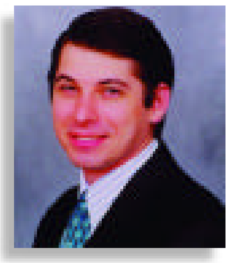
Reprinted from PowerBuilder Developer's Journal PowerBuilderJournal.com

PRODUCT REVIEW

PowerGen by E. Crane Computing

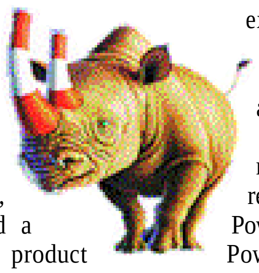


Eric Saperstein

Regeneration Is a Snap with PowerGen

Did you ever sit back late one evening and dream of the day when building PowerBuilder applications would be a simple, repeatable automated process? I have!

I have for quite some time now, as a matter of fact, because each time I had to set up another build – and then regenerate it three or four times to ensure that everything was compiled in inheritance order – I became increasingly frustrated. For PowerBuilder to last as a viable development environment,somebody had to come up with a method of making this process simple.



A PowerBuilder build should be like an automated C compile using the "make" facility. So in response to this widespread issue, E. Crane Computing delivered a solution called PowerGen. This product provides a front-end interface that completely handles a PowerBuilder build from start to finish. It's automated, it's simple and it's repeatable. For several months now I've been evaluating this product and designing a method for using it with our promotion and build processes. This article provides both a product review and an outline of the methodology I used.

What Is PowerGen?

PowerGen's primary function is to build PowerBuilder applications. PowerBuilder also builds PowerBuilder applications, but there are some significant differences in the way PowerGen builds versus how PowerBuilder builds. These differences justify the former's existence. The bottom line: PowerBuilder is missing what makes the build process simple and repeatable.

Why do we need a product that makes the process simple and repeatable? In my opinion, PowerBuilder is incomplete without it. PowerBuilder provides a structured environment for the rapid development of applications and makes good use of object-oriented technology, but turning out a product is a royal pain. The average turnaround time for our PowerBuilder applications to build is four to six hours, and from what I hear, that's a good time. This is unacceptable when compared to our C++ applications that turn out in less than 30 minutes.

Why Do You Need PowerGen? Because PowerBuilder Is Incomplete Without it!

PowerBuilder doesn't build as logically expected or handle Inheritance order properly during the regeneration or build processes. PB completes these processes by taking each PBL in an alphabetical directory listing. Consequently, it may require the regeneration of an application several times during one build to ensure that everything gets regenerated properly.

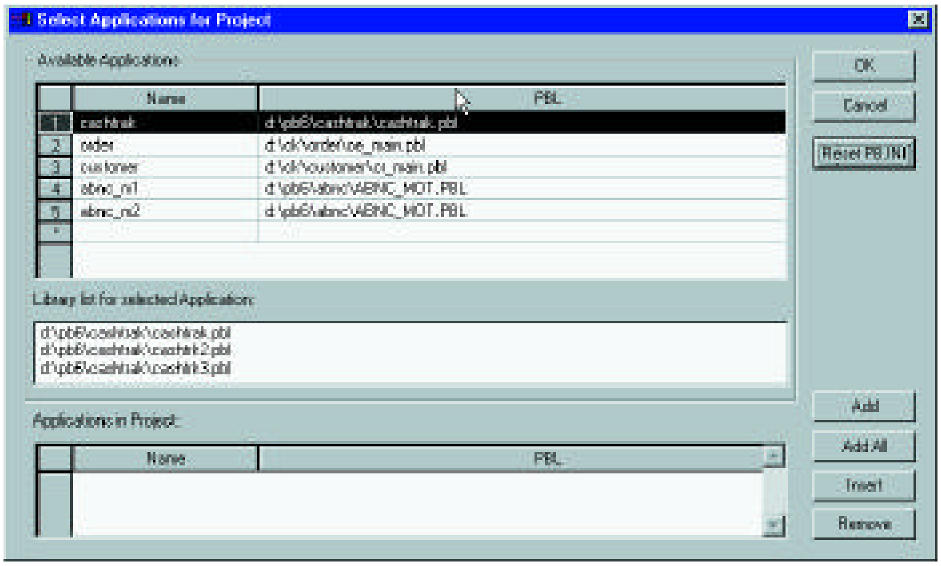
PowerBuilder provides no command line interface to allow automated or scheduled builds. Furthermore, a builder is required to use the PowerBuilder GUI for every new build. Using a GUI leaves room for human error and it forces some poor soul to be in early and/or leave late in order to complete the process.

PowerBuilder doesn't generate a log file to record each step of the regeneration and build process. Having no record of each build, including the libraries used, errors found, etc. means this information can't easily be included in the applications archive for future reference.

A PowerBuilder build requires you to regenerate every object each time a build is required; no mechanism to recompile one or a few objects and then re-create the executables is available. This wastes time when you make a small change to your source code that forces a full regeneration.

With PowerBuilder you can't safely stop a build in progress. Stopping a one-hour build process after 20 minutes because someone found a last-minute bug usually corrupts a PBL or two and requires everything to be checked out all over again.

PowerGen is a packaged product designed to provide a front-end interface to PowerBuilder so it can handle regeneration and build processes. The GUI or command line interface of this product is available in place of the PowerBuilder GUI interface when a regen or build is required. PowerGen acts as a command center; it doesn't directly access PBLs. The product sits in front of



Available Applications:

	Name	PBL
1	cashtrak	d:\pb6\cashtrak\cashtrak.pbl
2	order	d:\ok\order\oe_main.pbl
3	customer	d:\ok\customer\cs_main.pbl
4	abnc_n1	d:\pb6\abnc\ABNC_MOT.PBL
5	abnc_n2	d:\pb6\abnc\ABNC_MOT.PBL
*		

Library list for selected Application:

d:\pb6\cashtrak\cashtrak.pbl
d:\pb6\cashtrak\cashtrk2.pbl
d:\pb6\cashtrak\cashtrk3.pbl

Applications in Project:

	Name	PBL

Figure 1: The Select Application screen

PowerBuilder and forwards requests through the PowerBuilder's API directly to the PowerBuilder engines. The PB engine handles the actual work, but with PowerGen in command, it works in an organized fashion.

PowerGen is a simple application that provides a practical solution to all of the previous issues from a company dedicated to filling the gaps in the PowerBuilder development environment.

PowerGen Review

I've read hundreds of product reviews in the past few years and found that most set out to tell you about the nice features of the product and a few things that are wrong with it. That's valuable information, but it doesn't tell you what happens in the real world. I'm interested in simple, real-life implementations of the product so I can provide a practical review. Therefore,this review is based on a pilot implementation of PowerGen on a real project.

When I began my review of PowerGen, I determined two objectives. If one or both of these objectives could be achieved, this product would save hours of wasted time and aggravation.

My two objectives were to:

1. Develop a methodology using PowerGen to handle a complete build of several of our applications using a developer-maintained build script. The applications consisted of both proprietary and shared resources that update on a regular basis. This can cause changes to the list of required resources on a regular basis. For the purpose of this article, I assumed that the required files were already successfully checked out of the source repository and were ready for the build.
2. Develop a methodology to utilize PowerGen within our promotion process to complete the automatic scheduled regeneration of our application and shared source code. Some segments of our code were released as shared libraries, and these releases constantly updated to new release-based directories. This made the regeneration location dynamic.

The PowerGen Configuration

Breaking down any system requires finding out what files are important, what their purpose is and how the application uses them. It doesn't matter whether you're trying to learn how to maintain an operating system, database or word processor – the key files are the core to understanding them.

PowerGen is based on text configuration files with the ".gen" extension, or "Gen" files. The Gen file holds the key information about an application that PowerGen will follow to complete regeneration or build an executable. These Gen files can be maintained manually through a text editor, or through the PowerGen GUI. Maintenance of this file is simple; anyone with knowledge of a text editor can learn it in just a few minutes. Explanation of each line is available in the PowerGen manual.

This file allows you to pass along build information to create repeatable releases using PowerGen. A developer can create and maintain an application's Gen file, then pass it to the build team to kick off PowerGen with a command line each time a build or regen is required. A complete build or regen is now a one-step process that doesn't require human intervention. A sample Gen file can be found in Listing 1.

Objectives

I started writing this article with the intention of having to create a complex methodology to handle the build and regen process using PowerGen. Why not? Everything else I've had to do with configuration management for PowerBuilder has been complex and time consuming. As you'll see in the sections below, I was wrong. These few paragraphs describe the complete process from start to finish.

Objective I: The PowerGen Build

Incorporating PowerGen into an established *build* process is simple. All you need to do is add a line to your existing checkout script to kick off a regen using PowerGen with a preconfigured Gen file.

The following is a simplified example using the PowerBuilder 5.0 example project and PVCS Version Manager.

Hint: Using PVCS Version Manager, retrieve the latest version of example50 PBLs. This command can be modified to be based on version labels or to select only specific files. The PowerGen Gen file should be archived and retrieved using a get or similar command.

```
C:\get -Y -W X:\archives\
example50\*_v('\\c:\build\example50')
```

The PowerGen regeneration command:

```
C:\Start /W PowerGen
/option=Y:\unitref\example50\example50_build.gen
```

Objective II: The PowerGen Promotion Regen

Incorporating PowerGen into an established *promotion* process is simple. All you need to do is add a line to your existing script to kick off a regen using PowerGen with a preconfigured Gen file.

The following is a simplified example using the PowerBuilder 5.0 example project and PVCS Version Manager.

Hint: Using PVCS Version Manager, retrieve the latest version of example50 PBLs. This command can be modified to be based on version labels or to select only specific files. The PowerGen Gen file should be archived and retrieved using a get or similar command.

```
C:\get -Y -W X:\archives\
example50\*_v('\\Y:\unitref\ example50')
```

The PowerGen regeneration command:

```
C:\Start /W PowerGen ñR /option=Y:\uni-
tref\example50\example50_unit_regen.gen
```

That's it, a regen can be that simple. This means that the process can be implemented on a scheduling application! A promotion to unit reference libraries can take place, with the regen complete, every night, and a log file with error reports will be ready in the morning.

Open Issues

PowerGen is limited to building with a preconfigured Gen file to determine the location of all the libraries and to provide all required build/regen parameters. Several of our procedures involve promoting to a new unique reference directory each time a promotion takes place. This dynamic regen target makes it difficult to utilize PowerGen to perform the regeneration on a scheduled basis.

This issue also dictates that multiple Gen files may be required to support a given project. Note the difference in the PowerGen command lines in the build and regen processes listed above. This is because the unit reference libraries and build locations are not the same.

```
C:\Start /W PowerGen
/option=Y:\unitref\example50\example50_build.gen
C:\Start /W PowerGen ñR /option=Y:\uni-
tref\example50\example50_unit_regen.gen
```

If the target directory were dynamic, PowerGen would increase in versatility. E. Crane Computing is working on resolving this issue.

E. Crane Computing recently released version 3.0 of PowerGen. New to this version is a unique function called "Bootstrap Import." This function creates an entire application from exported objects, enabling a dynamic approach to source and release control. For more information about PowerGen version 3.0 and its many features, visit E. Crane's Web site at www.ecrane.com.

Conclusion

PowerGen is a simple product that provides a great service. The capability of scheduling automated builds and regeneration for PowerBuilder means no more late night dial-in sessions or four-hour daytime delays in promotions. The product is extremely easy to use – almost self-explanatory if you already understand PowerBuilder – and pays for itself in a very short time. ◆

About the Author

Eric Saperstein has over three years' experience with software configuration management, including projects involving Sybase, Oracle, PowerBuilder, Visual Basic and Visual C++. He has over eight years' experience in information systems including application development, networking and telecommunications. You can contact him at esaperstein@metlife.com.