New Progress Data Types
Paul Guggenheim

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
- 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:

  - Binary Large Objects (BLOBs)
Large Object Data Types

- Character Large Objects (CLOBs)

Large Object Data Types

- LONGCHAR

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Use</th>
<th>Type of Data</th>
<th>Limit</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>Database</td>
<td>Binary</td>
<td>1 GB</td>
<td>?</td>
</tr>
<tr>
<td>CLOB</td>
<td>Temp-Table</td>
<td>Character</td>
<td>1 GB</td>
<td>?</td>
</tr>
<tr>
<td>LONGCHAR</td>
<td>Temp-Table</td>
<td>Character</td>
<td>None</td>
<td>?</td>
</tr>
<tr>
<td>MEMPTR</td>
<td>Variable</td>
<td>Binary</td>
<td>None</td>
<td>?</td>
</tr>
</tbody>
</table>
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 \texttt{convmap.cp} file.
  - The codepage is stored with the datatype.
  - Only one codepage may be stored with a CLOB or LONGCHAR at a time.

Copyright © 2017 Paul Guggenheim & Associates

LOB Features

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

Copyright © 2017 Paul Guggenheim & Associates

LOB Restrictions

- Size Limit 1GB
- Cannot use LOBs in an index
- Cannot use LOBs in an array
- Cannot use LOBs in an \texttt{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:load-image(y) ok.
end.
In `viewpics.p`, the picture field for the student table is displayed.
- The BLOB field cannot be loaded into an image widget directly.

- 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 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 sffirstname 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.
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

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

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

CLOB Application
**CLOB Application**

- In viewrt2.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.

- 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.

- 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 (CT) is represented as UTC-06:00 or -360. It is 360 minutes earlier than UTC or GMT.

---

**World Time Zones**

- [Image of a world time zone map]
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.

- 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.

- 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**

```plaintext
/* 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**

```
Today Mtime Timezone Now
System Date (Today) 22/02/00
System Time Milliseconds (Mtime) 19,460,624
System Timezone (Timezone) 800
System Date Time Timezone (Now) 22/02/00 10:14:10.624-06:09
```

**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
time(now) label "Time (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

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

**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.

```plaintext
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 "+HHMM" will return the number of minutes from UTC.

```
TIMEZONE { [ datetime-tz-expression | char-expression ] }
```

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

```
NOW
```

- Optionally, if a DATETIME or DATETIME-TZ expression

```
/* set b to now without using now */

b = datetime-tz(today, mtime, timezone).
```

- Default: set window: title = "DATETIME-TZ Formats".

```
/* 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".
```
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.

e = d.

display a b c d e with 1 col.

DATETIME and DATETIME-TZ Data Types

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

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.

New Progress Data Types
Paul Guggenheim
DATETIME and DATETIME-TZ Data Types

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

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

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

/* 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

<table>
<thead>
<tr>
<th>Date</th>
<th>+/- Days</th>
<th>+/- milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-06-04 00:00:00</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2023-06-04 00:00:00</td>
<td>0</td>
<td>1,000</td>
</tr>
<tr>
<td>2023-06-04 00:00:00</td>
<td>5</td>
<td>40,000</td>
</tr>
<tr>
<td>2023-06-04 00:00:00</td>
<td>-5</td>
<td>140,000</td>
</tr>
<tr>
<td>2023-06-04 00:00:00</td>
<td>100,000</td>
<td>140,000</td>
</tr>
</tbody>
</table>

Copyright © 2017 Paul Guggenheim & Associates
Calculating Date and Time

In `calctdt.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.

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

Calculating Date and Time

![Add Interval Function](image)

**Final Date Time:** 22/22/2004 21:13:02.914
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, or 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`.

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

Any change in the `DATETIME` time, interval amount or interval unit will instantly change the result `DATETIME`.

**Interval Function**

<table>
<thead>
<tr>
<th>Start Date Time</th>
<th>End Date Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/03/2007 07:10:00.414</td>
<td>03/03/2007 07:10:09.416</td>
</tr>
</tbody>
</table>

**Interval Unit**

- Days
- Hours
- Minutes
- Seconds
- Milliseconds

**Interval Difference 948**
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

```c
/* 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

- 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.

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.

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.
New Progress Data Types
Paul Guggenheim

Displaying DATETIME-TZ Field from Another Timezone

```c
// reportdtz3.p

do with frame a:
  from 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.
  done 1.
  for each report by reportdtz with frame a:
    display reportid reportdescription reportdtz.
  end.
```

– 7
– 7
– 7
– 7

/* reportdtz3.p */

– 7
– 7
– 7
– 7

Paul Guggenheim & Associates

DateTimes, Blobs and Clobs
June 4, 2017

Displaying DATETIME-TZ Field from Another Timezone

<table>
<thead>
<tr>
<th>Report ID (Description)</th>
<th>Report DTZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>000004 Grade</td>
<td>10/18/06 10:36:45</td>
</tr>
<tr>
<td>000005 Student</td>
<td>10/18/06 10:37:40</td>
</tr>
<tr>
<td>000001 Student Diego</td>
<td>10/18/06 11:12:13</td>
</tr>
<tr>
<td>000002 Student Dirk</td>
<td>10/19/06 01:17:30</td>
</tr>
</tbody>
</table>

– 7
– 7
– 7
– 7

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.

– 7
– 7
– 7
– 7

Copyright © 2017 Paul Guggenheim & Associates
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 of 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) = 0 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.

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

**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 ?.
New Progress Data Types
Paul Guggenheim

Arrays as Parameters

```haskell
/* 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 */
```

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

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

```haskell
internal procedure is called and passes an unfixed indeterminate array as an output parameter,
block will then iterate for the number of extents defined.
```

**Arrays as Parameters**

- 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 m[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. /* i = 1 to extent(a) with down */

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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Use</th>
<th>Default Format</th>
<th>Initial Value</th>
<th>Label</th>
<th>Justify</th>
<th>Value Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATETIME</td>
<td>All</td>
<td>MM/DD/YYYY</td>
<td>?</td>
<td>Left</td>
<td>Special</td>
<td></td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>All</td>
<td>MM/DD/YYYY</td>
<td>?</td>
<td>Left</td>
<td>Special</td>
<td></td>
</tr>
<tr>
<td>BLOB (EXTRA)</td>
<td>Database</td>
<td>N/A</td>
<td>?</td>
<td>N/A</td>
<td>1 GB</td>
<td></td>
</tr>
<tr>
<td>CLOB (EXTRA)</td>
<td>Database</td>
<td>N/A</td>
<td>?</td>
<td>N/A</td>
<td>1 GB</td>
<td></td>
</tr>
<tr>
<td>LONGCHAR</td>
<td>Database</td>
<td>N/A</td>
<td>?</td>
<td>N/A</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

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.