An Amduus™ Information Works, Inc. Publication

Table of Contents

Publisher's Introduction

Yes, I asked on the PEG about what people would like to see in the next issue – I was pretty excited about the PHP integration goodies myself. BUT the guys who put together the goods say they have even better goodies than what is on their websites. So I will give them time to polish up the code and hopefully they will pass it along to me.

It appears that on the peg there are a lot of questions about background processors. I know from experience that if you are writing a Webspeed program – you are sooner or later going to need a background processor. A background processor allows you to run the more complex and time consuming operations someplace else than the Webspeed agent or the user's client. When there is timing involved, one doesn't want to tie up a Webspeed agent with work when you can pass it back to another process (even perhaps even on another computer!) nor deal with the browser time-out issues.

Examples of background processor tasks include connecting to other systems in an asynchronous manner such as credit card processors (who can be notoriously slow in deciding to OK a transaction.) Often in Webspeed applications one can use it to generate reports and then mail the report to the requesting user.

One can even use a background processor as a poor man's AppServer. This is especially useful for very large systems where you need to share the pain across multiple machines.

Have fun!

Scott Auge
sauge at amduus dot com

Zeno Background Processor

By Scott Auge
sauge @amduus.com

Introduction

There will come a time when your application grows beyond a simple GUI interface to a database. Just for the sake of time and for handling great deals of data processing you will want to somehow offload the work to another set of machines or at least another process.

For example, in a manufacturing system there is really no reason for a user's client to attempt to perform an inventory allocation to a BOM. This is something that usually isn't interactive, takes a long time to do, and can be requested of another machine to deal with it. Thus the the user can continue to perform their work without waiting for the hour glass to go away.

Another example is a web site that needs to generate a report. One always wants to keep the agents available to take in and display quick information. A common method to achieve this is to use a web page to get the parameters of a report and then offload the report to some background process. The background process will actually generate it and mail it out to the user or make it available in their “report” screen via a record creation in the database. If your web system needs to wait on a service someplace (think credit card validation), it is often nice to have a background processor that will be able to tell the browser what is going on.

If this sounds like something you need, then let me make you aware of the open source GPL licensed Zeno Background Processor. With the GPL you would be legally obligated with Zeno as you would with Lunix. Redistribution of the work or any derivations requires distribution of the source code. (A BSD license for Zeno is available for

purchase.)

You will be very surprised what you can do with this tool.

With some inventive programming, one can use Zeno to become a business event processor. One job will realize it needs to fire off other jobs and so in a way you have multiple modules all keeping each other up to date with what is happening out on the system. An example might be a sale of equipment, which has a BOM, which must have inventory allocated to it, which means purchase orders and factory orders might need to be made... all these things are business events and might be best handled in the background.

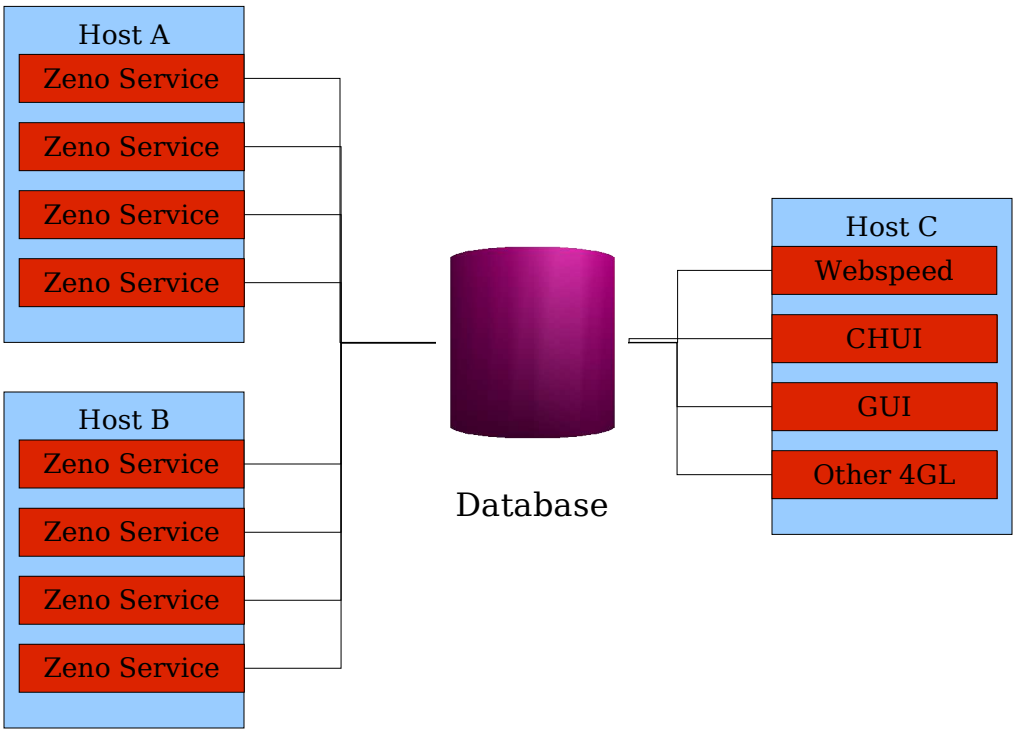
You can set up different "services." You can have multiple instances of the same service. The service instance(s) can be on the same or across multiple machines. By having multiple instances, you can perform load balancing – a service isn't put on-hold until the request is done – Zeno just grabs another process running the service and sends it there.

By using multiple machines – you can take one machine down and have the operations still continue across the remaining "up" machines. You can transparently add a machine. This allows for a great deal of flexibility in hardware, scalability, and makes it easier for 24 x 7 x 365 up time.

Zeno doesn't care where it gets it's jobs from – they can be from GUI, Webspeed, or CHUI applications. All these applications use a single procedure file called `jobclient.p` to perform interactions with Zeno via an easy to use and understandable set of APIs.

Architecture

Below is a pretty robust system composed of two machines containing the Zeno Background Processor and a third handling Webspeed and/or CHUI and/or GUI and/or Other 4GL background services. Host C can actually be a numbered set of clients to the database, it need not be one machine.



You can run the Zeno background processor on the same machine as your other progress code – it is simply a collection of "batch mode" processes connected to a database.

Setting up Zeno

Setting up Zeno is pretty straight forward. You will need to insert some new tables into the system, configure the scripts given to match the connection parameters to your

database, and then insert some records into the JobPrg table via an API in jobclient.p. These instructions are available in detail in the forthcoming Zeno documentation.

Administration of Zeno services

The administration scripts for Zeno are UNIX oriented. If there are any Windows programmers out there who want to port some MS tools to start, stop, and clean like the scripts do, please do and send them to me at sauge @ amduus.com. Feel free to make an add on program for Zeno for monitoring and searching Jobs records. In the future there will be a webspeed program that performs these actions.

Administration of Zeno is done via the `mngprc.bash` script. This script has multiple switches and parameters to allow you to start and stop a single process of the service. Once you have set it up with the proper connection parameters via the `set.bash` script, Zeno should be good to go.

Summary of Zeno mngprc.bash Switches And Arguments

When you type `mngprc.bash` without any arguments, it will provide this help message:


```
./mngprc.bash
-start          Start the processor (requires -prcname)
-stop           Stop a processor (requires -hostname -hostpid
                or -hostname -prcname)
```

```
-clean          Clean dead records (requires -tosubmitdate or
                -toprcstopdate)
-hostname       Specify the host the command should appeal to
-hostpid        Specify the PID of the process to send the command to
-prcname        Name the service the processor provides
-tosubmitdate   Used for -clean to delete records submitted up
                to and including this date
-toprcstopdate Used for -clean to delete records cleanly processed
                up to and including this date.
-pause          Amount of time to pause between job runs (throttle)
```

How to start a Zeno service

Starting a Zeno service instance is as easy as typing in:

```
mngprc.bash -start -prcname YourServiceName &
```

Each machine in the Zeno system will need the service(s) started in this manner ON THAT MACHINE. You can start up multiple instances of the same service. I usually have a `startzeno.bash` script on each of the machines that need to have services running on it.

How to stop a Zeno service

There are two ways to shut down a Zeno service. You can shut down an entire service and all of it's instances on a given machine with the command:

```
mngprc.bash -stop -prcname YourServiceName -hostname YourHost &
```

Since Zeno uses database records to communicate, you can actually shutdown the services on any host from any host.

Issuing the command

```
mngprc.bash -stop -hostpid ThePID -hostname YourHost &
```

allows you shut down a specific process in the Zeno system. The same rule applies – you can stop a Zeno process on any machine from any machine.

You must specify a hostname to shutdown the service on. There is no global shutdown available ... yet (programming challenge anyone?)

Performing Database Clean Up

If you start and stop many sessions of the Zeno processes, the database will fill up with some "cruft." Often it will clean up after it's self quite nicely for the Zeno administration records, however it is up to the user to purge Zeno Job request records.

You can purge Job request records up to and including the submission date OR up to and including the processed date. Using the submission date will eliminate all records up to that date. Using the processed date means those records that have not been processed (or have an error) will remain in the system.

The command

```
mngprc.bash -clean -tosubmitdate 12/12/2005 &
```

will eliminate all records submitted up to 12/12/2005. Using a date in the future will allow you to delete all the records. The date matches your date format for your Progress session.

The command

```
mngprc.bash -clean -toprcstopdate 12/12/2005 &
```

will eliminate all processed records up to the given date.

Cleaning up depends on how often your Zeno system is used and your tolerance for the size of the Jobs table.

Programming a Service for Zeno

Zeno comes with no "services." Lets define a service. A service is some unit of work that is useful to your application. Since applications are varied – it is really up to you to develop a service.

Examples of a service might be "Process.EndOfMonth" or "Process.Underwrite" or "Report.JobCost" or "Report.MonthlySales" or "Integration.CreditCardAuth" or "Integration.OpenLDAP." Some services, like OpenLDAP can only be used on UNIX machines – but you have Windows clients! Zeno can help "reach across the expanse."

Zeno is really a bit of a traffic control cop that can aid processes reaching out to other processes that can perform this kind of work on their behalf.

The APIs

Once you have written a service (more about that later) – which in general is really a 4GL program – you need to inform Zeno of it. It is easy enough with the DefineService() procedure.

```
DefineService(INPUT cServiceName, INPUT c4GLPrg)
```

You name the service a unique name and then tell it the 4GL file, perhaps ZenoGetOpenLDAP.p, to execute when the service is called upon. The service will do a RUN VALUE() on this file name, so if there are subdirectories or the like you will need to deal with that in the call or the PROPATH just like other 4GL calls.

If you call the API with a different 4GL program, the original will be removed and replaced by the new 4GL program. This can be helpful if you are putting new pieces in and they don't work – just flip the service back to the old 4GL program till the new stuff is working.

Example:

```
/*************************************************************************/
/* Define the service definition via APIs.                               */
/*************************************************************************/

RUN DefineService IN hjobclient (INPUT "tstsvc1", INPUT "tstsvc1.p").
RUN DefineService IN hjobclient (INPUT "tstsvc2", INPUT "tstsvc2.p").
```

Once a service is defined, it is defined until it is Undefined. The service outlasts the session that creates the service. (It is actually a record in the database.)

```
UndefineService(INPUT cServiceName)
```

To remove a service, simply call UndefineService() with the name of the service. The records will be removed from the JobPrg database.

Example

```
RUN RemoveService IN hjobclient ("tstsvc1").
RUN RemoveService IN hjobclient ("tstsvc2").
```

Creating the Service Code

Creating the service code really depends on what you need the service to do. Note that

the particular instance in Zeno handling the call will stop processing until the service is completed. So if the service takes a while, you may want to start up multiple instances of the service to handle multi-user systems.

All service code must have one character type input parameter and one character type output parameter.

The input parameter will contain the information sent via the SubmitJob() procedure. The output parameter will contain the information your service wishes to communicate back to the GetJobResults() procedure. See the service template file found in the source code for Zeno.

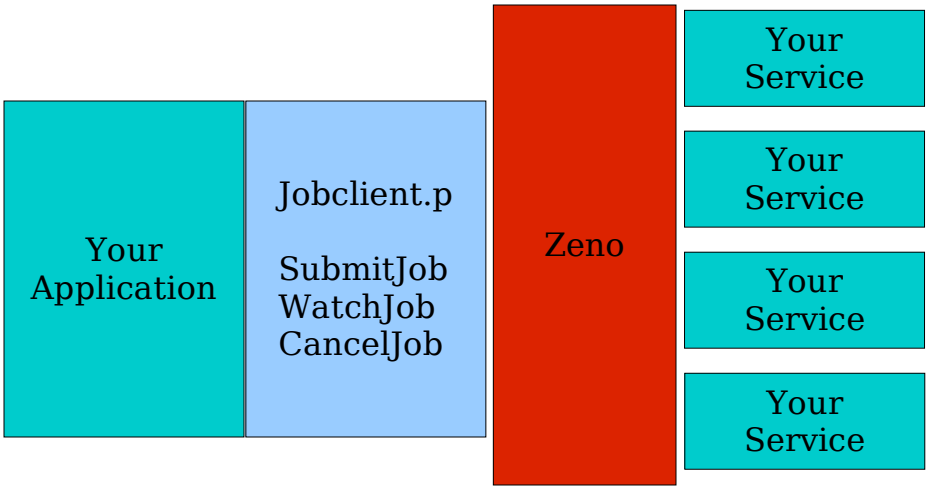
It is suggested you send the values you need in a NVP organization. Something like:

```
FirstName=Scott,LastName=Auge,OldPassword=blah,NewPassword=blah1
```

Then you would use the ENTRY() function or a set of your own parameter parsing functions to chop up the pieces in the manner you need them. Such a beastie is in the works for the next version of Zeno should it be desired.

Using the client code to communicate with the Zeno processor

There is a file called jobclient.p which provides a set of APIs to the Zeno system. It is through these APIs that you would interact with Zeno.



Do not interact with Zeno outside the APIs or you risk setting yourself up for problems on updates. Who knows what could be added to the system and what that addition means to present internal workings. One thing you can be sure of, is that we take all measures to try and keep the APIs functional and the same.

The APIs

```
SubmitJob (INPUT cServiceName, INPUT iPriority, INPUT cDataIn, OUTPUT cJobID)

cServiceName AS CHARACTER
iPriority AS INTEGER
cData AS ChARACTER
cJobID AS CHARACTER
```

It is through SubmitJob that you ask Zeno to do some work for you. You identify which service you wish Zeno to execute, the priority of the job, what data you are sending to Zeno, and a “name” for your job will be returned to you.

```
WatchJob (INPUT cJobID, OUTPUT cState)

cJobID AS CHARACTER
cState AS CHARACTER
```

Using WatchJob and the "name" of the job provided by SubmitJob, you can see what status the job is at. Here is a description of the various states:

State	Purpose
SUBMITTED	The job is waiting to be processed.
WIP	Zeno is currently processing the job.
ERROR	Zeno encountered an error trying to run the service.
DONE	As far as Zeno knows, the service was rendered without error.
CANCELLED	Job was cancelled.

```
GetJob (INPUT cJobID, BUFFER Job)

cJobID AS CHARACTER
Job AS BUFFER FOR JOB TABLE
```

The GetJob API allows you to read the values on the Job record. This is more of a utility API and not suggested for regular use. You really should use GetJobResult() to obtain the information set by the service you programmed.

```
GetJobResult (INPUT cJobID, OUTPUT cResult)

cJobID AS CHARACTER
cResult AS CHARACTER
```

The GetJobResult API allows you to read the value set into the OUTPUT parameter of your service program.

```
CancelJob (INPUT cJobID)

cJobID AS CHARACTER
```

Finally, there is an API that allows you to cancel a job. If the job is in WIP, you cannot cancel the job. Simply send the name of the job to Cancel job and it will mark the status CANCELLED.

Here is some example code showing how to use the APIs

```
/*************************************************************************/
/* Put out some jobs to do.                                              */
/*************************************************************************/

RUN SubmitJob IN hjobclient (INPUT "tstsvc3", INPUT 1, INPUT "Job1", OUTPUT cJobID). /*
101 */
RUN SubmitJob IN hjobclient (INPUT "tstsvc4", INPUT 1, INPUT "Job1", OUTPUT cJobID). /*
100 */
RUN SubmitJob IN hjobclient (INPUT "tstsvc1", INPUT 1, INPUT "Job1", OUTPUT cJobID).
RUN SubmitJob IN hjobclient (INPUT "tstsvc2", INPUT 1, INPUT "Job1", OUTPUT cJobID).

/*************************************************************************/
/* tstsvc2.p has 30 second pause 20*2 seconds should be long enough for  */
/* the process to pick it up and provide the service.                    */
/*************************************************************************/

DO iIter = 1 TO 20:
  RUN WatchJob IN hjobclient (INPUT cJobID, OUTPUT cState).
  DISPLAY cState.
  PAUSE 2.
END. /* DO iIter = 1 TO 10 */
```

Where to find the code

http://amduus.com/OpenSrc/SrcLib/Zeno/zeno.2005120080024.zip

Go into the Zeno directory to find possibly updated builds.

Scott Auge is the founder of Amduus information Works. He has been working with Progress technologies since Version 6. He works with UNIX platforms dealing with integration and web based applications.

Donations

Do you find something useful in the E-Zine now and then? Do you think it holds value for your education in the 4GL and to learn what is out there?

It does take money to produce this electronic magazine. Sure I can cut corners by not needing paper, ink, or postage – but bandwidth still costs x amount of dollars every month.

Even with Linux server, hardware does cost money – and the server is five years old. It's due for an upgrade.

Yes, I use OpenOffice.org to edit the document on, but throw in the laptop and that is a few more dollars.

Throw in a developer's Progress license – well... we all know how much that puppy is! It sure is useful for writing code with! I would like to get on version 10.

All of the above is going to cost about $10,000.00 this year. Bet you didn't know it costs so much for a free publication!

We all know there are not that many publications for Progress based technology. You can help yourself, and others, by donating to "the cause."

Your Name: _____

Organization: _____

___ $5.00 ___ $10.00 ___$20.00

Consider this address good until June 2005:

Scott Auge
222 East Riverside #302
Austin, TX 78704
United States of America

Thanks for your support!

Scott

Publishing Information:

Scott Auge publishes this document. I can be reached at sauge@amduus.com.

Amduus Information Works, Inc. assists in the publication of this document by providing an internet connection and web site for redistribution:

Amduus Information Works, Inc.
1818 Briarwood
Flint, MI 48507
http://www.amduus.com

Other Progress Publications Available:

This document focuses on the programming of Progress applications. If you wish to read more business oriented articles about Progress, be sure to see the Profile's magazine put out by Progress software http://www.progress.com/profiles/

There are other documents/links available at http://www.peg.com .

There is a web ring of sites associated with Progress programming and consultants available at http://i.webring.com/hub?ring=prodev&id=38&hub .

White Star Software publishes a commercial document called "Progressions." It is similar to this document but with different content. More information can be found at http://wss.com/. White Star also publishes Progress Programming books!

Article Submission Information:

Please submit your article in OpenOffice[1] 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.

Send your articles to sauge@amduus.com!

Thanks!

1 OpenOffice is a freely available Office Suite for Windows, Apple, and *NIX based operating systems. You can download it at http://openoffice.org. This document is edited on OpenOffice on Apple OSX.