The Progress Electronic Magazine

In this issue:

Publisher's Statement: 1
Article: Saving States with Webspeed Based Applications 2
Article: Shared Libraries on UNIX with Progress........ 13
Product Updates and Releases: 17
Jobs Available: 18
Contractors/Consulting Companies Available: 18
Community Announcements: 18
Publishing Information: 19
Article Submission Information: 19
Upcoming Articles: 19

Do you like what your reading? Feel free to distribute this document to co-workers!

Did you sign up to receive this E-Zine? Send email to scott_auge@yahoo.com to subscribe!

Publisher's Statement:

Wow! So many people interested in Progress out there – sometimes, when one is sequestered away in a cubicle someplace; one doesn't realize how vibrant the Progress community is! Out here in Silicon Valley I know of a few people creating apps with Progress, but sometimes one's vision is limited!

When I started this, I figured there would be 50 or 60 people interested – happily I am glad to say, there are literally hundreds of people reading this E-Zine. Very exciting! Perhaps one day it will reach into the thousands!

This months main focus is saving state information in a Webspeed application. As we know, developing applications on the web can be very different from the normal GUI/Character connection. Having these ideas available will help make developing web applications more secure and functional.

Since I do not have a web site anymore, I am also including selected texts from my archives so that they can still be available out there. This month I am including the page regarding shared libraries on UNIX.

Soon we will be hearing about other applications developed with Progress tools. Unlike the Progress Email Group, I happily accept such news (though it may not be printed!)

Some people will be like "oh no – this has become a marketing rag!' but I do not feel so.
I think that hearing about these applications and companies will 1) identify companies that use Progress – a common question asked. 2) Knowledge of this can help people who are interested in Progress jobs to find such companies. 3) Knowing of other applications available could help people build up their IT infrastructure based on Progress apps – something that is good for all of us!

To your success,

Scott Auge
Well done is better than well said.
---Gwen Haymore

What do you want to read about?

Send email to scott_auge@yahoo.com… GUI and language tricks are the arena of Progressions from White Star Software – this Zine focuses on UNIX/Web/System Integration programming…

Article: Saving States with Webspeed Based Applications

Written by Scott Auge scott_auge@yahoo.com

When we create web applications, we quickly discover there is a difficulty in programming stateless applications, and that is all our data in variables disappears when we travel from web page to web page.

One common way to keep this kind of information is to store the information in hidden fields on the page. Users might be able to create problems with the application by saving the page to disk and then manipulating these values – causing the program to potentially blow up.

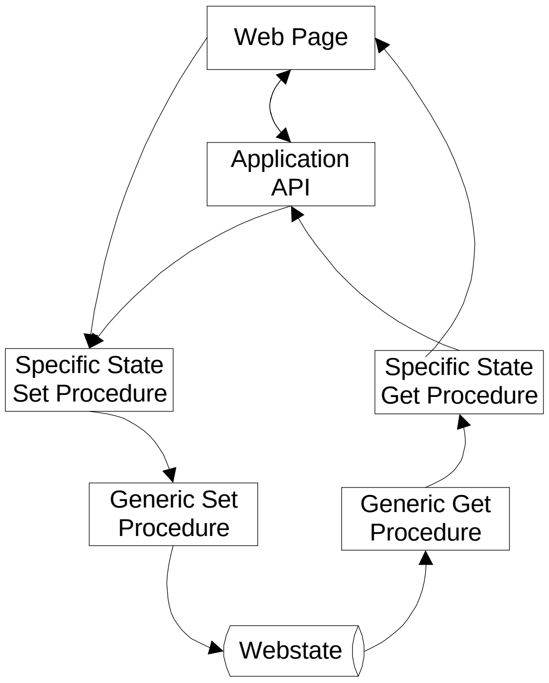
Sometimes we will want to create hyperlinks to other pages that will require data – creating these hyperlinks can be hard to read and confusing – I have found that a lot of data stored in the hyperlink can actually be stored in the database. This simplifies programming the link in terms of syntax. Also, there is the above-mentioned problem with curious or malicious users who might save the page to disk and fiddle with the values stored in the link.

Sometimes we wish to know if a user has been to a certain web page, or has passed some kind of test (such as they entered a valid credit card number or the like.) This state saving architecture will help identify that the user has passed these kinds of tests when interacting with other pages of the web application.

Implementing this kind of architecture requires a method of identifying the user. This can be achieved by using a cookie or one of those hidden fields. What we wish to store in these data spots are a random unique string to the user. Thus if the user decides to play around with these values, the likelihood of them switching over to another user's ID is very remote. This can be achieved with the `RandomString.p` program described in issue one of this publication.

Below is a diagram describing the use of the procedures one would want to create with this kind of architecture in mind. As can be seen, there is the top level which is the web page presented to the user. Below that might be some APIs that perform some business purpose or logic and can communicate errors back to the page. These two top level programs call into state programs that are tuned to handle specific states. These state programs then call the more generic state programs.

In this article, I will present an include file that would be in a Web page, as well as two programs tuned to handling the state "who is logged in" as well the generic state programs.



Calling Tree for Web State Architecture

Saving states requires some database work - the following is a report for the WebState table:

```
25/04/01 06:49:08       PROGRESS Report
Database: /db/dvd/data/dvd (PROGRESS)




==========================================================================
============================= Table: WebState ===========================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- -------------------
WebState                       webstate           4     2 WebState

  Description: Used to store session information about a web based log in.
```

```
 Storage Area: N/A


============================= FIELD SUMMARY =============================
============================= Table: WebState ===========================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                Data Type   Flags Format          Initial
----- ------------------------- ----------- ----- --------------- ----------
   20 Category                  char        i     x(8)
   40 Data                      char              x(8)
   30 Name                      char        i     x(8)
   10 SessionID                 char        i     x(8)

Field Name                      Label                 Column Label
------------------------------- --------------------- ----------------------
Category                        Category              Category
Data                            Data                  Data
Name                            Name                  Name
SessionID                       SessionID             SessionID


============================= INDEX SUMMARY =============================
============================= Table: WebState ===========================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- ---------------------------------
      key1                               1 + SessionID

pu    pukey                              3 + SessionID
                                           + Category
                                           + Name

** Index Name: key1
 Storage Area: N/A
** Index Name: pukey
 Storage Area: N/A


============================= FIELD DETAILS =============================
============================= Table: WebState ===========================
```

The `SessionID` is a unique random string that identifies which user the data is stored for.

The `Category` field is used to group pieces of data together. For example, if data is related to an address, then the category value might be “Address”.

The `Name` field actually names the data. One could think of it as a variable name within the database.

The `Data` field stores the data. Note that in this implementation, the value is a character type. This is because a character is the most basic form of storage – conversions between integer, real, logical, etc. can be performed with functions. It is assumed the programmer will know which type the value should be cast into to properly use it. This simplifies the arguments to the procedures that set and get the data.

A generic procedure is provided below to write values into the WebState table. If the state is available, the procedure will update it. If it is not available, it will create it. How to call the program can be seen later on in programs tuned to handle a certain state.

```
/*
 * WriteState.p
 * Written by Scott Auge
 *
 * Given a session id, category, and state name, set the value.
 *
 */

DEF VAR RCSVersion AS CHARACTER INIT "$Header:
/home/appl/dvd/src/State/RCS/WriteState.p,v 1.1 2001/04/02 02:58:25 root Exp
root $" NO-UNDO.

DEF INPUT PARAMETER pSessionID   AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pCategory    AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pName        AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pValue       AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pError      AS CHARACTER NO-UNDO.

{Error.i}

/* Determine if there is a state of this session.  If so, we will just over-
 * write the values.  If there is not, we will create a record.  Note we look
 * up the record with a NO-WAIT option in case it is used by some other
 * program.  It is up to the calling program to determine what it should do
 * under this condition.  (Recommended is to pause and try again.)
 */

FIND WebState EXCLUSIVE-LOCK
```

```
WHERE WebState.SessionID = pSessionID
  AND WebState.Category = pCategory
  AND WebState.Name = pName
  NO-WAIT NO-ERROR.

IF NOT AVAILABLE WebState THEN
  IF LOCKED WebState THEN DO:
    ASSIGN
    pError = {&LOCKED}.
    RETURN.
  END. /* NOT AVAILABLE AND LOCKED! */
  ELSE DO:
    CREATE WebState.
    ASSIGN
    WebState.SessionID = pSessionID
    WebState.Category = pCategory
    WebState.Name = pName.
  END.

ASSIGN
WebState.Data = pValue.

ASSIGN
pError = {&NOERROR}.
```

Of course, once a state is written into the table, a way to get the state out of the table needs to be available. This procedure is a generic procedure that would be called by more discrete procedures that deal with specific states.

```
/*
 * ReadState.p
 * Written by Scott Auge
 *
 * Given a session id, a category, and a name, return the session value.
 *
 */

DEF VAR RCSVersion AS CHARACTER INIT "$Header:
/home/appl/dvd/src/State/RCS/ReadState.p,v 1.1 2001/04/02 02:58:24 root Exp
root $" NO-UNDO.

DEF INPUT PARAMETER pSessionID  AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pCategory   AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pName       AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pValue     AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pError     AS CHARACTER NO-UNDO.

{Error.i}

FIND WebState NO-LOCK
WHERE WebState.SessionID = pSessionID
  AND WebState.Category = pCategory
  AND WebState.Name = pName
  NO-WAIT NO-ERROR .

IF NOT AVAILABLE WebState THEN
  IF LOCKED WebState THEN DO:
    ASSIGN pError = {&LOCKED}.
    RETURN.
  END.
  ELSE DO:
    ASSIGN pError = {&NORECORD}.
    RETURN.
  END.

ASSIGN pValue = WebState.Data.

ASSIGN pError = {&NOERROR}.
```

Once a user has logged out, or has not performed any interaction with the web application with the given SessionID for a given period of time, the state information should be eliminated. This procedure deletes all the information associated with a session.

```
/*
 * ClearState.p
 * Written by Scott Auge
 *
 * Given a state ID, clear all entries in the state table for the user.
 * BEWARE:  Maybe LOCKING conditions in this API!!!!
 */

/* RCS Stuff */

DEF VAR RCSVersion AS CHARACTER INIT "$Header:
/home/appl/dvd/src/State/RCS/ClearState.p,v 1.1 2001/04/02 02:58:22 root Exp
root $" NO-UNDO.

DEF INPUT PARAMETER pSessionID AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pError    AS CHARACTER NO-UNDO.
```

```
{Error.i}

FOR EACH WebState EXCLUSIVE-LOCK
WHERE WebState.SessionID = pSessionID:

  DELETE WebState.

END. /* FOR EACH WebState */

ASSIGN pError = {&NOERROR}.
```

(As an aside, one may wish to store a UserID to SessionID state record. Often when a user closes the browser without logging out and then logs in again, a new SessionID will be created for them. Using this record can allow the application to rewrite all the old SessionID values to the new SessionID value – thereby allowing the application to "remember" where the user left off – kind of a neat trick that cannot be achieved with "state based" connections.)

Once the more generic methods are available for use, we wish to build on those with other procedures that are more specifically tuned to a purpose. Since logging in a user is a common requirement for a program, this article will focus on these procedures.

As can be seen, these programs really are more like "wrappers" to the generic programs. They set the category and name of the storage spot for the data that is sent into them.

```
/* SetLogin.p
 * Written by Scott Auge
 *
 */

DEF VAR RCSVersion AS CHARACTER INIT "$Header:
/home/appl/dvd/src/State/RCS/SetLogin.p,v 1.1 2001/04/02 02:58:25 root Exp root
$" NO-UNDO.

DEF INPUT PARAMETER pCookie AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pUserID AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pError AS CHARACTER NO-UNDO.

RUN State/WriteState.p (INPUT pCookie,
                        INPUT "Login",
                        INPUT "Cookie",
                        INPUT pUserID,
                        OUTPUT pError).
```

```
/* GetLogin.p
 * Written by Scott Auge
 *
 * Obtain login from state table
 *
 */

DEF VAR RCSVersion AS CHARACTER INIT "$Header:
/home/appl/dvd/src/State/RCS/GetLogin.p,v 1.1 2001/04/02 02:58:24 root Exp root
$" NO-UNDO.

DEF INPUT PARAMETER pCookie   AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pLoginID AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pError   AS CHARACTER NO-UNDO.

RUN State/ReadState.p (INPUT pCookie,
                       INPUT "Login",
                       INPUT "Cookie",
                       OUTPUT pLoginID,
                       OUTPUT pError).
```

Here we see a program that can be included in a web page to determine if the user has logged in or not.

We do not put the ID for the customer record out on the net, but a session ID because I believe one should put as little usable information on the web as possible. There are some other functions `AlertBox` and `Relocate` that output javascript to pop-up dialog boxes and relocate the browser to different pages (perhaps another article!)

Secure.i

```
DEF VAR lCustomerID AS CHARACTER NO-UNDO.
DEF VAR lSessionID  AS CHARACTER NO-UNDO.
DEF VAR lError      AS CHARACTER NO-UNDO.

{AlertBox.i}
{Relocate.i}

ASSIGN lSessionID = GET-VALUE("SwapperSession").

RUN State/GetLogin.p (INPUT lSessionID,
                      OUTPUT lCustomerID,
                      OUTPUT lError).
```

```
IF lError <> {&NOERROR} THEN DO:

  AlertBox ("Your not logged in!").
  Relocate ("index.html").
  RETURN.

END.

FIND Customer NO-LOCK
WHERE Customer.CustomerID = lCustomerID
NO-ERROR.

IF NOT AVAILABLE Customer THEN DO:

  AlertBox ("Can’t find your customer record!").
  Relocate ("index.html").
  RETURN.

END.
```

To finish off this article, I will show how to store data in the database. Some applications may have the user go though multiple screens to gather information about some task to perform. In one of my experiences, the data was passed along in hidden fields, and when the user reloaded the final screen, which created a ticket, multiple tickets of the same issue were created.

By storing the data in the database, this kind of action can be prevented because upon use of the data in the task, the data can be removed. The logic can realize the missing data and stop this action from occuring. The following procedures can provide that functionality.

```
/* SetData.p
 * Written by Scott Auge
 *
 * Set a piece of data
 *
 */

DEF INPUT PARAMETER pCookie    AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pDataName  AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pDataValue AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pError    AS CHARACTER NO-UNDO.

RUN State/WriteState.p (INPUT pCookie,
                        INPUT "WIPData",
                        INPUT pDataName,
                        INPUT pDataValue,
                        OUTPUT pError).
```

The following allows the retrieval of that data.

```
/* GetData.p
 * Written by Scott Auge
 *
 * Get a piece of data
 *
 */

DEF INPUT PARAMETER pCookie     AS CHARACTER NO-UNDO.
DEF INPUT PARAMETER pDataName   AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pDataValue AS CHARACTER NO-UNDO.
DEF OUTPUT PARAMETER pError     AS CHARACTER NO-UNDO.

RUN State/ReadState.p (INPUT pCookie,
                       INPUT "WIPData",
                       INPUT pDataName,
                       OUTPUT pDataValue,
                       OUTPUT pError).
```

One may want to create a `ClrData.p` procedure that deletes the record from the WebState table.

One can create a table to help identify the names of the data:

Data Name	Data Purpose
CustomerID	Stores the current customer identification number
Street1	Street1 of address
HasEnteredCreditCard	Has the user entered a valid credit card?

An example piece of code using the SetData.p procedure would be:

```
RUN State/SetData.p (INPUT pCookie,
                     INPUT “HasEnteredCreditCard”,
                     INPUT YES,
                     OUTPUT pError).
```

About the author: Scott Auge is a programmer with Computational Research Systems and Analysts Express Inc. He has been programming in the Progress environment since 1994. Works have included E-Business initiatives and focuses on web applications on UNIX platforms. scott_auge@yahoo.com

Article: Shared Libraries on UNIX with Progress

Written by Scott Auge scott_auge@yahoo.com

What is a shared library?

A library contains at least one subroutine that can be used by multiple processes. Programs and subroutines are linked as before, but the code common to different subroutines is combined in one library file that can be loaded at run time and shared by many programs.

Why use shared libraries with Progress?

One of the main purposes of using shared libraries outside the Progress world is to shrink the amount of memory a program requires when multiple instances of it are running. Another use of shared libraries is to load only portions of the program into memory, leaving other portions that are not required until called upon. Also, if functionality of the routines are common to many programs, such as the square root function or the like, the library can be used by many programs thereby leaving one copy of the routine in memory instead of the number of programs needing the routine.

The main reason shared libraries are interesting in the Progress world however, is to extend the reach of the Progress client or Application Server to abilities and functionality outside the 4GL programs. This can include integration of the 4GL program with messaging software such as MQ Series or Crossworlds. It may be:

- the need to access other DBs such as DB2 through it's CLI client libraries (an admittedly painful method of handing cross technology programming without a dataserver)
- to call the C api's of a middleware package for tax computation (VERTEX)
- to call native compiled routines for speed issues

to utilize open source code (generally written in C or C++) The biggest main reason is so as not to need to use pro-build and the HLC interface. While this works pleasantly, it does take some research to figure out how to use. This is because it does not use the more common method of a makefile, but a script generated by a utility tool called ProBuild. Using the HLC interface also requires the re-linking of the client _progres file and distribution of that little baby all around. It is a bit of a nuisance for version changes as it is one more step that is not commonly understood unless the research has been done.

Discussion of a C function we wish to access via the Progress 4GL:

```
/* Test using .a's/.so’s with Progress v9 or better
 * Written by Scott Auge
 */

#include <stdio.h>

int testit (int i)
  {

   FILE *f;

   f = fopen ("/tmp/this.test", "wb");

   fprintf (f, "It Worked! Try %d\n", i);

   fclose (f);

   return 0;
  }
```

This is a very simple program that receives a number from the 4GL code and writes it to a file on the hard drive. Personally I find it not very intuitive for handling files on drives from Progress' 4GL. Hence this little program to show how to use - what is more familiar to me - the use of C APIs to manipulate file data.

Discussion of the Progress 4GL code we use to access the C function:

```
/* See if we can invoke a shared library file from Progress V9
 * Written by Scott Auge
 */

DEF VAR a AS INTEGER NO-UNDO.
DEF VAR b AS INTEGER NO-UNDO.
```

```
PROCEDURE testit EXTERNAL "/home/aa52271/code/c/progress.so/test.a"
CDECL:
  DEF INPUT PARAMETER j AS SHORT.
  DEF RETURN PARAMETER i AS SHORT.
END.

ASSIGN a = 1
       b = 0.

RUN testit (INPUT a, OUTPUT b).
```

This is the Progress 4GL code that calls the C routine expressed in the previous code segment. Note that the procedure expression has been changed to look for a function in a library, and that parameter passing should happen in the C order. Also note that there is a required return type parameter. At the moment of this writing, all routines must return a value, that is - they cannot be void types.

Calling the function is the same as calling a 4GL program, which is a refreshing change from the call statement needed for HLC programming. To aid 4GL programmers, one only needs to inform them of the parameters, to include a file that would define the procedures at the top of their program that might use this function, and to continue programming as they always have.

This is good because it is more elegant and standard.

Discussion of the makefile used to create a shared library under AIX:

```
all:    test.c
        cc -c test.c
        cc -bE:test.exp -bM:SRE -bnoentry -o test.a test.o

run:    run.c
        cc -o run run.c -L: test.a -bnoquiet

clean:
        rm *ped
        rm core*
        rm protrace*
        rm *.o
        rm test.a
        rm run
        rm run1
```

Discussion of the makefile used to create a shared library under Linux:

```
all:    test.c
        gcc -c test.c
        gcc -shared -o test.a test.o


clean:
        rm *ped
        rm core*
        rm protrace*
        rm *.o
        rm test.a
        rm run
        rm run1
```

Discussion of the ease of administration with shared libraries:

One of the nicer aspects of using shared libraries is for the administrator's tasks. On UNIX, the use of the make command and a makefile are far better known than the ProBuild utility and the running of it's script. The programmer can create a makefile that will compile, clean up, and deploy the shared libraries in the proper directories.

The process of adding in C integrations changes from:

Start probuild

Choose the parts that are needed (do I need networking? etc.)

Type in all the little .o's

Run the script

Copy the resulting _progres to where it needs to be

Have people restart their sessions, restart background processes to a more simple:

Type `make`

Distribute new .so

Have people restart their sessions, restart background processes

Conclusion:

Using shared libraries can make the integration needs of Progress 4GL programs much more simpler and even possible.

This is important as there seems to be more and more "off the shelf" programs becoming part of a company's infra-structure.

It handles such needs such as paging, credit card checking, etc. that require access to behaviors and functionallity outside the handling of database manipulations that the 4GL is best at doing.

About the author: Scott Auge is a programmer with Computational Research Systems and Analysts Express Inc. He has been programming in the Progress environment since 1994. Works have included E-Business initiatives and focuses on web applications on UNIX platforms. scott_auge@yahoo.com

Product Updates and Releases:

Do you have a new product or release of software – let me know and it will be included in the latest E-Zine. Price is $10.00 per article per issue.

Computer Based Training CD available for Webspeed Programming!

Are you curious about programming in Webspeed? Want to update your skills? See what you’re getting into?

A CD-Rom is now available with source code and presentations with voice-over explainations for learning fundamental programming with Webspeed 3.1.

Topics include how Webspeed presents HTML to the browser, how the browser presents information to Webspeed, and how Webspeed updates the database(s) it is attached to. Focuses on E4GL programming and how to architect your source code for easy maintenance and understanding.

Price is $250.00 US, re-sale for training classes negotiable. Windows 95 or better required, PowerPoint or the free PowerPoint reader from www.microsoft.com required. RealPlayer Basic needed for MP3 voice-overs. Soundcard required.

Contact: scott_auge@yahoo.com

Jobs Available:

Do you have a job opening available? Let me know and it will be included in the latest E-Zine. Price is $10.00 per listing per issue.

Contractors/Consulting Companies Available:

If you do work in the Progress world – let me know and I will be able to include you! Price is $10.00 per listing per issue.

Analysts Express Inc.

The Reliable Source for I.T. Consulting and Recruiting
Contact: James Arnold @ 888/889-9091

Jay A. Martin
DataChase, LLC
www.data-chase.com
info@data-chase.com

Providing contract programming and writing services.

Computational Research Systems (CRS)
scott_auge@yahoo.com

Creation of modules and products for re-sale. Internet/Intranet programming for E-Business and Customer Self-Serving web sites for manufacturing/service/law enforcement industries.

Community Announcements:

A place to announce Progress User Groups, Open Source Exchanges, etc. No charge!

Wonder where to find those Progress marketing articles are?

http://www.progress.com/analyst/
http://www.progress.com/profiles/index.htm
http://www.progress.com/success/index.htm

Publishing Information:

Scott Auge publishes this document. I can be reached at scott_auge@yahoo.com.

This electronic magazine is to be published once a month.

Currently there are over 200 subscribers and companies that receive this mailing!

Computational Research Systems assists in the publication of this document:

Computational Research Systems
4506 Carlyle Court Suite 712
Santa Clara, CA 95054

Article Submission Information:

Please submit your article in Word or text. Please include a little bit about yourself for the About the Author paragraph.

Looking for technical articles, *marketing Progress* articles, articles about books relevant to programming/software industry, white papers, etc.

Currently, payment for an article is not possible. But – articles CAN enhance your prestige and name recognition – and these things can translate into money!

Upcoming Articles:

Using Parameter records for easier configuration of your application/Installing Progress Webspeed/RDBMS on Linux – June 2001

Sending and receiving email with your progress application – July 2001

Background reporting processes for web applications – August 2001