The Progress Electronic Magazine

In this issue:

Publisher's Statement: 1
Coding Article: Receiving E-Mail From A POP3 E-mail Box 2
Management Article: Determining The Number Of Agents A Webspeed Application Needs. 9
Product Updates and Releases: 15
Jobs Available: 16
Contractors/Consulting Companies Available: 16
Community Announcements: 16
Publishing Information: 17
Article Submission Information: 17
Upcoming Articles: 17

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

Though intended for users of the tools provided by Progress Software Corporation, this document is NOT a product of Progress Software Corporation.

Publisher's Statement:

As you can see, I have converted over from issuing Word documents to Acrobat PDF files. This is because some people do not have Word 2000 and need to upgrade in order to properly read the document. Ah yes, Microsoft and their upgrade treadmill at work – but it is effecting me now! Hopefully, everyone will be able to read the document using the freely available Acrobat Reader software available from http://www.adobe.com on all sorts of platforms and operating systems. As an added benefit – it appears that PDF files are actually a couple K smaller than Word .doc files.

In this issue we get to nuts and bolts about e-mail. This is done completely within the Progress environment. I have chosen to do so because not everyone is on Windows to call into an OCX, and not everyone is on UNIX to just ask a tool available to push out some mail. Since this can be complex, I have split the article into two parts. Part 1 this month discusses how to receive e-

mail into your Progress application. Part 2 next month will be sending e-mails with your Progress application using the internet standard protocols.

I have decided to make the second article of this and future E-Zines management oriented. This month's E-Zine is a document that was on my web site before it was taken down: How to figure out how many web agents does one need to license for an application. In the future there will be items such as calculating Return On Investment, Project Management, and other IT and software management agenda items that focus on measuring resources and benefits.

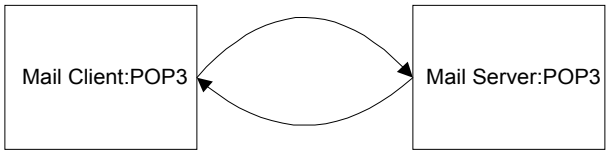
To your success,

Scott Auge
Founder, Amduus Information Works

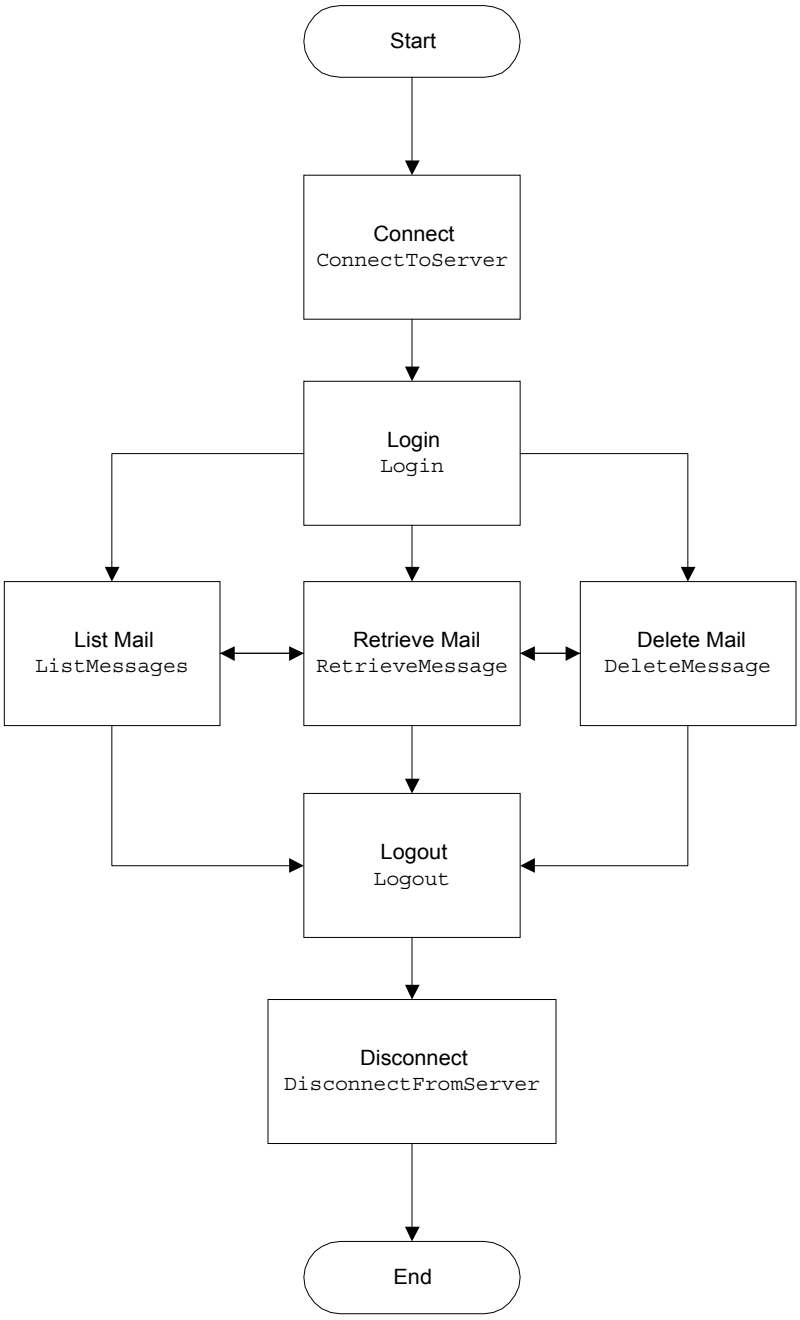
Coding Article: Receiving E-Mail From A POP3 E-mail Box

Written by Scott Auge scott_auge@yahoo.com

This first of a series of articles is for the retrieval and sending of mail. In this article, we will use the POP3 protocol to interact with a POP3 daemon that manages mailboxes. In short, the architecture appears as below. A mail client program interacts with a mail server. What we will be programming is the mail client portion. Long enough the mail server follows the POP3 protocol internet standard, the mail client should be able to interact with it irregardless of manufacturer or platform.



In order for these programs to talk to each other, a protocol of communication needs to be set up. The POP3 protocol is pretty simple. It is composed of "words" that will identify who the user is, how many messages a user has, the retrieval of a message, and the deletion of a message from the mail box. The following flow chart shows the basic steps involved in these activities. The name of the API to achieve the step is included.



The POP3 API is composed of a set of internal procedures. This code is stored in an include file so that it can be made easily available to little test programs to let you play with the source. One may want to put it into a persistent procedure for commercial use.

```
/* file: pop3api.i
 * Written by Scott Auge
```

```
 * scott_auge@yahoo.com
 *
 * This file is released into the public domain.
 *
 * Written by Scott Auge
 * Perform operations on POP3 mail servers
 *
 * Use this file as an include or a persistance procedure.
 */

DEF VAR RCSVersionpop3apii AS CHARACTER INIT "$Header:
/home/sdk/mail/source/RCS/pop3api.i,v 1.1 2001/01/11 15:35:10 root Exp
$" NO-UNDO.

{pop3.i}
{memptr.i}
```

The first step is to connect to the mail server. Connections to mail servers are accomplished by a TCP port connection on port 110. This is a commonly known port and generally reserved on systems. This code uses the Progress 4GL to connect to the mail server. It is pretty straight forward.

The complementary function is to disconnect from the server. This is accomplished with 4GL in the `DisconnectFromServer` procedure.

```
/*************************************************************/
/* Create the socket, connect to server.                     */
/*************************************************************/

PROCEDURE ConnectToServer:

  DEF INPUT PARAMETER MailServer AS CHARACTER NO-UNDO.
  DEF OUTPUT PARAMETER hHandle AS HANDLE NO-UNDO.
  DEF OUTPUT PARAMETER Result AS INTEGER NO-UNDO.

  DEF VAR hSocket AS HANDLE NO-UNDO.
  DEF VAR lConnectParameters AS CHARACTER NO-UNDO.

  CREATE SOCKET hSocket.
  hSocket:SET-SOCKET-OPTION ("TCP-NODELAY", "FALSE").
  hSocket:SET-SOCKET-OPTION ("SO-LINGER", "FALSE").

  ASSIGN lConnectParameters = '-H ' + MailServer + ' -S pop3'.

  hSocket:CONNECT(lConnectParameters) NO-ERROR.

  IF hSocket:CONNECTED() = FALSE THEN DO:
    ASSIGN Result = {&NOCONNECT}.
    RETURN.
  END.

  ASSIGN Result = {&NOERROR}
         hHandle = hSocket.
```

```
END. /* PROCEDURE ConnectToServer */

PROCEDURE DisconnectFromServer:

  DEF INPUT PARAMETER hSocket AS HANDLE NO-UNDO.

  hSocket:DISCONNECT().

END.  /* PROCEDURE DisconnectFromServer */
```

One a connection has been established to the server, the mail client needs to identify who is using it to the mail server. This is accomplished by a token sequence as follows:

```
USER [userlogin] <CR>
PASS [userpassword] <CR>
```

where <CR> is a carriage return.

The mail server will then return information detailing the success or failure of authentification. Currently this code does not take that into account. Some additional work will need to be done to handle circumstances where the userid and password are denied by the server.

```
/*************************************************************/
/* Login into the server as a user to access that mail box.  */
/*************************************************************/

PROCEDURE Login:

  DEF INPUT PARAMETER hSocket AS HANDLE NO-UNDO.
  DEF INPUT PARAMETER pUserID  AS CHARACTER NO-UNDO.
  DEF INPUT PARAMETER pPassword AS CHARACTER NO-UNDO.
  DEF OUTPUT PARAMETER pMessageText AS CHARACTER NO-UNDO.

  DEF VAR lPOP3Command AS CHARACTER NO-UNDO.

  ASSIGN lPOP3Command = "USER " + pUserID + "~n".
  RUN lmemput (mData, lPOP3Command).
  hSocket:WRITE(mData,1,LENGTH(lPOP3Command)) NO-ERROR.

  RUN ProcessServerResponse (INPUT hSocket, OUTPUT pMessageText).

  ASSIGN lPOP3Command = "PASS " + pPassword + "~n".
  RUN lmemput (mData, lPOP3Command).
  hSocket:WRITE(mData,1,LENGTH(lPOP3Command)) NO-ERROR.

  RUN ProcessServerResponse (INPUT hSocket, OUTPUT pMessageText).

END. /* PROCEDURE Login */
```

Messages are numbered under POP3. To aquire the messages available, one would enter the following token:

```
LIST <CR>
```

to which the server would reply:

```
1 <CR>
2 <CR>
3 <CR>
```

if there were three messages available.

The following code would translate that answer into a string “1,2,3” for easier manipulation in the 4GL with the ENTRY() and NUM-ENTRIES() functions.

```
/*************************************************************/
/* List messages available on the server.                    */
/*************************************************************/

PROCEDURE ListMessages:

  DEF INPUT PARAMETER hSocket AS HANDLE NO-UNDO.
  DEF OUTPUT PARAMETER pMessageText AS CHARACTER NO-UNDO.

  DEF VAR lPOP3Command AS CHARACTER NO-UNDO.
  DEF VAR i AS INTEGER NO-UNDO.
  DEF VAR t AS CHARACTER NO-UNDO.

  ASSIGN lPOP3Command = "LIST~n".
  RUN lmemput (mData, lPOP3Command).
  hSocket:WRITE(mData,1,LENGTH(lPOP3Command)) NO-ERROR.

  RUN ProcessServerResponse (INPUT hSocket, OUTPUT pMessageText).


  ASSIGN pMessageText = trim (pMessageText).

  /*--- remove hard carriage returns with spaces ---*/
  DO WHILE (INDEX(pMessageText, CHR(13)) <> 0):
    ASSIGN SUBSTRING(pMessageText, INDEX(pMessageText, CHR(13))) = " ".
  END.
  /*--- remove hard carriage returns with spaces ---*/
  DO WHILE (INDEX(pMessageText, CHR(10)) <> 0):
    ASSIGN SUBSTRING(pMessageText, INDEX(pMessageText, CHR(10))) = ",".
  END.

  DO i = 2 TO NUM-ENTRIES(pMessageText):
    ASSIGN t = t + ENTRY(1, ENTRY(i, pMessageText), " ") + ",".
```

```
  END.
  ASSIGN pMessageText = SUBSTRING(t, 1, LENGTH(t) - 3).

END. /* PROCEDURE ListMessages */
```

Once one has a list of messages, those messages can be retrieved from the server. Note that the messages are in a standard format, but it is quite readable by humans. Retrieval of a message entails sending the token command to retrieve a message as well as the message ID to retrieve:

```
RETR [MessageID] <CR>
```

to which the server will return the mail message. The `MessageID` would be one of the numbers obtained in the `ListMessages` procedure.

```
/*************************************************************/
/* Retreive message from the server.                         */
/*************************************************************/

PROCEDURE RetrieveMessage:

  DEF INPUT PARAMETER hSocket as HANDLE NO-UNDO.
  DEF INPUT PARAMETER pMessageID AS CHARACTER NO-UNDO.
  DEF OUTPUT PARAMETER pMessageText AS CHARACTER NO-UNDO.

  DEF VAR lPOP3Command AS CHARACTER NO-UNDO.

  ASSIGN lPOP3Command = "RETR " + pMessageID + "~n".
  RUN lmemput (mData, lPOP3Command).
  hSocket:WRITE(mData,1,LENGTH(lPOP3Command)) NO-ERROR.

  RUN ProcessServerResponse (INPUT hSocket, OUTPUT pMessageText).

END. /* PROCEDURE RetrieveMessage */
```

Deleting a message from the server is done by issuing the token command:

```
DELE [MesssageID] <CR>
```

Where one of the numbers from the ListMessage procedure is used as the MessageID. Note that when one deletes a message, the following messages are renumbered n – 1. When making multiple deletions, this will need to be taken into account to prevent deleting the wrong message.

```
/*************************************************************/
/* Delete a message from the server.                         */
/*************************************************************/

PROCEDURE DeleteMessage:

  DEF INPUT PARAMETER hSocket as HANDLE NO-UNDO.
```

```
  DEF INPUT PARAMETER pMessageID AS CHARACTER NO-UNDO.
  DEF OUTPUT PARAMETER lServerResponse AS CHARACTER NO-UNDO.

  DEF VAR lPOP3Command AS CHARACTER NO-UNDO.

  ASSIGN lPOP3Command = "DELE " + pMessageID + "~n".
  RUN lmemput (mData, lPOP3Command).
  hSocket:WRITE(mData,1,LENGTH(lPOP3Command)) NO-ERROR.

  RUN ProcessServerResponse (INPUT hSocket, OUTPUT lServerResponse).

END. /* PROCEDURE DeleteMessage */
```

A clean logout of the user is desired else the mail server connection will usually hang in someway till a timeout expires to automatically perform housekeeping. (Note I am not saying that a mail server will hang for ALL users for this situation, but a user not logging out and disconnecting cleanly may receive "Mail Box Locked" errors on re-login until this circumstance has been cleaned up. Under UNIX one would kill the mail daemon process for that user.)

The mail server is informed by the mail client of a desire to logout by issuing the following token:

```
QUIT <CR>
```

After doing so, the user will not be able to manipulate the mail box until the Login sequence of tokens has been re-issued.

```
/*************************************************************/
/* Logout of the server.                                     */
/*************************************************************/

PROCEDURE Logout:

  DEF INPUT PARAMETER hSocket as HANDLE NO-UNDO.
  DEF OUTPUT PARAMETER lServerResponse AS CHARACTER NO-UNDO.

  DEF VAR lPOP3Command AS CHARACTER NO-UNDO.

  ASSIGN lPOP3Command = "QUIT~n".
  RUN lmemput (mData, lPOP3Command).
  hSocket:WRITE(mData,1,LENGTH(lPOP3Command)) NO-ERROR.

  RUN ProcessServerResponse (INPUT hSocket, OUTPUT lServerResponse).

END.
```

The following code is pretty straight forward to pulling data off the socket connection. It makes some calls to procedures that take data from `MEMPTR` storage space, and place it into `CHAR` storage space.

```
/************************************************************/
/* Get a response from the server.                          */
/************************************************************/

PROCEDURE ProcessServerResponse:

  DEF INPUT PARAMETER hSocket AS HANDLE NO-UNDO.
  DEF OUTPUT PARAMETER lResponse AS CHARACTER NO-UNDO.

  DEF VAR i AS INTEGER INIT 0 NO-UNDO.

  /************************************************************/
  /* Sometimes we may need to wait a little bit for the server to */
  /* respond so we loop a bit till we get something interesting.  */
  /* If we dont, we just bail out of here.                    */
  /************************************************************/

  DO WHILE i < 6:
    IF hSocket:GET-BYTES-AVAILABLE() <> 0 THEN LEAVE.
    PAUSE 1 NO-MESSAGE.
    ASSIGN i = i + 1.
  END.

  IF hSocket:GET-BYTES-AVAILABLE() = 0 THEN RETURN.

  /************************************************************/
  /* We discover we have something to read from the socket, lets  */
  /* see what we got!  Pop it into raw memory, then put it into   */
  /* a 4GL varible.                                           */
  /************************************************************/

  RUN lmemset (mData, " ").
  hSocket:READ (mData, 1, GET-SIZE(mData), READ-AVAILABLE).

  RUN lmemget (mData, OUTPUT LResponse).

END. /* PROCEDURE ProcessServerResponse */
```

About the author: Scott Auge is the founder of Amduus Information Works. He has been programming in the Progress environment since 1994. His work have included E-Business initiatives and focuses on web applications on UNIX platforms.
scott_auge@yahoo.com

Management Article: Determining The Number Of Agents A Webspeed Application Needs.

Written by Scott Auge scott_auge@yahoo.com

Introduction:

This document is to help programmers and system administrators to determine the size of machines for web applications. It provides formulae for the calculation of memory and processor needs, as well as connection speeds. It also provides calculations for the number of web agents

needed for a Webspeed program using Progress Software Corporation (http://www.progress.com) development tool.

Calculating Renderings:

The first step is to determine the types of pages that will be rendered. If your web application has a login screen, this would be a page. If your application has a logout screen, this would be another page. If your application has static pages, a count of these should be made also. A quick worksheet to include this information is available below:

URL	Static	Description
login.html	No	Login to the web application
logout.html	No	Logout of web application
login_help.html	Yes	Help page for login
shw_user.html	No	Show User Accounts
...		

Calculating Hits:

Calculating a hit is different from rendering a page. A page is composed of HTML. Inside that HTML may be references to images. A hit is a connection to the web server to fully render a page. One hit is for the initial HTML sent to the browser for the page. The browser then parses the HTML looking for images it needs to retrieve to fully render the page. A retrieval of an image is also a hit on the web server.
To calculate the total hits to retrieve a page, one needs to determine the number of hits and the amount of data associated with a hit. A convenient worksheet to do so is below:

URL	login.html
HTML	3 Kb
logo.jpg	6 Kb
submit.jpg	10 Kb
line.jpg	4 Kb
Total Kb:	23 Kb
Total Hits:	4
Dynamic Creation Time:	1 Sec

We can see from the example that four hits are needed to render the page, and that the system will need to handle 23 Kb of data to process this page for one user. Also included is the usual time the page is loaded on a system that is acceptable. Some pages will take longer to create than others, simply because the algorithms involved or the amount of data coming from the database takes longer. Of course, we desire a page to be returned to the browser in under a second.

How "Hit Distribution" fits into the equation:

Through use of the application, one can determine the web page renderings that provide hits on the system. Hits are directed either to the web server for images, dynamically created pages, and static HTML. The brokers are only hit for dynamic HTML creation, however, brokers generally have more work to do pulling up data and handling business rules.
It is this Hit Distribution that helps determine the minimum size a machine needs to be. It is based on the breaking up of the time of the day into periods - a period can be 1 hour or 1 minute. The smaller segment of time the period represents, the better the results will be for hit analysis. A period need not be any smaller than the smallest Dynamic Creation Time of a page generated by Webspeed - so if your fastest web page will take 2 seconds to render, a period need not be any smaller than 2 seconds.
During this period, one examines the web pages that are rendered by the web application. This can help determine the number of hits asked of the web server, the agents, and the amount of data that is passed through the system.
One should run the application as a beta with some people using it to get a good measurement of the hit distribution. It goes to think that adding more people to the system will be a ratio of having the original beta people on the system, and that ratio can be applied to agents, web server processes, machine memory, connection requirements, and CPU requirements. If beta'ing is not possible, one will have to work the system as a user might to get the numbers. Of course, the more people that beta for you, the more accurate your analysis will be.
Below is a five minute measurement at the beginning of the morning. It should be noted that during certain times of the day, as well during the week, month, or year even - there may be differences in which web pages get the most attention.

Start Time	08:00
End Time	08:05
login.html	200
menu.html	200
workspace.html	200

We can see from this most basic example, that the total number of pages rendered were 600. By using the Calculating Hits worksheet, we can determine just how many accesses were asked of the web server and the transaction server. Also, we will have a general idea of how many kilobytes of data were asked for during this five minute time period.

URL	login.html	Totals
Start Time:	08:00	
End Time:	08:05	
Accesses	200	
Transaction Server Hits	1	200
Web Server Hits	4	800
Data Memory requirements	23 KB	4.49 MB

URL	menu.html	Totals
Start Time:	08:00	
End Time:	08:05	
Accesses	200	
Transaction Server Hits	1	200
Web Server Hits	12	2400
Data Memory requirements	170 KB	33.20 MB

URL	workspace.html	Totals
Start Time:	08:00	
End Time:	08:05	
Accesses	200	
Transaction Server Hits	1	200
Web Server Hits	1	200
Data Memory requirements	2 KB	0.39 MB

Start Time:	08:00
End Time:	08:05

Grand Accesses	600
Grand Transaction Server Hits	600
Grand Web Server Hits	3400
Grand Data Memory Requirements	38.08 MB

By creating additional work sheets like the above, one can see the total amount of work performed by the transaction server and the web server within that five minute period. From above, we can see that the machine would need at least 4.49MB of memory to handle data needs of the login.html screen for a period of five minutes (in the morning!)
As an aside, five minutes is 300 seconds and so for 200 accesses (at a second per request) the machine is spending 2/3 of that time processing login.html requests should we be only using one agent. This is definately a clue we are going to be interested in some number of agents. More on that in the Calculating Agents section of this document.
But memory wise, if we want to process all those requests within 1 second, we would need 4.49MB of memory dedicated to processing that page. This is over-kill as we could spread out the accesses over the entire five minutes evenly and find we only need 23KB of memory to work with to support that need (provided some people may need to wait five minutes to login!) But since the time period was 5 minutes, we know what memory needs we have for that time period - 4.49MB.
A five minute period is FOREVER in computer terms - it is better to use a five SECOND period, but that can be very time consuming to calculate all the five second periods in the day. Remember though, the smaller the period the more precise your needs will be, and that means less memory to buy as well less licenses to buy (for non-state holding applications).
Another way to do this is during beta testing, find those times that a lot is asked of the web server - this can be found in the web server logs and there are usually programs available for almost every web server that can read the logs and turn them into graphs of a sort. Then during the peak times calculate the needs as above.
Once the base numbers are available, one can use ratio's to determine needs for scaling up (or down.)

Calculating Agents:

Calculating the number of agents for a state based application is easy - how many users do you wish to use the application?
However, calculating the number of agents for a non-state based application becomes more interesting... .

In a non-state based application, the agents are shared between users. This makes for very efficient use of the memory, processor, and licensing cost of the application. In other words, once an agent has completed executing a business rule and rendering the HTML for the page to be returned, it can be used by another user - versus the more conventional way where memory, CPU cycles, and License are allocated to a user who may be off to a meeting or eating lunch!
However, it should be noted the application becomes a wee bit more difficult to program when it is non-state based. There are different techniques to accomplishing this.
A short way to calculate agents is to use the Grand Transaction Server Hits for the time period that has the largest Grand Transaction Server Hits value to determine the number of licenses. Again be aware of the length of your time period - smaller means a more accurate number of licenses.

Calculating Memory:

Calculating the amount of memory needed is based on how much data is expected to be processed in a time frame as well the number of agents and web server processes (or threads).
Note that web server memory will be based on what kind of web server is used. Some servers start new processes and some have a base process and then memory for each thread. See your web server documentation on how to calculate memory for it's process space.

Calculating Memory for Agents	
Number of Agents	100
Size of Agent	298162 (WS3.1 AIX)
Total:	28.43 MB
Calculating Memory for Web Server	
Number of Web Server Processes (or Threads)	200
Size of Process or Thread	1234713
Total:	235 MB
Calculating Memory for Rendering Data	
Maximum Rendering Data	38.08 MB
Operating System	
Minimum Memory	64 MB
Totals	
Recommended Memory	366 MB

Note that memory for brokers/agents/name server for the Progress transaction server is not included in this calculation as these are usually on a seperate machine.

Calculating Processor:

To be added later. Suggestions? One criteria is for the processor to handle the Maximum Rendering Data within seconds.

Calculating Connection Speeds:

This is based on the maximum rendering data required for a given time period for all time periods (anything less will have a choking point at that time period.) In this example, we are interested in moving 38.08 MB per second down the line in the five minute interval we investigated.

Calculating the Web Server Processes or Threads:

Based on the web server used. See it's manual.

About the author: Scott Auge is the founder of Amduus Information Works. He has been programming in the Progress environment since 1994. His work 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.

Amduus Information Works announces the availability of Memo Board

Memo Board is an open source share-ware package for Webspeed and Blue Diamond transaction servers. It provides a notice board on the web as well a site map. Purchasable add-on's include an employee address book that includes employee skills for staffing projects. Distributions were made on news://comp.databases.progress

Contact Scott Auge at 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

02-1

Amduus Information Works
4506 Carlyle Court Suite 712
Santa Clara, CA 95054
408-980-8447
scott_auge@yahoo.com

Creation of modules and products for re-sale. Internet/Intranet programming for E-Business in the marketing/manufacturing/service/law enforcement industries.

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

Providing contract programming and writing services.

01-07

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 300 subscribers and companies that receive this mailing!

Amduus Information Works assists in the publication of this document:

Amduus Information Works
4506 Carlyle Court Suite 712
Santa Clara, CA 95054

Article Submission Information:

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

Please submit your article in Microsoft Word format or as 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.

Upcoming Articles:

July 2001
Sending and receiving email with your progress application.
How many Webspeed agents?