

PROGRESSIONS

In an earlier article, (issue 51) Jay Martin and Ted Duke explained the intricacies of record scoping and locking. To oversimplify in recapping their observations, a record lock is held for the longer of record scope and transaction scope. We are going to explore some of the implications of scope with regards to internal procedures. The facts may surprise you. First, let's start with a simple program to understand the compile listing option.

```
/* bracket1.p */
CustBlock:
for each customer:
  update name.
end.
```

If you perform a compile listing on this program you will get the following output (in part):

```
Line Blk
--- ---
1 1 for each customer:
2 1 update name.
3 end.

File Name Line Blk. Type Tran Blk. Label
-----
bracket1.p 0 Procedure No
bracket1.p 1 For Yes CustBlock
Buffers: sports2000.Customer
```

What is important to see here is that the customer buffer is scoped to the for each block. This means that the record scope and transaction scope begin and end with the same block, CustBlock.

So, what happens if we move the update to an internal procedure? Notice in the output below that the transaction scope is, as expected, shifted to the internal procedure. However, the record scope is shifted to the procedure block.

```
Line Blk
--- ---
1 CustBlock:
2 1 for each customer:
3 1 run updateCustNum.
4 end.
5
6 procedure UpdateCustnum:
7 update customer.custnum.
8 end.
```

```
File Name Line Blk. Type Tran Blk. Label
-----
bracket2.p 6 Procedure Yes Procedure
UpdateCustnum
bracket2.p 0 Procedure No
Buffers: sports2000.Customer
bracket2.p 2 For No CustBlock
```

Given what we have learned, this would imply that the record locks for each of the customers would endure until the end of the procedure. However, if you inspect promon, or write record locking test programs, you will find that there is only one record locked at a time. What a relief. At least, using this mechanism doesn't make things worse! Let's go back to a program with no internal procedure, and update both the customer record, and each order within the customer record. Here is a sample program to do that.

```
CustBlock:
for each customer:
  customer.creditlimit =
customer.creditlimit * 1.
  OrderBlock:
  for each order of customer:
    order.ordernum = order.ordernum * 1.
  end.
  if customer.custnum = 5 then do:
    message "Pausing" view-as alert-box.
    return.
  end.
end.
```

[continued on page 18](#)



Convert EDITOR to ROWS By Armando Pensado

It is a very common programming practice to use of the Editor widget. This object is a very practical of many screen designs. Nevertheless, when we think about printing its contents, some problems arise.

A programmer can trick the Stream output by including an Editor widget in the output; however, it will be considered as one single row at the point of a page eject in the document. When there is no control on how many lines of data will be printed before the contents fo the Editor is to be printed, the big block of text in the Editor can fall at the end of the page, and the system will never know if the Editor's contents have to be broken between pages. The programmer will see that the editor will continue printing on the next page, and right after that, the Form Header is displayed. This is also true for long strings.

The purpose of this document is to give the programmer a tool to change the Editor contents into rows. The approach will include transforming the Editor's contents into a Temp-table. This way the printout can handle one row at a time and it will be easy to control at the moment of printing each line.

Implementation

The implementation requires the usage of a TEMP-TABLE defined within an include file, and calling an external procedure.

Definition Block

The programmer must locate the include file in the definition block as:

```
{EditToRowDef.i}
```

This will be described below.

Sending the Editor Contents for Transformation

Whenever the programmer requires transforming an EDITOR contents or big string into rows, execute the tool with the following syntax:

```
RUN EditorToRows.p
(INPUT <Editor-BigText>, /* CHARACTER */
 INPUT <Max-Length-per-row>, /* INTEGER */
 INPUT <IGNORE-RETURNS>, /* LOGICAL */
 OUTPUT TABLE tt-EditorRow). /* TEMP TABLE */
```

```
<Editor-BigText> -Text to be split into rows.
<Max-Length-per-row> - Number of Chars to
include per row.
<IGNORE-RETURNS> - Yes will remove all RE-
TURNS and gather all lines.
```

Retrieving the EDITOR split into rows

The way to call the resulting rows is:

```
FOR EACH tt-EditorRow:
  DISPLAY tt-EditorRow.RowNum
  tt-EditorRow.Descr FORMAT "X(60)".
END.
```

The following example show in a simple way the usage of the tool:

```
/*Definitions*/
{EditToRowDef.i}

/*Define tt-EditorRow*/

/* Look for a big string that requires
transformation*/

FIND FIRST MyRecord WHERE
MyRecord.MyField EQ x no-lock no-error.
```

Logic Behind the Scenes

Include File: EditToRowDef.i

The Temp-table definition is very simple:

```
/* _____
FILE: EditToRowDef.i
OBJ: Define Editor To Row
table definition
NOTE: Needed when Using
EditToRows.p program
_____*/
DEF TEMP-TABLE tt-EditorRow
  FIELD RowNum AS INT
  FIELD Descr AS CHAR
  INDEX RowNumIDX IS PRIMARY RowNum.
```

The Algorithm

Definition Block

```
/* ***** Definitions ***** */
/*---- Define tt-EditorRow ----*/
{EditToRowDef.i}

/* Parameter Definitions --*/
DEF INPUT PARAMETER i_FulleEditor
  AS CHAR NO-UNDO.
DEF INPUT PARAMETER i_FulleEditor
  AS CHAR NO-UNDO.
```

```

DEF INPUT PARAMETER i_MaxLengthRow
AS INT NO-UNDO.
DEF INPUT PARAMETER i_IgnoreReturn
AS LOG NO-UNDO.
DEF OUTPUT PARAMETER TABLE FOR tt-EditorRow.

```

```

/*General Variables -*/
&SCOPED-DEFINE DEF-MAX-LENGHT 50

```

Main Block

```

/* ***** Main Block ***** */
/*...STEP 1. Evaluate Paramters...*/
/* If the parameter i_FullEditor comes in
blank or ? value then return: there
is no reason to continue */
IF i_FullEditor EQ ? OR i_FullEditor EQ ""
THEN RETURN "EMPTY-STRING".

/* If the Maximum Length per row has not
been defined assign a default value */
IF i_MaxLengthRow = 0 THEN ASSIGN
i_MaxLengthRow = {&DEF-MAX-LENGHT}.

/* STEP 2. Main Method.....*/
/* Convert into rows the Editor contents
passed as a parameter */
RUN StringToRows IN THIS-PROCEDURE.

```

```
RETURN "OK".
```

Internal Procedure

```

/*-----*/
Purpose: Method to covert STRING
content into ROWS.
Parameters: <none>
Notes:
/*-----*/
DEF VAR t_RowNum AS INT NO-UNDO.
DEF VAR t_RowDescr AS CHAR NO-UNDO.

DEF VAR t_FstCharPos AS INT NO-UNDO.
DEF VAR t_LstCharPos AS INT NO-UNDO.

DEF VAR t_Char AS CHAR NO-UNDO.
DEF VAR t_NxtChar AS CHAR NO-UNDO.
DEF VAR t_Word AS CHAR NO-UNDO.

/* Remove RETURNS if they are not required
by the user */
f
IF i_IgnoreReturn THEN DO:
/* Gather all information taking
RETURN characters */
ASSIGN i_FullEditor =
REPLACE(i_FullEditor,CHR(10)," ").

```



The logo for Jargon Software, featuring the word "JARGON" in a bold, stylized font. The letter 'J' is large and has a green outline. The letters 'A', 'R', 'G', 'O', and 'N' are in black with a white outline. A small "TM" trademark symbol is to the right of the 'N'.

- Develop powerful Wireless applications in our new V3 Jargon Writer!
- Run wireless apps either online or offline when out of coverage area
- Create "bolt-on" solutions to your Progress app in days, not months.
- Integrate barcode scanners, printers and other wireless peripherals
- Deploy with AppServer or WebSpeed. Save money with AppServer.
- Enjoy easy, powerful wireless deployment for under \$150 per handheld.
- Leverage your 4GL experience - no need to master Java, C++, or HTML.
- Download a FREE Evaluation Copy or contact us for a personal online demo

JARGON SOFTWARE Inc
708 North First Street, Suite 432
Minneapolis, Minnesota 55401
612 338 1175 | fax 612 338 2974
www.jargonsoft.com (On-line demo!)
info@jargonsoft.com

4

```
END.

/* Start Process ..... */
ASSIGN t_RowNum = 1
      t_RowDescr = "".

/* loop on the whole string */
Lbl-Word:
DO t_FstCharPos = 1 TO
  LENGTH(i_FullEditor) - 1:

  /* read one character at a time */
  ASSIGN t_Char =
SUBSTRING(i_FullEditor,t_FstCharPos,1).

  /* If the character is a RETURN then
  create an empty row*/
  IF ASC(t_Char) EQ 10 THEN DO:
    CREATE tt-EditorRow.
    ASSIGN tt-EditorRow.RowNum = t_RowNum
           tt-EditorRow.Descr = ""
           t_RowDescr = ""
           t_Word = "".
    ASSIGN t_RowNum = t_RowNum + 1.
    NEXT Lbl-Word.
  END.

  /* Ignore Tabs */
  IF ASC(t_Char) EQ 9 THEN
NEXT Lbl-Word.

/*Ignore extra spaces */
IF t_Char EQ " " THEN NEXT Lbl-Word.

/* Continue reading characters until the
end of the word or the maximum
permitted size per row */
ASSIGN t_Word = t_Char
      t_LstCharPos = t_FstCharPos.

Lbl-Char:
DO WHILE TRUE:
  ASSIGN t_LstCharPos = t_LstCharPos + 1
         t_NxtChar =
SUBSTRING(i_FullEditor,t_LstCharPos,1).

  /* The word ended*/
  IF t_NxtChar EQ " " OR LENGTH(t_Word)
EQ i_MaxLengthRow OR
ASC(t_NxtChar) EQ 10 THEN LEAVE.

  ASSIGN t_Word = t_Word + t_NxtChar.
END. /* DO WHILE */

/* ADD: Add to the current row if there
is still space .....*/
/* First check if the rowdescr has
something already */
IF t_RowDescr NE "" AND
LENGTH(t_RowDescr) +
(LENGTH(t_Word) + 1) <= i_MaxLengthRow

THEN DO:
  ASSIGN t_RowDescr = t_RowDescr +
" " + t_Word
       t_Word = "".
END.

/* Second case if the row descr is
totally empty */
IF t_RowDescr EQ "" AND
(LENGTH(t_Word)) <= i_MaxLengthRow
THEN DO:
  ASSIGN t_RowDescr = t_RowDescr +
t_Word
       t_Word = "".
END.

/* CREATE ROW: Reasons to make a Row ... */
/* The t_Word didn't fit in the row.
Create row and move word -> Row Descr*/
IF t_Word NE ""
THEN DO:
  CREATE tt-EditorRow.
  ASSIGN tt-EditorRow.RowNum = t_RowNum
         tt-EditorRow.Descr =
t_RowDescr
         t_RowDescr = t_Word
/* <- Reassign the Word to new descr */
t_Word = "".
/* This can happen only once */
ASSIGN t_RowNum = t_RowNum + 1.
END.

/* Rowdescr is full or last char was a
END-OF-LINE*/
IF LENGTH(t_RowDescr) EQ i_MaxLengthRow
OR ASC(t_NxtChar) EQ 10
THEN DO:
  CREATE tt-EditorRow.
  ASSIGN tt-EditorRow.RowNum = t_RowNum
         tt-EditorRow.Descr = t_RowDescr
         t_RowDescr = t_Word.
/* t_Word is empty at this pont */
ASSIGN t_RowNum = t_RowNum + 1.
END.

/* Increment Counter */
IF ASC(t_NxtChar) EQ 10 OR t_NxtChar
EQ " " THEN
  ASSIGN t_FstCharPos = t_LstCharPos.
ELSE ASSIGN t_FstCharPos =
t_LstCharPos - 1.

END. /* LOOP Lbl-Word */

/* Create last line if something
remains */
IF t_RowDescr > ""
THEN DO:
  CREATE tt-EditorRow.
  ASSIGN tt-EditorRow.RowNum = t_RowNum
```

```
tt-EditorRow.Descr = t_RowDescr.
END.
```

The full code for the algorithm begins at right.

Armando Pensado

Armando graduated in 1987 as a Biomedical Engineer. In 1991 he took his first class in Progress and since then he has been in love with the tool. He is currently working for Financial Processing System (FPS) in California, and is the head of the team that develops Progress products for the company. He spends a good amount of time building tools that can improve the quality of his systems.

Armando has two Science awards and a few publications in the Biomedical engineering world. His hobbies include hiking in the mountains with his two boys and his wife; they enjoy their time together.

armando@fpsnet.com



```
&ANALYZE-SUSPEND _VERSION-NUMBER UIB_v8r12
&ANALYZE-RESUME
&ANALYZE-SUSPEND _UIB-CODE-BLOCK _CUSTOM
_DEFINITIONS Procedure
/*-----
File       : EditorToRows.p
Purpose    : Change the Contents of
an editor into ROW inside a TEMP-TABLE.
Syntax     : i_FullEditor -> Editor's
              content
              i_MaxLengthRow ->
Number Maximum of Characters by Row
              i_IgnoreReturn ->
Replace enters for spaces
```

Description : When the programmer requires printing (or sending to an ASCII or HTML File) the contents of an editor, the output will not understand that the widget represents several lines. Paging control including the one set by Progress will not work.

The system will think that the Editor is composed of a single line and even though it prints out several lines the counters consider only one row in the print out.

With this procedure the user will transform the Editor's content into several rows save it in a temp-table tt-EditorRow. Within this table it will be easy to control the number of lines printing, resolving the paging problem and editor printing.

```
Author(s)  : Armando Pensado
Created    : August 2001
Notes     :
-----*/
/* This .W file was created with the
Progress UIB.                               */
/*-----*/

/* ***** Definitions ***** */

/*Define tt-EditorRow ---*/
{EditToRowDef.i}

/* Parameters Definitions ---*/
DEF INPUT PARAMETER i_FullEditor
AS CHAR NO-UNDO.
DEF INPUT PARAMETER i_MaxLengthRow
AS INT NO-UNDO.
DEF INPUT PARAMETER i_IgnoreReturn
AS LOG NO-UNDO.
DEF OUTPUT PARAMETER TABLE FOR tt-EditorRow.

/*General Variables ---*/
&SCOPED-DEFINE DEF-MAX-LENGHT 50

/* _UIB-CODE-BLOCK-END */
&ANALYZE-RESUME
```

6

```
&ANALYZE-SUSPEND _UIB-PREPROCESSOR-BLOCK

/* *** Preprocessor Definitions ***** */

&Scoped-define PROCEDURE-TYPE Procedure

/* _UIB-PREPROCESSOR-BLOCK-END */
&ANALYZE-RESUME

/* **** Procedure Settings ***** */

&ANALYZE-SUSPEND _PROCEDURE-SETTINGS
/* Settings for THIS-PROCEDURE
   Type: Procedure
   Allow:
   Frames: 0
   Add Fields to: Neither
   Other Settings: CODE-ONLY COMPILE
*/
&ANALYZE-RESUME _END-PROCEDURE-SETTINGS

/* ***** Create Window ***** */

&ANALYZE-SUSPEND _CREATE-WINDOW
/* DESIGN Window definition
   (used by the UIB)
   CREATE WINDOW Procedure ASSIGN
       HEIGHT           = 8.35
       WIDTH            = 47.
/* END WINDOW DEFINITION */

&ANALYZE-RESUME

&ANALYZE-SUSPEND _UIB-CODE-BLOCK _CUSTOM
_MAIN-BLOCK Procedure

/***** Main Block ***** */

/* STEP 1. Evaluate paramters.....*/
/* If the parameter i_FullEditor comes
   in blank or ? value return, */
/* there is no case to continue */
IF i_FullEditor EQ ? OR i_FullEditor EQ
"" THEN RETURN "EMPTY-STRING".

/* If the Maximum Lenght per row has
   been not defined assign a default
   value */
IF i_MaxLengthRow = 0 THEN ASSIGN
   i_MaxLengthRow = {&DEF-MAX-LENGHT}.

/* STEP 2. Main Method.....*/
/* Convert into rows the Editor passed as
   a parameter */
RUN StringToRows IN THIS-PROCEDURE.

RETURN "OK".
/* _UIB-CODE-BLOCK-END */
&ANALYZE-RESUME
```

```
/* ***** Internal Procedures ***** */

&ANALYZE-SUSPEND _UIB-CODE-BLOCK _PROCEDURE
StringToRows Procedure
PROCEDURE StringToRows :
/*-----*/
   Purpose:      Method to covert STRING
                  content into ROWS.
   Parameters:   <none>
   Notes:
/*-----*/

DEF VAR t_RowNum      AS INT NO-UNDO.
DEF VAR t_RowDescr   AS CHAR NO-UNDO.

DEF VAR t_FstCharPos AS INT NO-UNDO.
DEF VAR t_LstCharPos AS INT NO-UNDO.

DEF VAR t_Char       AS CHAR NO-UNDO.
DEF VAR t_NxtChar    AS CHAR NO-UNDO.
DEF VAR t_Word       AS CHAR NO-UNDO.

/* Remove RETURNS if is not required by
   the user */
IF i_IgnoreReturn THEN
DO:
/* Gather all information taking
   RETURN characters */
   ASSIGN i_FullEditor =
   REPLACE(i_FullEditor,CHR(10)," ").
END.

/* Start Process -----*/
ASSIGN t_RowNum      = 1
       t_RowDescr = "".

/* loop on the whole string */
Lbl-Word:
DO t_FstCharPos = 1 TO
   LENGTH(i_FullEditor) - 1:

   /* read one character at a time */
   ASSIGN t_Char =
   SUBSTRING(i_FullEditor,t_FstCharPos,1).

/* If the character is an RETURN then
   create empty row*/
IF ASC(t_Char) EQ 10 THEN
DO: CREATE tt-EditorRow.
   ASSIGN tt-EditorRow.RowNum =
       t_RowNum
       tt-EditorRow.Descr   = ""
       t_RowDescr          = ""
       t_Word               = "".
   ASSIGN t_RowNum = t_RowNum + 1.
NEXT Lbl-Word.
END.

/* Ignore Tabs */
IF ASC(t_Char) EQ 9 THEN
NEXT Lbl-Word.
```

```

/*Ignore extra spaces */
IF t_Char EQ " " THEN NEXT Lbl-Word.

/* Continue reading characters until the
end of the word or the maximum
permitted size per row */
ASSIGN t_Word      = t_Char
       t_LstCharPos = t_FstCharPos.

Lbl-Char:
DO WHILE TRUE:
  ASSIGN t_LstCharPos =
         t_LstCharPos + 1
         t_NxtChar     =
SUBSTRING(i_FullEditor,t_LstCharPos,1).

/* The word ended*/
IF t_NxtChar EQ " " OR
   LENGTH(t_Word) EQ i_MaxLengthRow
OR
   ASC(t_NxtChar) EQ 10 THEN
  LEAVE.

  ASSIGN t_Word = t_Word + t_NxtChar.
END. /* DO WHILE */

/* ADD: Add to the current row if
there is still space .....*/

/* First check if the rowdescr has
something already */
IF t_RowDescr NE "" AND
   LENGTH(t_RowDescr) +
   (LENGTH(t_Word) + 1) <=
   i_MaxLengthRow THEN
DO:
  ASSIGN t_RowDescr = t_RowDescr +
         " " + t_Word
         t_Word      = "".
END.

/* Second case if the row descr is
totally empty */
IF t_RowDescr EQ "" AND
   (LENGTH(t_Word)) <= i_MaxLengthRow
THEN
DO:
  ASSIGN t_RowDescr = t_RowDescr +
         t_Word
         t_Word      = "".
END.

/* CREATE ROW: Reasons to make a Row
..... */

/* The t_Word didn't fit in the row.
Create row and move word -> Row Descr*/
IF t_Word NE "" THEN
DO: CREATE tt-EditorRow.
     ASSIGN tt-EditorRow.RowNum =
           t_RowNum
           tt-EditorRow.Descr =
           t_RowDescr
           t_RowDescr
           = t_Word
           = "".
           ASSIGN t_RowNum = t_RowNum + 1.
END.

/* Rowdescr is full or last char was a
END-OF-LINE */
IF LENGTH(t_RowDescr) EQ
i_MaxLengthRow OR ASC(t_NxtChar)
EQ 10 THEN
DO:
  CREATE tt-EditorRow.
  ASSIGN tt-EditorRow.RowNum = t_RowNum
         tt-EditorRow.Descr = t_RowDescr
         t_RowDescr
         = t_Word.
  /* t_Word is empty at this pont */
  ASSIGN t_RowNum = t_RowNum + 1.
END.

/* Increment Counter */
IF ASC(t_NxtChar) EQ 10 OR t_NxtChar
EQ " " THEN
  ASSIGN t_FstCharPos = t_LstCharPos.
ELSE ASSIGN t_FstCharPos =
         t_LstCharPos - 1.
END. /* LOOP Lbl-Word */

/* Create last line of something
remained */
IF t_RowDescr > "" THEN
DO: CREATE tt-EditorRow.
     ASSIGN tt-EditorRow.RowNum =
           t_RowNum
           tt-EditorRow.Descr =
           t_RowDescr.
END.

END PROCEDURE.

/* _UIB-CODE-BLOCK-END */
&ANALYZE-RESUME

```

SAX and XPath by Tony Lavinio

Among the features in Progress 9.1D, the SAX parser for XML was introduced. SAX, or Simple API for XML, is a mechanism for parsing XML as it is streamed through. This is as opposed to the DOM, or Document Object Model, which loads the entire document and lets you navigate it.

DOM has the advantage where back-and-forth navigation is necessary, but exacts a high premium in memory usage. SAX is thrifty in memory, but once a node goes by, you've got to either remember it or restart your parsing.

The SAX parser is implemented with callbacks, which are procedures with specific names that are called as the events occur. When the document is opened, the internal procedure named 'StartDocument' is called, and there is an analogous one called 'EndDocument'. 'StartElement' is called when each XML tag is hit, and 'EndElement' as each is completed. The 'External Program Interfaces' guide lists the full set of callbacks, and includes some sample code as to how they can be used.

XPath is a syntax for specifying a specific node or branch of nodes in an XML document. Although there is much subtlety in XPath expressions, (and as XPath2 rolls around, much complexity), the concepts involved were clearly explained by Ronald Bourret in "XPath in Five Paragraphs" located at <http://www.rpbouret.com/xml/XPathIn5.htm> on his web site.

XPath is the syntax used in XSLT and XQuery for locating nodes, and as such is worth having at least a passing familiarity with it.

In order to explore both SAX and XPath, and as part of my research work here at PSC Labs, I've written a program that uses the SAX parser to read an arbitrary XML document, and at each text node calculates the unique XPath expression to pinpoint that specific node. These are accumulated in a temp-table. There are two ways the program has of specifying the nodes: 1) by relative node number, using XPath's `node()[n]` construct, which counts any type of node; and 2) by relative text node number, using `text()[n]`, which only counts the text (SAX: "character") nodes.

Since SAX parsing gives you no context, this program also demonstrates how a temp-table can be used to track your location as you move through the document.

As part of its ongoing move to diversify its business base, Progress has recently purchased eXcelon Corporation, which provides such XML-focused products as XIS (an XML database) and Stylus Studio (an XML editing environment). The latter especially can help in learning about and developing with XPath, XSLT, and related technologies. I used Stylus Studio to help debug this program before I knew we were merging, and found it quite useful — especially the XPath expression bar at the top of each XML window.

Further reading on SAX can be found at <http://www.saxproject.org/>, and more on XPath can be found at <http://www.w3.org/TR/xpath>. For more, just type "SAX" or "XPath" into your favorite search engine.

I hope this program helps to demonstrate the mechanics behind using both SAX and XPath. Happy XMLing!

Tony Lavinio

Tony has returned to his roots at Progress Software, this time in the research group PSC Labs. He can be reached at the other end of tony@lavinio.net.

```
/* file: SAXPath.p */
/*-----*/
/*
```

```
SAXPath.p
```

```
This program uses the SAX parser to
navigate an XML document, and it
tracks the XPath expression to the current
node as it executes.
```

```
Copyright (C) 2002 Progress Software
Corporation
```

```
Author: Tony Lavinio, PSC Labs
```

```
Date: 2002-Oct-29
```

```
*/
/*-----*/
```

```
/* Setting this value will accumulate the
results and display them. */
&SCOPED-DEFINE test
```

```

&IF DEFINED(test) &THEN
DEFINE VARIABLE cc AS CHARACTER
    INITIAL ":u NO-UNDO.
&ENDIF
DEFINE VARIABLE hParser AS HANDLE NO-UNDO.

/* These next five lines are the main
portion of the program. */
/* Everything else of significance happens
in SAX callbacks. */

CREATE SAX-READER hParser.

hParser:HANDLER = THIS-PROCEDURE.
hParser:SET-INPUT-SOURCE('file':u,
    "saxpath.xml").

hParser:SAX-PARSE( ) NO-ERROR.

DELETE OBJECT hParser.

IF ERROR-STATUS:ERROR THEN DO:
    IF ERROR-STATUS:NUM-MESSAGES > 0 THEN
        /* unable to begin the parse */
        MESSAGE ERROR-STATUS:GET-MESSAGE(1)
        VIEW-AS ALERT-BOX INFORMATION.
    ELSE
        /* error detected in a callback */
        MESSAGE RETURN-VALUE
        VIEW-AS ALERT-BOX INFORMATION.
    END.

&IF DEFINED(test) &THEN
MESSAGE cc VIEW-AS ALERT-BOX INFORMATION.
&ENDIF

RETURN.

/*-----*/

DEFINE VARIABLE iPath AS INTEGER
NO-UNDO. /* Absolute path ordering */
DEFINE VARIABLE lChar AS LOGICAL
NO-UNDO. /* Are we in a text node? */
DEFINE VARIABLE cStack AS CHARACTER NO-UNDO.
/* Name-path of where we are */

DEFINE TEMP-TABLE ttPath NO-UNDO
    FIELD tiPath AS INTEGER
/* for absolute ordering within document */
    FIELD tcPath AS CHARACTER CASE-SENSITIVE
/* node-name path to this node */
    FIELD tiRel AS INTEGER
/* used in XML cursor/stack management */
    FIELD tiNode AS INTEGER
/*
    relative node number w/i parent element
*/
    FIELD tiText AS INTEGER
/* relative text node number w/i parent
    element */

```

```

FIELD tcXPath AS CHARACTER
/* XPath to this text node */
INDEX x_Path IS PRIMARY UNIQUE
    tcPath tiPath.

DEFINE QUERY qryPath FOR ttPath.

/*-----*/
PROCEDURE StartDocument:
    CLOSE QUERY qryPath.
    ASSIGN
        iPath = 0
        cStack = ":u
        lChar = FALSE.
    EMPTY TEMP-TABLE ttPath.
END PROCEDURE.

/*-----*/

PROCEDURE StartElement:
    DEFINE INPUT PARAMETER ip-namespaceURI
    AS CHARACTER NO-UNDO.
    DEFINE INPUT PARAMETER ip-localName
    AS CHARACTER NO-UNDO.
    DEFINE INPUT PARAMETER ip-qName
    AS CHARACTER NO-UNDO.
    DEFINE INPUT PARAMETER ip-attributes
    AS HANDLE NO-UNDO.
    DEFINE VARIABLE iRel AS INTEGER
    INITIAL 1 NO-UNDO.
    DEFINE VARIABLE cParent AS CHARACTER
    INITIAL ":u NO-UNDO.

    OPEN QUERY qryPath FOR
        EACH ttPath NO-LOCK
        WHERE ttPath.tcPath = cStack
        BY ttPath.tiPath DESC.
    GET FIRST qryPath.
    IF AVAILABLE ttPath THEN
        ASSIGN
            cParent = ttPath.tcXPath
            ttPath.tiNode = ttPath.tiNode + 1.

/* Push this location onto the XML
cursor/stack */
cStack = (IF cStack = ":u THEN ":u ELSE
cStack + ',':u)
    + ip-qName
    + (IF AVAILABLE ttPath THEN '=':u +
        STRING(ttPath.tiRel) ELSE ":u).

/* Have we seen this path before?
We want to make sure we're */
/* using the correct positional indicators
for this particular */
/* location and not just one that has the
same path names that */
/* we'd seen earlier in the document. */
OPEN QUERY qryPath FOR
    EACH ttPath NO-LOCK
    WHERE ttPath.tcPath = cStack
    BY ttPath.tiPath DESC.

```

10

```
GET FIRST qryPath.
IF AVAILABLE ttPath THEN
    iRel = ttPath.tiRel + 1.

CREATE ttPath.
ASSIGN
    iPath          = iPath + 1
    lChar          = FALSE
    ttPath.tiPath  = iPath
    ttPath.tiRel   = iRel
    ttPath.tiNode  = 0
    ttPath.tiText  = 0
    ttPath.tcPath  = cStack
    ttPath.tcXPath = cParent + '/' :u +
ip-qName
    + (IF cParent = "" THEN
        ":u
        ELSE
        SUBSTITUTE('&[1]':u,iRel)
        ).

&IF DEFINED(test) &THEN
    cc = (IF cc = ":u THEN ":u ELSE cc +
'~n':u)
    + "Element" + '=' :u + ttPath.tcXPath.

DEFINE VARIABLE iLoop AS INTEGER NO-UNDO.
DO iLoop = 1 TO ip-attributes:NUM-ITEMS:
    cc = (IF cc = ":u
    THEN ":u
    ELSE cc + '~n':u)
    + "Attr" + '=' :u
    + ttPath.tcXPath + "/"@ +
ip-attributes:GET-QNAME-BY-INDEX(iLoop).
END.
&ENDIF

END PROCEDURE.

/*-----*/

PROCEDURE Characters:
    DEFINE INPUT PARAMETER ip-charData
    AS MEMPTR NO-UNDO.
    DEFINE INPUT PARAMETER ip-numChars
    AS INTEGER NO-UNDO.

    /* If we actually have content here,
    let's properly mark our place in the
    XPath expression. */
    OPEN QUERY qryPath FOR
    EACH ttPath NO-LOCK
    WHERE ttPath.tcPath = cStack
    BY ttPath.tiPath DESC.
    GET FIRST qryPath.
    IF NOT lChar AND AVAILABLE ttPath THEN
    ASSIGN
        ttPath.tiNode = ttPath.tiNode + 1
        ttPath.tiText = ttPath.tiText + 1.
```

```
/* Remember that we can get more than one
Character's callback for the same text
node. So we keep track of the fact that
we're in one now, in case we get another
Character's callback before we switch to
the next node. */
lChar = TRUE.
/*
The current node can be referenced as:
ttPath.tcXPath + '/node()' :u +
STRING(ttPath.tiNode) + ':' :u
or
ttPath.tcXPath + '/text()' :u +
STRING(ttPath.tiText) + ':' :u
*/

&IF DEFINED(test) &THEN
    cc = (IF cc = ":u
        THEN ":u ELSE cc + '~n':u)
    + "Text" + '=' :u
    + ttPath.tcXPath + '/text()' :u +
    STRING(ttPath.tiText) + ':' :u.
&ENDIF

END PROCEDURE.

/*-----*/
PROCEDURE EndElement:
    DEFINE INPUT PARAMETER ip-namespaceURI
    AS CHARACTER NO-UNDO.
    DEFINE INPUT PARAMETER ip-localName
    AS CHARACTER NO-UNDO.
    DEFINE INPUT PARAMETER ip-qName
    AS CHARACTER NO-UNDO.

    /* pop an item off the XML cursor/stack */
    ASSIGN
    ENTRY (NUM-ENTRIES(cStack),cStack) = ":u
    cStack = RIGHT-TRIM(cStack,',' :u)
    lChar = FALSE.
END PROCEDURE.

/*-----*/

PROCEDURE EndDocument:
    CLOSE QUERY qryPath.

    /* Now's your chance to use the data in
    the temp-table, before it's cleared out.
    */
    EMPTY TEMP-TABLE ttPath.
END PROCEDURE.

/*-----*/
/* end-of-file: SAXPath.p */
```

Fathom Management Overview: Focusing on Jobs By Brian Bowman

Fathom Management is a new product created out of a need for a single tool to manage and monitor the Progress environment. The functionality of Fathom Management has its roots in other Progress tools. There was a desire to integrate the existing varied environments used to maintain and manage Progress OpenEdge products. This was a challenge as different products had different interfaces, utilized different configuration files and ran on a variety of platforms. Fathom Management's goal was to make these diverse products obsolete. Outlined below is the history of tools that preceded Fathom Management and the types of problems they try solve.

Many of the Progress database management tools are still in the V5 style of Progress. Promon is character based and requires screen scraping to achieve data capture for historical analysis. One of the biggest advances after promon was the implementation of Virtual System Tables (VST's). Progress chose not to implement an interface but to document the tables. This was to enable the user community to be able to create their own tools using the VST's. This was only marginally successful as users had to know what they were looking for prior to accessing the more than 350 data points. Most database utilities are still command line interface (CLI) based.

ProControl is limited only to windows platforms. It provides some of the features needed to manage the Progress database, but does not meet all of the requirements. It does however, have the capability of launching jobs. It is Progress' first Graphical User Interface tool (GUI) management tool.

Progress Explorer introduced additional capabilities to the OpenEdge environment including network access to the database and configuration capabilities. While important advancements, these new features did not completely meet the needs of the end user. It is windows-based and is tied to the Microsoft Management Console (MMC). Progress Explorer does not include the ProControl functionality of launching jobs from a Progress utility.

All of the Progress management tools mentioned

above lack a consistent User Interface and multiple platform support. Another issue that none of these tools address is the ability to capture data for historical trend analysis. An expert Progress database IT professional is required to identify what an end user will require in six months. Assuming such an IT professional exists, she would then be required to write and maintain code to capture the appropriate data, have the skills to turn the data into valuable easily consumable information, and be able to store the data for a user-defined period of time.

All of these factors made it clear that the database was the best place to start with a facelift. This is where Fathom Management comes into play.

Fathom Management contains many different components. It is integrated into the AdminServer to take advantage of the AdminServers' processing, knowledge of the environment, and capabilities. Fathom Management has a built in web server to handle communication. The internal java-based web server was implemented to guarantee compatibility and consistency across all platforms. The UI is completely browser based and written primarily in Java Servlet Pages (JSP). Fathom Management includes an integrated scheduler to handle manage scheduling. A Progress database is used to store historical data and track statistical information.

Fathom Management Features and Benefits at a Glance

Industry Standard Languages - Fathom is based on platform independent languages, from JAVA and JSP's to PERL for scripting of Jobs.

System Monitoring - Fathom Management has a component for system monitoring to capture system information and store it for historical analysis.

Direct Database Access - A client called the dbagent does the database monitoring. This client has very little overhead and accesses shared memory directly to gather information and monitor the status of the database. This can be done locally or remotely.

Configurable Rules - Each resource defined to Fathom Management has a set of rules that are configurable. These rules help evaluate when the resource is performing normally or outside the expected norm.

Alerting - There is also an alert component that

provides proactive notification when rules are broken and events occur. An example of this is the abnormal shutdown of a database. When this happens an alert is fired and the user can choose to perform any set of actions. Alerts can be emailed, viewed on the console, sent to a pager, or passed through SNMP.

Log File Monitoring - Log file monitoring is also built into Fathom Management. You can monitor any log file for specific strings or regular expressions. This automates log file monitoring for problems that occur.

Reporting - Reporting is handled by a set of canned 4GL reports written against the Fathom database. The database is also SQL-92 “enabled” for reporting with 3rd party tools such as Crystal Reports.

Trend Storage – All database and system information is stored for historical analysis. Job execution statistics, Alert details, and resource statuses are also stored in the database.

External Process Execution - Fathom Management also implemented the capabilities to schedule, execute, monitor, and capture output from external processes. These are called jobs in Fathom Management.

The remainder of this article provides additional details about jobs, explains how they work, provides examples, and explains some of the advantages of the job component of Fathom Management. It is one of the most powerful and extensible parts of Fathom Management. It combines the best of the new framework as well as the capacity to support existing user scripts that companies have developed over the years.

Jobs were created out of the need to be able to execute an external process and maintain control over that routine. The external process can be a C program, a 4GL program, a Perl script, or any other process that can run on a machine. This capability existed in ProControl, but did not exist in Progress Explorer. Controlling the process includes capturing any output from the process, capturing statistics about the process, and being able to perform actions based on the exit status of the process.

Upon execution of a process Fathom Management captures the start time, end time, and the exit status of the job. This information is stored in the Fathom Management database for historical purposes. Execution

of the job is via the Progress core OpenEdge jvmstart executable. This is the same mechanism that is used by the AdminServer to start other processes such as WebSpeed brokers, AppServer brokers, and the Name Server.

The exit status allows Fathom Management to be able to perform different actions based on the exit code. This provides Fathom Management with the capability to chain multiple jobs together and branch the processing based on job execution being successful or not.

Jobs can also be configured to run stand alone and as actions to be performed upon a rule violation. This makes them available to other jobs as well as alert actions. A good example of this is building a backup process. A job can be used to capture the database status and based on that shut down the database with a custom script. If that job is successful, an offline backup of the database can be performed. If the shutdown is not successful, one can do an online backup of the database can be performed. All of the information is captured in log files and available for viewing online through a browser. There is no need to go to the machine to see the status, change the job, and re-execute it.

Jobs are schedulable in the Fathom Management scheduler. The scheduler is capable of scheduling the job to run one time, at startup of Fathom Management, or recursively at different intervals.

Jobs also have the concept of templates built into them. Templates allow the user to quickly configure a new job from an existing template. This makes creating new jobs much quicker. Templates can also be exported from one system and imported on other systems allowing ISV's to create templates for their customers.

Fathom Management also ships a set of standard database jobs that are customized for databases. These include an online and offline backup job, a restore job, a truncate before-image job, a grow before-image job, a database analysis job, and a job to compact the historical trend data. The database analysis job is a good example of the capability of jobs in Fathom Management. It runs proutil dbanalysis against the database, loads the information into the Fathom Management database for future analysis, and optionally analyzes the index information to recommend if an index should be rebuilt. It can also automatically rebuild the indexes it finds with index compact.

Arthur Fink Consulting

www.ArthurFink.com

Listening to users

Designing systems that work

Listening to users is very important, because what we hear from users is important. Designing or building systems based only on our assumptions or ideas about what users need is always going to be a dangerous business. Let me be part of your user interface design process.

IAP GmbH

XCheck

Logfile analyzing
system checking

- usable with Windows or UNIX
- check activity of NS/DB/WS/httpd
- analyze logfiles of NS/DB/WS
- check drive space, space in DB
- execute self defined scripts
- analyze self defined logfiles
- get notified by e-mail, http
- or screen output

Viper

Visual Printing
and Enhanced Reports

- uses Windows printer drivers
- data processing with 4GL
- incl. layout designer (VFD)
- stores layouts DB or file based
- no runtime licence cost
- supports bmp/jpg/wmf images
- embedding rtf-texts (font,...)
- generates xml output (xslfo)
- generates pdf-files (email)
- supports WebSpeed /-Client

PCase

CASE-Extension for the
Data Dictionary

- view Progress-DB structures
- create/update DBs directly
- reengineer Progress-DBs
- read/write Progress df-files
- compare/maintain versions
- incl. DB Content Viewer
- incl. Open Report Interface
- autogenerates references
- print resizable ER-Diagrams
- report-, structure- or ERD view

QComp

Project management
compiling, analyzing

- compiles project file lists
- includes compiler server
- also compiles in char-mode
- uses different Progress vers.
- compiles for different OS
- contains xref-analyze frontend
- shows db structure & content
- keeps track of project errors

IAP GmbH

IAP GmbH • Moerkenstrasse 9 • D-22767 Hamburg • Germany
Tel. +49 40 30 68 03 - 0 • Fax +49 40 30 68 03 - 10
email: info@tools4progress.com

T4P
>> TOOLS FOR PROGRESS

Information and free testversions at www.tools4progress.com

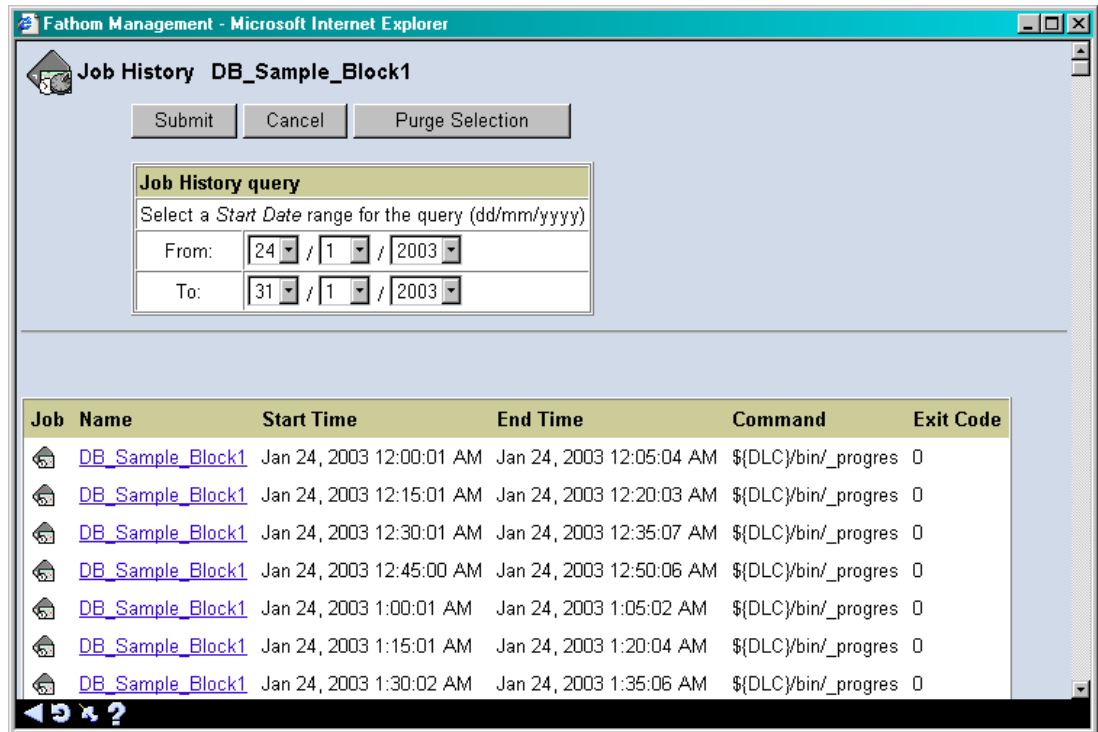


figure 1

There are additional standard jobs available on PSDN that can be downloaded, imported into Fathom Management, and scheduled. They currently include OpenEdge management, after-image file management, and Sonic notification.

Another feature of jobs is the capability to generate an alert if the job takes too long to run. Fathom Management proactively notifies the user that the process is taking too long. The user can configure the job to fire an alert after a certain amount of time. This allows Fathom Management to perform other actions based on the alert

Implementing existing scripts into Fathom Management is very quick and easy to do. One customer implemented their entire batch processing (70+ processes) in Fathom Management within a half day. Reporting can be scheduled to run during the off hours and the administrator can be notified if the process completed successfully or not. This allows the administrator to focus on other things.

A sample of what a report for a particular job would look like is shown in figure 1, above.

The job component of Fathom Management is just one part of Fathom. It is very versatile and can be used for just about any processing that you can imagine. It

can make a DBA's life much easier. Fathom Management will proactively notify the user of any problems. This allows the user to concentrate on other more strategic areas of the business.

Fathom Management can be downloaded for a free 30 day trial. For more information about Fathom Management please visit: www.progress.com/Fathom.

Fathom Management 2.0A is the current commercial release. It is available on most platforms and runs with Progress 9.1C and 9.1D. Fathom Management 2.1A beta is scheduled to begin in May 2003. To participate in the Fathom Management v2.1a beta program please contact Kristen Howell at khowell@progress.com.

Brian Bowman

Brian has worked in the computer industry for 15 years, 10 of them have been with Progress Software. He has done application development, technical support, training, and consulting in the US and Europe. His focus while at Progress has been on database tuning and administration. Brian currently works on the Fathom Management development team for Progress. When he is not playing with computers he is playing outdoors with his two children. You can reach Brian at bowman@progress.com

On Listening to Users Who Don't Talk by Arthur Fink

No, this is not about some Zen paradox, or some philosophical mystery! Most of us want to please our users with systems that really do work, and yet the users give us very little feedback. Besides, they don't know what's possible in the technology, so how can they? And so we produce systems that we believe will give the users what they need. What an act of faith! Unfortunately, much too often, it's faith that falls flat. [And most of what I've just said is wrong.]

Several years ago one of my clients was close to going under. They had purchased a Progress application from an unscrupulous or inept vendor, and were now helpless to control their business. Yes, they could take orders, but it was hard to process returns, and even harder to find out what items were overstocked or about to run out. Planning was virtually impossible. Simple tasks occupied the users for much too long. My mission was to make a few critical and lifesaving "fixes", and then to serve as an expert witness in the legal proceedings that would surely follow.

Thankfully the legal path was abandoned quite early. My client wasn't going to get blood from a stone (or a poorly performing Progress var), and no cash settlement would get their business going right. Instead, I was asked to do a much more major rewrite of the user interface. And now the real story begins.

I sat next to an operator who was taking orders. Wearing a "training" headset, I could hear both sides of the conversation as the customer asked questions, and tentatively moved towards ordering certain items, and the operators feverishly tried to determine what was in stock, when delivery could take place, etc. Then the operator had to fill in lots of extra information on the screen: No, this item is not part of a set; no, it's not monogrammed; no, it's not shipped to a different address, and on and on! Then the customers wanted to change something — quantity two instead of one. Again, there was a long series of questions to ask just to enter this change. The whole process was lengthy, awkward, constraining, and mostly unnecessary.

But this was a professional setting, I was the outside

consultant, and the operator was at first afraid of being too candid. She just smiled and breathed out as this whole process was completed. "Did you really need to fill out that much information," I asked. "Of course," she replied. "This silly new system requires it." And thus we broke the ice. I asked her which of the information would get used later in the customer service, shipping, or return departments —the only departments that might directly work with this order. And now I could get a candid response —"None of it".

My visit with the returns department was even worse —or should I say better? It was the system that was horrible, and the users who guided me through this tortuous maze. As they saw that I was listening intently, taking notes, asking serious but not loaded questions . . . they opened up more and more. Pretty soon I had pages and pages of notes about what was wrong —seriously wrong— with the system this company had purchased. More important, I had lots of good suggestions about how it might be re-worked.

This shouldn't be news. I listened, as others (who had something to say) talked. But it is news because this so rarely happens. Let me elaborate. After completing these interviews, I spoke with some of the technical staff at the original system vendor. I knew that I'd need a working relationship with them if I was ever to dig into the gizzards of their system, and so I tried to communicate in the most attentive and polite manner. But I was immediately hit with hard questions: "Why was my client bad-mouthing the vendor at the catalog retailer meetings? . . . Couldn't they be more appreciative of the wonderful aspects of the system . . . those parts of the system that really DID work?" I told my tales of woe, the litany of complaints from users, and dysfunctional things that I had observed directly. Their response —and this is my main point— "But the users told us none of this . . . they seemed to like the system so much!".

How could two sets of consultants, listening to the same user community, get two such different readings? The first consultants wanted only approval, couldn't respond to suggestions or criticism with anything other than "what you have is what you've got" and thus weren't really asking questions but were soliciting acceptance. I was genuinely seeking the truth, and tried to encourage any clear reporting. Furthermore, to get things moving, I was asking some leading questions, offering feedback designed to encourage more story-telling (for example,

"Why are you writing down this number from screen A, and then typing it back into the computer on screen B?"). And the fact that I was taking careful notes, asking people to elaborate, and offering no excuses or explanation of why things were as bad as they were only further reinforced my credibility and safety.

Listening to users is important, because what we hear from users is important. Designing or building systems based only on our assumptions or ideas about what users need is always going to be a dangerous business. Listening sounds so basic. What needs to be said about it? However, I've found that listening is an art and a skill, something that needs to be taught and practiced, and something that is not always easy to do. Some of my clients hire me simply to listen to their users, hearing things that might have been apparent to anybody but that I am privileged to record.

Based on such experience, I present these eight principles of good listening:

1. Listening starts with respect. If you don't believe users have something of value to share, are discerning and creative reporters of their own experience . . . then you'll never hear what they are saying.

2. Feedback helps the process. Let the users know what you're hearing. That encourages them, and may lead to helpful corrective feedback. (e.g. "Do I hear you saying that you never use those fields, and that they really don't need to be on the screen?")

3. "Listen" with different modalities. Sometimes users won't talk about what they are doing, but a careful observer will see their behavior and comment upon it. At that point, users may then offer a verbal summary of what you already saw.

4. Listening needs to be part of a feedback cycle. If all you do is listen, users will tire of your presence. But come back with memos, system design memos, new screen layouts, and real code changes . . . and they realize that talking with you is important, and brings results.

5. Don't answer! Users pick up right away if you're being defensive. Listening is about hearing—not about answering.

6. Body language matters. If you're really listening,

look at the person. Maintain focus. Show that you are engaged.

7. Cultivate broader relationships. Try to spend time in the "break room". Chat about things other than system design. Connect with people as people, and let them connect with you.

8. You missed something important! It's always true. Either you didn't hear, or the user didn't quite say it. So stay tuned, and keep engaged. Listening is a process—not an event.

By the way, does this sound like system consulting, or marriage survival, or how to raise your teenager? It should sound like all these, because listening is critical in all aspects of life. People with whom we're engaged need to know that we are genuinely seeking their insights, their feelings, their advice. They need to know that we are present, and that we can be moved to action based on what we hear. Each moment of listening is a critical moment.

Arthur Fink

Arthur is a consultant from Peaks Island, Maine (a real island off the coast of Portland), really does spend most of his productive work time listening to users! He's been working with Progress since the company began, and worked with the founders of Progress (Data Language Corporation) before that. Although he still does consulting and training on Progress programming (both GUI and ChUI), much of his focus now is on helping with user interface and database design. Contact him at Arthur Fink Consulting, arthur@ArthurFink.com, or 207.766.5722.



WHITE STAR BOOKS AND PUBLICATIONS



MAKING GOOD PROGRESS: An absolutely comprehensive book on Progress. Almost 1,000 pages and 35 chapters on everything you need to know about the application development process --from design to tuning.

\$130 (Priority ship included)

\$144 USD Canada (air ship)

\$160 USD International (air ship)

DISASTER PREVENTION: A booklet by Adam Backman on what you need to know to guarantee your system's integrity. Insights on RAID, replication, on-line back ups and much more. There's a lot more to backing up your system than just putting a tape in the drive.

\$24 US

\$27 USD Canada/International

HIGH PERFORMANCE REPORTS: This guide will lead you through the steps of analyzing user requests and designing reports that will be efficient and maintainable; how to analyze and rework slow reports to speed them up.

\$23 US

\$26 USD Canada/International

CODE SMARTER, PROGRAMMING FOR THE FUTURE: All of the features for "Future Proof" programming and how to use them: the preprocessor, internal procedures, dynamic functions, publish/subscribe, super procedures, dynamic buffers, widgets and browsers . . .

\$26 US

\$29 USD Canada/International

WHAT DO I DO NOW? An introduction to the principles of event-driven programming and the many new features and concepts of V7/V8, this booklet will get you up to speed with the preprocessor, schema and UI triggers, internal procedures, the query command and new issues in scoping and transactions.

\$25 US

\$28 USD Canada/International

PROGRESSIONS: NOW IN ELECTRONIC FORMAT!!! A technical journal by Progress programmers, Progressions covers a broad range of subjects for all levels of expertise. It features articles (with code) on such topics as report tricks, database design, undocumented commands, editorials, product reviews, as well as articles on Versions through 8.2 for both GUI and character mode. 1 year subscription is 6 issues. Back issues are available: US. \$7.00 each Canada/International US \$8.00.

PRICING FOR ALL: US, CANADA, INTERNATIONAL \$60.00 USD/one year

VERSION 8/9 FOR \$89: Can't find Version 8/9 information? It's right at your fingertips! We have 11 back issues of Progressions—our V8/V9 Package. Each issue contains one or more articles on useful Version 8/9 topics: insights, tips, the editor, dynamic menus, launching windows apps, triggers, benchmarks, strip menus, application security, designing a GUI interface, internal procedures and much more. Progressions V8/V9 Package -11 issues

US \$89 USD

\$99 USD Canada/International

We accept Visa, MasterCard, AMEX (Include Expiration Date) fax credit card orders 970.963.3548 or email to info@wss.com. We also accept company checks drawn on a US Bank. Sorry, no purchase orders. Shipping and Handling included.

Checks must be in U.S. Dollars payable to White Star Software. CA residents add 7.75%

White Star Software

970.963.3545 info@wss.com www.wss.com 970.963.3548 FAX

At the pause, this is what promon shows:

Record Locking Table:

Usr Name	Rec-id	Table	Lock	
6 Administ	103	101	EXCL	*
6 Administ	252	18	EXCL	**
6 Administ	254	18	EXCL	
6 Administ	255	18	EXCL	
6 Administ	256	18	EXCL	
6 Administ	268	18	EXCL	
6 Administ	276	18	EXCL	
6 Administ	647	18	EXCL	
6 Administ	648	18	EXCL	
6 Administ	649	18	EXCL	
6 Administ	650	18	EXCL	
6 Administ	651	18	EXCL	
6 Administ	658	18	EXCL	

* These are the customer records

** These are the order records

Each of the order records are locked, since the transaction scope is at the customer block. This is to be expected. So what happens if we use the internal procedure as shown above to update the credit limit. Here is the program:

```
CustBlock:
for each customer:
  run updatebalance.
  OrderBlock:
  for each order of customer:
    order.ordernum = order.ordernum * 1.
  end.
  if customer.custnum = 5 then do:
    message "Pausing" view-as alert-box.
    return.
  end.
end.

procedure updateBalance:
  customer.creditlimit =
  customer.creditlimit * 1.
end.
```

At the pause, **this** is what promon shows:

Record Locking Table:

Usr Name	Rec-id	Table	Lock
6 Administ	103	101	SHR

Surprisingly enough, there is only one record locked, and this is the customer record. It is still in scope, and so the lock remains. The conclusion you can draw from this is that it may be possible to reduce the number of record locks in a procedure by calling an internal procedure. Given what the compile listing shows us, this might not be intuitive. However, you can confirm it by trying to locate one of the records for customer 5 exclusive-lock. None of them will be locked, even with

a share lock! This is quite surprising, and has implications, primarily for the number of record locks maintained by the lock table.

This should not excuse lazy or careless programming. Among other things, it invokes some overhead to call an internal procedure rather than updating the record in-line. However, in a large, invoicing program or other routine where a large number of records must all be updated as a block, this can help reduce the number of record locks being held at any one time. Note, however, that if the outer loop (in this case, containing the customer record) must be accomplished as a block, then you will have to state something like:

```
do transaction:
  for each customer:
    .
    .
  end.
end.
```

In this case, every record you touch with a lock will stay locked, no matter whether you call an internal procedure or not. So don't make the assumption that calling an internal procedure will ALWAYS bail you out. Use good programming sense, and always test what you code. It is one thing to understand the rules, and another thing to conclusively prove that you have correctly implemented your understanding of those rules.

Progressions

Jupiter:	John Campbell
Saturn:	Connie Campbell
Venus:	Harriet Coates
Pluto:	Michael Bartlett

Progressions is published bi-monthly, which means 6 times a year, not 24. Now that we are an electronic newsletter, we are offering the same price for everyone, no matter where you live! The annual subscription price is \$60.00 USD.

We accept Visa, Mastercard, and/or American Express (please include your expiration date and whether or not you have enough of a credit line for us to head for Katmandu or simply the border). All checks must be drawn on a US bank; checks and money orders must be in US dollars. In order to protect our contributors, the information in this periodical is copyrighted 1992-2003 White Star Software, Inc., and/or the author. All rights reserved worldwide and to the end of the cosmos.

Progressions
PO Box 250 Carbondale, CO 81623
970.963.3545 Voice 970.963.3548 Fax

Provide your programmers the essential PROGRESS® training nutrition they deserve!

For additional nutrition, try PGA Consulting

Feed your thin clients fortified DWP!

PGA COURSES
The Training of Gurus

See our website for nutrition information
www.pgasmarts.com

Extreme PROGRESS™

Care and Feeding of SMARTOBJECTS 2™

Introduction to PROGRESS Programming™

Beyond V6™ (Jump-Start to V8 Character)

Non-Programmer's PROGRESS™

for each customer
name:screen-value
enable all
update order

prompt-for item-num
display invoice-total
output to printer

wait-for close of this-procedure

Feeding Programmers for 16 Years

Paul Guggenheim & Associates, Inc. is an authorized DWP (Dynamic WebClient Platform) and Jargon Software reseller



Paul Guggenheim & Associates, Inc.
1788 Second St., Suite 201 Highland Park, IL 60035
Phone: (847) 926-9800 Fax: (847) 926-9805
Web: www.pgasmarts.com



Table of Contents

Start by doing what is necessary, then what is possible, and suddenly you are doing the impossible.

-- St. Francis of Assisi

Convert EDITOR to ROWS	Armando Pensado	2
SAX and XPath	Tony Lavinio	8
Fathom Management Overview with a Focus on Jobs	Brian Bowman	11
On Listening to Users Who Don't Talk	Arthur Fink	15

GOT HEADACHES?

Do Any of these Describe You?

- ◆ You have a new Progress-based package you're implementing but want some customizations.
- ◆ The users of your existing application are asking for numerous modifications.
- ◆ You want to upgrade; you've made customizations; you don't want to spend a fortune "going back" to vanilla code but you don't want to lose what you've implemented.

Do You Need to:

- ◆ Add or remove some fields in an existing screen, change some labels or defaults, or even rewrite the screen entirely...
- ◆ Add your own tracking screens in the middle of the application flow...
- ◆ Put your own messages, validation or extended business rules into the application...
- ◆ Add fields or indexes to existing tables...
- ◆ Get better rollup information, and get it in real-time...

TailorPro is the solution to all these problems, and many more. It is a Progress-based application which allows you to insert your own application logic anywhere in your current programs. It enables you to add fields not tracked by your application, and perform data entry to them in the "middle" of existing screens. No source code or DB changes required!

TailorPro Saves You Money by:

- ◆ Cutting operating costs in your company by making your users more efficient.
- ◆ Dramatically reducing the implementation costs for custom application components.
- ◆ Eliminating the future costs of upgrading a customized application.

Look into the **TailorPro** features on our web demo: www.wss.com/tailorpro

White Star Software

voice: 970.963.3545

tailorpro@wss.com

fax: 970.963.3548