PRODUCT REVIEW

PowerGen

by E. Crane Computing

The proven workhorse for PowerBuilder build automation

WRITTEN BY
AL SOUCY

We discovered PowerGen when we were looking for a command line utility for importing objects into PowerBuilder. At the time (PowerBuilder V5.0) we were struggling with a source control system whose SCC-API interface was having all kinds of trouble with PowerBuilder (as most source control systems did at the time). As a result we cobbled together our own check-in/check-out utility using PowerGen's command line operations.

In the course of that exercise we invited the vendor, E. Crane Computing, to come to our offices and make a PowerGen presentation. What we found out, of course, is that importing objects was just a sideline of PowerGen's major strength of automating the entire build process for PowerBuilder applications. We've been using PowerGen ever since and it's a staple in our environment.

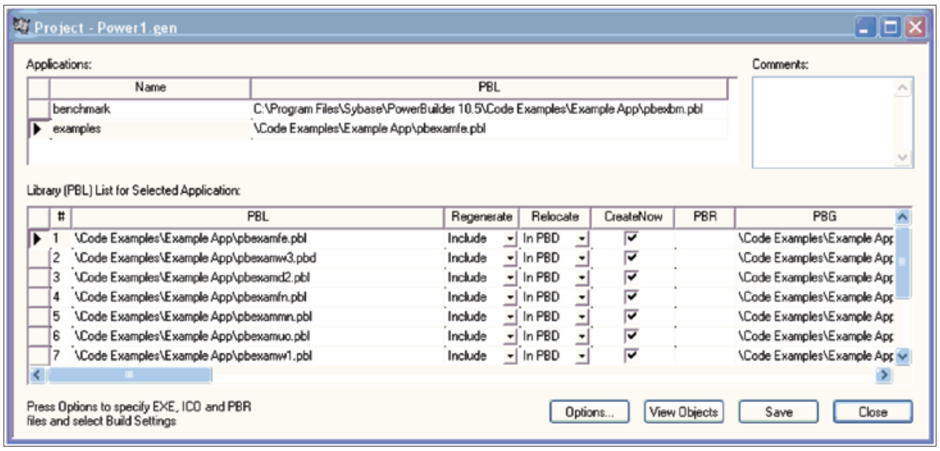
What is PowerGen?

Since its first release PowerGen has been focused exclusively on automating the build process for PowerBuilder applications. First released for PowerBuilder V4, it has allowed for a consistent build methodology through all succeeding PB versions.

It has two major functions. The first is producing PB deliverables from PBLs. To do this it offers regeneration, PBD and DLL creation, and EXE creation. To support the users' complete automation requirements it also includes, copy, import, export, and optimize functions. With a separate utility VersionEdit, delivered with PowerGen, it can also modify version resource information. (VersionEdit also works with any standard Windows executable such as EXE and DLL.)

The second major function is producing PBLs from source objects. Starting with only the PB objects in their *.sr* exported source format, PowerGen can create PBLs and repopulate them with their constituent objects. First introduced for PB 5.0, this function is called Bootstrap Import (a term since appropriated by other products). It allows "source traceable builds," an essential element of a good Software Configuration Management (SCM) process. It also lets you maintain only object-level source in your Source Control system, without resorting to versioning PBLs, which was never a good practice.

A sister function of the Bootstrap Import, the Synchronize function, will update a set of PBLs with modified source. In the Synchronize function PowerGen examines each object in the PBL and compares it with the exported source. It updates the object in the PBL only if it's changed. It also adds new objects, represented in export files, but not present in the PBL, and removes objects that exist in the PBL, but have no corresponding source. The advantage of the Synchronize function over the Bootstrap Import is that the Synchronize function is usually an order of magnitude faster.



Applications:

	Name	PBL
	benchmark	C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl
▶	examples	\Code Examples\Example App\pbexamfe.pbl

Comments:

Library (PBL) List for Selected Application:

	#	PBL	Regenerate	Relocate	CreateNow	PBR	PBG
▶	1	\Code Examples\Example App\pbexamfe.pbl	Include	In PBD	✓		\Code Examples\Example App
	2	\Code Examples\Example App\pbexamw3.pbd	Include	In PBD	✓		\Code Examples\Example App
	3	\Code Examples\Example App\pbexamd2.pbl	Include	In PBD	✓		\Code Examples\Example App
	4	\Code Examples\Example App\pbexamfn.pbl	Include	In PBD	✓		\Code Examples\Example App
	5	\Code Examples\Example App\pbexammn.pbl	Include	In PBD	✓		\Code Examples\Example App
	6	\Code Examples\Example App\pbexamuo.pbl	Include	In PBD	✓		\Code Examples\Example App
	7	\Code Examples\Example App\pbexamw1.pbl	Include	In PBD	✓		\Code Examples\Example App

Press Options to specify EXE, ICO and PBR files and select Build Settings

FIGURE 1 | A POWERGEN PROJECT

The PowerGen 'Project'

A PowerGen project consists of one or more PB applications.

New projects are created from existing PB applications defined in Targets, Workspaces, and PB.INI files. A project can also be created by choosing the application PBL and adding the libraries individually.

When presented in the PowerGen's GUI, the applications in the project are shown in the top list and the PBLs in the selected application are shown in the bottom list. Note that the paths used in the project can be specified as "relative paths." This allows for greater portability between specific build environments. PowerGen provides a lot of control for the build process. Each PBL is marked as included or excluded from the regeneration process. This has proven efficient for applications sharing PBLs, because the shared PBLs are regenerated the first time they appear and are then subsequently just referenced in the library list. This saves time in the build. Likewise PBDs or DLLs can be created selectively for each PBL, avoiding redundant operations. Each PBL can be turned into its own PBD or its objects can be included with the EXE. Another useful feature is that PowerGen will relocate the deliverables, PBDs/DLLs/EXE to a specified directory. Finally, as part of the build process PowerGen can, optionally, optimize the PBLs.

One capability unique to PowerGen is that it can create a mix of PBDs and DLLs for a single application. Although DLLs (machine code) haven't been widely used, there are cases where specific functions in an application will benefit. If those functions are confined to just one or a few PBLs then PowerGen can create DLLs for just those PBLs. This functionality provides an opportunity to optimize the delivery environment without unduly burdening the build environment.

The project is saved as a separate file with a .gen extension. The file is ASCII and is fully documented with the understanding that users may want to modify or create their projects programmatically. Although the information saved in the project file has been expanded through the various releases of PowerGen, each new PowerGen version can open any previous version's projects. See Listing 1.

Most of PowerGen's options are saved in the project file although a few are saved in the registry. The ones saved in the registry are judged to be more germane to the build environment than a specific project. For example, the option of whether the resulting PB applications exhibit "New Visual Styles" is saved in the Registry.

Command Line and GUI

All of PowerGen's functions can operate from the GUI or the command line. PowerGen 1.0 (for PB4) was delivered with a command line interface and it has been maintained in compatible form to the current release V6.5 (supporting PB5 through 10.5).

The priority of compatibility in the command line syntax means that a PowerGen script written for PowerGen 1.0 will operate without change with PowerGen 6.5.

The command line syntax generally consists of "switches" introduced by a slash (/) with parameters separated by spaces. Command lines can be simple. For example, Pwrgn105.exe /A=Examples.gen will build all of the applications defined in the project, Examples.gen. This means it will regenerate the PBLs and create PBDs and EXEs as defined in the project.

Note that the name of the PowerGen EXE corresponds to the version of PB that you're using; Pwrgn105.exe corresponds to PB10.5.

A more complicated example is:

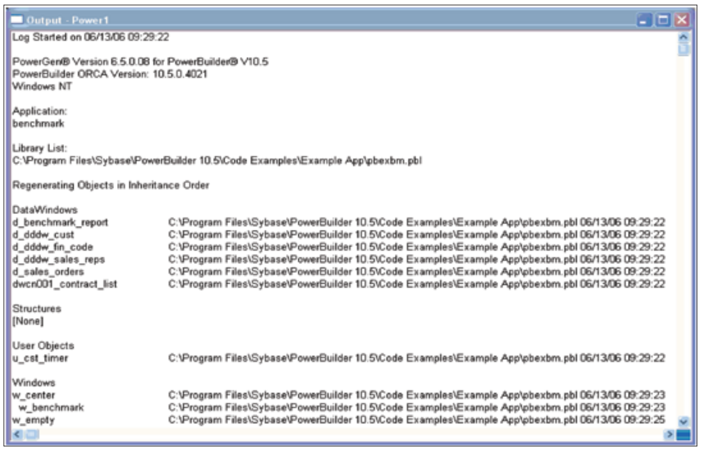
```
Pwrgn105.exe /K=Examples.gen /PBG /P=PBExamfe.pbl /RP /L=Example.log
/A=Example
```

This command line synchronizes an individual PBL, PBExamfe.pbl in the example application defined in the project file, Examples.gen. The /PBG switch indicates that the synchronize function should use the corresponding PBG file, PBExamfe.pbg, to get information about the objects that belong in the PBL. The /RP switch specifies doing a full regeneration following synchronization and the /L parameter names the file where the output log is saved.

PowerGen signals a failure from a command line operation by writing a file containing an error code. Scripts can be branch-based in the absence or existence of the file. For example:

```
If Exist Power1.err Goto ERROR
(next operation)
Goto END
:ERROR
Echo Error Building Examples
:END
```

You can see the scripts that we use at NH-DHHS on the E. Crane Web site on their support pages, http://ecrane.com/scripts.htm.



```
Output - Power1
Log Started on 06/13/06 09:29:22

PowerGen® Version 6.5.0.08 for PowerBuilder® V10.5
PowerBuilder ORCA Version: 10.5.0.4021
Windows NT

Application:
benchmark

Library List:
C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl

Regenerating Objects in Inheritance Order

DataWindows
d_benchmark_report        C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:22
d_dddw_cust               C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:22
d_dddw_fin_code           C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:22
d_dddw_sales_reps         C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:22
d_sales_orders            C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:22
dwcn001_contract_list     C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:22

Structures
[None]

User Objects
u_cst_timer               C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:22

Windows
w_center                  C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:23
  w_benchmark             C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:23
w_empty                   C:\Program Files\Sybase\PowerBuilder 10.5\Code Examples\Example App\pbexbm.pbl 06/13/06 09:29:25
```

FIGURE 2 |

The Output Log

PowerGen displays detailed output in a scrollable window, while the operation is running.

The log shows information about the environment (versions), the application being processed, and then about every detail of the build. Details can be customized to add or subtract information. The logs are sufficiently complete to meet all audit and compliance requirements.

Any error, either in configuration or in a particular operation, is written in a red font and summarized at the end of the log. This diagnostic information is especially important when configuring a build for the first time. At the same time that the GUI is presenting PowerGen's output, a log file is written with the same information. There are many options as to how and where this log is written. For example, PowerGen can generate a name for the file that corresponds to the starting time and date of the operation.

Bootstrap Import

PowerGen first introduced the Bootstrap Import for PB V5.0. Its purpose is to rebuild an application completely from just the exported object syntax (*.sr* files). So PowerGen enabled source traceability in our SCM processes long before it was possible with PB.

Later this became a critical issue for PB8.0 when PB's source control architecture changed and there were no longer "registered" PBLs to use for a build. As soon as PowerGen offered the Bootstrap Import we chose to use it rather than rely on the registered PBLs, because it wasn't possible to verify the contents of those PBLs.

PowerGen's Bootstrap Import uses a three-phase import process. In the first phase global types and basic object signatures are established. In the later phases the complete object information is added back in. This progressive import is required to avoid fatal errors caused by a fully formed object that may contain many references to objects that haven't been imported. The finesse required to import selected parts of an object in successive phases requires an automated method. Attempting to do it by importing complete objects (which we tried), even in a carefully selected order, is futile.

Because PowerGen introduced the Bootstrap Import in an early version of PB, it has had to deal with several interesting issues.

Until PB8.0 was released there was no information available about which objects belonged in which PBLs. Obviously this was essential information for rebuilding PBLs from object sources. So PowerGen introduced the Object List File (OLF) that provided the mapping between object files and PBLs. There are two forms of the OLF. In one all of the objects are enumerated along with their corresponding PBL:

```
.\Code Examples\Example App\pbexbm\benchmark.sra,.\Code Examples\Example
App\pbexbm.pbl,"Comment"
.\Code Examples\Example App\pbexbm\d_benchmark_report.srd,.\Code Examples\Ex-
ample App\pbexbm.pbl
.\Code Examples\Example App\pbexbm\d_dddw_cust.srd,.\Code Examples\Example
App\pbexbm.pbl
Etc.
```

In this form each line contains two-four comma-separated parameters. The first names the object file, the second the PBL, and the optional third and fourth the object comments and PBL comments respectively.

In the second form the object file names can include a wild card:

```
.\Code Examples\Example App\pbexbm\*.sr?,.\Code Examples\Example App\pbexbm.
pbl,"Comment"
```

In this form every file with an *.sr* form in the pbexbm subdirectory will be included in the PBL.

OLF's (and PBGs) can be created automatically by PowerGen, with many options to specify directory configurations for the source files, which form to use, whether or not to include comments, etc.

We've organized our source control repository to have a separate folder for the sets of objects for each PBL. This lets us use the OLF wild card form and not worry about whether an enumerated list is accurate. In other words, we're letting the source control structure dictate the PBL configuration. I think it's a better SCM practice than relying on a list that has to be maintained outside of the source control system. Even PB's PBG file, introduced in PB8, which provides the mapping function, is prone to be inaccurate.

Another issue arose from supporting older versions of PB. In versions of PB before V7.0 the "headers" of object source files, those lines beginning with $PBHeader and $PBComments, were removed from the source file when it was added to the source control. That meant that the comments were lost and that the object name and type had to be determined from the PBScript syntax rather than the header information. Comments are saved in the OLF files as optional parameters.

PowerGen versus ORCAScript

We've examined the use of ORCAScript for our build automation. ORCAScript is a command line-only utility that started shipping with PB 9.0 for automating build procedures.

We quickly decided to stick with PowerGen for several reasons.

We have a fully automated process, based on PowerGen, that operates flawlessly. And the process hasn't required any changes through all the versions of PB and PowerGen that we've used. As the adage goes, don't fix what isn't broken.

ORCAScript is only available for PB 9.0 and later. We need support for earlier versions.

ORCAScript doesn't provide enough detail in its output to diagnose build or configuration problems easily. Many of its error messages are difficult to interpret.

We've experienced cases where ORCAScript couldn't complete a build successfully (and PowerGen could).

ORCAScript is a command line-only interface. We find it far more productive, using PowerGen, to have a GUI that we can use to prototype our build procedures and then easily translate to a script.

Our Experience

We develop and maintain 40 PowerBuilder applications at New Hampshire's Department of Health and Human Services. The applications are used extensively in our welfare and health services delivery agencies. Example applications are for child-care licensing and managing adult and elderly care. Throughout the state the applications are used by hundreds of users.

Using PowerGen we have defined a process that includes a full Bootstrap Import and build whenever an application version is promoted to system integration testing and user acceptance testing. We use AllFusion Harvest from CA for our software configuration management tool and have versioned our PowerBuilder applications at the object level since we adopted PowerGen. When we're ready to release a version to test we "get" the labeled objects from Harvest and then run PowerGen scripts that completely automate the import and build process.

We've been using PowerGen at NH-DHHS for eight years. During that time it's saved thousands of man-hours and delivered a consistent high-quality result.

And, because of the importance given to compatibility between PowerGen versions (and an initially well-conceived design) we haven't had to change our build process through all the versions of PowerBuilder that we've used. In and of itself this has been a huge timesaver and has let us remain very confident in the quality of our build procedures.

Author Bio

Al Soucy is software configuration manager at New Hampshire's Department of Health and Human Services. In that role Al manages software configuration for dozens of PowerBuilder applications as well as applications written in Java, .NET, and COBOL (yes, COBOL).

LISTING 1

```
$PowerGenVersion=10
$ProjectName="",".\..\..\Code Examples\Example App\powergen.log",1,""
$DefaultApplication=""
$DefaultLibrary=""
$ApplicationName="examples",0,0,0,0,0,0,1
$ApplicationLibrary="\Code Examples\Example App\pbexamfe.pbl",0
$EXEPath="\Code Examples\Example App\examples.exe","",1,0
$ICOPath=""
$PBRPath=""
$PBDPath="",0,0
$SourceControl="","","","",1,0
$Library="\Code Examples\Example App\pbexamfe.pbl","",0,2,0,"\Code Examples\
Example App\pbexamfe.pbg"
$Library="\Code Examples\Example App\pbexamw3.pbd","",0,2,0,"\Code Examples\
Example App\pbexamw3.pbg"
$Library="\Code Examples\Example App\pbexamd2.pbl","",0,2,0,"\Code Examples\
Example App\pbexamd2.pbg"
$Library="\Code Examples\Example App\pbexamfn.pbl","",0,2,0,"\Code Examples\
Example App\pbexamfn.pbg"
$Library="\Code Examples\Example App\pbexammn.pbl","",0,2,0,"\Code Examples\
Example App\pbexammn.pbg"
$Library="\Code Examples\Example App\pbexamuo.pbl","",0,2,0,"\Code Examples\
Example App\pbexamuo.pbg"
$Library="\Code Examples\Example App\pbexamw1.pbl","",0,2,0,"\Code Examples\
Example App\pbexamw1.pbg"
$Library="\Code Examples\Example App\pbexamw2.pbl","",0,2,0,"\Code Examples\
Example App\pbexamw2.pbg"
$Library="\Code Examples\Example App\pbexamd1.pbl","",0,2,0,"\Code Examples\
Example App\pbexamd1.pbg"
$Library="\Code Examples\Example App\pbexamsa.pbl","",0,2,0,"\Code Examples\
Example App\pbexamsa.pbg"
```

Figure 2 - A PowerGen Project File