PowerGen White Paper

Concepts and methodologies recommended for using PowerGen to improve:

- The Build Process
- Source Traceable Builds for PowerBuilder Applications
- Synchronization of PowerBuilder Applications to Source Control

Incorporating these methodologies will ensure that your build is accurate, efficient, and repeatable, and that your application releases are reproduced from a single source.

E. Crane Computing
16 Centre Street
Concord, New Hampshire 03301
603-226-4041 http://www.ecrane.com

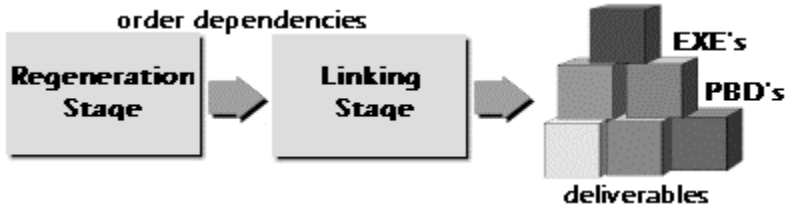
Concepts

PowerGen focuses on some of the unique problems facing PowerBuilder development shops in managing releases and software configurations. These are critical aspects of the PowerBuilder life cycle, since 70% of the cost of the overall product development effort is applied to maintenance activities, of which release management and configuration control are key elements.

The Build Process

The build process for PowerBuilder applications has similarities with those of other development environments:

- There is a regeneration stage in which PowerBuilder objects (and/or external objects) are recompiled and replaced with the compiled form of the object's code. As objects are modified during the software development process, regeneration helps make sure that all objects remain synchronized with the changes.
- There is a linking stage in which compiled PowerBuilder objects are combined into deliverable components (EXE's and PBD's).
- Order dependencies exist for both the regeneration and the linking stages. For example, it is often necessary to regenerate one object before another, or one component before another. This is especially important for PowerBuilder objects that inherit behavior from their ancestors. Ancestors must be regenerated before their descendants.



- A given application may be delivered as a set of executables and dynamic libraries, which share common code. Further, some components may support programming interfaces that enable one to communicate with the other.
- Some components of the application may be acquired and are not reproducible from the source.

PowerGen's Consistent Build Process

A consistent, rigorous build process is essential. It gives developers a more stable foundation, established by repeating the build as consistently and as often as possible. It gives release managers the ability to maintain a certifiable association of released software with the source from which it is built. And it gives QA staff the assurance that the process is viable and provides a consistent method for producing a quality software product.

PowerGen increases quality and productivity of the build through a variety of unique automation features. It automatically regenerates all objects in inheritance order, thereby eliminating both errors and time spent performing this manually.

- PowerGen's build functions may all be run from the command line. This enables you to script an entire build process and eliminate manual, error-prone checklists.
- PowerGen lets you specify a complete set of deliverable components in the build. With PowerGen you can define a PowerBuilder project that contains multiple executables. For each application you want to include, you specify the executable(s) and the PowerBuilder dynamic libraries, if any.
- PowerGen produces a complete set of deliverable components from source files.
- PowerGen handles order dependencies of objects in both the regeneration and linking stages. It automatically regenerates PowerBuilder objects in "inheritance order" rather than library order.
- PowerGen manages build resources as efficiently as possible. It lets you exclude specified libraries from the regeneration, to avoid rebuilding third party or common class libraries unnecessarily.
- PowerGen provides an "Incremental Regeneration" capability that reduces the time for build/debug cycles without compromising build integrity.

- PowerGen's results are documented. It produces a detailed log file that documents the results of the build. It lists all the objects that were regenerated, all PBD's and EXE's created, regeneration and/or linking errors, and the time and date for each object referenced.
- PowerGen makes the build repeatable. Once you've defined a PowerBuilder project with PowerGen, you can rebuild your application simply and easily in a single step.
- PowerGen eliminates the PowerBuilder-imposed limitations on importing and exporting objects, letting you define a set of PowerBuilder objects to export to one or multiple text files.

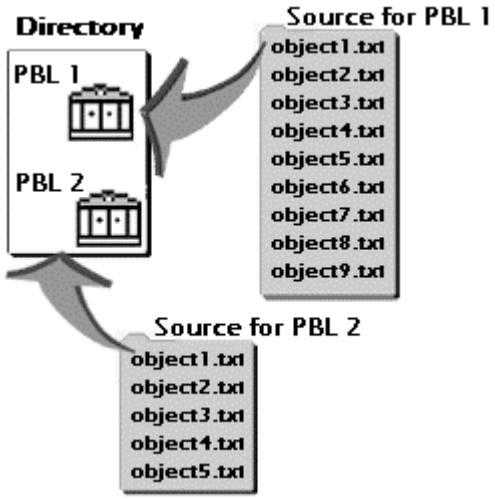
Source and Release Control

A source control system operates on individual files. When multiple developers are working on a set of controlled source, it is beneficial if the source files are small enough (in size and scope) so that two developers are not likely to want to change the same source at the same time. When source files are large, one developer must either wait for the other to complete changes or make changes independent of the other and then merge the two sets of changes.

PowerBuilder is an object-oriented rapid application development environment. PowerBuilder development is based on objects, or small blocks of code designed for a single reusable purpose. One of the challenges for source control implementers in PowerBuilder is that the source for individual objects is embedded in PowerBuilder Libraries, PBL's. PBL's are typically quite large in scope containing tens or even hundreds of individual objects. For the reasons mentioned above, this organization makes it irrational to control PowerBuilder source at the PBL level.

PowerBuilder does provide a mechanism for exporting individual objects to text files. These individual objects suit the requirements of source control, but create another problem. When the time comes to construct a new version of an application, PowerBuilder provides no mechanism to create the required PBL's from the individual objects that have been controlled in the source control system.

What is desired in practicing a good source control methodology is the ability to re-create the application strictly from sources; to start with an empty directory and a set of source objects and end up with a set of PBL's that can be built into the deliverables.



Source and Release Control Problems with PowerBuilder

By using PowerBuilder alone, it is not possible to recreate the application "from the ground up" in the manner described above. You *can* use PowerBuilder to import an individual object into a PBL, but because of the intricate dependencies that exist between PowerBuilder objects, this import ability depends on other objects already being part of the PBL. For example, if Object-A references Object-B, Object-B must exist in the library before Object-A can be imported (using PowerBuilder's import function). If Object-B also references Object-A, you're in a catch-22 situation, because you can't import either one due to their cyclical references. This is a classic "Bootstrap problem" in that it presents dependencies between operations that make it seem impossible to get started.

Source and Release Capabilities with PowerGen

PowerGen's Bootstrap Import and its Synchronization functions solve the inherent problem of source control with PowerBuilder. Through an iterative process, PowerGen first imports objects from which the cross-dependencies have been removed. In subsequent iterations, it imports more and more of the body of the objects, until they have all been completely restored in the PBL's. This is a simplified explanation, since the dependencies are subtle and varied; avoiding import errors requires an extensive analysis of these dependencies.

PowerGen's Bootstrap Import and Synchronization functions eliminate a huge gap in applying rigor to the source control process for PowerBuilder applications.

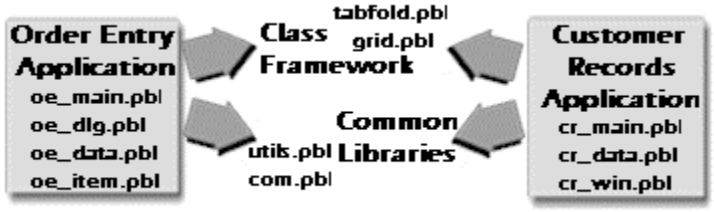
Synchronization quickly and efficiently updates PBL's to accurately reflect object sources that are controlled in a source control system. In contrast to the Bootstrap Import function, Synchronization selects only the source objects that are different from the ones in the PBL's and imports them.

The Bootstrap Import process also lets you perform other functions as well, such as general cleanup of objects in your PBL's and PBL migration.

A PowerGen Example

Here's an example of how Creative Kids, a company that sells children's toys through catalog sales, uses PowerGen to increase the quality of their build, improve source and release control, and enhance productivity at the same time.

Creative Kids has developed two PowerBuilder applications: one handling order entry and the other handling customer records. Each application has its own unique set of PowerBuilder libraries; both of these applications access a third-party Class Framework library as well as a set of in-house Common Libraries.



Before implementing PowerGen, Creative Kids went through a labor-intensive build process that required one to two man-days. Here's what their weekly build process involved:

Establish the "Build PBL's"

1. Copy PBL's from development directories
2. Check objects in PBL's against objects in source control system to ensure PBL's represent a known set of source
3. Import updated objects into the correct PBL

Build the Order Entry Application

4. Open Project for application
5. Perform a full rebuild
6. Optimize the PBL's
7. Copy EXE and PBD's to the Test area
8. Test the application

Build the Customer Records Application

(Note: the steps for this application must be different than the above in order for the common libraries to be shared between the two applications.)

9. In Library Painter Regenerate objects in Customer application PBL's
10. Create the PBD's and EXE for the Customer application
11. Optimize the PBL's
12. Copy EXE and PBD's to the Test area
13. Test the application

Test the Deliverables

14. Test all the deliverable components for both applications

Both developers and maintainers at Creative Kids became increasingly frustrated at the large amount of manual work required, the number of errors occurring during each build, and the time required for each release. They needed to do something.... and decided to implement PowerGen.

Using PowerGen, Creative Kids' build process improved significantly. Errors occurring during each build were almost eliminated. And the time required per build was reduced to that needed to schedule a nightly batch build process.

First, Creative Kids went through a step-by-step process of defining its applications and putting associated data into PowerGen. Then they interactively tested the applications to make sure the build worked as planned. Once the build was working properly, the weekly 14-step process described above became a single button click to activate PowerGen. Their build process became repeatable, error-free, and documented. By using PowerGen's Synchronization function and Bootstrap Import function, Creative Kids was confident that its builds could be traced to its source control system. And they saved labor days of time each week as well.

Step-by-Step

This section describes the step-by-step methodologies recommended for

- The Build Process
- Source Traceable Builds for PowerBuilder Applications
- Synchronization of PowerBuilder Applications to Source Control

Incorporating these methodologies will ensure that your build is accurate, efficient, and repeatable, and that your application releases are reproduced from a single source.

The Build Process: Step-by-Step

The recommended build process described here follows these steps:

- Step 1: Define your applications and deliverable components
- Step 2: Start PowerGen and create a new project
- Step 3: Specify the applications in your PowerGen project
- Step 4: Specify the executable path and support files
- Step 5: Specify regeneration information
- (Optional) Step 6: Specify any changes to code generation defaults
- Step 7: Specify PBL information
- Step 8: Specify build options and define log file preferences
- Step 9: Save the project
- Step 10: Test the build
- Step 11: Test the resulting components

Step 1: Define Your Applications and Deliverable Components

A PowerGen Project is the collection of information about your PowerBuilder applications that PowerGen needs to carry out its build tasks. This information includes, in part, the names of the applications and their library lists.

Before you start PowerGen, define the complete set of delivered applications to be included in your PowerBuilder project. You perform this step in PowerBuilder; PowerGen will read this information when you create your PowerGen project.

Executable(s)

- The executable is the main program for the application. Define at least one executable per application. You are not restricted to a single executable with PowerGen; define all you want included.

PowerBuilder Dynamic Libraries

- When PowerBuilder builds a dynamic library, it copies the compiled versions of all objects from the source library (PBL) into the dynamic library (PBD). Your application may not include any dynamic libraries, but if it does, define all those that you want included.
- The inclusion or exclusion of dynamic libraries brings a host of pros and cons that should be considered when defining your PowerBuilder project.

Few PBD's	
+	Easier installation
+	Easier packaging
+	Easier configuration control and support
-	Low degree of modularity disadvantages both development and maintenance
-	Functions within the application(s) may be duplicated

Many PBD's	
+	High degree of modularity makes development and maintenance easier
+	Functions can easily be shared between applications.
+	"Spot" updates possible
-	Difficult installation
-	Difficult packaging
-	More difficult configuration control and support

Step 1 Example

Creative Kids has two applications – Order Entry and Customer Records—and defines the following deliverable components for each of them. Notice that the dynamic libraries in both the Class Framework library and in-house Common libraries are shared between the Order Entry application and Customer Records application.

Applications	Executables	Dynamic Libraries
Order Entry	order.exe	oe_main.pbd oe_dlg.pbd oe_data.pbd oe_item.pbd utils.pbd com.pbd tabfold.pbd grid.pbd
Customer Records	customer.exe	cr_main.pbd cr_data.pbd cr_win.pbd utils.pbd com.pbd tabfold.pbd grid.pbd

Step 2: Start PowerGen and Create a New Project

The first time you start PowerGen, the program asks you to enter the full path to the PB.INI file you want to reference for this project. In PB8 you will see a list of defined Workspaces rather then being prompted for the PB.INI file. In this example we will use the PB.INI as a source of application definitions.

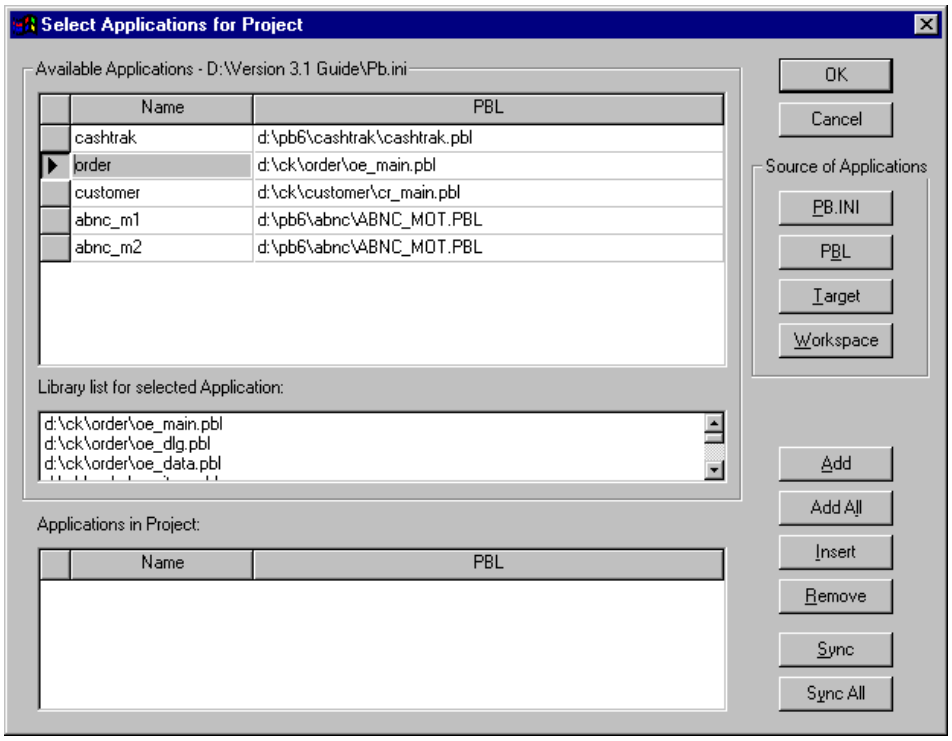
Once you specify the PowerBuilder INI file, PowerGen displays a list of all the applications it finds there, including the set of applications you identified in Step 1. Remember that this is a list of all applications in your INI file only and that PowerGen projects can be created from a combination of four different sources :

- PB.INI file
- PB Target (PB 8 only)
- PB Workspace (PB8 only)
- Application PBL

When a source is selected, PowerGen presents a list of applications available from that source. You may change sources to pick applications from different sources to be part of your PowerGen project. In Step 2 you will do that -- those applications you defined in Step 1 from the appropriate sources.

Step 2 Example

The following PowerGen screen shows all of Creative Kid's applications, including the two applications Creative Kids defined in Step 1. Note that each application is shown with its library list (the set of PBL's from which the application will be derived). The list of PBL's displayed in the Library List box depends on the application currently highlighted in the Applications box.



At this point, no applications are listed in the bottom part of the window, because no applications have been added to the project yet. That will happen in Step 3.

Step 3: Specify the Applications in Your PowerGen Project and Save the Project

Now you need to specify the applications you want included in your PowerGen project. PowerGen will build these applications in the order it sees in this project list. Once you've specified the applications, you need to save the project, giving it an appropriate file name and saving it to the directory containing the application PBL.

Add Applications

The Select Applications for Project dialog box lists all the applications in the last source you had specified.

If you need to select a different source, click the appropriate PB.INI, PBL, Target, or Workspace button and browse to locate the application source.

Regardless of the source, to add an application to your PowerGen project, simply highlight it in the Available Applications area and click the Add button. The selected application will appear in the Applications in Project area. Note that within a selected application, PowerGen will always build things in the correct order automatically. When you're finished adding applications, the application list at the bottom of the screen should match the list of applications you defined in Step 1. When the Applications in Project list is complete and in the appropriate order, click OK .

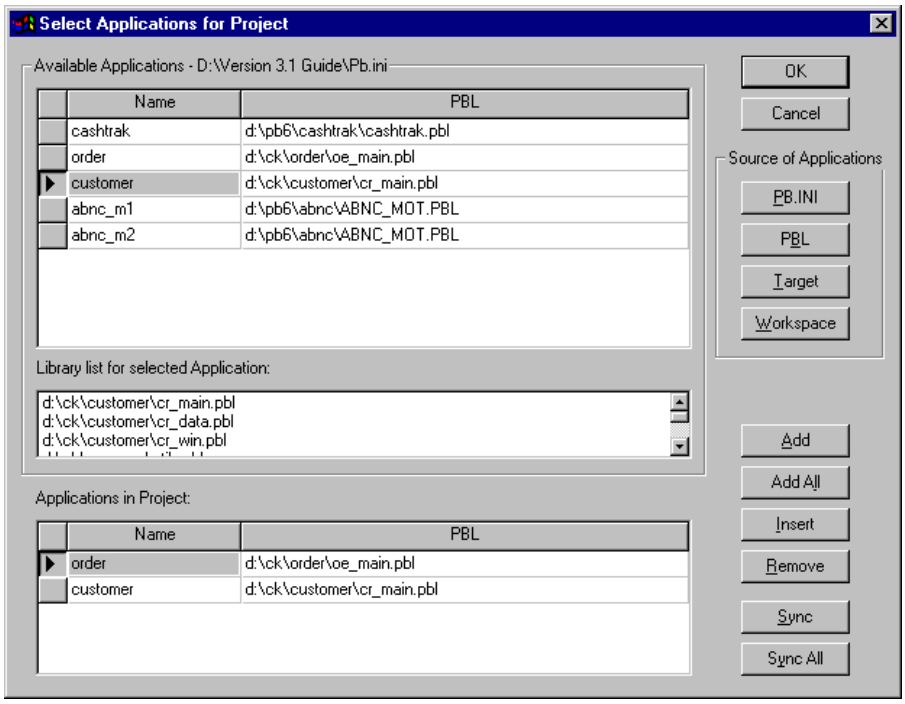
Save the Project

The main PowerGen window for the project appears. PowerGen automatically assigns a project name with a GEN extension to each new project (Power1.gen, for example). It is recommended that you specify an appropriate name for your project and then save it into the same directory as the application PBL.

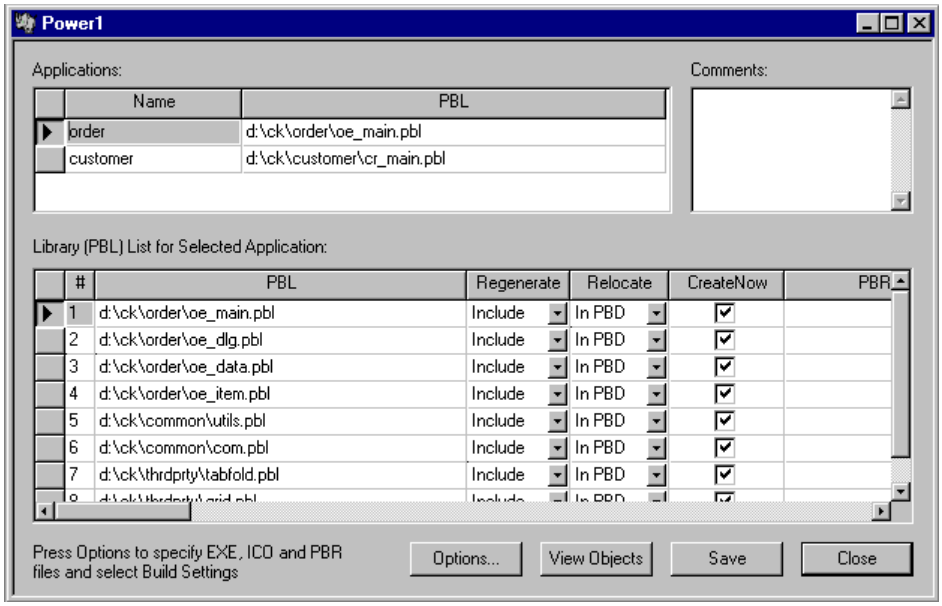
To save the project with an appropriate name, click the Save button or choose Project → Save. The Save As dialog appears, where you can give the project a name with a GEN extension and browse to locate the file in the same directory as the application PBL. Then click OK to save the project information.

Step 3 Example

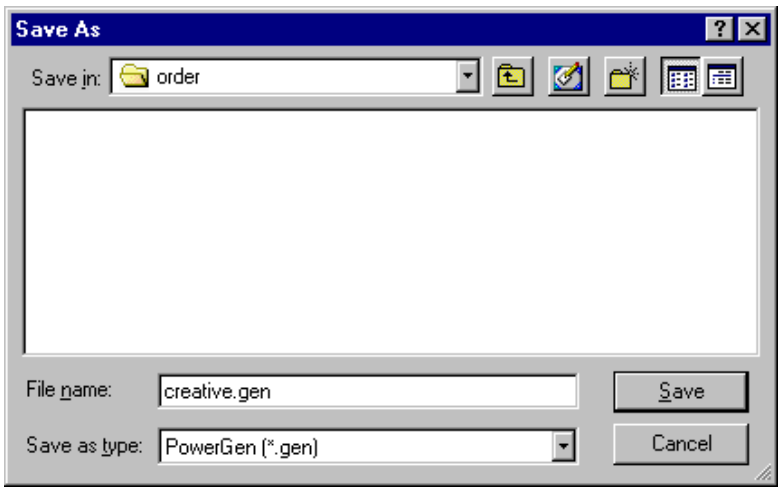
The source for both of the Creative Kids applications is in the PB.INI. Creative Kids adds the Order and Customer applications to the PowerGen project.



Once they click OK, the main PowerGen window for their project opens, displaying the two selected applications and appropriate libraries for each application.



Since Creative Kids does not want the assigned Power1 file name for their project, they click the Save button and name their project creative.gen, and save it into the same directory as the application PBL.



Step 4: Specify the Executable Path and Support Files

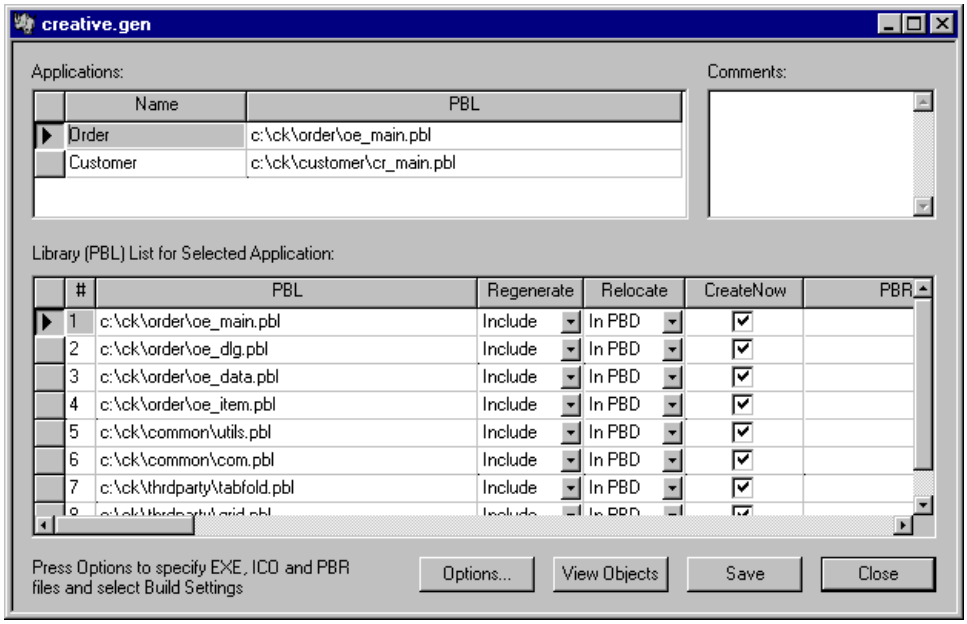
For each application you've included, specify a path to the executable file name (EXE) and, optionally, a path to an icon file name and a PowerBuilder resource (PBR) file name.

An icon file holds the picture that will be shown in Windows to indicate its presence in a group or to show that the application is open (running) but in a minimized window. *(Note that the icon must be specified here to be part of the EXE, even if it has already been assigned to the Application Object. This is a requirement of the ORCA interface.)* A PBR lists all dynamically assigned resources in an application, letting PowerBuilder include them in the dynamic library when it builds it.

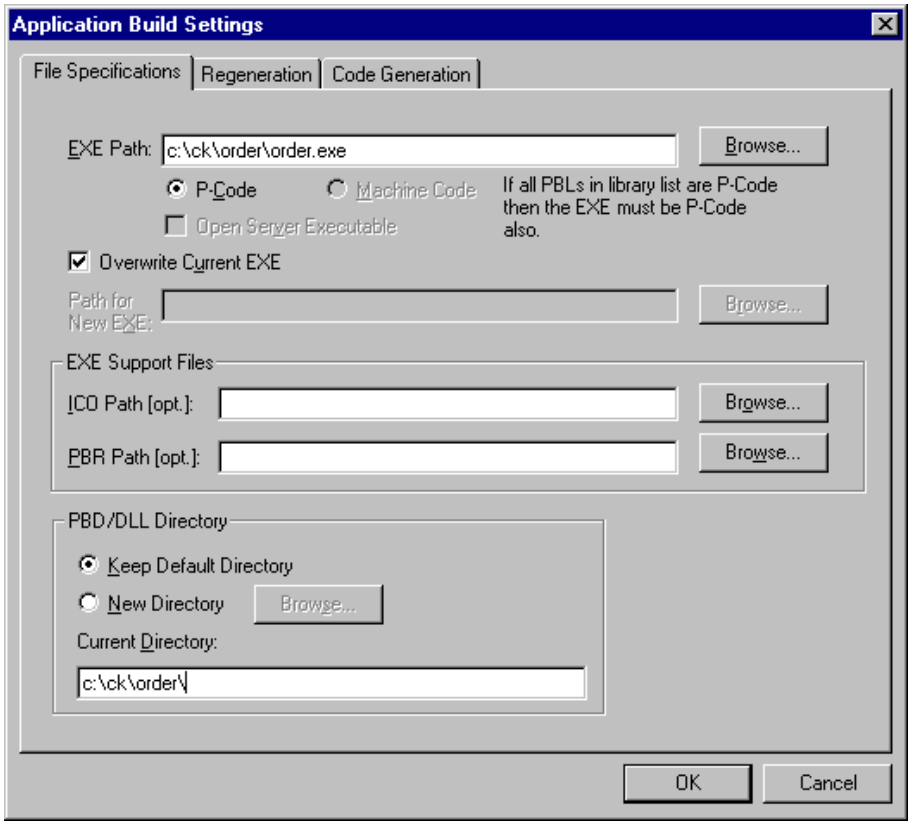
If you want to browse through the directories to locate the executable, icon file, or PBR file, click its corresponding Browse button. You'll see a list of all your directories and files. When you've found the file you're looking for, click on it in the list and it will appear in the appropriate PowerGen field.

Step 4 Example

The named project, creative.gen, appears in the Window title area and the two applications Creative Kids specified in Step 3 are still listed in the Applications box of the PowerGen window.



Creative Kids uses the Options button to open the Application Build Settings dialog box and set File Specifications for the Order application.



They use the Browse button to specify the path and name of the executable file in the EXE Path text box and the executable support files in the other text boxes. Creative Kids keeps the PBD's for the Order application in the default directory (which is the directory where the PBL is located).

Creative Kids would then specify a path to the executable for the Customer application as well.

Step 5: Specify Regeneration Information

You can specify object regeneration information for an entire project, or on an application-by-application basis.

- To specify object regeneration information at the project level, you would choose Options → Project... and then click on the Regeneration folder and choose the appropriate options.
- If you are specifying object regeneration information at the application level, you do so after specifying the executable path. Because the Application Build Settings dialog box is still open, you simply click on the Regeneration folder and then choose the appropriate options.

You can specify whether you want PowerGen to perform an incremental regeneration on the objects in the referenced libraries, which objects PowerGen should target for regeneration, and whether to export objects after regenerating them.

Defaults:

PowerGen's default settings for regeneration information are explained below. *We recommend using all default settings **except** for the last one listed – regenerating duplicate objects.*

Full regeneration. PowerGen does *not* perform an incremental regeneration unless you specify otherwise. Use a full regeneration to be sure that the deliverables represent the exact state of each object in each library.

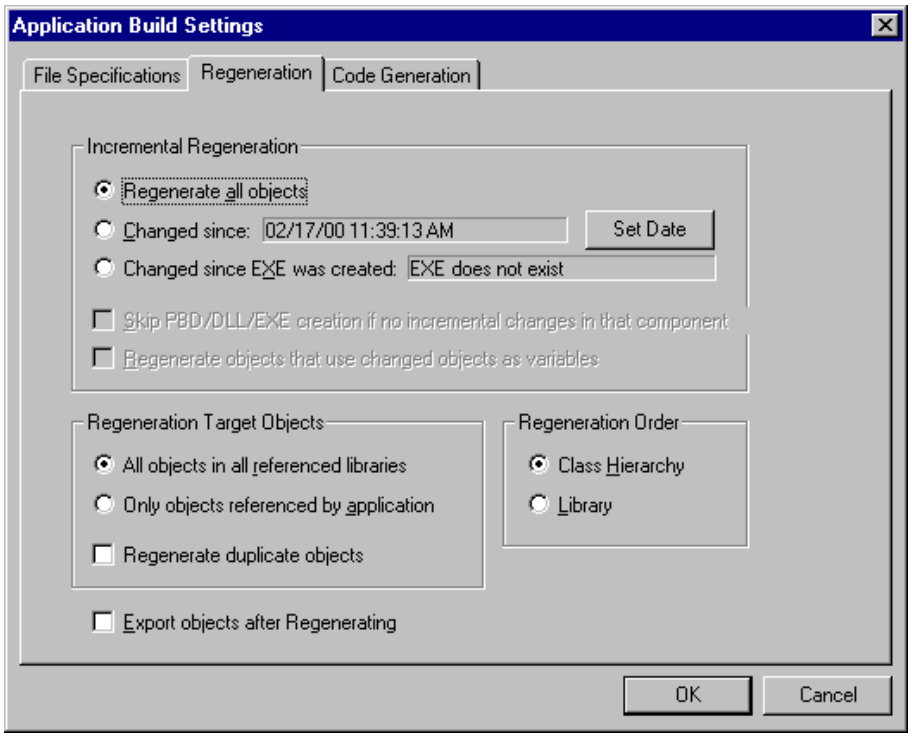
Use incremental regeneration to integrate changes made by developers in the course of producing interim releases that are intended to keep object modification in sync. If you select incremental regeneration, you may skip PBD creation if none of the objects in the library have been regenerated and/or choose to regenerate objects that use changed objects as variables.

Regenerate all objects in all referenced libraries. PowerGen regenerates all objects in all of the referenced libraries for the selected application unless you specify otherwise. If you have multiple applications in a single library and want PowerGen to regenerate only those objects referenced by the current application, it would be appropriate to choose the alternate option.

Regenerate Duplicate Objects. PowerGen will regenerate duplicate objects unless otherwise specified. This default exists to make PowerGen V4.2 compatible with previous versions of the product. However, regenerating duplicate objects will increase build time. Unless you need to maintain consistency with a previous version of PowerGen, we recommend that you choose NOT to regenerate duplicate objects.

Step 5 Example

This screen shows the Regeneration folder at the Application Build Settings dialog box:



Creative Kids is specifying a full regeneration for the Order application and regenerating all objects in all of the referenced libraries for the Order application. They have chosen *not* to regenerate duplicate objects.

Step 6 (Optional): Specify Any Changes to Code Generation Defaults

PowerGen's code generation defaults (the type of p-code and machine code) have been carefully selected based on the vast majority of PowerBuilder development projects. In this step, you're not selecting whether to use p-code or machine code (you'll do that in Step 8, when specifying PBL information); rather, you're setting what type of p-code and machine code you'll use in your project and/or application. *In almost all cases you will keep PowerGen's code generation defaults and skip this step*.

Note that there may be only one form of machine code generation and one form of p-code generation for the whole application. For example, if you select the machine code to be optimized for speed, that setting will apply to all machine code deliverables.

If you need to make changes to the defaults:

- To change code generation defaults at the project level, you would choose Options → Project... and then click on the Code Generation folder and choose the appropriate options.
- If you are changing code generation defaults at the application level, you do so after specifying the executable path. Because the Application Build Settings dialog box is still open, you simply click on the Code Generation folder and then choose the appropriate options.

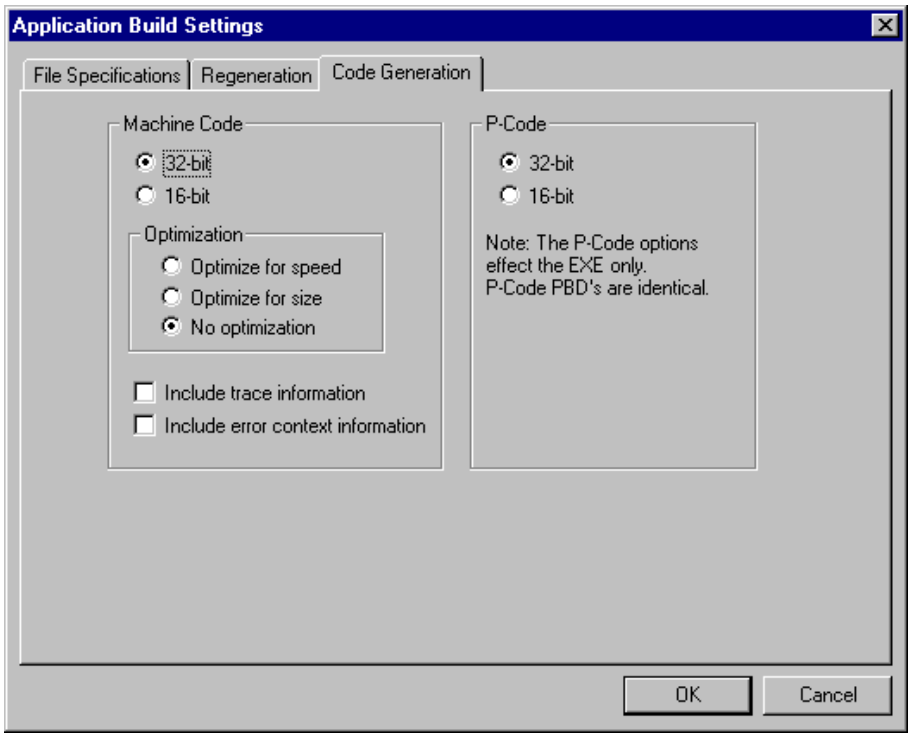
Regeneration Options & Defaults

PB Version	Code Formats Options
Version 5.0	32-bit p-code 16-bit machine code no optimization 32-bit machine code no optimization (default)
Version 6.X	16-bit p-code 32-bit p-code (default) 16-bit machine code no optimization 32-bit machine code no optimization (default)
Version 7.0	32-bit p-code 32-bit machine code no optimization (default)
Version 8.0	32-bit p-code 32-bit machine code no optimization (default)

When specifying the type of machine code output, the default is not to optimize. If you change the defaults, you may optimize for speed or for size. For machine code output, you may also choose to set the Trace Information option if you intend to trace program execution (by using the /debug command line argument with your application). Set the Error Context option to display information about the specific object, event, and script line number in your application when a runtime error occurs.

Step 6 Example

Creative Kids skips this step in the process. The screen below shows the Code Generation folder at the Application Build Settings dialog box. Because Creative Kids is using PowerBuilder Version 7.0, their default options are 32-bit for p-code and 32-bit for machine code, with no optimization.



Step 7: Specify PBL Information

For each PBL or PBD included in each application's library list, specify the following:

- Whether or not you want the library objects regenerated when the current application is built.
- Whether you want to locate the library objects in a PBD, EXE, or DLL (locating them in a PBD is the default)
- If you're locating the objects in a PBD or DLL, then also specify whether you want to create the PBD at the same time as the EXE for the current application. This is the mechanism used to exclude common libraries you do not want to create redundantly.
- If you're locating the objects in a PBD (p-code deliverable), the type of p-code code and options selected at the application and/or project level are applied.
- If you're locating the objects in a DLL (machine code deliverable), the type of machine code and options selected at the application and/or project level are applied. Although PowerBuilder lets you produce it, machine code is ***not*** generally recommended for the following reasons:
  - Build times are excessive
  - Build- and run-time environments are less robust than p-code
  - Deliverables are much larger

  In some instances there may be selected portions of an application that will benefit from machine code. If that were the case, selecting to locate the PBL objects in a DLL would occur in this step.

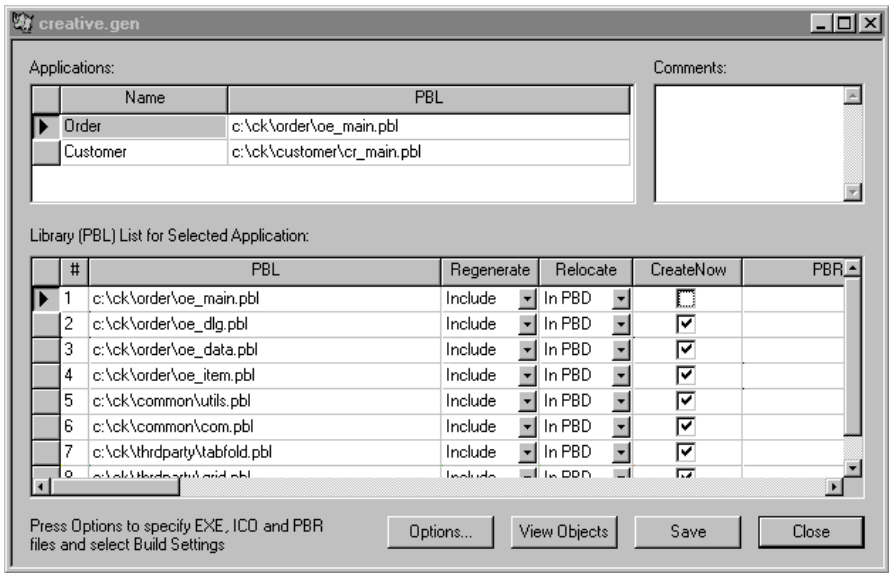
  (PowerGen has a unique ability to select machine code or p-code on a PBL-by-PBL basis.)

NOTES:

- If you are building both 32-bit and 16-bit p-code versions of your program, keep in mind that only the executables are affected. You would *include* all the PBL's for regeneration when building the first version and *exclude* the PBL's from regeneration when building the second version.
- If a write-protected PBL is included in the library list, it must be excluded from being regenerated (although it will still be used to resolve inheritance dependencies). From a write-protected PBL you *can* create dynamic libraries.
- If a PBD is included in the library list it must be excluded from being regenerated and marked *not* to be created.

Step 7 Example

The screen displayed here shows the OE_MAIN library within the Order Entry application. Creative Kids is including the objects in this PBL in the regeneration and locating its objects in the PBD.



The list below shows the specifications Creative Kids will enter for each library within each application in their PowerGen project.

Applications	Libraries in List	Include/Exclude from Regen	Object Location	Create PBD now?
Order Entry	oe_main.pbl	Include	PBD	Yes
	oe_dlg.pbl	Include	PBD	Yes
	oe_data.pbl	Include	PBD	Yes
	oe_item.pbl	Include	PBD	Yes
	utils.pbl	Include	PBD	Yes
	com.pbl	Include	PBD	Yes
	tabfold.pbl	Include	PBD	Yes
	grid.pbl	Include	PBD	Yes
Customer Records	cr_main.pbl	Include	PBD	Yes
	cr_data.pbl	Include	PBD	Yes
	cr_win.pbl	Include	PBD	Yes
	utils.pbl	Exclude	PBD	No
	com.pbl	Exclude	PBD	No
	tabfold.pbl	Exclude	PBD	No
	grid.pbl	Exclude	PBD	No

Note that when common libraries and third party or class framework libraries are specified within the Order Entry application, they are included in the regen and are built with the other libraries in the application. They are *excluded* from the regen (avoiding redundant regeneration) and not built when specified in the Customer Records applications. This results in shorter build times and makes the process more repeatable, since you are not performing the same step (building common PBD's) more then once.

Step 8: Specify Build Options and Define Log File Preferences

Build Options

Before you save your project and begin building each of the applications in it, you can specify several build options. You may choose:

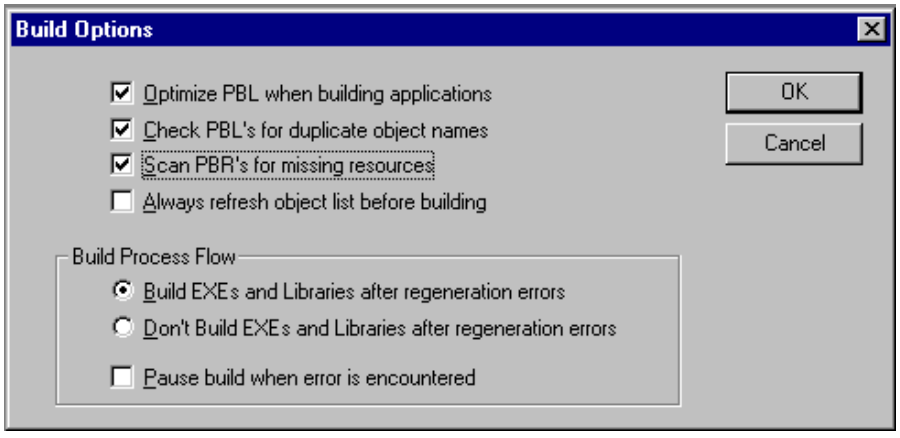
- To optimize the PBL's during the build. The default is *not* to optimize.
- To check PBL's for duplicate object names and report them in the build log in the order in which they are found in the library list. This is the default.
- To scan PBR's for missing resources before attempting to rebuild PBD's and EXE's. This eliminates the unwanted message dialog from being displayed during a build. The default is *not* to scan.
- To refresh the object list before building the project. The default is *not* to refresh the list. If the contents of a library are changing while your PowerGen project is open, using this option ensures that no objects are missed during a build. The option setting information is saved in the Registry.

You can also specify how you want the build process to flow when regeneration errors occur during the current PowerGen session. The default is to build EXE's and libraries after regeneration errors and not to pause the build when an error is encountered.

NOTE: These build options are preserved across PowerGen sessions and are saved in the Registry.

Step 8 Example

Creative Kids has selected to optimize PBL's when building applications and to build EXE's and libraries even if regeneration errors occur.



Define Log File Preferences

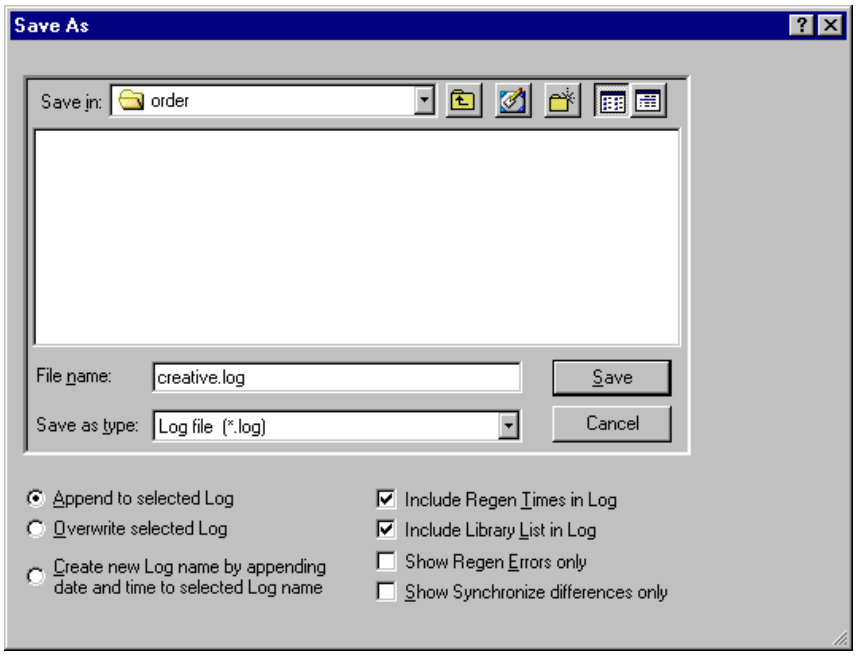
PowerGen's log file is created by default, named POWERGEN.LOG, and is located in the directory where the target application PBL is located.

Each time you build your entire application or any part of it, Synchronize PBL's, Bootstrap Import objects within selected applications, or export or import one or more objects, new log messages are written to the log file, giving you an on-going record of your progress. You may choose to have messages appended to an existing log file or written to a new file based on the date and time.

Set your log file preferences by choosing Options → Log File. The default options are to append the information about this build to the selected log file, to include the regen times in the file, and to include the library list in the file.

Step 8 Example

At this point, Creative Kids has kept the default log file preference options.



When Creative Kids is in the final stages of testing their build, they might select the "Show Regen Errors only" option, to target only those errors and abbreviate the log file output as a result.

Step 9: Save the Information Needed to Build the Project

Now you've entered all the specific information needed to build your project with PowerGen. Save this information by choosing Project → Save. PowerGen will automatically save the information to the same project file you specified in Step 3.

Step 10: Test the Build

Before actually building the entire project, test that the build works as you planned. Make a backup copy of all your components before you begin testing, so if errors occur you can restore all the elements properly.

To test the build, it's recommended that you interactively test each application in the list by:

1. Regenerating objects

1. Highlight the first application in the list and choose Application → Regenerate.
2. When the regeneration is completed, review PowerGen's log file, looking for errors and for an indication that all objects were re-generated once and only once.
3. Then go into the PowerBuilder Library painter to check that everything was regenerated correctly.
4. Once you're satisfied that this aspect of the build works properly, continue testing the application by optimizing the PBL's.

2. Optimizing PBL's

1. With the first application in the list still highlighted, choose Application → Optimize.
2. When optimized, check the directories to make sure that the optimized PBL's are smaller than they were before being optimized. This operation saves the original in a backup file with the same name as the original and a .bak extension.

3. Creating PBD's

1. With the first application in the list still highlighted, choose Application → Create PBD's.
2. When created, look for errors in PowerGen's log file.
3. Check the directories to make sure that the PBD's were created properly.
4. Once you're satisfied that this aspect is working properly, continue testing the application by creating the executable.

4. Creating EXE's

1. With the first application in the list still highlighted, choose Application → Create EXE.
2. When created, look for errors in PowerGen's log file.
3. Check the directory to make sure that the executable was created properly. Make sure the EXE can be started with no errors. Check that the desired icon is displayed with the EXE.
4. Once you're satisfied that this aspect is working properly, continue testing the application by repeating this process with each of the other applications, in appropriate order.

When the individual applications have been tested, choose Project → Build All to build the entire PowerGen project in a batch. Review the log file, looking for errors and making sure that PBD's and EXE's were created.

Step 10 Example

The following is an excerpt from the log file created from regenerating the objects in the "Order" application.

```
PowerGen Version 4.2
Log Started on 7/1/01  8:41:17am

Application:
order

Library List:
d:\ck\order\oe_main.pbl
d:\ck\order\oe_dlg.pbl
d:\ck\order\oe_data.pbl
d:\ck\order\oe_item.pbl
d:\ck\order\utils.pbl
d:\ck\order\com.pbl
d:\ck\order\tabfold.pbl
d:\ck\order\grid.pbl

Regenerating Objects in Inheritance Order

DataWindows
d_ord_codelist_values_unaggr      d:\ck\order\oe_main.pbl
d_ord_codelist_values_aggr        d:\ck\order\oe_main.pbl
d_ord_pane_attrib                 d:\ck\order\oe_dlg.pbl
```

```
Structures
str_pane_layout_object          d:\ck\order\oe_dlg.pbl
str_pane_layout_font            d:\ck\order\oe_dlg.pbl
str_pane_layout_color           d:\ck\order\oe_dlg.pbl

Windows
w_ord_base_browser              d:\ck\common\utils.pbl
    w_ord_codelist_browser      d:\ck\order\oe_main.pbl
w_ord_codelist_export           d:\ck\order\oe_main.pbl
w_ord_codelist_import           d:\ck\order\oe_main.pbl
w_ord_codelist_import_create    d:\ck\order\oe_main.pbl
w_ord_codelist_values_modify    d:\ck\order\oe_main.pbl
w_ord_frame                     d:\ck\common\utils.pbl
Error C0019: Incompatible field ii_active_sheet_count—for type w_ord_frame at
line 25 of function wf_pop_sheet of object w_ord_frame
Line no: 25 Column no: 0
Error C0015: Undefined variable: i—at line 26 of function wf_pop_sheet of object
w_ord_frame
Line no: 26 Column no: 0
w_ord_page_layout_browser       d:\ck\common\com.pbl
...
Log Ended on 7/1/01 8:45am
```

Based on the Log File preference options selected in the previous step, Creative Kids' log file lists all of the activities carried out in the build process, including regeneration and PBD and EXE creation. Errors are listed with the object that caused the error. The log output will also appear in the Output Window while the build is running.

Step 11: Test the Resulting Components

Test all the deliverable components to make sure the build produced operational applications.

Source Traceable Builds for PowerBuilder Applications: Step by Step

PowerGen's Bootstrap Import process lets you reproduce reliable releases from a set of controlled source. Through an iterative process, the Bootstrap Import first imports objects from which the cross-dependencies have been removed. In subsequent iterations, it imports more and more of the body of the objects, until they have all been completely restored in the PBL's.

This methodology gives you a rigorous approach to source and release control, since each application is "built from source" rather than relying on a set of existing PBL's. The methodology described here follows these steps:

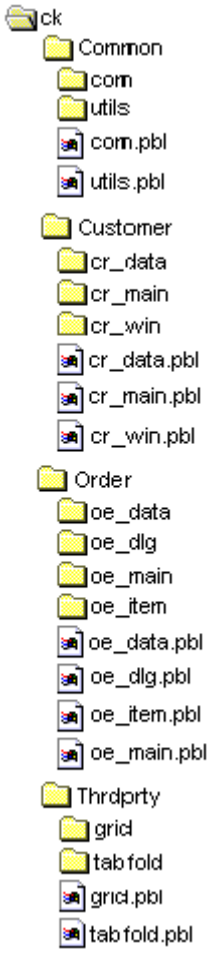
- Step 1: Get object source from your source control system and create PBL subdirectories.
- Step 2: Create an object list file (OLF) for each application.
- Step 3: Delete the original PBL subdirectories.
- Step 4: Set options to be applied during the Bootstrap Import
- Step 5: Use PowerGen to Bootstrap Import each application's objects.
- Step 6: Proceed with PowerGen's build process.

Step 1: Get Object Source from your Source Control System and Create PBL Subdirectories

The first step in producing source traceable builds with PowerGen is to get the object source from your source control system. It's recommended that you create a subdirectory for each PBL in the application and that you create the subdirectory below the directory where the PBL itself is located. Once you've created all the appropriate subdirectories, then place the associated objects for the PBL into each of them.

Step 1 Example

Creative Kids has one Class Framework library, one set of in-house Common Libraries, and two applications – Order Entry and Customer Records. They create the following subdirectories for object source and place the appropriate objects in them.



Note that the Common, Thrdprty, Order, and Customer directories contain the existing PBL's as well. In Step 3, these will be deleted so that PowerGen can Bootstrap Import the new object source and create new PBL's from it.

Step 2: Create an Object List File (OLF) for Each Application

An Object List File is a text file that associates the object source with a PBL in the library list. Based on the library list, PowerGen automatically creates the OLF for you, assuming that each PBL subdirectory is located below the directory where the PBL is located. Use PowerGen to create an OLF for *each* application you are going to build.

The OLF is a multi-line text file where each line has the format:

source_file_spec, target_PBL[, object_comments[, target_PBL_comments]]

Target PBL comments, if included, will appear as the comment string associated with the PowerBuilder library. The object comments can be specified for each object. If there are no object comments, the object_comments can be left blank, but delimited by commas. The object comments and PBL comments can be enclosed in quotes if the comments contain commas.

Step 2 Example

Creative Kids is building the Order Entry application first, and is including the Class Framework and Common Library PBL's in this application. PowerGen creates an OLF for the Order Entry application that looks like this:

ORDER.OLF

```
d:\ck\order\oe_main\*.sr*, d:\ck\order\oe_main\oe_main.pbl
d:\ck\order\oe_dlg\*.sr*, d:\ck\order\oe_dlg\oe_dlg.pbl
d:\ck\order\oe_data\*.sr*, d:\ck\order\oe_data\oe_data.pbl
d:\ck\order\oe_item\*.sr*, d:\ck\order\oe_ item \oe_ item.pbl
d:\ck\common\utils\*.sr*, d:\ck\common\utils\utils.pbl
d:\ck\common\com\*.sr*, d:\ck\common\com\com.pbl
d:\ck\thrdprty\tabfold\*.sr*, d:\ck\thrdprty\tabfold \tabfold.pbl
d:\ck\thrdprty\grid\*.sr*, d:\ck\thrdprty\grid\grid.pbl
```

The OLF for the Customer Records application is shown below. Note that the Class Framework and Common Library PBL's are not included here, since they are being created with the Order Entry application.

CUSTOMER.OLF

```
d:\ck\ customer\cr_data\*.sr*, d:\ck\customer\cr_data\cr_data.pbl
d:\ck\customer\cr_main\*.sr*, d:\ck\customer\cr_main\cr_main.pbl
d:\ck\ customer\cr_win\*.sr*, d:\ck\customer\cr_win\cr_win.pbl
```

Step 3: Delete Libraries You Want to Recreate

In this step, PowerGen deletes the PBL's you want to recreate. These are the libraries you'll be recreating during Bootstrap Import, so deleting them now ensures that you'll be reproducing the release from source.

Step 3 Example

Creative Kids now uses PowerGen to delete all libraries from the Common, Thrdprty, Order, and Customer subdirectories. No PBL's now exist, and each subdirectory contains its associated object source.

Step 4: Set Options to be Applied During the Bootstrap Import

PowerGen lets you specify a variety of import options, such as the format of your source PBL's and whether to write temporary files during the Bootstrap Import, to regenerate all objects after the last Import phase, and to register objects for source control after the import.

Import files are normally in the same syntax as if they were exported from PowerBuilder, with header information that describes the object's name, type, and comments, followed by the source code for the object.

```
$PBExportHeader$w_mywindow_info.srw
$PBExportComments$Comments for my window
```

In some versions of PowerBuilder, the SCC API source control interface strips the $PBHeader information from the object source before saving in the source control system. For this reason, PowerGen allows you to use import files with this syntax variation.

By default, PowerGen automatically detects whether or not the $PBHeader… lines are included at the beginning of external object source file and automatically imports the appropriate syntax. This allows you to import files without having to examine the source files and to import files with and without headers in the same import function.

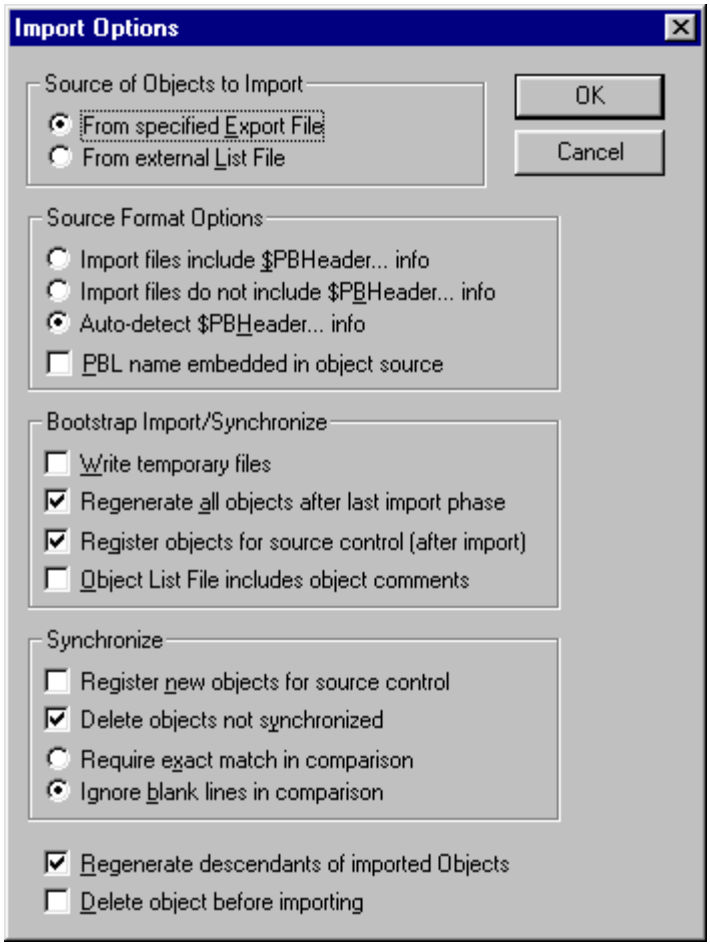
You may also choose to write temporary files to disk during the Bootstrap Import. PowerGen creates temporary objects that are imported in the process of constructing the libraries from scratch. This is required to avoid conflicts caused by unresolved references between two objects. As an aid in diagnosing problems in the Bootstrap Import process, these temporary objects can be saved to disk.

PowerGen automatically regenerates all the objects after the last iteration of the Bootstrap Import has been completed. You may disable this feature.

And finally, you may choose to have PowerGen automatically register objects for source control once Bootstrap Import is completed.

Step 4 Example

Creative Kids is keeping the defaults to automatically detect $PBHeader…information and to regenerate objects after the last Bootstrap Import phase. They are also requesting PowerGen to automatically register the objects for source control when Bootstrap Import is completed.



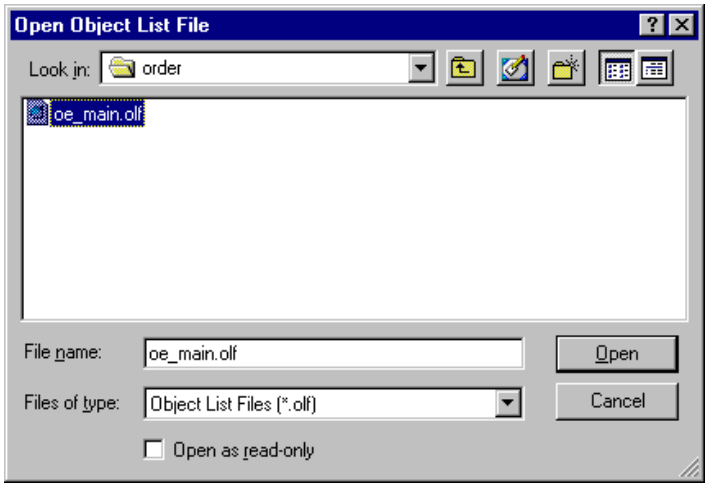
Step 5: Use PowerGen to Bootstrap Import Each Application's Objects

To begin the Bootstrap Import process, choose Application → Bootstrap Import, and select the OLF for the first application you want to build.

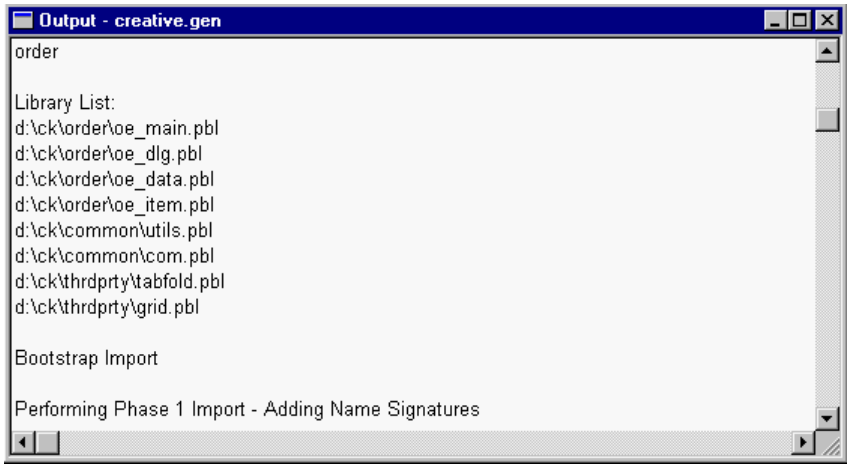
PowerGen then begins the Bootstrap Import process, displaying the results in an output window and stored in the specified log file for the current project.

Step 5 Example

Creative Kids first Bootstrap Imports objects in the Order application.

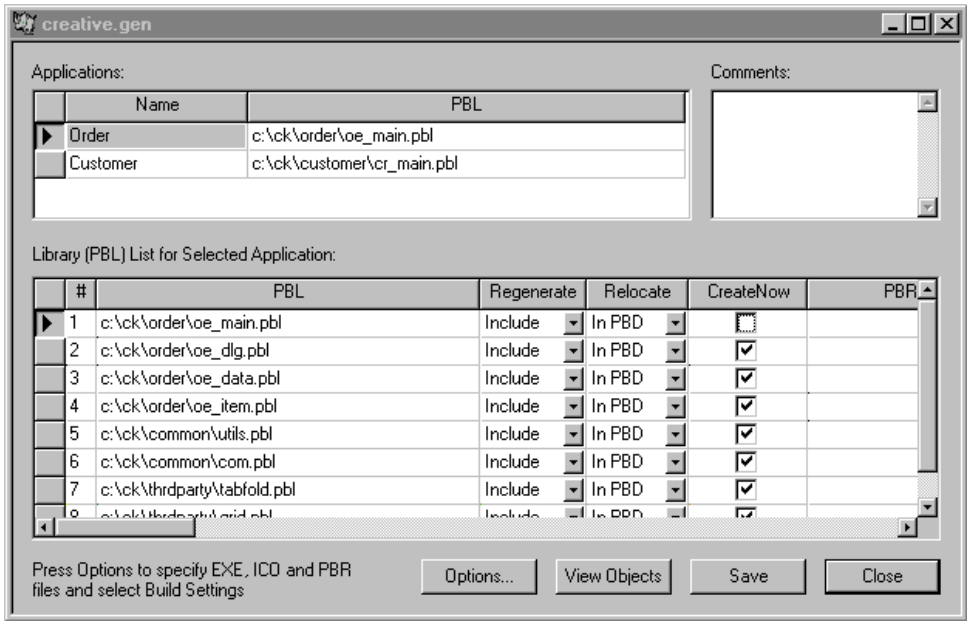


And then views the results in the Output Window. In the early phases of the import errors may appear in the log; these may be ignored. The final error total will only include errors encountered in the final phase.



Creative Kids then Bootstrap Imports the Customer application and views the results in the Output Window as well.

At the conclusion of this process, PowerGen's application list looks identical to its appearance before Bootstrap Importing, but now the libraries listed are populated with source.



Step 6: Proceed with PowerGen's Build Process

With all PBL's in all your applications now populated with source, you're ready to proceed with PowerGen's build process.

- If you have already specified the executable path, support files, regen information, code generation information, PBL information, and build options for your applications (Steps 4 through 8 in the Step-by-Step Build Process described previously), then you would now proceed to testing the build (Step 9).
- If you haven't previously specified this information, then proceed with the PowerGen build process at Step 4.

Synchronization of PowerBuilder Applications to Source Control: Step by Step

PowerGen's "Synchronization" process lets you refresh your application PBLs from a set of controlled source. The Synchronization process selects which objects required updating or deletion and identifies new objects to be added to the PBLs. Like Bootstrap Import, Synchronization first imports objects from which the cross-dependencies have been removed. In subsequent iterations, it imports more and more of the body of the objects, until they have all been completely restored in the PBL's. Because it only deals with changes to PBL's, Synchronization runs in a fraction of the time that Bootstrap Import runs.

This methodology gives you a rigorous approach to maintaining development PBLs in sync with objects in source control. The methodology described here follows these steps:

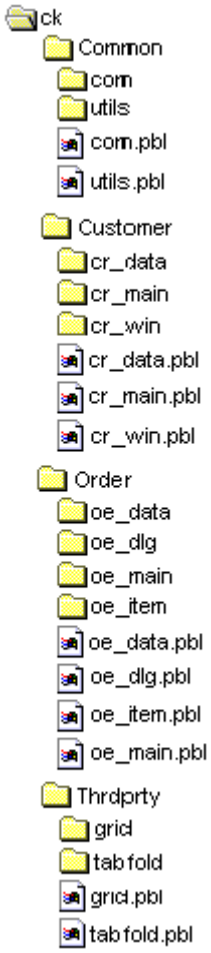
- Step 1: Get object source from your source control system and create PBL subdirectories.
- Step 2: Create an object list file (OLF) for each application.
- Step 3: Set options to be applied during Synchronization.
- Step 4: Use PowerGen to Synchronize each application's objects.
- Step 4: Proceed with PowerGen's build process.

Step 1: Get Object Source from your Source Control System and Create PBL Subdirectories

The first step in the source and release control process with PowerGen is to get the object source from your source control system. It's recommended that you create a subdirectory for each PBL in the application and that you create the subdirectory below the directory where the PBL itself is located. Once you've created all the appropriate subdirectories, place the associated objects for the PBL into each of them.

Step 1 Example

Creative Kids has one Class Framework library, one set of in-house Common Libraries, and two applications – Order Entry and Customer Records. They create the following subdirectories for object source and place the appropriate objects in them.



Note that the Common, Thrdprty, Order, and Customer directories contain the existing PBL's as well. These will be Synchronized with the updated source files.

Step 2: Create an Object List File (OLF) for Each Application

An Object List File is a text file that associates the object source with a PBL in the library list. Based on the library list, PowerGen automatically creates the OLF for you, assuming that each PBL subdirectory is located below the directory where the PBL is located. Use PowerGen to create an OLF for *each* application you are going to build.

The OLF is a multi-line text file where each line has the format:

source_file_spec, target_PBL[, object_comments[, target_PBL_comments]]

Target PBL comments, if included, will appear as the comment string associated with the PowerBuilder library. The object comments can be specified for each object. If there are no object comments, the object_comments can be left blank, but delimited by commas. The object comments and PBL comments can be enclosed in quotes if the comments contain commas.

Step 2 Example

Creative Kids is Synchronizing the Order Entry application first, and is including the Class Framework and Common Library PBL's in this application. PowerGen creates an OLF for the Order Entry application that looks like this:

ORDER.OLF

```
d:\ck\order\oe_main\*.sr*, d:\ck\order\oe_main\oe_main.pbl
d:\ck\order\oe_dlg\*.sr*, d:\ck\order\oe_dlg\oe_dlg.pbl
d:\ck\order\oe_data\*.sr*, d:\ck\order\oe_data\oe_data.pbl
d:\ck\order\oe_item\*.sr*, d:\ck\order\oe_ item \oe_ item.pbl
d:\ck\common\utils\*.sr*, d:\ck\common\utils\utils.pbl
d:\ck\common\com\*.sr*, d:\ck\common\com\com.pbl
d:\ck\thrdprty\tabfold\*.sr*, d:\ck\thrdprty\tabfold \tabfold.pbl
d:\ck\thrdprty\grid\*.sr*, d:\ck\thrdprty\grid\grid.pbl
```

The OLF for the Customer Records application is shown below. Note that the Class Framework and Common Library PBL's are not included here, since they are being Syncrhonized with the Order Entry application.

CUSTOMER.OLF

```
d:\ck\ customer\cr_data\*.sr*, d:\ck\customer\cr_data\cr_data.pbl
d:\ck\customer\cr_main\*.sr*, d:\ck\customer\cr_main\cr_main.pbl
d:\ck\ customer\cr_win\*.sr*, d:\ck\customer\cr_win\cr_win.pbl
```

Step 3: Set Options to be Applied During the Synchronization

PowerGen lets you specify a variety of import options, such as the format of your source PBL's and whether to write temporary files during the Synchronization, to regenerate all objects after the last Import phase, and to register new objects for source control after the Synchronization.

Import files are normally in the same syntax as if they were exported from PowerBuilder, with header information that describes the object's name, type, and comments, followed by the source code for the object.

```
$PBExportHeader$w_mywindow_info.srw
$PBExportComments$Comments for my window
```

In some versions of PowerBuilder, the SCC API source control interface strips the $PBHeader information from the object source before saving in the source control system. For this reason, PowerGen allows you to use import files with this syntax variation.

By default, PowerGen automatically detects whether or not the $PBHeader… lines are included at the beginning of external object source file and automatically imports the appropriate syntax. This allows you to import files without having to examine the source files and to import files with and without headers in the same import function.

You may also choose to write temporary files to disk during the Synchronization. PowerGen creates temporary objects that are imported in the process of constructing the libraries from scratch. This is required to avoid conflicts caused by unresolved references between two objects. As an aid in diagnosing problems in the Synchronization process, these temporary objects can be saved to disk.

PowerGen automatically regenerates all objects after the last iteration of the Synchronization has been completed. You may disable this feature.

You may choose to have PowerGen automatically register new objects for source control once Synchronization is completed.

When a new object is added to a PBL during Synchronization, the default is for PowerGen to automatically mark it as Registered.

You may choose to have the Synchronize function delete objects from the PBL's that do not have corresponding source files. If objects are deleted, PowerGen will attempt to regenerate the objects that referenced the deleted objects.

The Synchronization function compares the object source in the PBL and the source in the object source file in order to determine which objects have changed. The comparison can be selected as either an exact match or as one in which blank lines are ignored. Blank lines are commonly added or removed when PowerBuilder imports or exports an object. These lines have no effect on the object's function so can safely be ignored. This is the default.

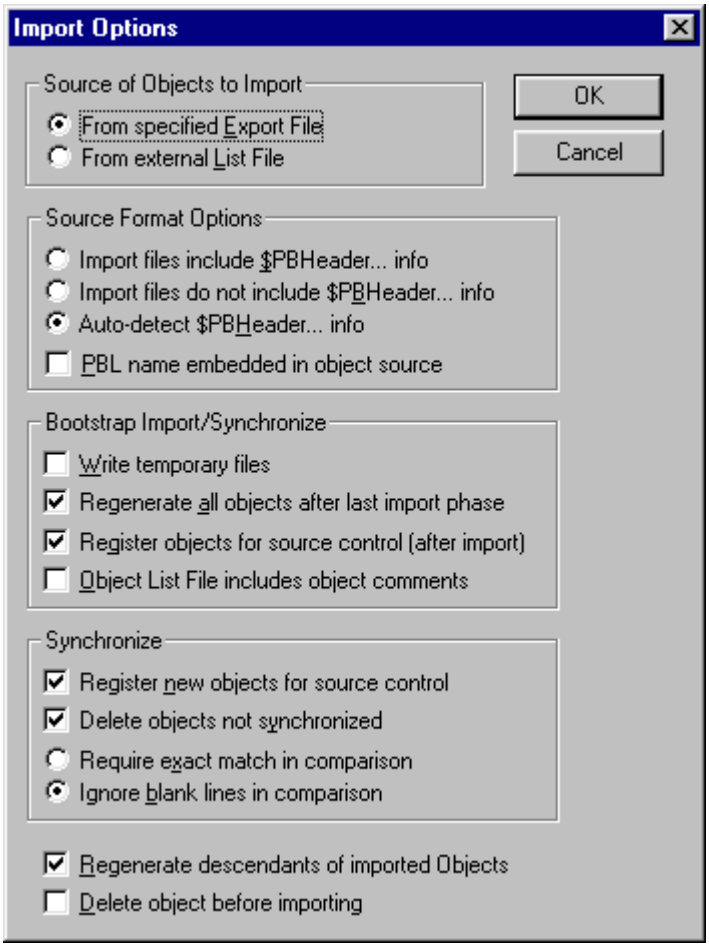
Set Log File Options for Synchronization

PowerGen lets you specify whether you want the log file, when displaying Synchronization information, to show only the objects that have been modified (i.e., added, deleted, or imported) because a change was detected. Choose this option by selecting Options → Log File and then making the appropriate selection at the bottom of the Log File Save As dialog.

You would most likely not choose this option initially, but later when fewer errors are likely to be encountered.

Step 3 Example

Creative Kids is keeping the defaults to automatically detect $PBHeader…information and to regenerate objects after the last Synchronization phase. They are also requesting PowerGen to automatically register the objects for source control when Synchronization is completed, to register new objects for source control, to delete PBL objects not synchronized, and to ignore blank lines in comparison.



Because Creative Kids does not want the log file to show only Synchronize differences, they do not need to open the Log File options and make any changes there.

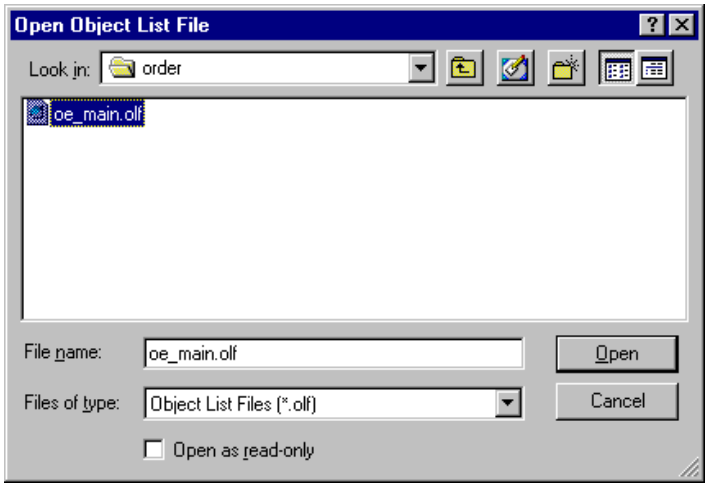
Step 4: Use PowerGen to Synchronization Each Application's Objects

To begin the Synchronization process, choose Application → Synchronize, and select the OLF for the first application you want to build.

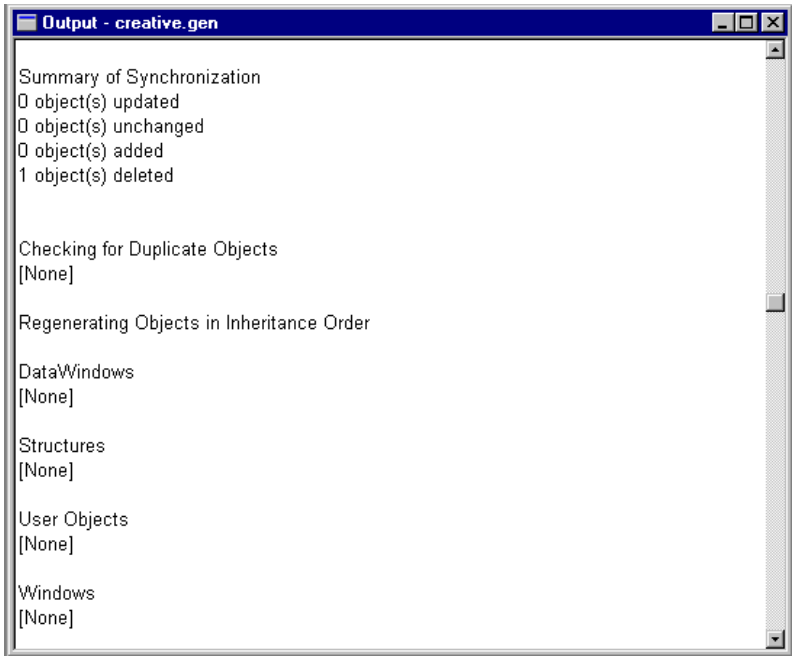
PowerGen then begins the Synchronization process, displaying the results in an output window and stored in the specified log file for the current project.

Step 4 Example

Creative Kids first Synchronizes objects in the Order application.



And then views the results in the Output Window. In the early phases of the import errors may appear in the log; these may be ignored. The final error total will only include errors encountered in the final phase.



Creative Kids then Synchronizes the Customer application and views the results in the Output Window as well.

Step 5: Proceed with PowerGen's Build Process

With all PBL's in all your applications now Synchronized, you're ready to proceed with PowerGen's build process.

- If you have already specified the executable path, support files, regen information, code generation information, PBL information, and build options for your applications (Steps 4 through 8 in the Step-by-Step Build Process described previously), then you would now proceed to testing the build (Step 9).
- If you haven't previously specified this information, then proceed with the PowerGen build process at Step 4.