The Progress Electronic Magazine

In this issue:

Did you sign up to receive this E-Zine? Send email to sauge@amduus.com to subscribe or fill out the forms at http://www.amduus.com/online/dev/ezine/EZineHome.html ! It's free! (Though donations are certainly welcome – whatever you feel is fair!)

Publisher's Statement:

This month's E-Zine has an exciting article for Graphing tools using the 4GL and perl. This is personally exciting, because while the E-Zine has over 400 subscribers on the list, it is the first article to not be written by me! (Hint! Hint! OK, there was one other, but it was more of an ad than an article.) The code is written and runs on Linux/UNIX machines, but should be transferable to Windows machines once one has Perl for windows and that sort of thing. (I specialize in the UNIX arena, and tend to stay away from Windows due to a "bad experience.")

On the peg list (www.peg.com) there were some questions about how to scope a Progress software project. Scoping is different from programmer productivity because it attempts to determine the number of programmers needed for a project as well as how long it will take. There are whole books written about this, but I will try to put something together that tackles the issues directly related to Progress apps.

Also something I need to say since I am no longer employed! I have written some routines that effect security in web applications. This means people who need turn key security in their Webspeed application need only become familiar with this code and user guide. In short, it will generate .htpasswd and .htaccess entries for Apache web servers on the fly for static files. As well it has routines to determine who is using the site, and if they have access to that dynamically generated web page. The code is oriented towards E4GL but could be re-worked to HTML-Mapping. It works on Webspeed and Blue Diamond, and any Webspeed alternatives that support E4GL source code. Needless to say, this code is not free. Write for more information.

To your success,

Scott Auge
Founder, Amduus Information Works, Inc.
sauge@amduus.com

Coding Article: Pie Charts

Written by Petr Vavrinec vap@iol.cz

Introduction

Recently I needed a quick way of generating some pie charts for my personal home use. As I firmly believe that all report generation belongs to the server and also happen to have a nice Linux box running PROGRESS (actually it's SuSE 6.3 and 9.1A, respectively), I succeeded in creating a simple Linux-based solution. In this article I would like to share my experience with honorable `E-Zine' readers. I created a nice little example with the “Customer” table from the “sports” database.

Why Perl?

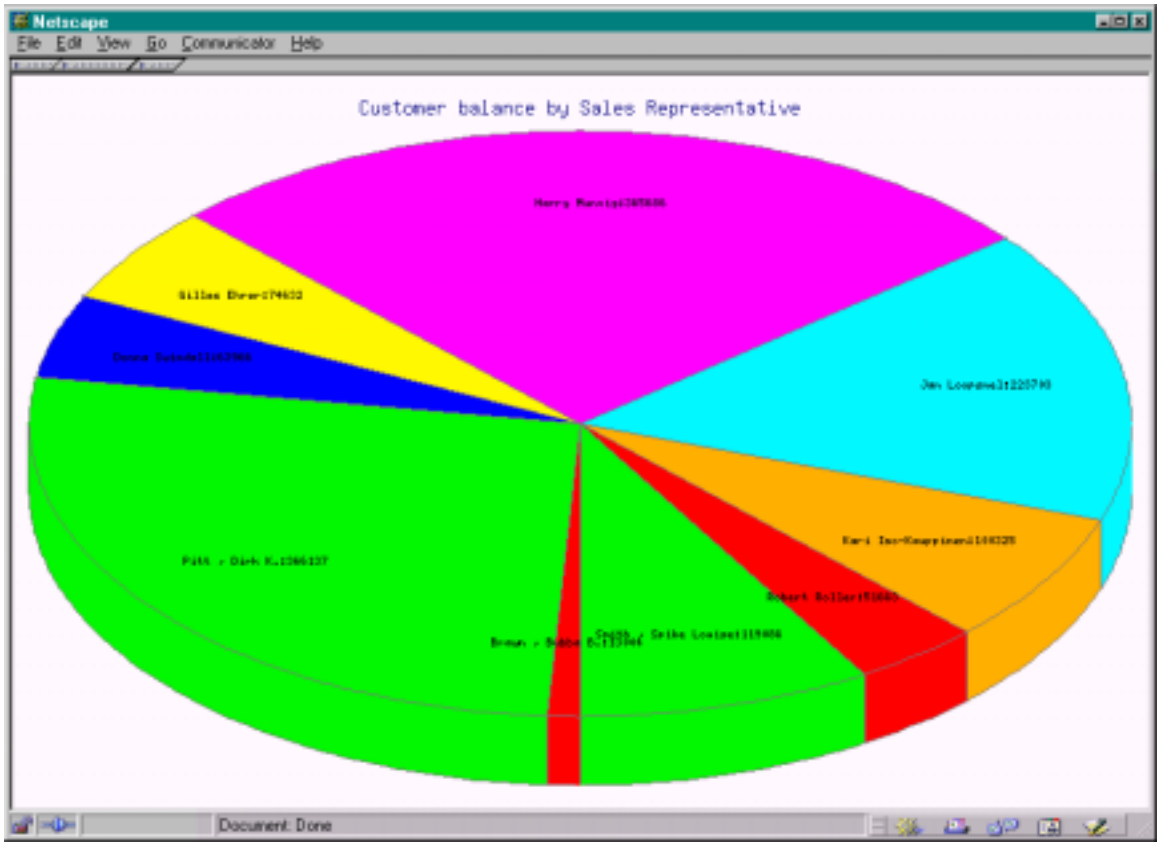
My solution is based on perl, as it seems to me a robust environment, ideal for parsing data being transferred from PROGRESS 4gl to the chart generating engine. Perl offers a generous quantity of modules suitable for almost every possible task, including, of course, pie chart generation.

Installation

First I looked around CPAN (see RESOURCES) to find out what this Perl Archive offers. I found the “GD::Graph” module, that looked very promising, so I downloaded the sources GDGraph-1.33.tar.gz, unpacked it and a quick glance to the README showed that I needed yet the “GD” and “GDTextUtils” modules. So I downloaded yet two more tarballs, “GD-1.33.tar.gz” and “GDTextUtil-0.80.tar.gz”. Obviously, I started with installing the “GD” module. It is just a perl interface to the “GD” graphics library, that should already be installed on every decent Linux installation. Or at least in every Linux installation that has an ambition to act as a graphical-output-generating box. The remaining two modules make the graph generation using “GD” library easy even for us lazy programmers.

Example with “sports” Database

If you have everything installed, you can have a look at my little example of a pie chart generating device. First, look at perl script “pie.pl”. Near the top you can see a simple parser that processes some parameters and data being sent to the script via standard input. The parameters are stored into special variables and data into an array. When everything is processed, two data arrays are joined to an array of arrays. Next we can move on to graph generation. The perl module makes it real easy with everything one needs to create a new graph object with some options (ok, properties, for you OO guys...).



The corresponding PROGRESS source, “pie.p” is simple too. At the top you'll see the parameter settings. Here you can set the basic stuff that controls the final appearance of your pie chart. The script `pie.pl' expects our data on standard input, so we will pipe our output to stdin of `pie.pl' (output through). First we will send the parameters, in a form that expects our parser, i.e. in a name=value pairs. When we have finished with parameters, we can start sending the data. As you can maybe remarked from the title, I created an example of Customer balance by Sales Representative. That means that we will loop through the Customer records, sorted by Sales Repesentative and calculate the balance for every Salesrep. Our parser expects the data in a name<TAB>value pairs.

Now you can press <F1> and the engine will create a `png' image with your graph in your working directory. Or the impatient can have a look at image `customer_balance.png' in this article.

If you need a different image format than default `png' and have ImageMagick suite of image processing tools installed, you can convert the image to almost anything you like; just use the &ExportFormat parameter.

Conclusion

This article should show you one of the many possible ways of how to create a server-side graph generating engine. I'm already satisfied with it, as I already said that I needed it only for my personal home use. For use in a professional application, one should install Type1 fonts. Also don't forget to check the documentation and samples that come with the “GD::Graph” sources. You will see that the module is able to produce also other types of graphs and that the settable options used in my little example are only a small subset. Happy Linuxing!

Resources

CPAN Web Site: http://www.cpan.org
ImageMagick Web Site: http://www.imagemagick.org
gd library Web Site: http://www.boutell.com/gd/

Progress Code as an example to using pie.pl chart generator

```
/* file: pie.p
 * Written by Petr Vavrinec
 * vap@data-norms.cz
 *
 * This file is released into the public domain.
 *
 * Example how to use the `pie.pl' pie chart generator.
 *
 */

  /*************************************************************************
   * settings                                                              *
   *************************************************************************/
  &scop OutputFileName customer_balance
  &scop Title          Customer balance by Sales Representative

  &scop ChartHeight    500
  &scop ChartWidth     800
  &scop PieHeight       50

  /* ExportFormat you can use for direct conversion into different graphical
   * format, like e.g. gif,jpeg. Default is png.
   * Use only if you have ImageMagick (especially its `convert' tool) */
  &scop ExportFormat


  /*************************************************************************
```

```
   * definitions                                                             *
   ****************************************************************************/
  define variable dBalance as decimal no-undo.

  /*************************************************************************
   * open the pipe                                                         *
   *************************************************************************/
  output through "pie.pl":u.

  /*************************************************************************
   * send the parameters                                                   *
   *************************************************************************/
  put unformatted "Title=":u + "{&Title}" skip
                  "File={&OutputFileName}":u skip
                  "Height={&ChartHeight}":u skip
                  "Width={&ChartWidth}":u skip
                  "PieHeight={&PieHeight}":u skip
&if "{&ExportFormat}" ne "" &then
                  "ExportFormat={&ExportFormat}":u skip
&else
  &scop ExportFormat png
&endif  /* &ExportFormat */
                  .

  /*************************************************************************
   * main data loop                                                        *
   *************************************************************************/
  for each Customer no-lock break by Sales-Rep:

    assign dBalance = dBalance + Customer.balance.

    if last-of(Sales-Rep) then do:
      find SalesRep where
           SalesRep.Sales-Rep = Customer.Sales-Rep
           no-lock no-error.

      /*********************************************************************
       * send the data                                                     *
       *********************************************************************/
      put unformatted (if available(SalesRep) then SalesRep.Rep-Name
                                              else Customer.Sales-Rep)
                       ":":u dBalance
                      "~t":u dBalance skip.

      assign dBalance = 0.
    end.  /* last-of Sales-Rep */
  end.  /* for each Customer... */
```

```
  /************************************************************************
   * close the pipe                                                       *
   ************************************************************************/
  output close.

  return.
```

The pie.pl Program

```
#!/usr/bin/perl -w

### load the module
use GD::Graph::pie;

### definitions
my @x;

my @y;
my @data;
my $title = "Pie chart demo";
my $file = "pie";
my $height = 500;
my $widht = 500;
my $pie_height = 30;
my $export_format = "png";

### main data loop: parameters are preceded by its name;
### data (x,y values) are separated by <TAB>;
### every item is on its own line
while (<STDIN>) {
        chomp;
        if (/^[tT]itle=(.*)/) {
                $title = "$1";
        } elsif (/^[fF]ile=(.*)/) {
                $file = "$1";
        } elsif (/^[hH]eight=(.*)/) {
                $height = $1;
        } elsif (/^[wW]idth=(.*)/) {
                $width = $1;
        } elsif (/^[pP]ie[hH]eight=(.*)/) {
                $pie_height = $1;
        } elsif (/^[eE]xport[fF]ormat=(.*)/) {
                $export_format = "$1";
        } elsif (/^(.*)\t(.*)/) {
```

```
                push @x, "$1";
                push @y, $2;
        }
}

### data array
@data = ( [@x], [@y] );

### graph generation
my $graph = new GD::Graph::pie($width, $height);

$graph->set(
        title => "$title",
        pie_height => $pie_height,
        transparent => 0
);


$graph->plot(\@data);

open(OUT, ">$file.png") or die "Cannot open for write: $!";
binmode OUT;
print OUT $graph->gd->png();
close OUT;

### optional output conversion (using ImageMagick's `convert' tool)
if ("$export_format" !~ /png/) {
        system("convert $file.png $file.$export_format");
}
```

About the author: Petr Vavrinec lives in a tiny Czech village, counting nine souls (that includes four members of his family). In his spare time he likes playing with his two dogs, one cat and two kids (not necessarily in that order of importance). He can be reached at vap@data-norms.cz.

Management Article: Scoping Progress Based Software

Written by Scott Auge sauge@amduus.com

On the Progress Email Group main list, there were some questions regarding the scoping of applications. I have been in the business for some time and thought this would be a good way to drop some tips.

There are two main ways to try and look at software adds or changes. One is with an analytical approach that tries to put some facts together to come to a conclusion. The other is a more empirical way that I believe, is more accurate to how long it will take things to get done.

One thing I want to make perfectly clear though, is that a programmer/analysts productivity is NOT based on number of lines of code written or function points completed. A programmer analyst's productivity should also include the learning and modification of the process that the software will operate under. The questions and answers needed of users or designers to create the proper software. The testing of the code. The documentation of the code. Unfortunately in many organizations, the creation of a user manual or user instructions of some kind.

Also, while we try to make programming a cut and dry assembly line fashion process, it simply is not. Programming is very much an intellectual pursuit, and some people are more intellectual than others, and if it is not that, then it is the problems facing the person on that day. Unlike assembly line productivity statistics, programmers are rarely solving the same problem over and over, or writing the same code over and over. Hence, one can never get a good statistic on what is happening "on the floor."

Looking at it analytically

Are the programmers familiar with the software?

One will need time to grow familiar with an application's source code. This timing should be taken into account. If the programmer is accustomed to working in the source code, then a smaller amount of time can be expected than from someone who .

Are the programmers familiar with the problem space?

Programming includes not only understanding the syntax and behaviors of a language, but how to use them to solve a problem. One programmer might be well experienced in the workings of an inventory system, but placed onto a project for handling neural network programming, they may need some time to catch up to what is going on. A lot of the times, it is not even that diverse, such as moving from programming logistical algorithms to accounting routines. Programming under account would require the programmer has an idea of GAAP and other such standards organizations in order to meet those requirements.

Is the code modular or spaghetti?

Everyone knows that modular code is far easier to maintain than the spaghetti variety. Oft times one "just wants to get it out" and "figuring it out while writing it" becomes the MO. The bad thing with spaghetti code is that it takes longer to learn what is going on, as well changes in one place can have unexpected changes in other places simply because of logic. Expect more time and energy expended working in spaghetti code.

Is the code documented?

Oft times it is much easier to read in a book what a certain routine does, versus reading the code. For example, there are many API's and functions written in other languages that perform black box functions. Having documentation that reads "if you want to create a part, send these parameters to this function" is far faster to work with than searching the code base for routines that look like they might do the trick.

How complex is the change?

Changes can have a measure of complexity in them also. For example, the simple changing of a label on the screen is less complex than creating a field in the database for screen input data which in turn is less complex than an algorithm for doling out inventory to work stations on a machine shop floor.

By tracking the types of changes that are done, one can begin a list of items with a complexity measure to them. They should be compared to each other, and can be high in resolution. As an example, changing a field in an accounting module might be more complex than changing a field in a sales leads tracking module within the same piece of software.

Do you know what you need to do yet?

Any kind of estimating is going to require some thinking time to figure out what needs to be figured out. By looking at previous work, one can get an idea of what needs to be done where to figure out the ins and outs of what is to be done.

What other functions are expected to be influenced?

Often, a change in one place, such as adding a field to a screen and database table, requires some additional work else where, such as adding that bit of information in a report. It might be even more complex if the user wishes to be able to sort on that new data in a set of reports.

If data is passed between to applications, which is quite often the situation in many companies with integrations, then an examination of how the integration points might be effected should be included.

Looking at it statistically

Determining some measuring points:

- *Time to add a field to a feature, module or package*
- *Time to add a field to the database*
- *Time to add a field to a screen*
- *Time to add a field to a report*
- *Time to create a certain piece of code*
- *Time to update a certain piece of code*
- *Time to document a piece of code*
- *Time to change User Manual documentations*
- *Time to write Test Plan for changes*

All of these measurements are going to be different for different parts of a package. So it is best to come up with a way to categorize your measurements so that they will be accurate. To think of it in terms of working on a car, replacing a tire on the suspension system is far different from replacing a motor in the electric windows. So while it is a "part replacement" action, the two differing aspects of the same car would be expected to have different timings.

Such a topic is hard to write out in a short article. Here are some books I have read that have really great ideas and techniques (in order that I believe them to be, best on top followed by lessor):

E.M. Bennatan, "On Time, Within Budget" ISBN 0-89435-408-6
Alan M. Davis, "201 Principles of Software Development" ISBN 0-07-015840-1
T. Capers Jones, "Estimating Software Costs" ISBN 0-07-913094-1
Steve McConnell, "Rapid Development" ISBN 1-55615-900-5

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

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
http://www.amduus.com
scott_auge@yahoo.com sauge@amduus.com

Creation of modules and products for re-sale as well customized Internet/Intranet programming for E-Business in the marketing/manufacturing/service and 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?

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

Product Announcements:

Survey Software

Amduus Information Works, Inc. is creating survey software. This software can be used on a web site to query a population of people about their views and needs. The population could be internal to a company or external to yield a better understanding of the marketplace. Documentation for the application will be available at http://www.amduus.com for free download.

The software ships with source code for better adaptability to your company's application landscape and needs. The software was developed and designed on Linux with Progress Version 9 and Webspeed 3.1. The software can operate on AIX, Linux, Solaris, HP-UX and UNIXWare. The software will also run on Blue Diamond available for free from Amduus Information Works, Inc.

Customers and resellers are welcome to contact Scott Auge at sauge@amduus.com for more information. Street price is $1,000 per machine without Progress licenses.

The software can be rented out at $100.00 per survey per week of taking results.

Mail List Software

The publishing of this E-Zine has created the need to create some software that can handle the creation of mailing lists, web pages to subscribe and unsubscribe as well as a command line to distribute this file. This software is available for $50.00 per site. The User Guide is available at http://www.amduus.com/Manuals/mls.doc . Purchase before October 15th and get a free update that will have a subscription verification feature and a list server.

Security SDK

The SDK works on E4GL source applications on Webspeed, Blue Diamond, or other Webspeed alternatives that use E4GL coding. Just one more thing brewing up at Amduus Information Works!

Publishing Information:

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

Currently there are over 300 subscribers and companies that receive this mailing! This mailing is not sent unsolicited, so it is not SPAM.

Amduus Information Works, Inc. assists in the publication of this document:

Amduus Information Works, Inc.
4506 Carlyle Court Suite 712

Santa Clara, CA 95054

Article Submission Information:

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.