters

- **Noextent** is called 3 times with 3 different array variables. The first one is fixed at 4 elements, the second is fixed at 7 elements and the third is an unfixed indeterminate array.
- The extent function returns the number of elements defined for an array. For an unfixed indeterminate array the number of extents is **?**.



Arrays as Parameters

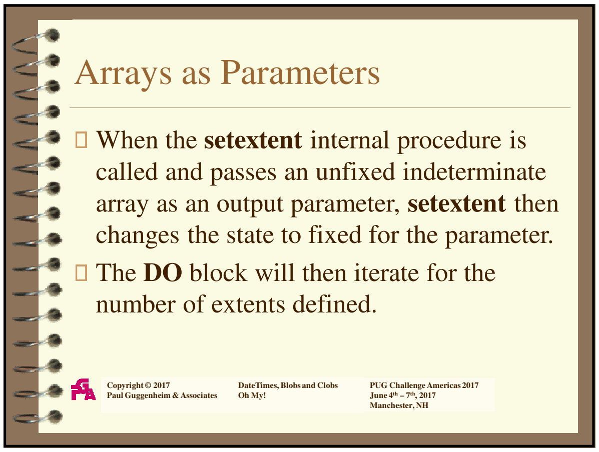
```
/* ary2.p */
def var a as char extent.
def var i as int.

run setextent (output a).

do i = 1 to extent(a) with extent(a) down
         title string(extent(a)) + " Extents":
  display i a[i].
end. /* i = 1 to extent(a) with down */

procedure setextent:
  def output parameter ipa as char extent 4 init
      ["abc","def","ghi","jkl"].
end.
```



Arrays as Parameters

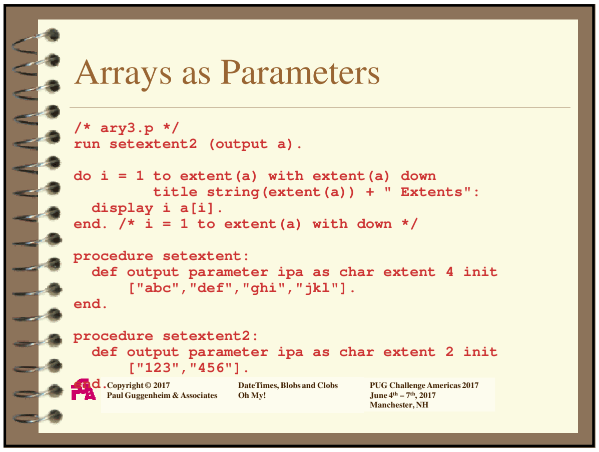
```
4 Extents
       i a
  ---------- ----------
       1 abc
       2 def
       3 ghi
       4 jkl
```

- In **ary2.p**, an internal procedure called **setextent** with a single output parameter with a fixed extent of four is defined.



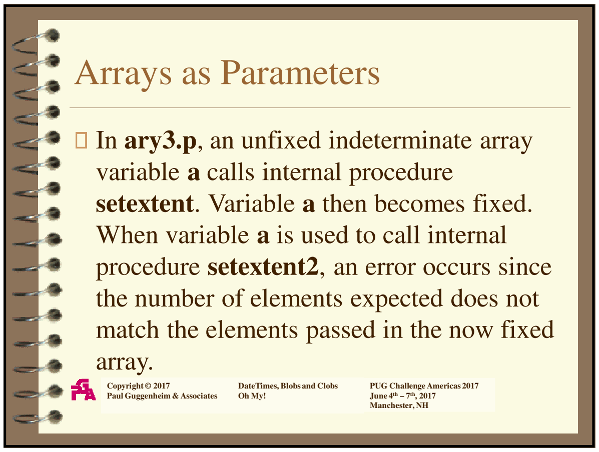
Arrays as Parameters

- When the **setextent** internal procedure is called and passes an unfixed indeterminate array as an output parameter, **setextent** then changes the state to fixed for the parameter.
- The **DO** block will then iterate for the number of extents defined.



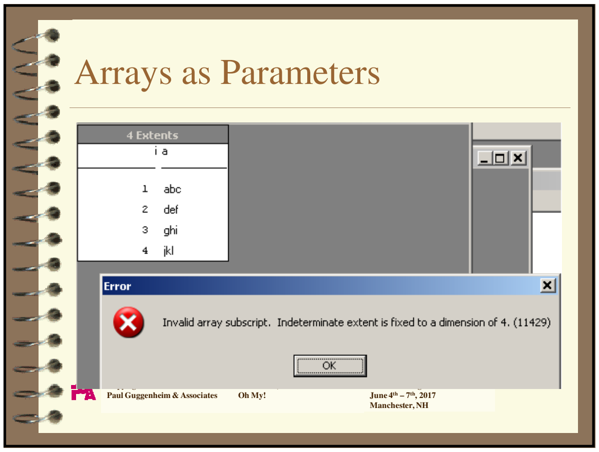
Arrays as Parameters

```
/* ary3.p */
run setextent2 (output a).

do i = 1 to extent(a) with extent(a) down
         title string(extent(a)) + " Extents":
  display i a[i].
end. /* i = 1 to extent(a) with down */

procedure setextent:
  def output parameter ipa as char extent 4 init
      ["abc","def","ghi","jkl"].
end.

procedure setextent2:
  def output parameter ipa as char extent 2 init
      ["123","456"].
end.
```



Arrays as Parameters

- In **ary3.p**, an unfixed indeterminate array variable **a** calls internal procedure **setextent**. Variable **a** then becomes fixed. When variable **a** is used to call internal procedure **setextent2**, an error occurs since the number of elements expected does not match the elements passed in the now fixed array.

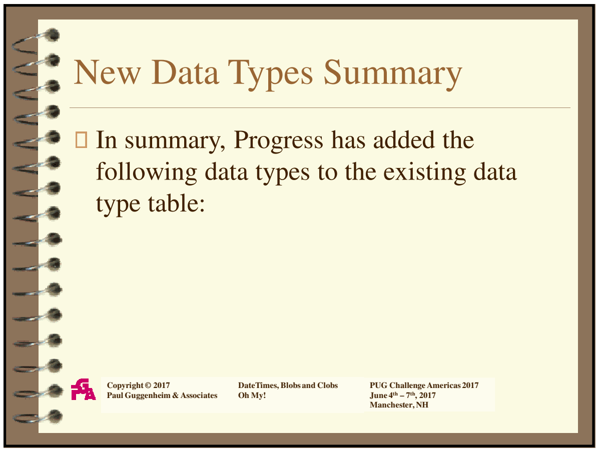


Arrays as Parameters

```
4 Extents
          i a
          
          1 abc
          2 def
          3 ghi
          4 jkl
```

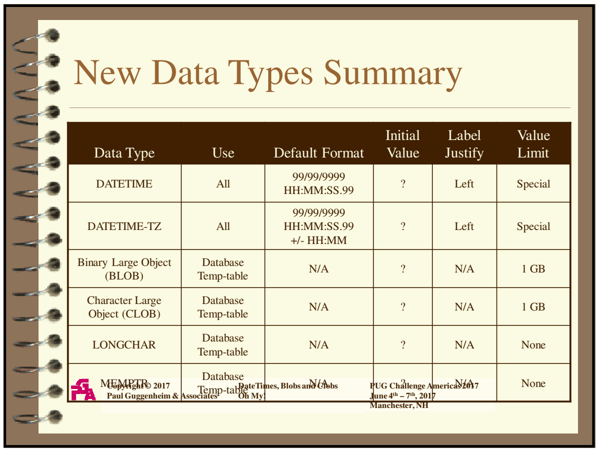
Error

Invalid array subscript. Indeterminate extent is fixed to a dimension of 4. (11429)



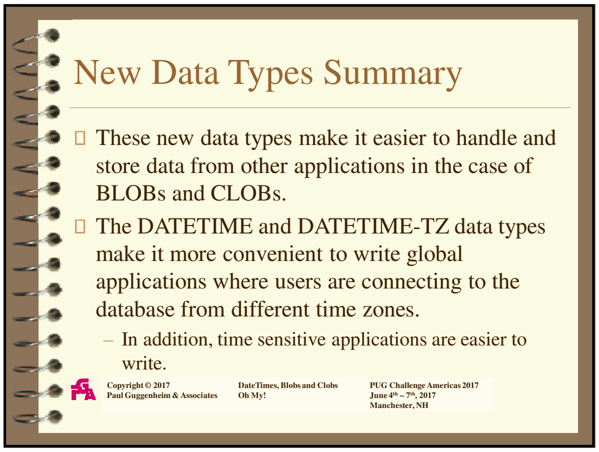
New Data Types Summary

- In summary, Progress has added the following data types to the existing data type table:



New Data Types Summary

Data Type	Use	Default Format	Initial Value	Label Justify	Value Limit
DATETIME	All	99/99/9999 HH:MM:SS.99	?	Left	Special
DATETIME-TZ	All	99/99/9999 HH:MM:SS.99 +/- HH:MM	?	Left	Special
Binary Large Object (BLOB)	Database Temp-table	N/A	?	N/A	1 GB
Character Large Object (CLOB)	Database Temp-table	N/A	?	N/A	1 GB
LONGCHAR	Database Temp-table	N/A	?	N/A	None
MEMPTR	Database Temp-table	N/A	?	N/A	None



New Data Types Summary

- These new data types make it easier to handle and store data from other applications in the case of BLOBs and CLOBs.
- The DATETIME and DATETIME-TZ data types make it more convenient to write global applications where users are connecting to the database from different time zones.
  - In addition, time sensitive applications are easier to write.