

Migration Guidelines for PowerServer Web

Appeon® PowerServer® 2020
FOR WINDOWS & UNIX & LINUX

DOCUMENT ID: ADC37817-01-2020-01

LAST REVISED: March 25, 2020

Copyright © 2020 Appeon. All rights reserved.

This publication pertains to Appeon software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Appeon Inc.

Appeon, the Appeon logo, Appeon PowerBuilder, Appeon PowerServer, PowerServer, PowerServer Toolkit, AEM, and PowerServer Web Component are trademarks of Appeon Inc.

SAP, Sybase, Adaptive Server Anywhere, SQL Anywhere, Adaptive Server Enterprise, iAnywhere, Sybase Central, and Sybase jConnect for JDBC are trademarks or registered trademarks of SAP and SAP affiliate company.

Java and JDBC are trademarks or registered trademarks of Sun Microsystems, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Appeon Inc., 1/F, Shell Industrial Building, 12 Lee Chung Street, Chai Wan District, Hong Kong.

Contents

1	Web RAD with PowerBuilder and PowerServer	1
1.1	Traditional approach to Web development	1
1.2	Advantages of PowerServer for Web RAD	1
1.3	Leveraged functionality with server components	2
1.4	PowerServer Web RAD methodology	2
1.4.1	PowerServer-standard PowerBuilder coding	2
2	Migration Process	4
2.1	Introduction	4
2.2	Installing required softwares	5
2.3	Understanding the general limitations of the PowerServer solution	6
2.4	Defining migration objective	7
2.4.1	Defining migration objective for non-PFC application	7
2.4.2	Defining migration objective for PFC application	8
2.4.2.1	Corporate PFC architecture	8
2.4.2.2	Defining migration objective progressively	9
2.5	Upgrading the original application	9
2.5.1	Upgrading obsolete PBLs	9
2.5.2	Upgrading PFC applications	9
2.6	Preparing the target application	10
2.6.1	Special processing required for PFC applications	10
2.6.2	Processing application based on migration objectives	10
2.7	Pre-configuring for the Web applications	11
2.7.1	Why is pre-configuration necessary	11
2.7.2	Four pre-configuration tasks	12
2.8	Modifying unsupported features	12
2.8.1	How to identify unsupported features	12
2.8.2	Feature modification methods	13
2.9	Enhancing the application with Web or PowerServer features	13
2.10	Trial deployments and debugging	13
2.10.1	Special deployment steps for distributed applications	14
2.10.2	Debugging deployed applications	14
2.11	Fine-tuning the runtime performance	14
2.12	Go-live deployment	14
2.13	Go live	15
3	Migration FAQ	17
3.1	How do I rapidly build a new Web application with PowerServer?	17
3.2	Does PowerServer support every PowerBuilder feature?	17
3.3	Do PowerServer-deployed Web applications support external resources?	17
3.4	Why classify my application into types?	17
3.5	What are the different application types?	18
3.6	What are the recommendations for converting the different application types?	18
3.7	What are the basic requirements for rewriting complex applications?	19
3.8	When would I need to modularize my application?	19
3.9	What are the benefits of modularizing my application?	19

3.10	What are the basic principles for modularizing an application?	19
3.11	Can you give an example of the modularization process?	19
4	Enhancing an application with Web or PowerServer features	21
4.1	PowerServer open interfaces	22
4.1.1	Description of the open interfaces	22
4.1.2	Applying PowerServer open interfaces in PowerServer-deployed applications	23
4.2	Appeon client functions	23
4.2.1	Description of Appeon client functions	24
4.2.2	Using Appeon Client functions	26
4.3	Loading an application in SAP Sybase Enterprise Portal	26
4.3.1	Restrictions on supporting Enterprise Portal	26
4.3.2	Tasks required to load the application in Enterprise Portal	27
4.4	Single sign-on	27
4.5	Integrating PowerServer Web applications with JSP/ASP	27
5	Processing PFC Applications	28
5.1	Auto-Sensing Environment	28
5.2	Suggested Modifications to PFC	28
5.3	Unsupported user objects	28
5.4	Unsupported standard class objects	29
5.5	Unsupported object properties	30
5.6	Unsupported object functions	34
5.7	Unsupported object events	35
5.8	Unsupported system functions	35
5.9	Unsupported Shared variables	35
5.10	Unsupported Function not found	35
5.11	Unsupported calls	36
5.12	Differently behaved features	38
6	Migration Tutorial	42
6.1	Introduction	42
6.1.1	Preparing for the tutorial	42
6.1.2	Relevant files	43
6.2	Loading the Tutorial PowerBuilder Application	44
6.2.1	Creating a Workspace	44
6.2.2	Loading the tutorial PBL file	46
6.2.3	Configuring ODBC data source	48
6.2.4	Running the tutorial application	50
6.3	Configuring PowerServer Toolkit	53
6.3.1	Configuring basic settings	54
6.3.2	Selecting PBL file(s)	55
6.3.3	Configuring deployment settings	55
6.3.3.1	Starting PowerServer and Web server	56
6.3.3.2	Adding a PowerServer profile	56
6.3.3.3	Adding a Web server profile	59
6.3.3.4	Selecting a deployment profile	62
6.3.4	Selecting DB Type(s)	62
6.3.5	Declaring transaction object(s)	64
6.4	Analyzing Unsupported Features	69

6.4.1 Unsupported features analysis	69
6.4.2 Optimizing and full build	71
6.5 Deploying the Tutorial PowerBuilder Application	74
6.6 Running the Web Application	75
Index	79

1 Web RAD with PowerBuilder and PowerServer

Developers can utilize PowerBuilder and PowerServer to achieve rapid Web application development (Web RAD). Web RAD with PowerBuilder and PowerServer greatly reduces the software development life cycle and I.T. development complexity, while preserving end-user productivity.

1.1 Traditional approach to Web development

In order to write even a small Web application with traditional methods, it takes a great deal of time, and the resulting Web application usually has an oversimplified user interface. Web application developers must master various new skills, such as ASP/JSP/PHP, Ruby, Python, Perl, JavaScript, Ajax, XML, HTML, Java, .NET, and J2EE when using Microsoft or Java technologies.

1.2 Advantages of PowerServer for Web RAD

Using PowerBuilder and PowerServer to develop powerful Web applications has the following advantages:

1.2.1 Provides the fastest path to the Web

PowerServer achieves unparalleled reductions in project cycle time and costs by eliminating the traditional Web application design, coding, and testing effort associated with rewriting applications for the Web. When your objective is to build a new application, PowerServer provides the most productive development approach for building enterprise-class applications that adhere to the most stringent Web standards.

1.2.2 Provides the best Web GUI possible

PowerServer Web applications precisely replicate the desktop application user interface with HTML running in standard Windows Web browsers. PowerServer graphical capabilities make the end user far more productive than any possible rewrite and eliminates all user-retraining time and costs.

1.2.3 Minimizes business risks

PowerServer greatly reduces business risks associated with building new software applications, because the time-tested PowerBuilder program code and DataWindows run at the core of the Web application.

1.2.4 Leverages current organizational skills

PowerServer:

- Relies solely on PowerBuilder skills and IDE to develop and maintain the application.
- Reuses the existing application source code and database, preserves investments in designing, building and testing the application UI, application business logic, data-access logic, and the database.

- Operates upon a single set of native PowerBuilder source code for deployment to both desktop and Web environments.

1.3 Leveraged functionality with server components

PowerServer's support to .NET/COM components enables you to access enhanced Web functionality. When developing new Web applications with PowerBuilder and PowerServer, you can use the .NET/COM components in PowerServer as a bridge to access many other functions on the Web and some Client Server application features that may even transcend the Web limitations. For detailed description, refer to Section 1.3.7, "Calling .NET/COM server components (.NET only)" in *Workarounds & APIs Guide*.

1.4 PowerServer Web RAD methodology

Web RAD (Rapid Application Development) with PowerBuilder and PowerServer is for writing a new PowerBuilder application and converting it to the Web using PowerServer.

There are three steps in a PowerServer Web development project:

Step 1: Analyze the project requirements. According to these requirements, lay out the function specifications that should be met for the project.

Step 2: Develop a new PowerBuilder application and implement the functions in it with PowerBuilder programming that conforms to the PowerBuilder coding standards.

During the coding process, utilize the Code Insight tool in PowerServer Toolkit to make sure the new code does not contain unsupported features.

While you develop a new PowerBuilder application, keep in mind one of the more recent and stringent standards, the Section 508 Amendment to the Rehabilitation Act. Refer to the whitepaper available at http://www.sybase.com/content/1035234/PB_508_Compliance_wp.pdf for details of the standard.

Step 3: Migrate the PowerBuilder application to the Web with PowerServer by following the migration process described in [Chapter 2, Migration Process](#).

Most of the time and effort in PowerServer Web RAD methodology is spent working in the PowerBuilder IDE to code the desktop PowerBuilder application.

1.4.1 PowerServer-standard PowerBuilder coding

PowerBuilder coding standards are recommended ways to write PowerBuilder code so that it can be recognized and translated into corresponding Web languages (HTML, JavaScript and XML) by PowerServer.

The PowerBuilder coding standards are laid out in Supported PB Features for PowerServer Web. When coding the new PowerBuilder application, you should refer to this document to ensure the PowerBuilder coding standards are followed.

After writing the new PowerBuilder application, use the PowerServer Toolkit integrated into the PowerBuilder IDE to automatically convert the PowerBuilder application to the Web. Some further modifications to the PowerBuilder application may be needed in order to make the application fully functional. Finally, convert the PowerBuilder application and deploy the Web application to servers for production use. For detailed instructions on Web deployment,

refer to Chapter 5, *Deploying PowerBuilder Applications in PowerServer Toolkit User Guide*.

2 Migration Process

2.1 Introduction

Instead of providing detailed instructions on each step, this chapter emphasizes the overall process. Each section may reference other places within this document or other PowerServer documents and support information.

If your goal is to write a completely new PowerBuilder application and deploy it to the Web, first refer to [Chapter 1, *Web RAD with PowerBuilder and PowerServer*](#), and then refer to instructions in this chapter for migrating the new PowerBuilder application to the Web.

The following figure illustrates the general process of deploying an existing PowerBuilder application to the Web. The process is divided into two phases:

- Phase One: Planning - Evaluating strategies for your Web migration objective.

Step 1: [Install required software](#)

Step 2: [Understand the general limitations of the PowerServer solution](#)

Step 3: [Define migration objective](#)

- Phase Two: Implementation - Implementing your Web migration objective.

Step 4: [Upgrade the original PowerBuilder application](#)

Step 5: [Prepare the target PowerBuilder application](#)

Step 6: [Pre-configure for Web application](#)

Step 7: [Remove or workaround unsupported features](#)

Step 8: [Enhance the application with Web or PowerServer features](#)

Step 9: [Trial deployments](#)

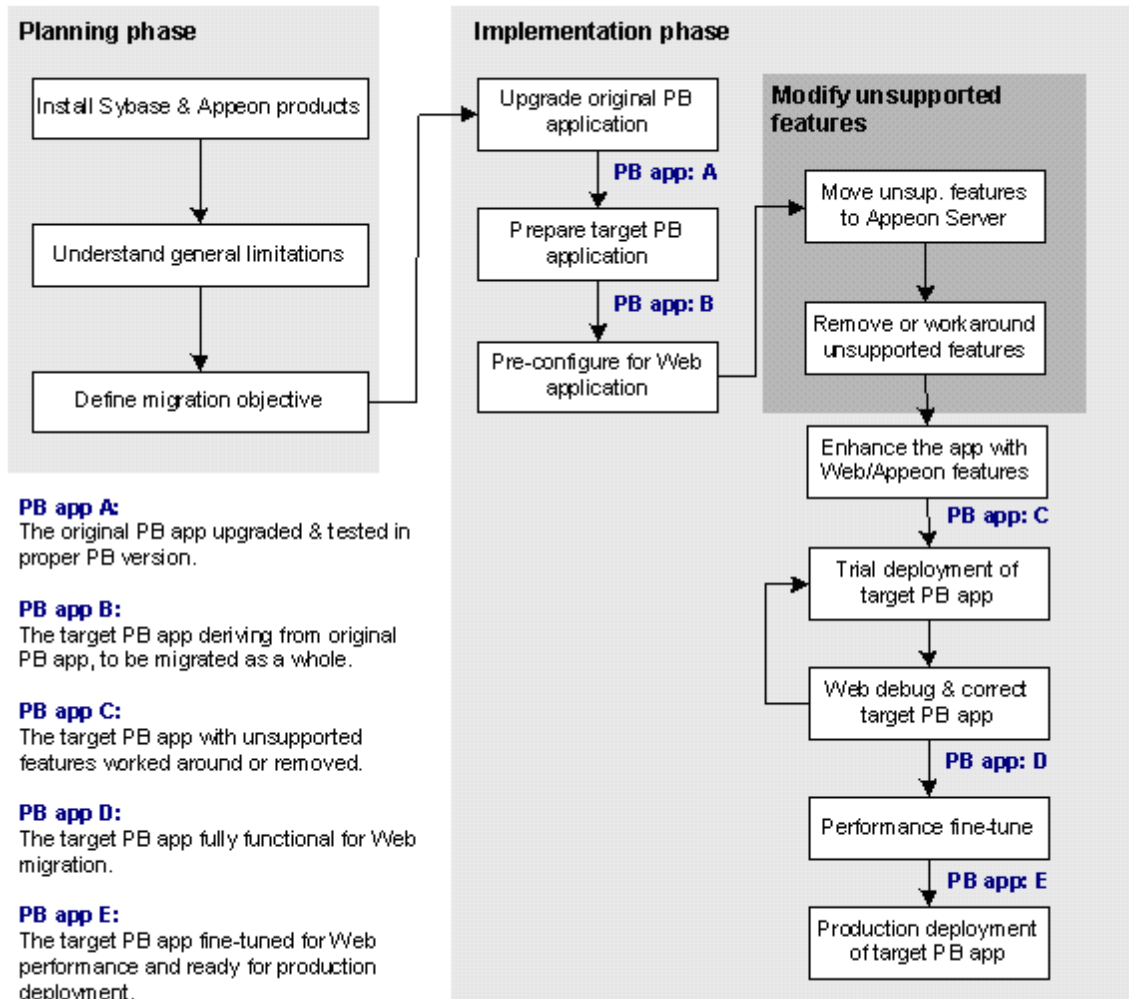
Step 10: [Debug Web application](#)

Step 11: [Performance fine-tune](#)

Step 12: [Go-live deployment](#)

Step 13: [Go live](#)

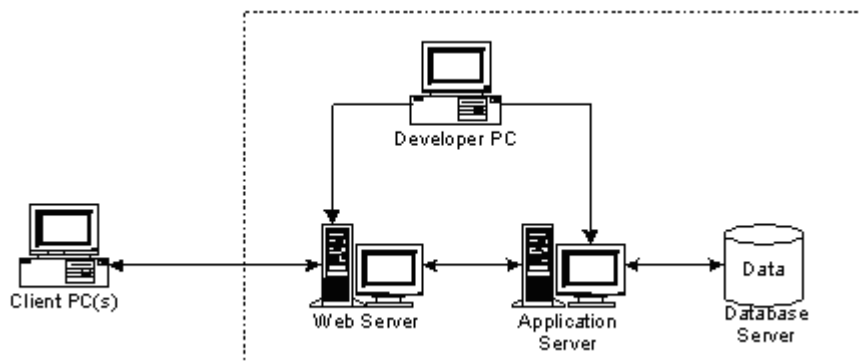
Figure 2.1: PowerServer Web Migration Process



2.2 Installing required softwares

The first step in the Web migration process is to set up the software environment.

Figure 2.2: PowerServer works in a standard n-Tier Web architecture



The following softwares should be installed to support the architecture:

- Windows Server 2019/2016/2012 R2 or Windows 10/8/8.1/7
- Microsoft Internet Explorer 11, Microsoft Edge, Google Chrome, Mozilla FireFox, or Opera
- PowerBuilder
- Database that your application uses
- Application Server (any of the following types):
 - Microsoft .NET Framework & Microsoft IIS
 - Oracle WebLogic
 - IBM WebSphere
 - TmaxSoft JEUS
 - JBoss
- PowerServer, which includes:
 - PowerServer (Which includes AEM, PowerServer Status Monitor, and PowerServer Web Component)
 - PowerServer Toolkit (Windows only)
 - PowerServer Web Component
 - PowerServer Help (Windows only)

For detailed instructions on installing the required softwares, refer to Installation Guide for .NET.

2.3 Understanding the general limitations of the PowerServer solution

There are some general limitations in the migration capabilities of PowerServer. These limitations affect how well a PowerBuilder application is suited for Web migration with PowerServer. It is very important to be aware of these limitations in advance.

Table 2.1: Limitations of the PowerServer solution

Limitation	What it limits...	Try to work around the limit by...
Coding style	We recommend not using certain coding styles such as generic code.	Avoiding the coding styles that do not work well with PowerServer based on advices in Supported PB Features for PowerServer Web.
Database	PowerServer supports the following database types:	Always using the supported database types.

Limitation	What it limits...	Try to work around the limit by...
	SAP ASE, SAP SQL Anywhere, Microsoft SQL Server, Oracle, IBM DB2, SAP IQ, MySQL, Informix, or PostgreSQL For details, see Installation Guide for .NET.	
Unsupported features	The Supported PB Features for PowerServer Web provides a list of supported and unsupported features.	Working around unsupported features by following the Chapter 5, <i>Workarounds for Unsupported Features</i> in <i>Workarounds & APIs Guide</i> .

For more information regarding the limitations, we recommend you read Chapter 1, *Basic Requirements and Recommendations* in *Supported PB Features for PowerServer Web*.

2.4 Defining migration objective

There are three typical migration objectives:

- Convert an existing PowerBuilder application to the Web.
- Extract a portion from an existing PowerBuilder application, such as a set of Windows and key functions, and migrate that portion to the Web.
- Migrate some DataWindows from the original PowerBuilder application to the Web.

2.4.1 Defining migration objective for non-PFC application

Step 1: Calculate the total size of the PBLs that make up the original PowerBuilder application.

Step 2: Identify the major unsupported features in the existing PowerBuilder application.

Step 3: Briefly assess the amount of work needed to modify the unsupported features in the existing PowerBuilder application. The assessment standards are as follows:

- The more complex the PowerBuilder application is, the more difficult it is to modify its unsupported features. The complexity can be characterized by advanced coding techniques including deep inheritance, etc.

If there is a large amount of business logic on the Client-side (e.g. in Windows and behind controls) and the logic is complex (e.g. it fires 20 events before the business rule is completed), then your application will most likely benefit from moving the business logic into n-Tier NVOs. This takes additional work and needs to be factored in.

- The more unsupported features the PowerBuilder application has, the more effort it takes to modify them.

[Section 2.8, “Modifying unsupported features”](#) provides methods on working around unsupported features.

Step 4: Decide the migration objective:

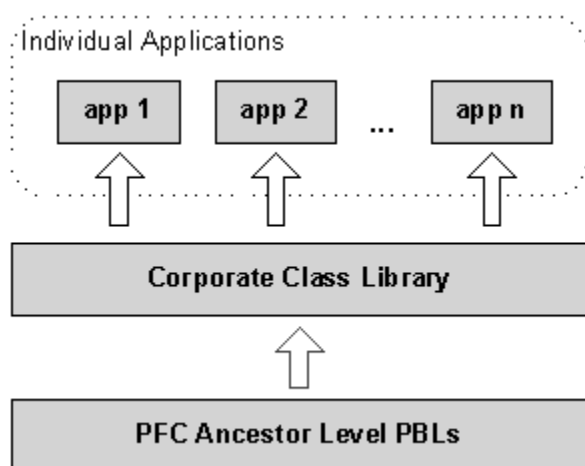
- If it is imperative that your company move business to the Web quickly, select the first migration objective and deploy the entire PowerBuilder application to the Web; otherwise select the second or third migration objective.
- Based on the assessments in the previous step, if substantial effort is required to modify the unsupported features in the original PowerBuilder application, you may want to consider converting part of the application to the Web (objective 2). Regardless of the approach taken, it will always take less work to deploy a PowerBuilder application to the Web with PowerServer than to rewrite it with J2EE/.NET, as PowerBuilder is highly productive, especially for your PowerBuilder team.
- The third possible objective is to deploy only your application DataWindows to the Web (separately from the application). This involves building a much smaller new PowerBuilder application and reusing the existing DataWindows. This is a logical solution when some key reporting functionality or data needs to be available on the Web in a very short time and it would otherwise take significant effort to deploy the entire application.

2.4.2 Defining migration objective for PFC application

2.4.2.1 Corporate PFC architecture

The following diagram illustrates the typical architecture for PFC applications.

Figure 2.3: Typical PFC applications architecture



The Corporate Class Library extends the PFC Ancestor Level, and contains customized and enhanced functionality for reuse by various applications within the entire corporate class library. The PFC Extension Level PBLs are included in the Corporate Class Library, and utilized to extend the PFC Ancestor Level. For example, corporate analysts may add intermediate extension level(s) between the PFC Ancestor Level and the Extension Level, or they can use the existing PFC Extension Level.

Depending on the business need and corporate complexity, sometimes the PFC Ancestor Level is extended further into several layers containing corporate departmental standards and business rules. These several layers are all included in the Corporate Class Library.

2.4.2.2 Defining migration objective progressively

If your PFC application's architecture has a large corporate layer, we recommend that you first aim to migrate a small application to the Web or portion of your existing PFC application to get your framework working on the Web, then expand to deploy a larger portion of your application or the entire application. This progressive approach prevents you from being confronted by many problems at the same time; otherwise, you may be overwhelmed with unsupported features that exist in both the application and the Corporate Class Library.

Step 1: Deploy a small PFC application that consists of the following three parts:

- A small amount of application-specific logic separated from the Individual Application.
- Corporate Class Library
- PFC Ancestor Level

In this step, the application-specific logic should be kept small and simple so you can focus on migrating the PFC Ancestor Level and the Corporate Class Library onto the Web. The PFC Ancestor Level and Corporate Class Library are more important because they are the base classes for all corporate applications.

Step 2: Gradually increase the size and complexity of Individual Applications, and change your focus to fixing the Individual Applications for Web deployment.

2.5 Upgrading the original application

PowerServer 2020 supports PowerBuilder 2019 R2. Any PowerBuilder source code that is intended for deployment to the Web **MUST** first be upgraded to be 100% compatible with the required PowerBuilder version. If your application is not upgraded, you will encounter runtime errors with the deployed Web application.

2.5.1 Upgrading obsolete PBLs

When loading PBLs from a previous version (e.g. PowerBuilder 5) into a later version, many error messages may appear in the PowerBuilder Output window. Since inheritance propagates are an issue from the ancestor object to all its descendant objects, some of the errors reported may actually come from a single source.

One common type of error found when upgrading PowerBuilder applications is string corruption. To correct these errors, edit the source code (by selecting *Edit Source* from the context menu on the error item displayed in the Output window). Errors will disappear when performing a Full Build to the application target.

2.5.2 Upgrading PFC applications

PFC applications intended for PowerServer Web migration **must** be upgraded to be compliant with PFC 9 or later. Some legacy PFC applications are based on the PFC 5.0 PBLs that have been migrated to the newer version, and you should upgrade those applications to use the new PFC 9 or later PBLs.

The PFC PFCOLD.PBL library contains obsolete objects that are not supported for PowerServer Web migration. If the original PowerBuilder application is a PFC application that uses objects in the PFCOLD.PBL library, you should remove the objects when upgrading the application to PFC 9 or later.

2.6 Preparing the target application

In this step, you need to process the original PowerBuilder application to create the target application. This work focuses on transforming the original PowerBuilder application into a target application that complies with PowerServer architectural guidelines set in Supported PB Features for PowerServer Web.

2.6.1 Special processing required for PFC applications

PowerServer supports most PFC features. As such, under most situations the existing PFC code can also work under the PowerServer environment without any modifications. However, there are a few features that are not supported. For that handful of unsupported features, PowerServer would devise a workaround or alternate implementation of those features. Then, the PFC framework would be enhanced with any auto-sensing feature that would detect whether the PFC framework is being run as Client/Server or as PowerServer Web. If Client/Server, then for those features that PowerServer does not support it will execute the original Client/Server implementation. If Web, then only for those features that PowerServer does not support it will execute the modified implementation. Refer to [Processing PFC Applications](#) for detailed instructions on how to modify the unsupported PFC features.

After modifying these unsupported features, you can deploy the PFC application just as how you deploy a normal PowerBuilder application.

2.6.2 Processing application based on migration objectives

The following table shows what processing tasks you need to perform based on your migration objective.

Table 2.2: Process application based on migration objectives

If you plan to migrate...	Do the following...
Entire original application	<ol style="list-style-type: none"> 1. Test the original PowerBuilder application and correct any applications functionality or user interface problems. Note: This step is for detecting and removing problems that may have existed in legacy PowerBuilder applications or problems caused by upgrading legacy PowerBuilder applications. 2. Perform a full build and optimize the original PowerBuilder application in PowerBuilder IDE.
A portion of the original application	<ol style="list-style-type: none"> 1. Extract the desired portion from the original PowerBuilder application into a new PowerBuilder application target. 2. Test the application to ensure there are no bugs and that it functions as expected.

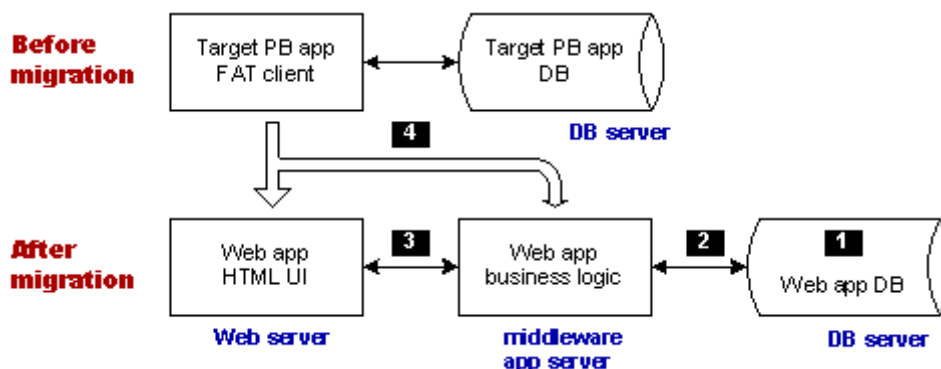
If you plan to migrate...	Do the following...
	3. Perform a full build and optimize this "extracted" portion of your application in the PowerBuilder IDE.
Some DataWindows in the original application	<ol style="list-style-type: none"> 1. Create a new PowerBuilder application target in the same workspace that holds the original PowerBuilder application. 2. Move the desired DataWindow objects from the original PowerBuilder application to the new PowerBuilder application target. 3. Add Windows, Menus, and a general UI for the application. Code simple business logic to the new PowerBuilder application in order to make it fully functional. 4. Test the new PowerBuilder application and correct any problems with its functionality and user interface. 5. Perform a full build and optimize the new PowerBuilder application in the PowerBuilder IDE.

2.7 Pre-configuring for the Web applications

2.7.1 Why is pre-configuration necessary

The following illustration of the PowerServer migration solution helps explain why these pre-configurations are necessary.

Figure 2.4: Before and after PowerServer Web migration



Before the Web migration, the heavy client of the target PowerBuilder application interacts with the Database Server. If the application is distributed, the client also interacts with the application server that hosts and executes important business logic for the target application.

After the Web migration, the Web application has an n-Tier Web architecture involving PowerServer as the middleware to hold business logic. This structure is called Browser Server, because the Web application is accessed via Web browsers such as Internet Explorer.

Making the prospective Web application function in an n-Tier Web environment requires more than just hosting the HTML user interface and the business logic on the servers.

2.7.2 Four pre-configuration tasks

As indicated by the numbers in [Section 2.7.1, “Why is pre-configuration necessary”](#), four pre-configuration steps are necessary:

Task #1: Manually set up the Web application database server (non-PowerServer task). This task is no different from setting up the PowerBuilder application database server.

Task #2: Manually set up communication between the back-end DB server and the middleware PowerServer. Specifically, set up JDBC data sources (non-PowerServer task).

Refer to Chapter 3, *Database Connection Setup* in *PowerServer Configuration Guide for .NET* or in *PowerServer Configuration Guide for J2EE* for instructions on this task.

Task #3: Set up communication between the Web application UI (hosted on the Web server) and the Web application business logic (hosted on PowerServer). This is done via the Web server configuration for the application (PowerServer task). This task is not required if PowerServer and Web server are set up on the same machine, because the communication will be automatically set.

Refer to *Web Server Configuration Guide* for instructions on this task.

Task #4: Configure PowerServer Toolkit so that it has sufficient information to automate the task of converting and deploying the target PowerBuilder application (PowerServer task).

Refer to *PowerServer Toolkit User Guide* for instructions on this task.

2.8 Modifying unsupported features

Some PowerBuilder features cannot be supported because:

1. There are architectural differences between desktop applications and Web applications. The functionality in a Web application is often limited with regards to desktop applications.
2. The current version of PowerServer does not support every PowerBuilder programming feature.

Unsupported features, if not modified, will be commented out in the generated Web files. The code that contains the unsupported features and other code that is dependent on those unsupported features will stop working.

This step involves modifying the unsupported features that have some functional impact on the running of the application. Some cosmetic features, such as "Border" property, can be ignored during the modification process if they will not affect the application.

2.8.1 How to identify unsupported features

Three primary sources of information can guide you in identifying PowerServer-unsupported features in the PowerBuilder applications:

- **Unsupported Features Analysis (UFA) report** - This tool is included in the PowerServer Toolkit. It scans the PowerBuilder application for its unsupported features and assists you in making changes to unsupported features. For instructions on how to use the UFA, refer to Chapter 4, *Using UFA Tool* in *PowerServer Toolkit User Guide*.

- Code Insight - This tool is also included in the PowerServer Toolkit toolbar. It helps you directly identify unsupported code in the PowerBuilder painter. For instructions on how to use the Code Insight, refer to Chapter 11, *Developing with Code Insight in PowerServer Toolkit User Guide*.
- Supported PB Features for PowerServer Web - This Help file is a searchable HTML file that can be launched by clicking the PowerServer Help button in the PowerServer Toolkit. It lists the supported and unsupported features in details and provides recommendations for writing convertible PowerBuilder code following the PowerServer coding standards.

2.8.2 Feature modification methods

You can modify unsupported code by following the instructions in Chapter 5, *Workarounds for Unsupported Features in Workarounds & APIs Guide*. The Guide provides examples of some common unsupported features and ways to work around them.

The following modification method is included in *Workarounds & APIs Guide*, and worth highlighted here because of its importance:

- (Windows only) Using the Apeon workaround for distributed DataWindows.
Apeon Workaround PBL provides two objects *apeondatawindow* and *apeondatastore*, and four functions *GetFullState*, *SetFullState*, *GetChanges* and *SetChanges*, for supporting distributed DataWindows.

2.9 Enhancing the application with Web or PowerServer features

This is an optional step. Some enhancement features can be automatically included in the deployed application, but some features require you to make changes in the PowerBuilder application first. The following lists the key enhancement features that you should add in the PowerBuilder application to make them effective in the deployed application:

- PowerServer open interfaces that can be called to manage all the applications run at PowerServer
- Apeon client functions that can be called to get client information or enable Apeon DataWindow menu
- Integration with other applications via CommandLine argument, or HyperLink controls

[Chapter 4, *Enhancing an application with Web or PowerServer features*](#) provides you instructions on how to implement a number of Web or PowerServer features in the deployed application.

2.10 Trial deployments and debugging

Having completed the previous steps, the target PowerBuilder application is ready for the first trial deployment.

Trial deployments are the intermediate deployments before the final go-live deployment. For detailed instructions on deployment and running of PowerServer Web applications, refer to the PowerServer Toolkit User Guide.

2.10.1 Special deployment steps for distributed applications

Special deployment steps are required for distributed applications with or without distributed DataWindows.

2.10.2 Debugging deployed applications

There may be issues in the trial deployment. For example, a feature in the Web application may work differently from the target PowerBuilder application or not work at all. These issues are usually caused by unsupported features or known issues (documented in the *PowerServer Release Bulletin*). Since UFA cannot detect every single different feature between Client/Server application and Browse/Server application, it is possible that your application contains an unsupported feature.

It is essential to debug the Web application and find the cause of the problem. You can use PowerServer Debugger to debug a PowerServer Web application. For detailed instructions, refer to Chapter 6, *Debugging PowerServer Web applications* in *PowerServer Toolkit User Guide*.

After modifying the target PowerBuilder application according to the debugging result, perform another trial deployment to see if the Web errors still exist in the Web application. The "Trial deployment -> Debug Web application" process may need to be repeated many times before the Web application functions properly.

2.11 Fine-tuning the runtime performance

Given the architectural differences between Client/Server applications and Web applications, each of these has advantages and disadvantages. Typically, Web applications offer better server scalability, while Client/Server applications provide a superior user experience (i.e. rich GUI) and better runtime performance.

PowerServer Web applications offer the rich PowerBuilder GUI with the scalability of n-Tier Web architecture. The rich PowerBuilder GUI ensures that the superior user experience and high user productivity is preserved. The server scalability is comparable to J2EE applications or .NET applications, if not better, since PowerServer moves more processing from the server to the Client. The PowerServer Performance Tuning Guide has laid out the PowerBuilder programming features and coding styles that lead to poor Web performance, and it provides suggestions/workarounds. Refer to the PowerServer Performance Tuning Guide for more information.

2.12 Go-live deployment

Go-live deployment deploys the target PowerBuilder application to your production servers.

Check the following settings in the PowerServer Toolkit Application Profile **before** starting the final deployment to your production servers:

- Set the Web file generation mode to Release - This encrypts the JavaScript code in the Web application, thereby protecting your organizations business rules and intellectual property.
- Turn off all Report options - This will reduce the size of Web files and boost the runtime performance of the deployed Web application.

- Check that the required database types are properly configured and selected.
- Check that the required additional files (such as INI files, .NET/COM components, image files, DLL/OCX files) are properly configured or selected.

Refer to the PowerServer Toolkit User Guide for detailed instructions on how to perform application deployment.

Check the following settings in AEM **after** performing the final deployment to your production servers:

- Configure and ensure that transaction objects and data sources in AEM can be successfully connected.
- Check that Run Mode for the go-live application is set to Normal Mode.
- Check Configuration Summary for the go-live application to ensure they apply to the go-live environment, such as DLL/OCX configurations, 32-bit/64-bit IE compatibility etc.
- Change the AEM default username and password as appropriate to ensure the security of AEM

Refer to PowerServer Configuration Guide for .NET or PowerServer Configuration Guide for J2EE for detailed instructions on how to configure AEM.

2.13 Go live

Go-live is the last step in the Web migration process for the application to go live successfully without serious problems, it is strongly recommended that you have done the following tasks in all related areas to reduce the likelihood of go-live problems:

Unsupported Features:

- Ensure there are no unsupported features in the application. At a minimum, the UFA report should not list any unsupported features.

Performance fine-tune:

- Read and follow the best practices defined in the PowerServer Performance Tuning Guide.

Functionality:

- Verify all application functionality is working properly. Functionality should be tested on all the browser version(s) that will be supported. For example, IE 7 32-bit, IE 8 32-bit, IE 8 64-bit etc. Please note: if the application uses third-party DLLs/OCXs/OLEs then 64-bit version of those files need to be packaged in order for the Web application to run correctly on 64-bit browsers.

Hardware/Software configuration:

- Select appropriate hardware to install PowerServer. PowerServer can give some suggestions based on the # of users, but in general use the most powerful server that customer has budget for.

- Configure IIS settings properly. PowerServer can give some suggestions based on the # of users and type of hardware that will be used.
- Understand the Web browser and system configuration/settings requirements and ensure that all users are aware of these requirements and that their browser settings are correct. If third-party DLLs/OCXs/OLEs are used the Web browser and system settings will need to be significantly changed, so if possible for ease of deployment and user satisfaction it is most preferred to avoid any third-party DLLs/OCXs/OLEs.

Performance and scalability:

- Verify the application response time over "realistic" network connection. What we mean by realistic is to use the same network connection (bandwidth and latency) that the average user will use. Test all application functionality to ensure that the application performs well.
- Verify the application scalability. There are two ways to do this. One is to use a scalability testing tool such as LoadRunner. The other option (and more preferred) would be to do a partial beta test. Essentially, get a group of users together to simultaneously use the system to see if any problems. The problems you are looking for is significant degradation in performance when multiple users on the system and/or system instability (e.g. IIS or database stops responding or crashes).

Beta test and controlled roll-out:

- Perform a beta test. Roll out the application to a smaller group of real users to see if any major problems. Expectations should be set with the users that this is a "beta" not a "live" situation. The users should be prepared that problems are likely to happen and they should be able to tolerate downtime of the system for several days or longer to correct any problems that are found.
- Perform a controlled roll-out. After the beta test has run for sufficient time with no major problems then increase the # of users.

Now go live! After the controlled-rollout has run for sufficient time with no major problems then go live.

3 Migration FAQ

3.1 How do I rapidly build a new Web application with PowerServer?

Step 1: Create a new application.

Quickly create a new PowerBuilder Client/Server application workspace in your PowerBuilder IDE.

Step 2: Code for the application.

Enable the PowerServer Code Insight tool that is included in PowerServer Toolkit, and then write code. With the Code Insight, you can get instant information on whether the new code is supported by PowerServer.

Step 3: Migrate the application automatically with Application Deployment Wizard.

After previous steps, your PowerBuilder application now contains features that are 100% supported by PowerServer. You do not need to modify any unsupported features. You can easily migrate your Client/Server application to Browser/Server application with the deployment wizard in PowerServer Toolkit toolbar.

3.2 Does PowerServer support every PowerBuilder feature?

The current version of PowerServer does not support all PowerBuilder syntax. Unsupported PowerBuilder syntax falls into two categories:

Category 1: PowerBuilder features that are Client/Server architecture specific, and cannot be implemented in n-Tier Web applications.

Unsupported features in this category must be reworked for n-Tier architecture.

Category 2: PowerBuilder features that are currently unsupported by PowerServer, but will possibly be supported in the future.

[Section 2.8, “Modifying unsupported features”](#) gives you high-level instructions on how to modify unsupported features. Chapter 5, *Workarounds for Unsupported Features in Workarounds & APIs Guide* provides examples on some common unsupported features and ways to work around them.

3.3 Do PowerServer-deployed Web applications support external resources?

PowerServer supports using external resources in Web applications, such as INI and image files, OCX and OLE controls, etc.

3.4 Why classify my application into types?

The PowerBuilder-to-Web process can be simple or complex depending on the type of application being migrated. PowerServer provides guidelines for different types of applications. Once you have identified the type of application, follow the specific guidelines to get your application on the Web even faster.

3.5 What are the different application types?

PowerBuilder applications can be classified as one of three types.

Type 1 applications meet the following requirements and can be automatically deployed to the Web:

- The application does not contain any functionality that cannot be implemented on the Web.
- The application does not contain any unsupported features.
- These types of applications tend to be below 50 MB (application PBLs including Framework).

Type 2 applications contain some functionality that cannot be implemented on the Web, and/or some unsupported features. The size of the application may be large, and/or the user objects in the application may have complex interdependencies. Modify the objects or code in the application and make it compliant with Supported PB Features for PowerServer Web.

Type 3 applications contain functions that cannot be implemented on the Web but are critical to the functionality of the application and have very complex frameworks that do not align well with PowerServer supported features, such as PFC-based applications. Depending on the business requirements and project scope, the developer can leverage the RAD capabilities of PowerBuilder to get an application on the Web much faster than a typical J2EE or .NET rewrite while preserving the rich PowerBuilder GUI.

3.6 What are the recommendations for converting the different application types?

The following are recommendations for handling the different types of applications.

Type 1 applications: This type is ready to be migrated. There is no additional work to do. PowerServer reads the PowerBuilder PBLs and generates the n-Tier Web application. The Web application will have DataWindows, business logic, and UIs identical to the original PowerBuilder application.

Type 2 applications: The developer needs to remove the unsupported features and Client Server specific functions from the application that cannot be implemented on the Web to ensure that the Web deployed application functions in the same manner as the PowerBuilder application. If the application is large and complex, PowerServer suggests splitting it into smaller applications to simplify the debugging of the application. Remove as many minor and unnecessary features as possible. Simplify the code and object interdependencies to make the application more straightforward. The DataWindows and a good portion of the PowerBuilder source code can be utilized. The amount of PowerBuilder source code that can be reused depends on the actual application's complexity and the user's requirements.

Type 3 applications: These generally cannot be migrated to the Web as-is, but the DataWindows and other useful objects can be exported and used to rapidly build a more straightforward PowerBuilder application using standard PowerBuilder programming.

The ultimate aim of the guidelines is to help you modify the PowerBuilder application and make it well supported by PowerServer. The existing DataWindows may be used. Depending on the application type, modify a fraction, a part, or all of the PowerBuilder source code.

3.7 What are the basic requirements for rewriting complex applications?

The Supported PB Features for PowerServer Web outlines a set of requirements and recommendations that should be followed. See Supported PB Features for PowerServer Web for more information. Essentially, you must re-code portions of the application to make the application more straightforward (by, for example, removing complex object interdependencies and reducing the number of DataWindows and DataStores in a given Window).

3.8 When would I need to modularize my application?

If the application is a Type 2 or Type 3 application, you may need to split the application into smaller applications and deploy them separately depending on the actual migration requirements.

3.9 What are the benefits of modularizing my application?

The following are advantages of splitting an application into smaller applications:

- Smaller applications have better performance when deployed to the Web.
- It is easier and more efficient to debug smaller applications and rework the unsupported features.
- It allows several developers to work on one project simultaneously and improves developer productivity because one or more developers can specialize on particular modules. The modules can each be deployed to the Web and linked together through a unified HTML interface.

3.10 What are the basic principles for modularizing an application?

The following are the basic principles for splitting an application into smaller applications (modularizing):

The functionality of each small application should be independent from the others. Each application to be deployed should be able to execute an independent and practical task. Following this principle, it is better to integrate closely related functional points in a small application and remove the loose object references. For example, to split a purchase order application into smaller parts, the order management logic (module) can be placed in a small application and the relevant supplier information management logic in another small application.

On the basis of the first principle, try to balance the complex objects evenly across all the small applications.

The applications should be as small as possible.

3.11 Can you give an example of the modularization process?

The following example highlights the process of modularization of an application:

A purchase order application needs to be deployed to the Web. It is a large and complex application that provides the following functionalities at a high-level:

- Order Input
- Order Management
- Supplier Management
- Resource Planning
- Reporting

According to business requirements, the most urgent application functions that must be brought on to the Web as soon as possible are Order Input, Order Management, and Reporting.

Based on the above business requirements, the first phase of the conversion process should involve extracting the Order Input, Order Management and Report functionalities from the original application and importing them back into PowerBuilder. This creates a new smaller application.

Using Supported PB Features for PowerServer Web and Apeon Code Examples, you can work around or modify the unsupported source code in the smaller application to bring it into compliance and deploy it to the Web.

4 Enhancing an application with Web or PowerServer features

The following table gives you a general idea on how to enhance applications with Web or PowerServer features.

Table 4.1: Enhance a deployed application with Web/PowerServer features

Type of the Feature	Description of the Feature	How to add it in the deployed application
Typical Web features	<ul style="list-style-type: none"> • Accessibility of the application via URLs or SAP Sybase Enterprise Portal • Firewall compatibility. The Web file transfer is HTTP over port 80. • SSL and digital certificate security • Superior scalability. Typical applications support 60-80 concurrent users per server CPU that maps to 300-500 named/end users per server CPU. • More... 	<p>Such features are automatically added in the deployment application, with no need for any special coding.</p> <p>Refer to Section 4.3, “Loading an application in SAP Sybase Enterprise Portal” for details on how to load the application in Enterprise Portal.</p>
PowerServer open interfaces	Use PowerServer open interfaces to manage PowerServer-deployed Web applications using PowerBuilder code.	<p>The interface(s) should be called in the PowerBuilder application to take effect in the deployed application.</p> <p>Refer to Section 4.1, “PowerServer open interfaces” for details.</p>
Apeon client functions	Use Apeon client functions to get client information or server information, or use Apeon enhanced features in the applications.	<p>The client function(s) should be called in the PowerBuilder application to take effect in the deployed application.</p> <p>Refer to Section 4.2, “Apeon client functions” for details.</p>
Web integration	<ul style="list-style-type: none"> • Provides flexible & open Java, .NET, & Web Services integration • Connectivity to Java/EJB, PB NVO, C/C++ DLL, COM/ActiveX Components on Application Servers • Connectivity to Web Services, J2EE, and .NET 	<p>There are different ways for implementing the integration of a PowerServer Web application with other applications.</p> <p>Refer to Section 4.4, “Single sign-on” and Section 4.5, “Integrating PowerServer Web applications with JSP/ASP” for details.</p>

Type of the Feature	Description of the Feature	How to add it in the deployed application
	<ul style="list-style-type: none"> Connectivity to Messaging Queues (MQSeries, JMS, etc.) 	

4.1 PowerServer open interfaces

PowerServer open interfaces give you the opportunity to manage services provided by PowerServer through PowerBuilder code. You can easily manage PowerServer-deployed Web applications with the open interfaces below:

- `getAllClients`
- `getAllSessions`
- `getSessionByID`
- `getSessionCount`
- `killAllSessions`
- `killSessions`
- `rollbackAllTransactions`

4.1.1 Description of the open interfaces

Refer to Chapter 4, *PowerServer open interfaces* in *Workarounds & APIs Guide* for the syntax of PowerServer open interfaces. Note that you can write code for calling the interface in the PowerBuilder application, but the code does not take effect in the PowerBuilder application; it only takes effect in the PowerServer-deployed application.

4.1.1.1 `getAllClients`

`getAllClients` gets the IP addresses of all client machines which corresponds to the active sessions that are opened for the specified application in the specified PowerServer.

4.1.1.2 `getAllSessions`

`getAllSessions` returns the detail information of active sessions with XML format, which are opened for the specific application in the specific PowerServer.

4.1.1.3 `getSessionByID`

`getSessionByID` returns the detail information of the specified session with XML format.

4.1.1.4 `getSessionCount`

With `getSessionCount` method, you can get the following three types of information.

- The total number of active sessions opened for the specified application in the specified PowerServer.
- The total number of active sessions in a specified PowerServer.

- The total number of sessions opened for the specified application in a PowerServer cluster.

To get the number of sessions in a PowerServer cluster, you need to first configure the cluster in AEM.

By using the `GetSessionCount` method, you can easily get the total number of active sessions in a specified PowerServer using PowerBuilder code and apply the information in other open interfaces such as `KillSession` to manage the sessions. For example, you can first call `GetSessionCount` and then call `KillSession` in the PowerBuilder application to make the deployed application kill all sessions in PowerServer when there are up to 100 active sessions in the server.

4.1.1.5 killAllSessions

`killAllSessions` kills all active sessions in a PowerServer or a PowerServer cluster and rolls back all associated transactions. To kill all sessions in a PowerServer cluster, you need to first configure the cluster in AEM.

4.1.1.6 killSessions

`killSessions` kills the specified session(s) in a PowerServer or a PowerServer cluster and rolls back the associated transactions. To kill sessions in a PowerServer cluster, you need to first configure the cluster in AEM.

4.1.1.7 rollbackAllTransactions

`rollbackAllTransactions` rolls back all transactions in a PowerServer or a PowerServer cluster. To roll back all transactions in a PowerServer cluster, you need to first configure the cluster in AEM.

4.1.2 Applying PowerServer open interfaces in PowerServer-deployed applications

To apply a PowerServer open interface in a PowerServer-deployed Web application, you only need to perform the following two steps.

1. Call the PowerServer open interface in the PowerBuilder application. For how to call PowerServer open interfaces in a PowerBuilder application, refer to Section 4.2, “Calling PowerServer open interfaces via EJB component” in *Workarounds & APIs Guide* or Section 4.3, “Calling PowerServer open interfaces via .NET component” in *Workarounds & APIs Guide*.
2. Deploy the PowerBuilder application to PowerServer the same way you would deploy a normal PowerBuilder application.

4.2 Apeon client functions

Apeon client functions are a set of PowerBuilder functions encapsulated in Apeon workaround PBLs. You can add Apeon client functions in your PowerBuilder applications to achieve the following in PowerServer-deployed applications:

- Get client information, such as the IP address, or the browser type of the client

- Get server information, such as total numbers of the active session on a cluster or a single PowerServer.
- Use PowerServer Web enhanced features, such as Apeon DataWindow menu, PDF print, and LDAP Logon.

4.2.1 Description of Apeon client functions

The following table provides the client functions and their return values. Note that some Apeon client functions are just interfaces provided by the Apeon system; they have empty function bodies and empty return values for the PowerBuilder applications.

Table 4.2: Client functions with their return values

Functions	Return value when executed in PowerBuilder	Return value when executed in PowerServer application
AppeonGetAppeonUserName()	None. This function has no effect in PowerBuilder.	The user name typed into the Apeon Login dialog.
AppeonGetBrowserVersion()	None. This function has no effect in PowerBuilder.	The version of the Internet Explorer browser that runs the current Web application.
AppeonGetClientID()	The unique session identifier for PowerBuilder client.	The unique session identifier for the Web client.
AppeonGetCacheDir()	None. This function has no effect in PowerBuilder.	The cache directory that is used by the current Web application.
AppeonGetClientIP()	The IP address of PowerBuilder client machine.	The IP address of the Web client.
AppeonGetClientType()	"PB"	"WEB" if the application runs on a Web browser, or "MOBILE" if the application runs on a mobile device.
AppeonGetHttpInfo (string attribute)	None. This function has no effect in PowerBuilder.	The HTTP header information from a particular request.
AppeonGetIEHandle()	None. This function has no effect in PowerBuilder.	The Internet Explorer handle for the current Web application.
AppeonGetIEURL()	None.	The URL of the current PowerServer application.

Functions	Return value when executed in PowerBuilder	Return value when executed in PowerServer application
	This function has no effect in PowerBuilder.	
ApeonGetOSType()	The type of the OS that runs the PowerBuilder client application.	The type of the OS that runs the current Web application.
ApeonGetRemainingdays (String as_type, ref string as_error)	None. This function has no effect in PowerBuilder.	The remaining day(s) of license or technical support.
ApeonGetServerType()	None. This function has no effect in PowerBuilder.	Returns 1 if the application runs on a PowerServer that is installed to a J2EE Server (such as WebLogic). Returns 2 if the application runs on a PowerServer that is installed to a Microsoft .NET Framework \IIS.
ApeonIsIn64Browser	None. This function has no effect in PowerBuilder.	Returns "true" if the application runs on a 64-bit IE browser. Returns "false" if the application runs on a 32-bit IE browser.
ApeonLDAPLogon (string as_username, string as_password)	None. This function has no effect in PowerBuilder.	Login username and password for the LDAP server.
ApeonPopMenu (Datawindow adw_dw, Integer nx, Integer ny)	None. This function has no effect in PowerBuilder.	None. At the execution of the function, ApeonDataWindow menu is displayed at the specified position on the specified DataWindow control. ApeonPopMenu has higher priority than ApeonPopMenuOn.
ApeonPopMenuOn (Datawindow adw_dw, Boolean ab_show)	None. This function has no effect in PowerBuilder.	None. At the execution of the function, ApeonDataWindow menu pops up when you right

Functions	Return value when executed in PowerBuilder	Return value when executed in PowerServer application
		click on the specified DataWindow.
AppeonPrint2File (datawindow adw, string asoutpath, string asoutname, long alouttype)	None. This function has no effect in PowerBuilder.	Integer. Returns 1 if it succeeds in printing the specified DataWindow/DataStore as a PDF or an image file of BMP, JPG or GIF format, otherwise, returns -1.
AppeonSwitchRealTimeCalc	None. This function has no effect in PowerBuilder.	Returns 0 (default value) if need to perform real-time calculation. Returns 1 if no need to perform real-time calculation.

4.2.2 Using Appeon Client functions

Although the main purpose of Appeon Client functions is to implement certain functionalities in PowerServer-deployed applications, you must call the functions in your PowerBuilder application to make the functions effective.

To use Appeon Client functions in PowerServer-deployed applications, refer to Section 1.3.2, “Appeon Client Functions” in *Workarounds & APIs Guide* for more detail.

4.3 Loading an application in SAP Sybase Enterprise Portal

SAP Sybase Enterprise Portal (EP) is an open and scalable portal framework that provides an intuitive, secure and customized environment for end-users; rapid development and deployment tools for developers; and an easy-to-use console for administrators. PowerServer-deployed applications can run in SAP Sybase Enterprise Portal or Unwired Accelerator.

4.3.1 Restrictions on supporting Enterprise Portal

PowerServer supports Enterprise Portal 6.1 Info Edition and Enterprise Edition. Be aware of the following restrictions when you load PowerServer Web applications in Enterprise Portal:

- Only ONE PowerServer Web application can be viewed inside one EP Page. Multiple PowerServer Web applications can be viewed within multiple pages. For example, you can have Sales App Demo on one EP Page and Appeon Code Examples on another EP Page and both function correctly.
- Exiting from PowerServer Web applications do not close Internet Explorer.
- Only one response window is allowed for an Internet Explorer browser at a given time.

4.3.2 Tasks required to load the application in Enterprise Portal

You need to perform the following four tasks to get a deployed application loaded in the portal after the application migration:

Task #1: Add the application into a portlet. Create an HTML element in the portlet and assign the URL of the PowerServer-deployed application to the HTML element.

Task #2: Add the application into a page. Create a new page, add the application into it, and approve the application.

Task #3: Add the page into a page group. Create a new page group and add the page created in task #2 into the page group.

Task #4: Create accounts for viewing the application. Create accounts and assign rights to the accounts for running the application.

For step-by-step instructions on conducting each of the preceding tasks, refer to the relevant documentation provided by SAP Sybase Enterprise Portal (available at <http://www.sybase.com/products/archivedproducts/enterpriseportal/technicalsupport>).

4.4 Single sign-on

Single sign-on can be implemented in the PowerServer deployed Web applications using the following two methods:

- Using a server component to manage user information
- Applying command line argument

For detailed steps of using the above methods, refer to Section 6.10, “How to log in the PowerServer Web application with single sign-on” in *Workarounds & APIs Guide*.

4.5 Integrating PowerServer Web applications with JSP/ASP

You can integrate PowerServer Web applications with JSP/ASP applications using the following three methods:

- Applying Apeon CommandParm and Hyperlink features
- Using Internet Explorer frame
- Integration through intermediate n-Tier server

For detailed steps of using the above methods, refer to Section 6.14, “How to integrate PowerServer Web applications with JSP/ASP” in *Workarounds & APIs Guide*.

5 Processing PFC Applications

PowerServer supports most PFC features. As such, under most situations the existing PFC code can also work under the PowerServer environment without any modifications. However, there are a few features that are not supported. For that handful of unsupported features, PowerServer would devise a workaround or alternate implementation of those features. Then, the PFC framework would be enhanced with any auto-sensing feature that would detect whether the PFC framework is being run as Client/Server or as PowerServer Web. If Client/Server, then for those features that PowerServer does not support it will execute the original Client/Server implementation. If Web, then only for those features that PowerServer does not support it will execute the modified implementation.

After modifying these unsupported features, you can deploy the PFC application just as how you deploy a normal PowerBuilder application.

5.1 Auto-Sensing Environment

The general idea of making PFC work for both PowerBuilder and PowerServer is calling a global function to judge the application's running environment. This global function will help to judge whether to run the original PFC code or Apeon compliant code.

The global function "apeongetclienttype" is included in the Apeon Workarounds PBL. You will need to first add the `apeon_workarounds.pbl` library to your application and then call the `apeongetclienttype` global function. For detailed instructions, refer to Chapter 1, *Apeon Workarounds PBL Reference in Workarounds & APIs Guide*.

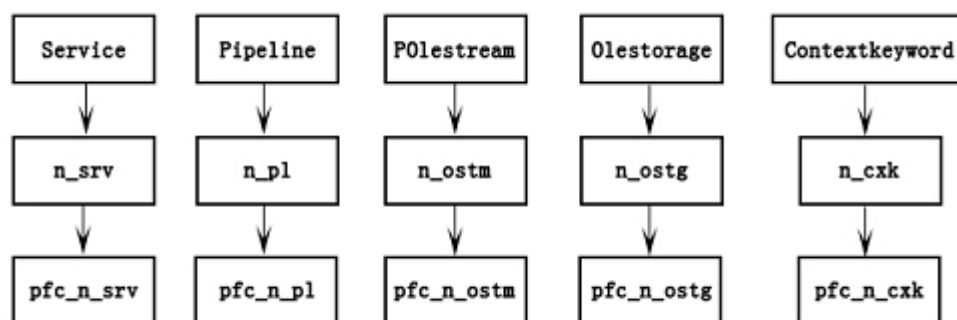
5.2 Suggested Modifications to PFC

The following sections give detailed suggestions for how to modify current PFC codes so that those PowerServer unsupported or modified PFC features work in both PowerBuilder and in PowerServer. Code modifications are only made where absolutely necessary and the "Auto-Sensing Environment" feature will be used to ensure under no circumstance are existing Client/Server users affected by any PowerServer modifications. PowerServer suggested modification will follow the following format:

```
IF apeongetclienttype() = "PB" then
  Original code in PFC
else
  Apeon compliant code in PFC
End if
```

5.3 Unsupported user objects

PFC framework contains the following user objects as unsupported (in the last two lines):

Figure 5.1: Unsupported User objects in PFC

Recommendation: Although these user objects are defined, they are not referred to in PFC framework, therefore you do not need to modify them and you should avoid referring to these objects in your application.

5.4 Unsupported standard class objects

These four standard class objects are unsupported:

- Contextinformation
- Classdefinition
- Scriptdefinition
- Variabledefinition

Recommendation: Avoid referring to the following object functions, this can reduce the impact on the functionality:

- pfcmain.pbl\pfc_w_master\of_setresize
- pfcapsrv.pbl\pfc_n_cst_security\of_setsecurity
- pfcapsrv.pbl\pfc_n_cst_metaclass\of_findmatchingevent
- pfcapsrv.pbl\pfc_n_cst_metaclass\of_findmatchingvariable
- pfcapsrv.pbl\pfc_n_cst_metaclass\of_getancestorclasses
- pfcapsrv.pbl\pfc_n_cst_metaclass\of_isfunctiondefined
- pfcapsrv.pbl\pfc_n_cst_metaclass\of_iseventimplemented
- pfcapsrv.pbl\pfc_n_cst_metaclass\of_iseventdefined
- pfcapsrv.pbl\pfc_n_cst_metaclass\of_isancestorclass

If you really need to use Classdefinition, try to use the following workaround:

Original script:

```
lb_defined = inv_metaclass.of_isFunctionDefined &
(lp_o_tocheck.ClassDefinition, "of_PostUpdate", ls_args)
```

Modified script:

```
If lpo_tocheck.TriggerEvent("pfc_descendant") = 1 Then
  lb_defined = True
Else
  lb_defined = False
End if
```

5.5 Unsupported object properties

Table 5.1: Unsupported object properties that need no modification

Object	Properties	Calls	Suggestion
Application	ToolbarUserControl	4	Need no modification.
	ToolbarText	5	
	ToolbarPopupMenuText	3	
	DwMessageTitle	4	
	DdeTimeOut	3	
OLEControl	ClassShortName	1	Need no modification.
	ClassLongName	1	
Transaction	Lock	3	Need no modification.
	Sqlreturndata	1	
Message	Returnvalue	1	Need no modification.
	Number	1	
	Handle	1	
Connection	Trace	2	Need no modification.
	Options	2	
	Connectstring	2	
GraphicObject	Classdefinition	1	Need no modification.

Table 5.2: Unsupported object properties that need modification

Object	Properties	Calls	Suggestion
Window	Classdefinition	1	<p>Need modification.</p> <p>Calling this property has no impact on the functionality. However, it has impact on the position and size of the control in a DataWindow when the DataWindow is zoomed in or zoomed out. Refer to the detailed information below.</p> <p>DETAILED INFORMATION</p> <p>Location: Of_setresize(boolean) function of pfc_w_master in pfcmain.pbl</p> <p>PowerServer Unsupported Code: Lcd_class = this.ClassDefinition</p> <p>Line Number: 45</p>

Object	Properties	Calls	Suggestion
			<p>Change the code to:</p> <pre> if ApeonGetClientType() = 'PB' then classdefinition lcd_class lcd_class = this.ClassDefinition li_vars = UpperBound (lcd_class.VariableList) For li_v = 1 to li_vars If lcd_class.VariableList[li_v].Name = "width" then & li_origwidth = Integer (lcd_class.VariableList[li_v].InitialValue) If lcd_class.VariableList[li_v].Name = "height" then & li_origheight = Integer (lcd_class.VariableList[li_v].InitialValue) If li_origwidth > 0 And li_origheight > 0 then & Exit Next inv_resize.of_SetOrigSize (li_origwidth, li_origheight) else inv_resize.of_SetOrigSize (this.width, this.height) end if </pre>
PowerObject	Classdefinition	7	<p>Need modification</p> <p>This property is often used for detecting a function of a object.</p> <p>It is suggested to modify the whole code segment, calling a function no matter whether it exists or not. Refer to the detailed information below.</p> <p>DETAILED INFORMATION</p> <p>Location 1: Of_validation(powerobject) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: Lcd_class = this.ClassDefinition</p> <p>Line Number: 123</p> <p>Change the code to:</p> <pre> if ApeonGetClientType() = 'PB' then lb_defined = inv_metaclass.of_isFunctionDefined & (lpo_tocheck.ClassDefinition, "of_validation", ls_args) else lb_defined = True End if </pre> <p>Location 2: Of_updatespending(powerobject) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: Lb_defined = inv_metaclass.of_isFunctionDefined</p>

Object	Properties	Calls	Suggestion
			<p>(lpo_tocheck.ClassDefinition, "of_UpdatesPending", ls_args)</p> <p>Line Number: 130</p> <p>Change the code to:</p> <pre data-bbox="683 450 1398 680"> if ApeonGetClientType() = 'PB' then lb_defined = inv_metaclass.of_isFunctionDefined & (lpo_tocheck.ClassDefinition, "of_UpdatesPending", ls_args) Else lb_defined = True End if </pre> <p>Location 3: Of_updateprep(powerobject) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: Lb_defined = inv_metaclass.of_isFunctionDefined (lpo_tocheck.ClassDefinition, "of_UpdatePrep", ls_args)</p> <p>Line Number: 122</p> <p>Change the code to:</p> <pre data-bbox="683 1084 1398 1314"> if ApeonGetClientType() = 'PB' then lb_defined = inv_metaclass.of_isFunctionDefined & (lpo_tocheck.ClassDefinition, "of_UpdatePrep", ls_args) else lb_defined = True End if </pre> <p>Location 4: Of_update(powerobject,boolean,boolean) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: Lb_defined = inv_metaclass.of_isFunctionDefined (lpo_tocheck.ClassDefinition, "of_Update", ls_args)</p> <p>Line Number: 145</p> <p>Change the code to:</p> <pre data-bbox="683 1675 1398 1906"> if ApeonGetClientType() = 'PB' then lb_defined = inv_metaclass.of_isFunctionDefined & (lpo_tocheck.ClassDefinition, "of_Update", ls_args) Else lb_defined = True End if </pre> <p>However, with the suggested changes made, only DataWindows placed directly on windows would update. For DataWindows placed on tab pages to</p>

Object	Properties	Calls	Suggestion
			<p>update correctly, add the code to the Case statement in the function to call of_update() on UserObject and Tab controls to make updates on tab pages work correctly.</p> <p>Location 5: Of_postupdate(powerobject) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: Lb_defined = inv_metaclass.of_isFunctionDefined (lpo_tocheck.ClassDefinition, " of_PostUpdate ", ls_args)</p> <p>Line Number: 123</p> <p>Change the code to:</p> <pre data-bbox="683 752 1394 949"> if ApeonGetClientType() = 'PB' then lb_defined =inv_metaclass.of_isFunctionDefined & (lpo_tocheck.ClassDefinition, "of_PostUpdate", ls_args) else lb_defined = True End if </pre> <p>Location 6: Of_isselfupdatingobject(powerobject) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: Lb_defined = inv_metaclass.of_isFunctionDefined (apo_control.ClassDefinition, "of_Update", ls_args)</p> <p>Line Number: 60</p> <p>Change the code to:</p> <pre data-bbox="683 1317 1394 1514"> if ApeonGetClientType() = 'PB' then lb_defined =inv_metaclass.of_isFunctionDefined & (apo_control.ClassDefinition, "of_Update", ls_args) else lb_defined =True End if </pre> <p>Location 7: Of_accepttext(powerobject,boolean) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: Lb_defined = inv_metaclass.of_isFunctionDefined (lpo_tocheck.ClassDefinition, "of_AcceptText", ls_args)</p> <p>Line Number: 128</p> <p>Change the code to:</p> <pre data-bbox="683 1917 1394 1998"> if ApeonGetClientType() = 'PB' then lb_defined = inv_metaclass.of_isFunctionDefined </pre>

Object	Properties	Calls	Suggestion
			<pre>& (lpo_tocheck.ClassDefinition, "of_AcceptText", ls_args) else lb_defined = True End if</pre>
DataWindow control	Hsplitscroll	1	<p>Need modification.</p> <p>Refer to the detailed information below.</p> <p>Location: Of_position(dragobject,boolean) function of pfc_n_cst_dropdown in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: If ldw_object.hsplitscroll</p> <p>Line Number: 237</p> <p>Change the code to:</p> <pre>if ApeonGetClientType() = 'PB' then li_hsplit = Integer (ldw_object.Describe ("DataWindow.HorizontalScrollSplit")) li_hpos1 = Integer (ldw_object.Describe ("DataWindow.HorizontalScrollPosition")) li_hpos2 = Integer (ldw_object.Describe ("DataWindow.HorizontalScrollPosition2")) if ldw_object.hsplitscroll then if li_hsplit > 4 and li_pointerx > li_hsplit then li_hpos = li_hpos2 - li_hsplit - of_GetSystemSetting (DW_HSPLITBAR_WIDTH) else li_hpos = li_hpos1 End if else li_hpos = li_hpos1 End if else li_hpos = 0 End if</pre>

5.6 Unsupported object functions

Table 5.3: Unsupported object functions that need no modification

Object	Functions	Calls	Suggestion
NonVisualObject	GetContextService	1	Need no modification.
Window	Settoolbar	5	Need no modification.
	Gettoolbar	3	
	Gettoolbarpos	2	
	Settoolbarpos	2	
SingleLineEdit	Canundo	1	Need no modification.
RichTextEdit	Canundo	1	Need no modification.
OLEControl	UpdateLinksDialog	1	Need no modification.

Object	Functions	Calls	Suggestion
	PasteSpecial	1	
	Paste	1	
	Cut	1	
	Copy	1	
EditMask	Canundo	1	Need no modification.
DataStore	ReselectRow	1	Need no modification.

5.7 Unsupported object events

Table 5.4: Unsupported object events that need no modification

Object	Events	Suggestion
RichTextEdit	PrintHeader	Need no modification.
	FileExists	
ListView	Sort	Need no modification.
Application	SystemError	Need no modification.

5.8 Unsupported system functions

Table 5.5: Unsupported system functions that need no modification

System functions	Calls	Suggestion
ShowHelp	18	Need no modification.
FindClassDefinition	6	Need no modification.

5.9 Unsupported Shared variables

Table 5.6: Unsupported shared variables that need no modification

Shared variables	Calls	Suggestion
snv_property	5	Need no modification.

5.10 Unsupported Function not found

Table 5.7: Unsupported functions that need no modification

Functions	Calls	Suggestion
dwObject.GetChild	1	Need no modification.

Table 5.8: Unsupported functions that need no modification

Functions	Calls	Suggestion
dwObject.Print	3	Need modification. Refer to the detailed information below. DETAILED INFORMATION Location 1: Pfc_print() event of pfc_u_dw in pfcmain.pbl PowerServer Unsupported Code: Li_rc = lds_selection.Print(true, true)

Functions	Calls	Suggestion
		<p>Line Number: 75</p> <p>Change the code to:</p> <pre>if IsValid (lds_selection) then If ApeonGetClientType() = 'PB' then li_rc = lds_selection.Print (true, true) else li_rc = lds_selection.Print (true) end if destroy lds_selection else If ApeonGetClientType() = 'PB' then li_rc = this.Print (true, true) else li_rc = this.Print (true) end if end if</pre> <p>Location 2: Pfc_print() event of pfc_n_ds in pfcmain.pbl</p> <p>PowerServer Unsupported Code: Li_rc = lds_selection.Print(true, true)</p> <p>Line Number: 87</p> <p>Change the code to:</p> <pre>if IsValid (lds_selection) then if ApeonGetClientType() = 'PB' then i_rc =lds_selection.Print (true, true) else li_rc = lds_selection.Print (true) end if destroy lds_selection else If ApeonGetClientType() = 'PB' then li_rc = this.Print (true, true) else li_rc = this.Print (true) end if end if</pre>

5.11 Unsupported calls

Table 5.9: Unsupported calls

Unsupported calls	Suggestion
Calling the SQLPreview event by Update or ReselectRow functions is unsupported.	<p>Need modification. Refer to the detailed information below.</p> <p>DETAILED INFORMATION</p> <p>Location: Of_setupdatestyle function of the Pfc_n_cst_dwsrv_linkage object in pfc_dwsrv.pbl</p> <p>Add the following code:</p> <pre>if ai_style = TOPDOWN_BOTTOMUP then ai_style = BOTTOMUP end if if ai_style = BOTTOMUP_TOPDOWN then</pre>

Unsupported calls	Suggestion
	<pre> ai_style = BOTTOMUP end if </pre>
<p>Dynamically calling overloaded function of an uncertain object is unsupported.</p>	<p>Need modification. Refer to the detailed information below.</p> <p>DETAILED INFORMATION</p> <p>Location: Of_update(powerobject,boolean,boolean) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p>PowerServer Unsupported Code: li_rc = lpo_tocheck.Function Dynamic of_Update (ab_accepttext, ab_resetflag, lpo_updaterequestor)</p> <p>Line Number: 147</p> <p>Change the code to:</p> <pre> u_dw ldw_update n_ds lds_update u_lvs llvs_update u_tvs ltvs_update if ApeonGetClientType() = 'PB' then If lb_defined then li_rc = lpo_tocheck.Dynamic of_Update (ab_accepttext, ab_resetflag, lpo_updaterequestor) if li_rc < 0 Then Return -1 Continue End if else if lb_defined then Choose Case Typeof (lpo_tocheck) Case DataWindow! ldw_update = lpo_tocheck li_rc = ldw_update.of_Update (ab_accepttext, ab_resetflag, lpo_updaterequestor) if li_rc < 0 then Return -1 Continue Case ListView! llvs_update = lpo_tocheck li_rc = llvs_update.of_Update (ab_accepttext, ab_resetflag, lpo_updaterequestor) If li_rc < 0 Then return -1 Continue Case TreeView! ltvs_update = lpo_tocheck li_rc= ltvs_update.of_Update (ab_accepttext, ab_resetflag, lpo_updaterequestor) if li_rc < 0 Then return -1 Continue Case DataStore! lds_update = lpo_tocheck li_rc= lds_update.of_Update (ab_accepttext, ab_resetflag, lpo_updaterequestor) if li_rc < 0 Then return -1 Continue End choose End if end if end if </pre>

5.12 Differently behaved features

Table 5.10: Differently behaved features

Object	Suggestion
MDI frames	<p data-bbox="587 398 1362 432">Need modification. Refer to the detailed information below.</p> <p data-bbox="379 450 560 483">WorkSpaceX</p> <p data-bbox="587 450 1342 524">Location: Of_setposition () function of pfc_w_statusbar in pfcwnsrv.pbl</p> <p data-bbox="587 542 810 575">Line Number: 72</p> <p data-bbox="587 595 919 629">Suggested Modifications:</p> <p data-bbox="587 649 1337 723">Step 1: Add the rect structure variable to the pfcwnsrv.pbl with the following format.</p> <p data-bbox="587 743 981 777">global type rect from structure</p> <p data-bbox="587 797 699 831">long left</p> <p data-bbox="587 851 695 884">long top</p> <p data-bbox="587 904 716 938">long right</p> <p data-bbox="587 958 745 992">long bottom</p> <p data-bbox="587 1012 699 1046">end type</p> <p data-bbox="587 1066 1187 1140">Step 2: Declare a local external function in the pfc_w_statusbar.</p> <p data-bbox="587 1160 1366 1234">Function long GetWindowRect (long hwnd , ref rect lpRect) Library "user32"</p> <p data-bbox="587 1254 1286 1328">Step 3: Rewrite the scripts in the Current PFC into the following:</p> <pre data-bbox="587 1348 1358 2040"> long ll_bottompos, ll_rtn, ll_TopPos, ll_LeftPos rect lr_WinRect ll_rtn = getwindowrect (handle (iw_parentwindow), lr_WinRect) ll_LeftPos = PixelsToUnits (lr_WinRect.Left, XPixelsToUnits!) ll_TopPos = PixelsToUnits (lr_WinRect.Top, YPixelsToUnits!) ll_bottompos = PixelsToUnits (lr_WinRect.bottom, YPixelsToUnits!) If ApeonGetClientType() = 'PB' then // The Y Position of the Status Bar is the Bottom of // the Frame Window minus the MicroHelpHeight minus // the MicroHelpBorderHeight. ll_microhelp_ypos = & (iw_parentwindow.y + iw_parentwindow.height + li_ypos_extra) - & (iw_parentwindow.mdi_1.microhelpheight + ii_borderheight) // The desired X Position of the Status Bar is the // Frame Right End minus the StatusBar window. Also // subtract the extra spacing on win95. ll_desiredstatubar_xpos = & </pre>

Object	Suggestion
	<pre> iw_parentwindow.x + iw_parentwindow.workspacewidth() + & ii_borderwidth - (ii_winmaxwidth + 12 + li_xpos_extra) // The Frame X Position. ll_frame_xpos = iw_parentwindow.x + (2*ii_borderwidth) + 16 if ll_desiredstatubar_xpos < ll_frame_xpos then // Status Bar would extend to the left of the frame. this.move(ll_frame_xpos , ll_microhelp_ypos) // Make the Statubar the width of the frame. this.width = workspacewidth(iw_parentwindow) - (20 + li_xpos_extra) else // Normal as large as defined Status Bar window. this.move(ll_desiredstatubar_xpos , ll_microhelp_ypos) this.width = ii_winmaxwidth end if else // The Y Position of the Status Bar is the Bottom of // the Frame Window minus the MicroHelpHeight minus // the MicroHelpBorderHeight. ll_microhelp_ypos = & (ll_TopPos + iw_parentwindow.height + li_ypos_extra) - & (iw_parentwindow.mdi_1.microhelpheight + ii_borderheight) // The desired X Position of the Status Bar is the // Frame Right End minus the StatusBar window. // Also subtract the extra spacing on win95. ll_desiredstatubar_xpos = & ll_LeftPos + iw_parentwindow.workspacewidth() + & ii_borderwidth - (ii_winmaxwidth + 12 + li_xpos_extra) ll_frame_xpos = ll_LeftPos + (2*ii_borderwidth) + 16 if ll_desiredstatubar_xpos < ll_frame_xpos then // Status Bar would extend to the left of the frame. this.move(ll_frame_xpos , ll_bottompos - 70) // Make the Statubar the width of the frame. this.width = iw_parentwindow.workspacewidth() - (20 + li_xpos_extra) else // Normal as large as defined Status Bar window. this.move(ll_desiredstatubar_xpos , ll_bottompos - 70) this.width = ii_winmaxwidth end if end if end if </pre>
MDI frames	<p>Resize Service</p> <p>Need modification. Refer to the detailed information below.</p> <p>Step 1: Change the wintype property for the w_examplemain object from MDI to main.</p> <p>Step 2: Modify the pfc_open event in the n_exampleappmanager of the appexmfe.pbl as shown below:</p> <p>Current scripts in PFC:</p> <pre> Open (w_examplemain) Appeon compliant scripts in PFC </pre>

Object	Suggestion
	<pre data-bbox="587 241 1396 302">Open (w_frame) opensheet(w_exemplemain, w_frame, 0, layered!)</pre> <p data-bbox="587 324 1396 481">Step 3: Copy the following scripts from the pfc_preopen event for the w_exemplemain object in the appxmfe.pbl to the pfc_preopen event for the w_frame object in the exmmain.pbl and then comment out the following scripts in this event.</p> <pre data-bbox="587 504 1396 1137">// Set the frame window with the application manager. gnv_app.of_SetFrame(w_exemplemain) gnv_app.of_SetMicrohelp(true) // Enable the Status Bar Services of_SetStatusBar(true) if IsValid(inv_statusbar) then inv_statusbar.of_SetBorderType(0) inv_statusbar.of_SetGapWidth(40) inv_statusbar.of_Register ('debugsrv', 'bitmap', 'dbsvc.gif', 80) inv_statusbar.of_Register ('sqlspysrv', 'bitmap', 'sssvcoff.gif', 80) inv_statusbar.of_Register ('debuglogwin', 'bitmap', 'dlgwnoff.gif', 80) inv_statusbar.of_Register ('sqlspywin', 'bitmap', 'sswinoff.gif', 80) if gnv_app.ienv_object.Win16 then inv_statusbar.of_SetGDI(true) inv_statusbar.of_SetUser(true) end if inv_statusbar.of_SetTimer(true) end if</pre>
Any	<p data-bbox="587 1149 1396 1182">ClassName Need modification. Refer to the detailed information below.</p> <p data-bbox="587 1205 1396 1238">DETAILED INFORMATION</p> <p data-bbox="587 1261 1396 1328">Location: Of_updatespending(powerobject) function of pfc_n_cst_luw in pfcapsrv.pbl</p> <p data-bbox="587 1350 1396 1417">PowerServer Unsupported Code: If ClassName(la_rc) = 'integer' or ClassName(la_rc)='long' Then</p> <p data-bbox="587 1440 1396 1473">Line Number: 133</p> <p data-bbox="587 1496 1396 1529">Change the code to:</p> <pre data-bbox="587 1552 1396 2038">if ApeonGetClientType() = 'PB' then If ClassName(la_rc) = 'integer' or ClassName(la_rc) = 'long' Then // Functionality was found. If la_rc < 0 Then Return -1 If la_rc >= 1 Then lb_updatespending = True If TypeOf (lpo_tocheck) = DataWindow! Then la_rc = lpo_tocheck.Function Dynamic of_IsRoot() If ClassName(la_rc) = 'boolean' Then lb_updatespending = la_rc Else Return -1 End If End If End If End If lb_performedtest = True</pre>

Object	Suggestion
	<pre>End If else If ClassName(la_rc) = 'integer' or ClassName(la_rc) = 'long' or ClassName(la_rc) = 'number' Then If la_rc < 0 Then Return -1 If la_rc >= 1 Then lb_updatespending = True If TypeOf (lpo_tocheck) = DataWindow! Then la_rc = lpo_tocheck.Dynamic of_IsRoot() If ClassName(la_rc) = 'boolean' then lb_updatespending = la_rc else Return -1 End if End If End If lb_performedtest = True End If end if</pre>

6 Migration Tutorial

6.1 Introduction

This tutorial is a series of five exercises in which you convert and deploy the Apeon tutorial PowerBuilder application to the Web. By following the tutorial, you will get hands-on experience in Web migration using PowerServer.

The Apeon tutorial PowerBuilder application is a small program involving database interaction. The application is used in the tutorial as the target PowerBuilder application to be worked on for a complete Web migration.

This tutorial is a simplified and practical example of the PowerServer Web migration methodology laid out in [Chapter 2, Migration Process](#). It can serve as a starting point for any developer who wants to convert their existing PowerBuilder applications. After the successful migration of the tutorial application, you can try to convert more complex, real PowerBuilder applications into a more complicated network environment.

How you will proceed:

Lesson 1	Load the Apeon tutorial application into PowerBuilder.
Lesson 2	Configure PowerServer Toolkit.
Lesson 3	Analyze the tutorial application for unsupported features.
Lesson 4	Deploy the tutorial application to the Web.
Lesson 5	Run the Web application.

What you will learn:

Lesson 1	How to create a PowerBuilder workspace, load application PBL files, set up an ODBC data source, and run the application from the PowerBuilder IDE.
Lesson 2	Add the application profile, the PowerServer profile, the Web server profile, the deployment profile, the data source profile, and the transaction object mapping in the PowerServer Toolkit Configuration Wizard.
Lesson 3	Perform unsupported feature analysis, perform a Full build, and optimize the tutorial application.
Lesson 4	Deploy the tutorial application to the Web using the PowerServer Application Deployment Wizard.
Lesson 5	Run the deployed Web application in the Web browser.

How long it will take:

You can complete the entire tutorial in ninety minutes. Each lesson has also been designed so that you can stop after any lesson and continue at another time.

6.1.1 Preparing for the tutorial

The following preparations are required before starting this tutorial:

- Set up one Workstation.

This tutorial assumes the most simplified network environment. Only one physical machine is used for all the different roles in the n-Tier Web architecture: client PC, Web Server, application server, database server, and developer PC.

Refer to Installation Guide for .NET for specific system requirements.

- Install other required softwares:

Windows Server 2019, 2016, or 2012 R2, Windows 7, Windows 8.1, or Windows 10

PowerBuilder 2019 R2

Internet Explorer 11, Microsoft Edge, Google Chrome, Mozilla FireFox, or Opera.

Microsoft .NET Framework 4.x & IIS 7.5, 8.0, or 10

Note:

1. IIS is included with Windows, but not installed by default, you will need to install IIS first.
 2. IIS will be used as both the application server and the Web server in this tutorial.
 3. If PowerServer is installed to the other application servers (such as WebLogic, WebSphere, JBoss, or JEUS), you can still follow the steps in this tutorial for configuring and deploying the Apeon tutorial application while paying close attention to the parts specified as working differently for different application servers.
- Install Apeon PowerServer (on .NET) edition

Install PowerServer and PowerServer Toolkit components. The demo application that the tutorial uses will be automatically installed during the PowerServer Toolkit installation.

If you intend to install PowerServer and PowerServer Toolkit on different machines, most of the instructions in this tutorial, with the exception of some settings associated with PowerServer, such as PowerServer profile and data sources, will still be relevant. For instructions, refer to the PowerServer Toolkit User Guide.

6.1.2 Relevant files

The tutorial will use the following files:

File Name	appeontutor.pbl	appeonsample.db
Location	\appeondemo\Tutorial (e.g. C:\Users\Public\Documents \Apeon\Toolkit 2020\appeondemo \Tutorial)	\appeondemo\Database (e.g. C:\Users\Public\Documents \Apeon\Toolkit 2020\appeondemo \Database)
Description	The PowerBuilder Library (application source code) used in the tutorial.	The SAP SQL Anywhere database file used for the tutorial application.

6.2 Loading the Tutorial PowerBuilder Application

This tutorial starts with two files: *appeontutor.pbl* and *appeonsample.db*. The *appeontutor.pbl* file contains the source code for the Apeon tutorial PowerBuilder application. First, follow the instructions to make the tutorial application run in PowerBuilder.

In this section, you will learn how to:

- [Create a new PowerBuilder Workspace](#)
- [Load the tutorial PBLs into the Workspace](#)
- [Configure the ODBC data source for the tutorial application](#)
- [Run the tutorial application](#)

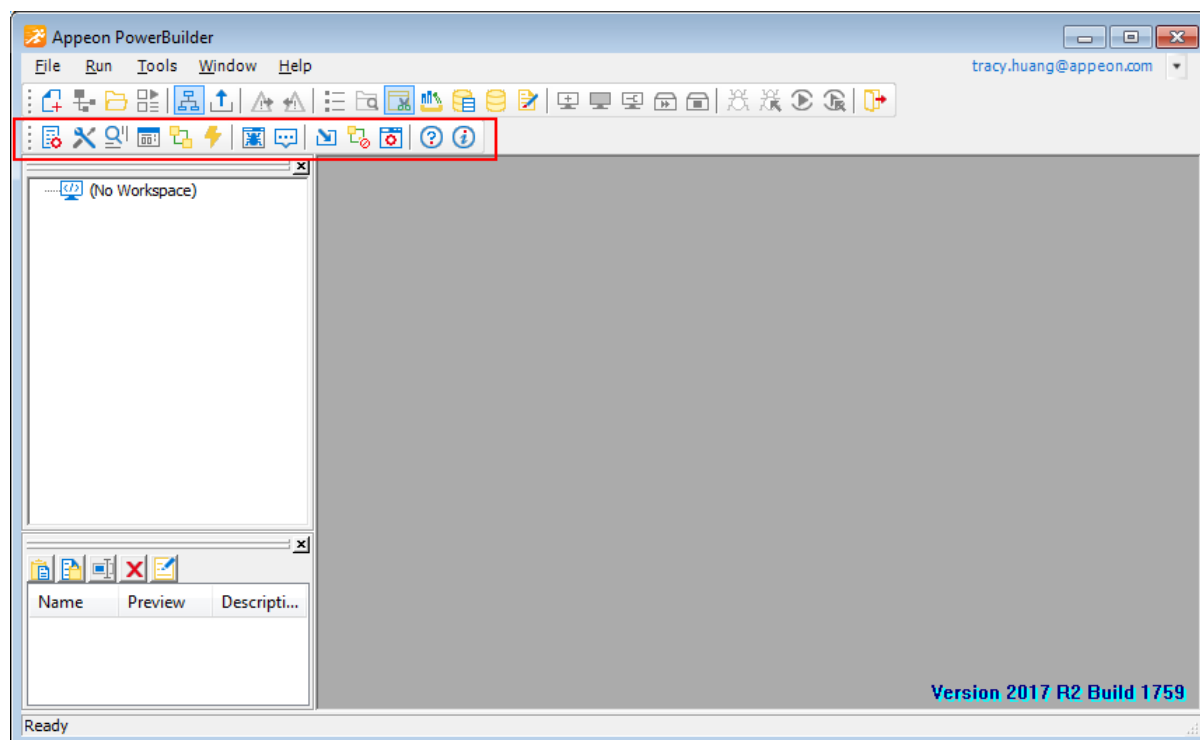
6.2.1 Creating a Workspace

You can have only one PowerBuilder Workspace open at a time, but you can add as many targets or applications to the Workspace as you want, including opening and editing objects in multiple targets.

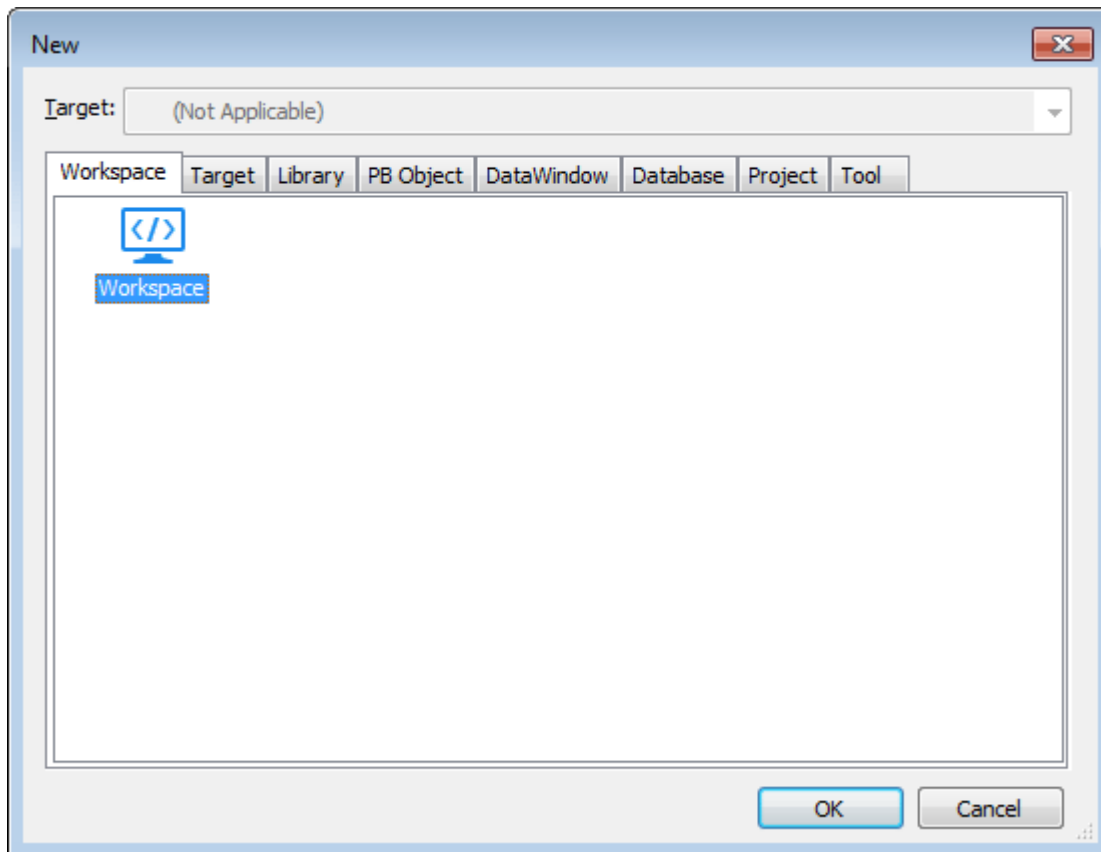
To create a new Workspace for the Apeon tutorial PowerBuilder application:

Step 1: Start PowerBuilder. The PowerBuilder IDE starts with the PowerServer Toolkit loaded.

Figure 6.1: PowerServer Toolkit loaded into PowerBuilder



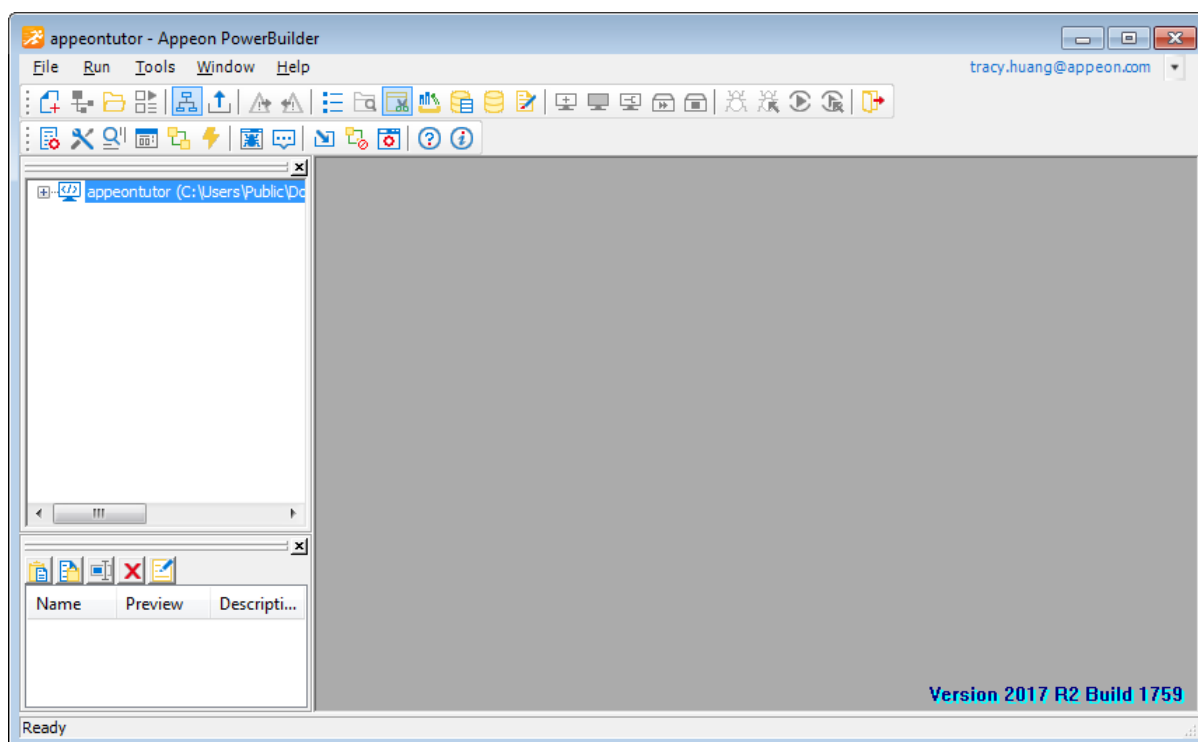
Step 2: Select **File** > **New** from the PowerBuilder menu bar, and the New dialog box appears.

Figure 6.2: Adding a new Workspace

Step 3: Select the **Workspace** icon in the **Workspace** tab and click **OK**. The New Workspace dialog box displays. Navigate to the `%Appeon%\PowerServer\Toolkit\appeondemo\Tutorial` folder, for example, `C:\Users\Public\Documents\Appeon\Toolkit 2020\AppeonDemo\Tutorial`.

Type `appeontutor` in the **File name** text box and click **Save**.

The `appeontutor` workspace has been added and appears as the top item in the PowerBuilder system tree.

Figure 6.3: Newly created Workspace

6.2.2 Loading the tutorial PBL file

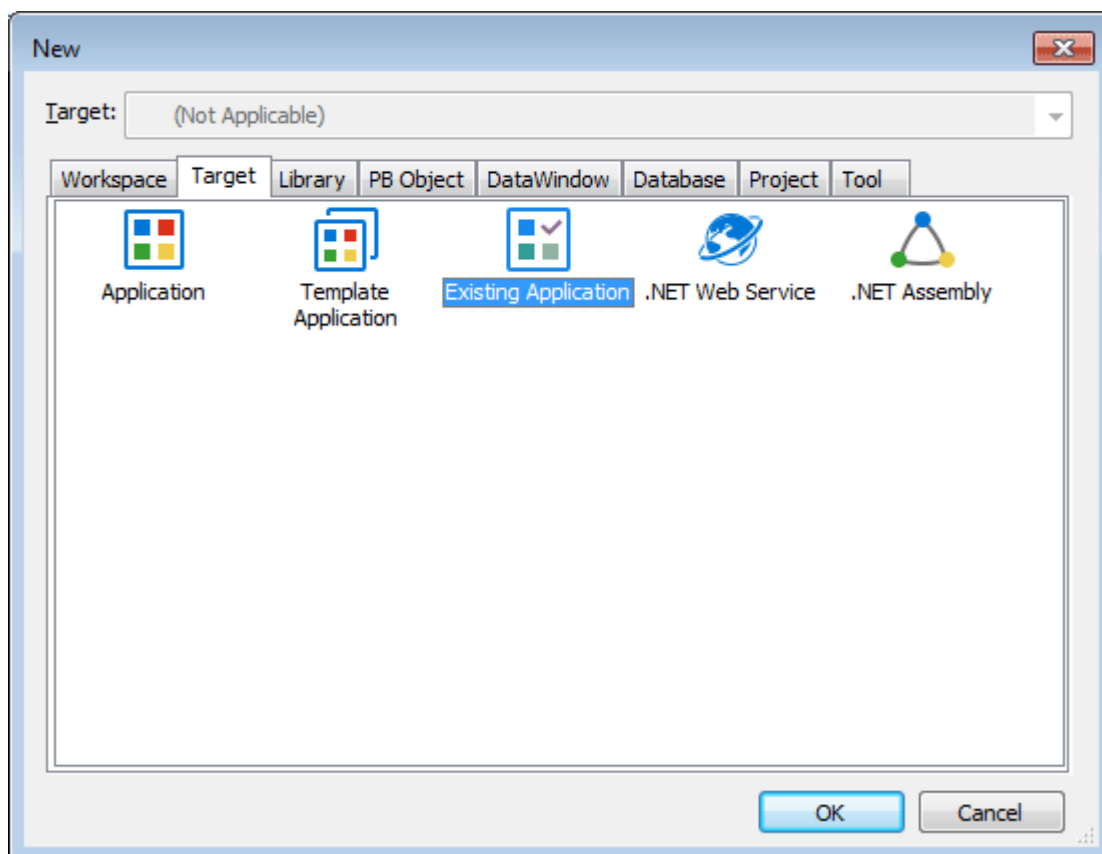
A PowerBuilder Library (*.pbl file) is a collection of compiled object definitions and source objects (including scripts) stored in the same location. The PowerBuilder painters and wizards store various objects in libraries, such as Applications, Windows, DataWindows, Menus, Functions, Structures, and User Objects.

Now load the PBL file of the Apeon tutorial application into the newly created Workspace:

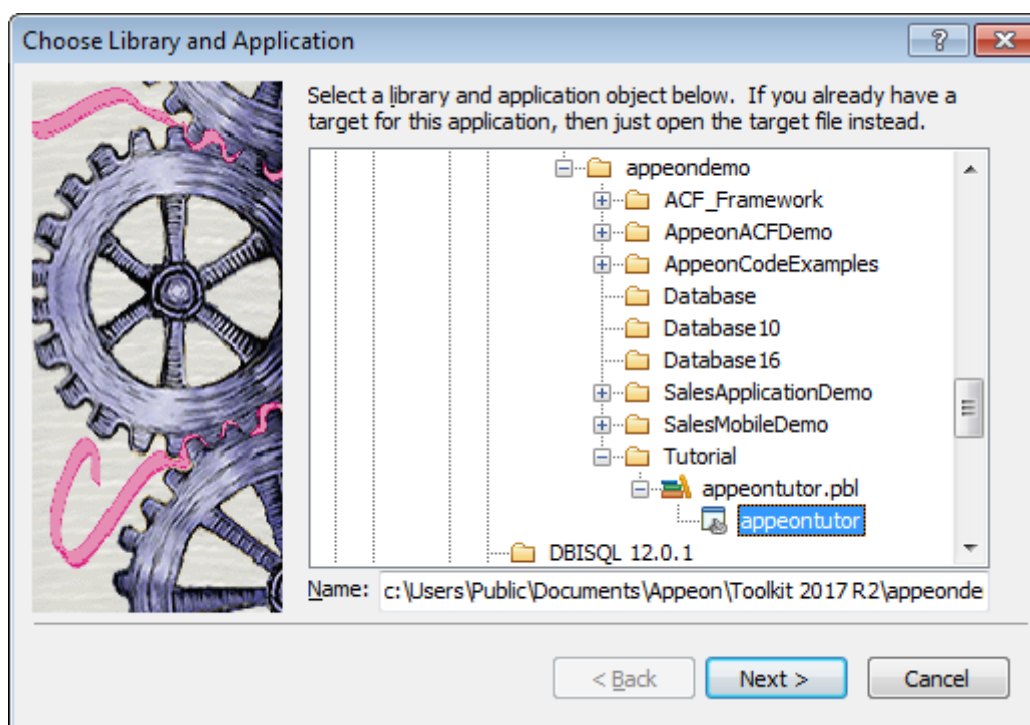
Step 1: Select **File** > **New** from PowerBuilder menu bar, and the New dialog box appears.

Click the **Target** tab if it is not already selected.

Step 2: Select the **Existing Application** icon and click **OK**.

Figure 6.4: Target tab

Now the Choose Library and Application dialog box is displayed. Select the *appeontutor* application under the *appeontutor.pbl*, and click **Next**.

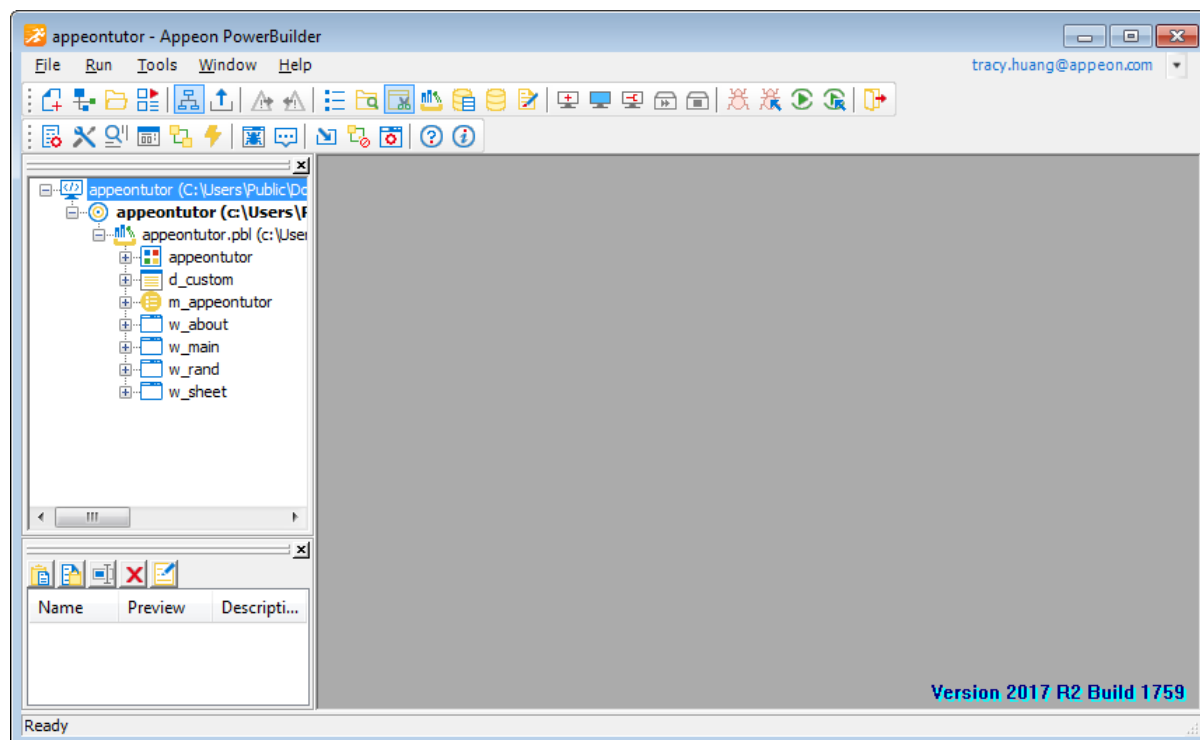
Figure 6.5: Choose Library and Application

Step 3: Click **Next** in the **Set Library Search Path** dialog box.

Step 4: Click **Finish** in the **Specify Target File** dialog box.

Step 5: The Apeon tutorial PowerBuilder application has now been added to the *appeontutor* Workspace and is displayed in the system tree.

Figure 6.6: Newly added "appeontutor" Workspace



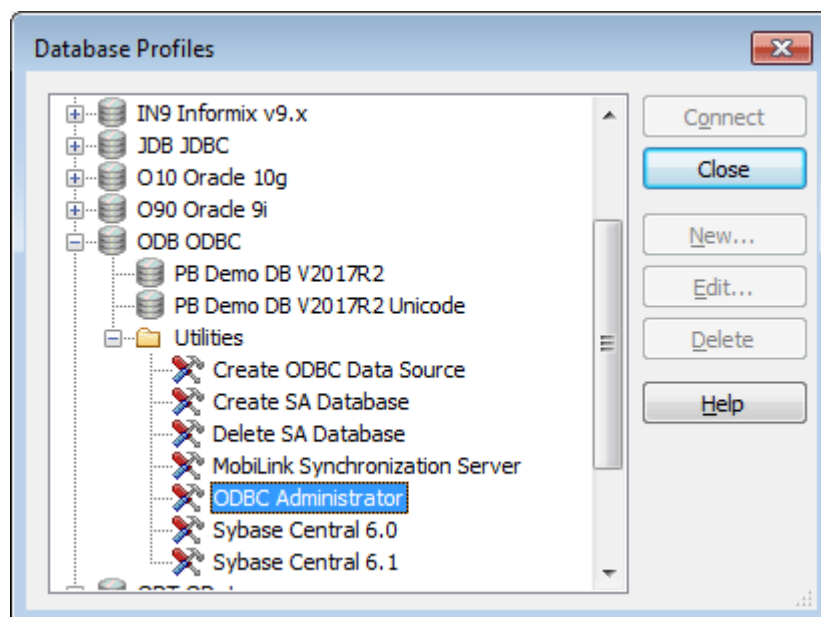
6.2.3 Configuring ODBC data source

An ODBC data source stores the parameters used to connect to the indicated data provider through the Open Database Connectivity interface. By setting up the ODBC data source that identifies the SAP SQL Anywhere database file for the Apeon tutorial PowerBuilder application, the tutorial application is able to establish a connection with the SAP SQL Anywhere database through a reference to the ODBC data source name.

To set up the ODBC data source for the tutorial application:

Step 1: Select **Tools > Database Profile** from the PowerBuilder menu bar, and the Database Profiles dialog box displays.

Step 2: Choose **ODB ODBC > Utilities**, and then double click **ODBC Administrator**.

Figure 6.7: Database Profiles

Step 3: Select the **System DSN** tab in the **ODBC Data Source Administrator** dialog box and click **Add**.

Step 4: Select the **SQL Anywhere 16** driver to connect to the *appeonsample.db* database file. Select **SQLAnywhere 16** and click **Finish** in the **Create New Data Source** dialog box.

Step 5: The ODBC Configuration for SQL Anywhere dialog box is displayed. Type the necessary settings into the different tabs as specified in the following table, and leave other fields at their default settings.

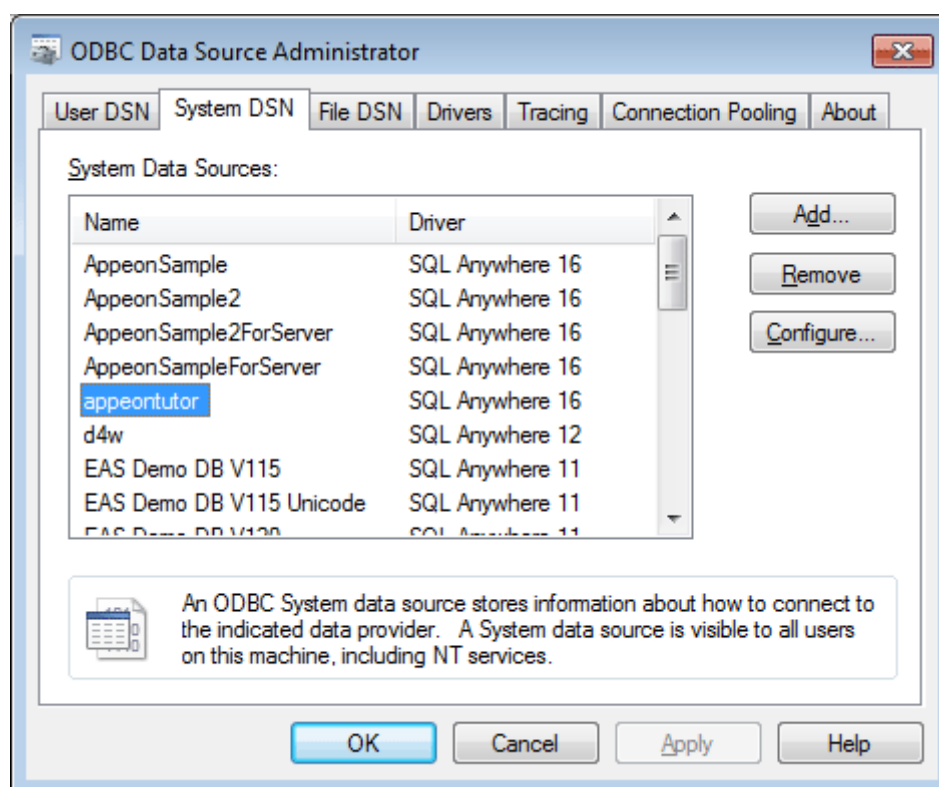
Table 6.1: Data source settings

In this tab...	In this field...	You should...
ODBC	Data source name	Type <i>appeontutor</i>
Login	User ID	Type <i>dba</i> (case sensitive)
	Password	Type <i>sql</i> (case sensitive)
	Action	Select <i>Start and connect to a database on this computer</i>
	Database file	Click Browse to select the <i>AppeonSample.db</i> file. For example, C:\Users\Public\Documents\Apeon\Toolkit 2020\appeondemo\Database\AppeonSample.db.
	Database name	Type <i>appeontutor</i>
	Server name	Type <i>appeontutor</i>

Step 6: Go back to the **ODBC** tab and click **Test Connection**. Ensure the connection test is successful before continuing.

Step 7: Click **OK** to close the ODBC Configuration for SQL Anywhere dialog box.

The *appeontutor* is added as a system data source under the System DSN tab.

Figure 6.8: ODBC Data Source Administrator

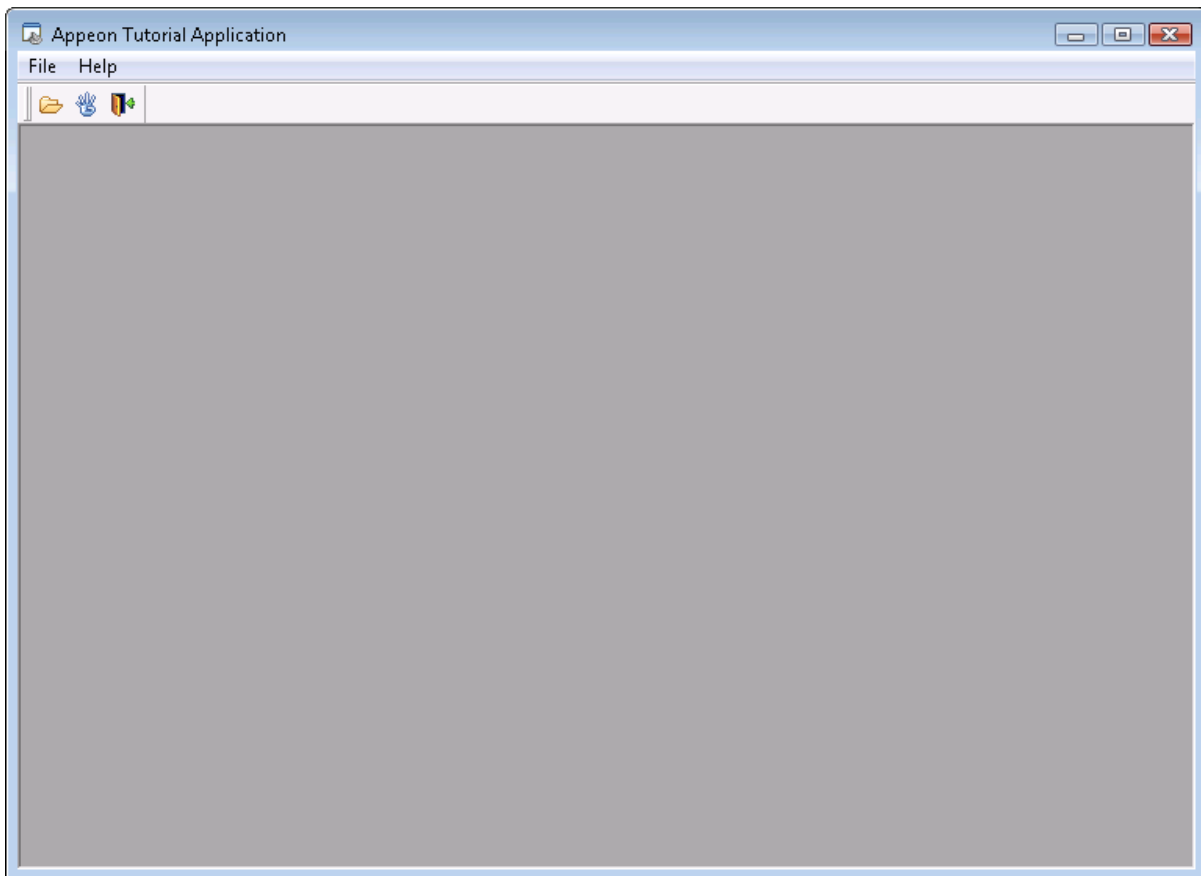
Step 8: Click **OK** to close the dialog box and click **Close** to exit the Database Profiles dialog box.

6.2.4 Running the tutorial application

The database connection parameters (PowerBuilder Transaction properties) have been preset in the tutorial PowerBuilder application that will connect to the data source named *appeontutor*. You can run the tutorial application in PowerBuilder now.

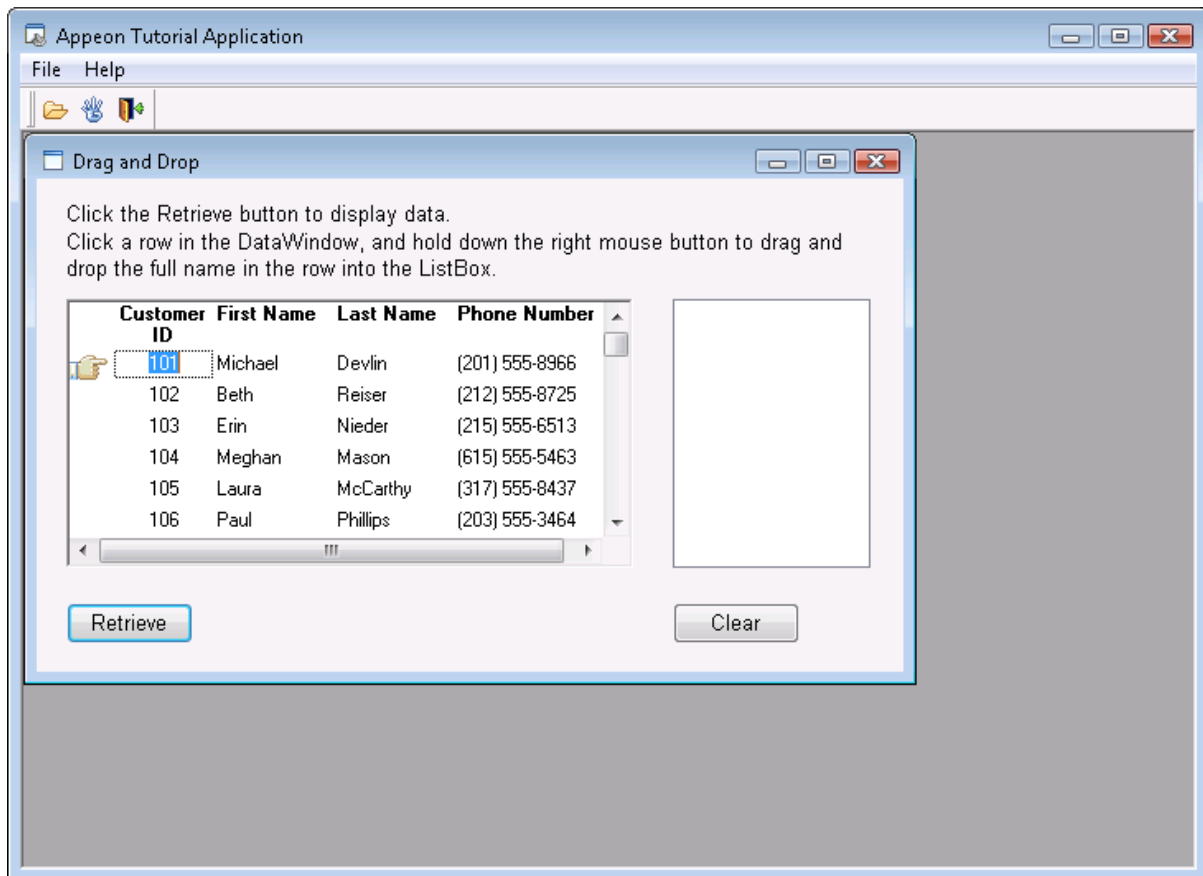
To run the Apeon tutorial PowerBuilder application:

Step 1: In the PowerBuilder IDE, choose **Run > Run appeontutor** from the PowerBuilder menu bar, or click the **Run** icon in PowerBar1. The Apeon tutorial PowerBuilder application starts.

Figure 6.9: Appeon Tutorial Application

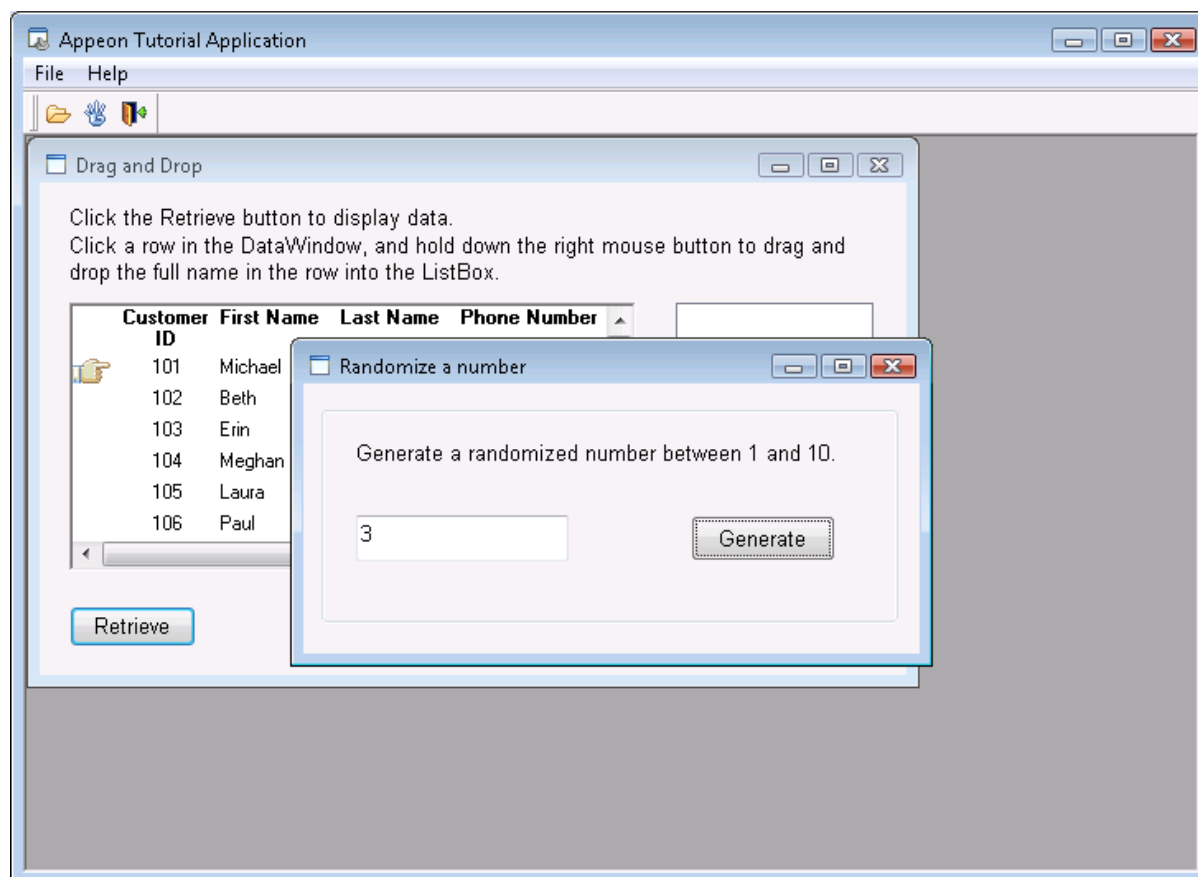
Step 2: In the Appeon Tutorial Application, select **File > Drag and Drop** to open the Drag and Drop window.

The Appeon tutorial PowerBuilder application has an MDI window, two sheet windows, and an About window. In the Drag and Drop window, click the **Retrieve** button to display data. Click a row in the DataWindow and hold down the right mouse button to drag and drop the full name in the row into the right ListBox. Click the **Clear** button to reset the ListBox.

Figure 6.10: Drag and Drop

Step 3: Select **File > Randomize** to open the Randomize a number window.

This window uses the PowerBuilder *Randomize* and *Rand* functions to randomly generate an integer between 1 and 10, and display it in a window.

Figure 6.11: Randomize a number

Step 4: Close the tutorial application, and return to the PowerBuilder IDE.

6.3 Configuring PowerServer Toolkit

PowerServer Toolkit, a component of Apeon PowerServer, extends the capabilities of PowerBuilder, allowing a new or existing PowerBuilder application to be converted into a *bona fide* Web Application, using only PowerBuilder skills.

PowerServer Toolkit provides a set of tools that enable the entire PowerBuilder-to-Web process to take place within the PowerBuilder IDE. These tools are accessed via a toolbar in the PowerBuilder IDE. The PowerServer Toolkit automatically loads each time PowerBuilder is opened.

Figure 6.12: PowerServer Toolkit

The Apeon tutorial PowerBuilder application should run through the following configuration tasks before PowerServer Toolkit can automate the task of Web conversion and deployment of the tutorial application.

- [Configuring basic settings](#)
- [Selecting PBL file\(s\)](#)

- [Configuring deployment settings](#)
- [Selecting DB Type\(s\)](#)
- [Declaring transaction object\(s\)](#)

All these configuration tasks can be completed within the PowerServer Toolkit Configuration Wizard. To access this wizard, click the **Config Wizard** button (🔧) on the PowerServer Toolkit.

Read the requirements on the Welcome screen and click **Next** to proceed.

6.3.1 Configuring basic settings

The basic settings refers to the settings of an application that are essential for deployment, including the application profile name, Web folder, and project type.

Figure 6.13: Configure basic settings window

The screenshot shows the 'PowerServer Toolkit Configuration Wizard' window. The left sidebar indicates the current step is 'Configure basic settings'. The main configuration area includes the following fields and options:

- Application Profile Name:** Text box containing 'appeontutor'.
- Web Folder:** Text box containing 'appeontutor'.
- Project Type:** Dropdown menu with 'Web' selected.
- Device Type:** Dropdown menu with 'Both' selected.
- Mobile App Name:** Text box containing 'appeontutor'.
- Mobile App Icon:** Text box with a 'Browse...' button.
- Mobile App Description:** Text area.
- Use this background color for the mobile app: [White color swatch]

At the bottom of the window, there are buttons for 'Help', '< Back', 'Next >', 'Cancel', and 'Finish'.

Step 1: Specify the settings as follows:

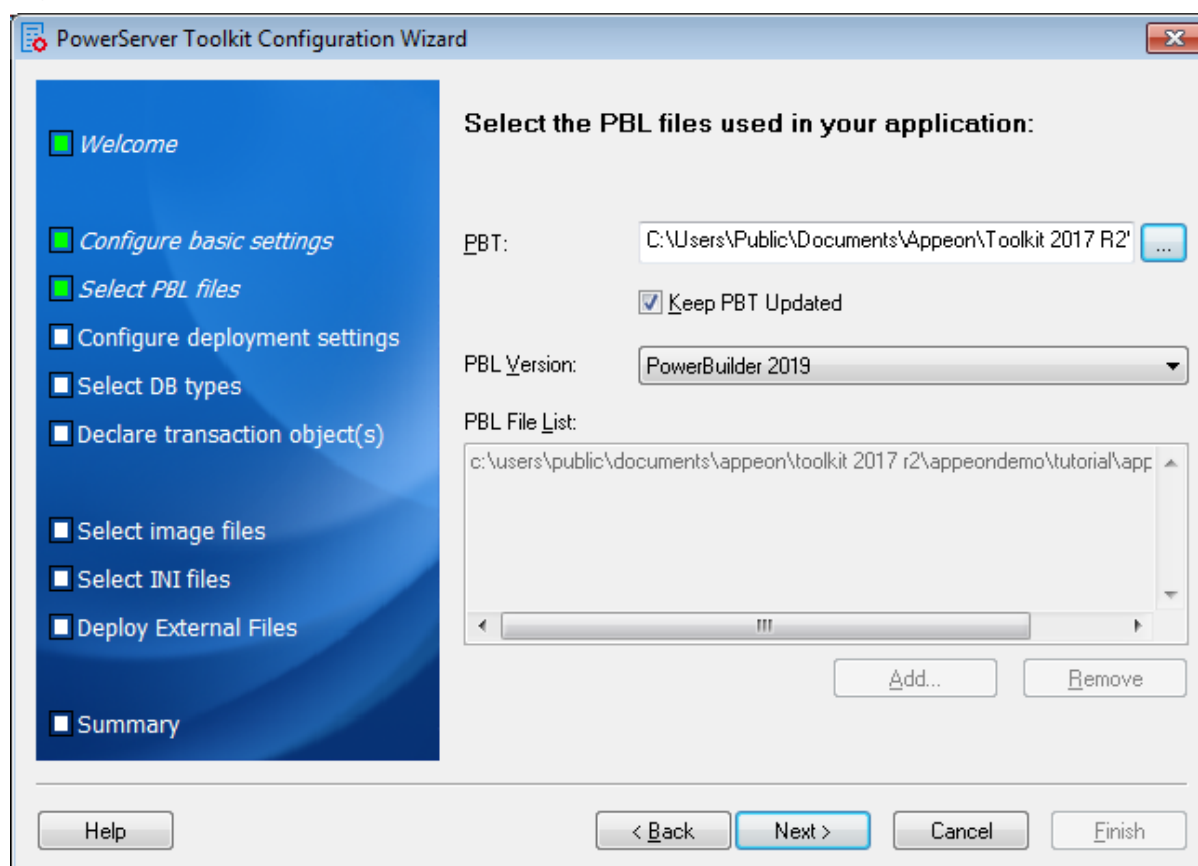
- In the **Application Profile Name** field, input *appeontutor*. The **Web Folder** field is automatically filled with what you enter in the **Application Profile Name** field.
- Select **Web** from the **Project Type** dropdown list box.

Step 2: Click **Next** to proceed.

6.3.2 Selecting PBL file(s)

Specify the version and the location of source code of the PowerBuilder application.

Figure 6.14: Select PBLs



Step 1: Verify that *apeontutor.pbt* is currently selected in the PBT field. Or click the browse (...) button to select the *apeontutor.pbt* file.

When the *apeontutor.pbt* file is selected, the *apeontutor* PBL file is automatically added to the PBL File List.

Step 2: Check the **Keep PBT Updated** checkbox.

Step 3: Choose a PowerBuilder version from the **PBL Version** dropdown list box.

Step 4: Click **Next** to proceed.

6.3.3 Configuring deployment settings

The deployment settings associate the PowerServer(s) and Web server(s) as a group used for application deployment. In this tutorial, the deployment profile will associate *PowerServerTutor* and *WebServerTutor*.

In this section, you will learn:

- [Starting PowerServer and Web server](#)
- [Adding a PowerServer profile](#)
- [Adding a Web server profile](#)

- [Selecting a deployment profile](#)

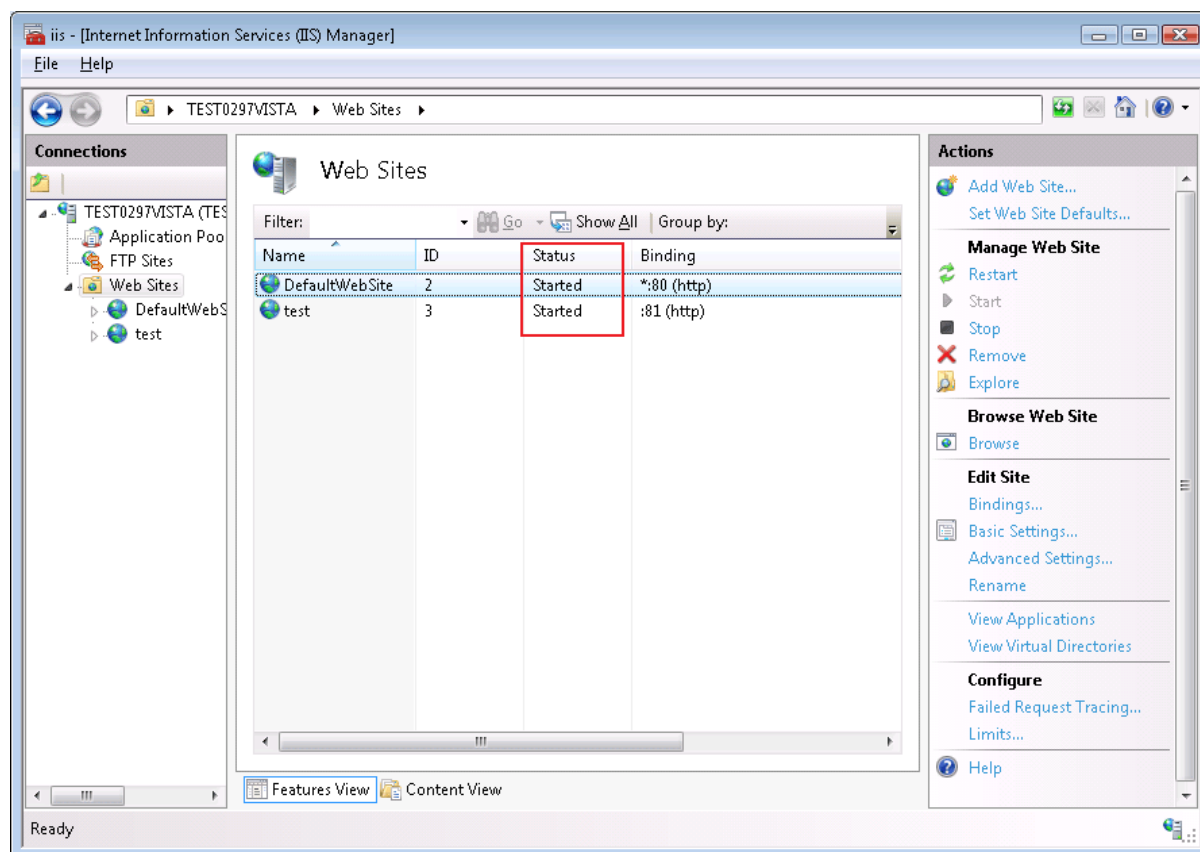
6.3.3.1 Starting PowerServer and Web server

When adding the PowerServer profile and Web server profile, you need to verify the connection to the PowerServer and the Web server; therefore, you should start them before configuration. If you want to start PowerServer installed to WebLogic, WebSphere, JBoss, or JEUS, refer to Installation Guide for .NET for detailed instructions. In this tutorial, IIS will be used as both the PowerServer and the Web server. Therefore, you only need to start IIS. Refer to the following steps for how to start the IIS/PowerServer.

Select Windows **Start > Programs > Apeon > PowerBuilder 2019 R2 > PowerServer > IIS Manager**.

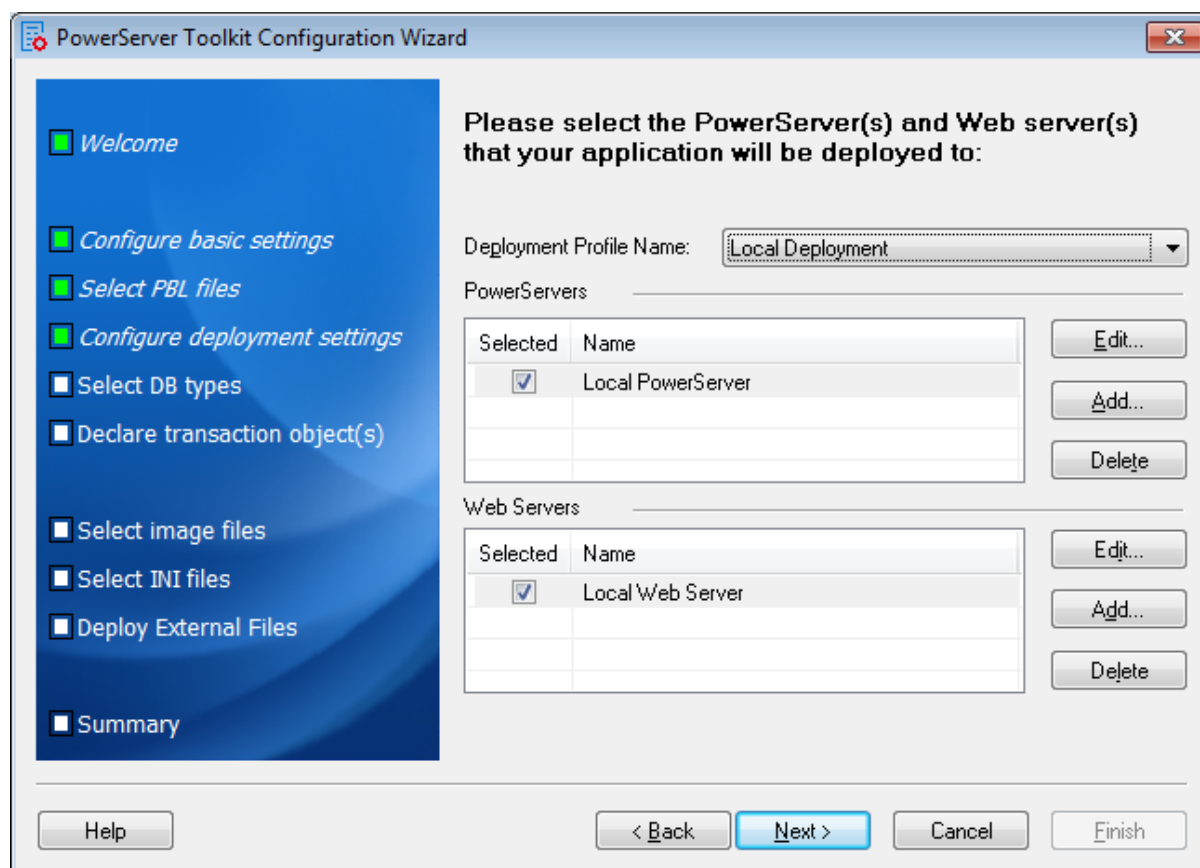
Make sure that the Web Site where PowerServer is installed is started, as shown in the following figure. If not, please select the name listed on the **Web Site** page and click **Start** on the Actions pane to start it. If it is started, the Start menu will be grayed out. The following figure indicates that PowerServer and Web server are ready for use.

Figure 6.15: IIS/PowerServer



6.3.3.2 Adding a PowerServer profile

Each PowerServer profile contains settings of a PowerServer/application server used by the PowerServer deployment.

Figure 6.16: PowerServer profile

To add the PowerServer profile for the deployment of the tutorial application:

Step 1: Click the **Add** button in the **PowerServer** group box.

The PowerServer Profile Configuration dialog box appears.

Figure 6.17: PowerServer Profile Configuration

The screenshot shows a 'PowerServer Profile Configuration' dialog box with the following fields and options:

- Profile Settings:** Profile Name: PowerServerTutor
- PowerServer Settings:** Server Type: J2EE .NET; Server: localhost; Server Port: 80
- AEM Settings:** AEM: http://localhost:80/AEM; Connection Method: HTTP HTTPS
- Deployment Security Settings:** Deployment Security: Disabled; Get State button; Username and Password should be configured in AEM. (Username and Password fields are empty)
- Test Settings:** Test PowerServer Settings... button

The following table lists instructions for how to specify the properties for a PowerServer profile.

Table 6.2: Instructions for creating a PowerServer profile

Property	Instructions
Profile Name	Input "PowerServerTutor" as the name of the PowerServer profile.
Server Type	Make sure that .NET is selected. If PowerServer is installed to WebLogic, WebSphere, JBoss, or JEUS, then select "J2EE".
Server	Input "localhost". To deploy to a remote PowerServer, enter the IP address or machine name of the remote server.
Server Port	Input "80". If PowerServer is installed to WebLogic, input "7001"; if installed to WebSphere, input "9080"; if installed to JEUS, input "8088"; if installed to JBoss, input "8080". To deploy to a remote PowerServer, enter the HTTP or HTTPS port of the remote server.
AEM URL	The URL for AEM will be automatically generated after Server and Server Port are specified.

Property	Instructions
Connection method	Leave it as default.
Deployment Security	Leave it as default.
Username	Leave it as default.
Password	Leave it as default.

For detailed instructions about these settings, refer to Section 3.2.3.2, “PowerServer profile settings” in *PowerServer Toolkit User Guide*.

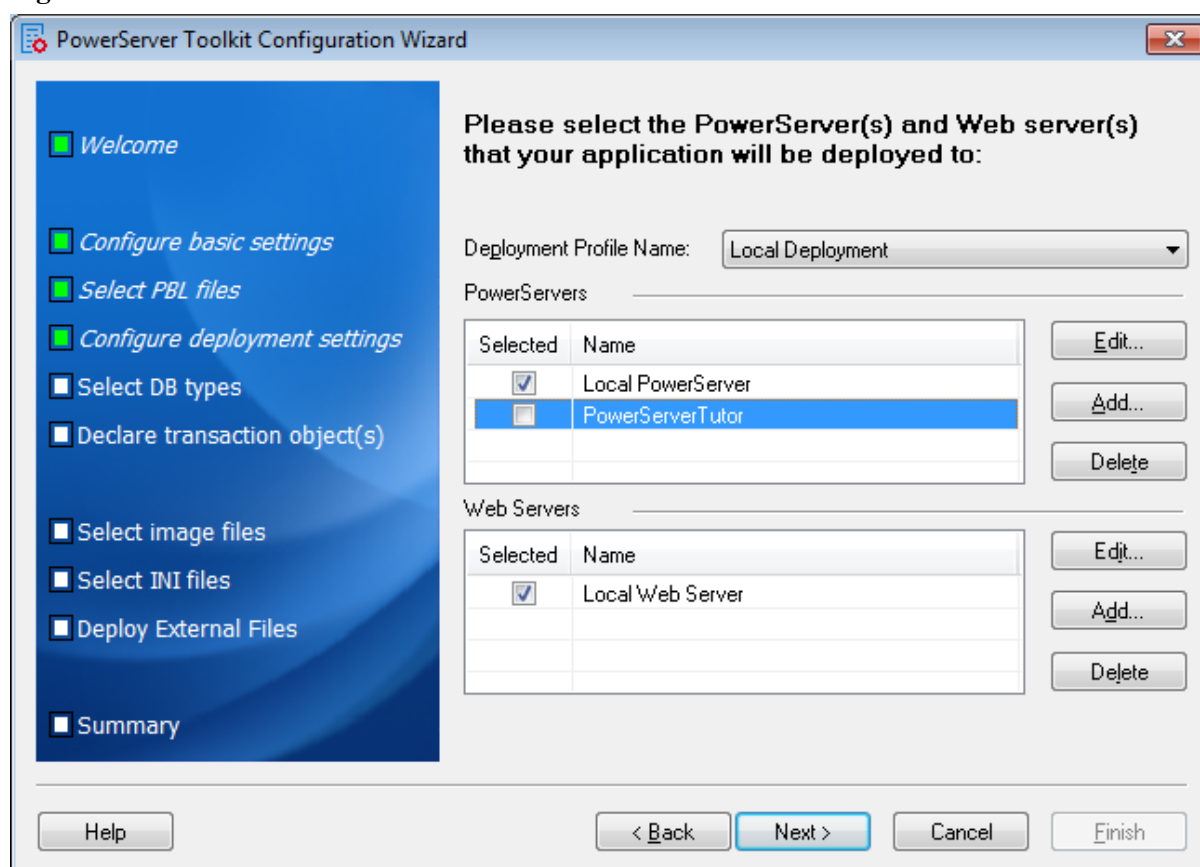
Step 2: Click **Test PowerServer Settings**.

PowerServer Toolkit will try to connect to PowerServer with the parameters you specified. Make sure the test is successful before you continue.

Step 3: Click **OK**.

PowerServerTutor is added to the PowerServers list.

Figure 6.18: PowerServer Tutor is added



6.3.3.3 Adding a Web server profile

Each Web server profile contains settings of a Web server used by the PowerServer deployment.

Since IIS will be used for the deployment of the Apeon tutorial PowerBuilder application, you need not to start it because IIS/PowerServer is already running.

To add the Web server profile for the deployment of the tutorial application:

Step 1: Click the **Add** button in the **Web Servers** group box.

The Web Server Profile Configuration dialog box appears.

Figure 6.19: Web Server Profile Configuration window

The screenshot shows the 'Web Server Profile Configuration' dialog box with the following settings:

- Profile Settings:** Profile Name: WebServerTutor; File Transfer Type: Local Server.
- Web Server Settings:** Server Type: Microsoft IIS; Server: localhost; Server Port: 80.
- Local Server Settings:** Web Root Path: C:\inetpub\wwwroot. A 'Browse...' button is present below the path.
- FTP File Transfer Settings:** FTP Port, FTP Username, and FTP Password fields are empty. There is a checkbox for 'Require explicit FTP over TLS/SSL' which is unchecked.
- Test Settings:** A button labeled 'Test Web Server Settings...' is highlighted with a dashed blue border.

At the bottom right, there are 'OK' and 'Cancel' buttons.

The following table lists instructions for how to specify the properties for a Web server profile.

Table 6.3: Instructions for creating a Web Server profile

Property	Instructions
Profile Name	Input "WebServerTutor" as the name of the Web Server profile.
File Transfer Type	Select Local Server. To deploy to a remote Web Server, select Remote Server, and then specify the FTP Port, FTP Username and FTP Password of the remote server.
Server Type	Select Microsoft IIS. If using WebLogic HTTP server, select WebLogic; if using WebSphere HTTP server, select WebSphere; if using JEUS HTTP server, select JEUS; if using JBoss HTTP server, select JBoss.

Property	Instructions
Server	Input "localhost". To deploy to a remote Web Server, enter the IP address or machine name of the remote server.
Server Port	Input "80". If PowerServer is installed to WebLogic, then input "7001"; if installed to WebSphere, then input "9080"; if installed to JEUS, then input "8088"; if installed to JBoss, then input "8080". To deploy to a remote Web Server, enter the HTTP or HTTPS port of the remote server.
Web Root Path	Click Browse to navigate to the path to the IIS Web site home directory (for example: C:\Inetpub\wwwroot).

For detailed instructions about these settings, refer to Section 3.2.3.3, “Web Server profile settings” in *PowerServer Toolkit User Guide*.

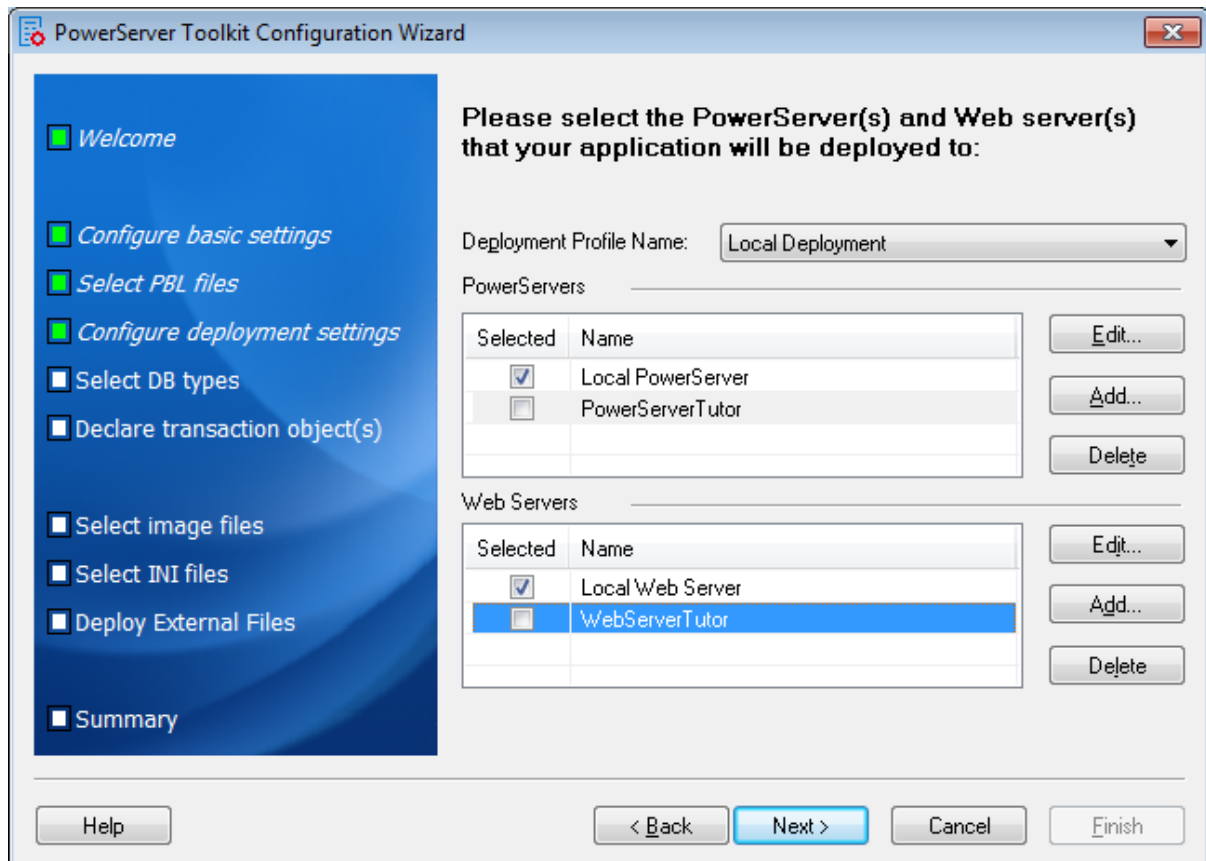
Step 2: Click **Test Web Server Settings**.

PowerServer Toolkit will try to connect to the Web server with the IP (localhost) and port (80). Make sure the test is successful before you continue.

Step 3: Click **OK**.

WebServerTutor is added to the Web Servers list.

Figure 6.20: WebServerTutor is added



6.3.3.4 Selecting a deployment profile

A deployment profile associates PowerServer(s) and Web server(s) as a group used for Web deployment.

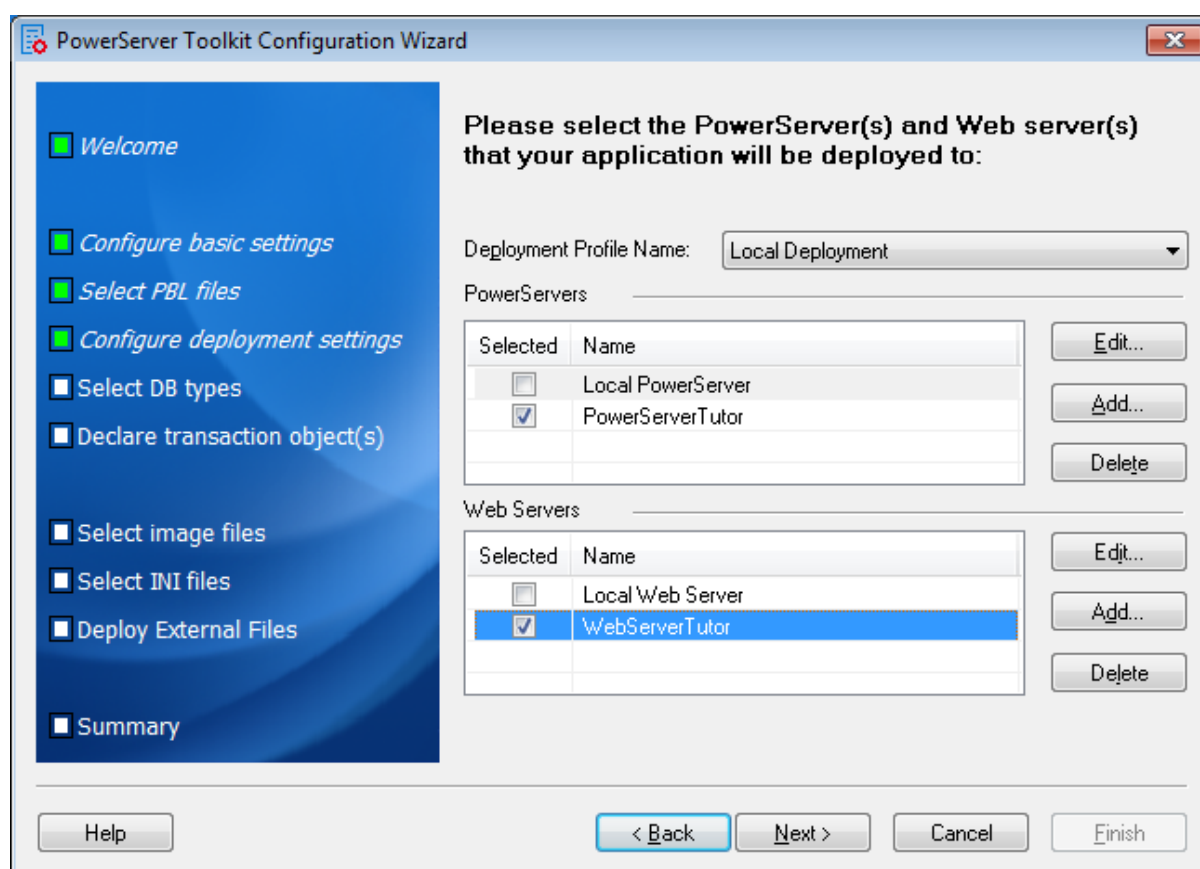
Step 1: In the Deployment Profile Name list box, select *Local Deployment*.

If no deployment profile is available, create one by following instructions in the Section 3.2.4.2, “Deployment profile settings” in *PowerServer Toolkit User Guide*.

Step 2: Select *PowerServerTutor* in the PowerServer list and select *WebServerTutor* in the Web Servers list.

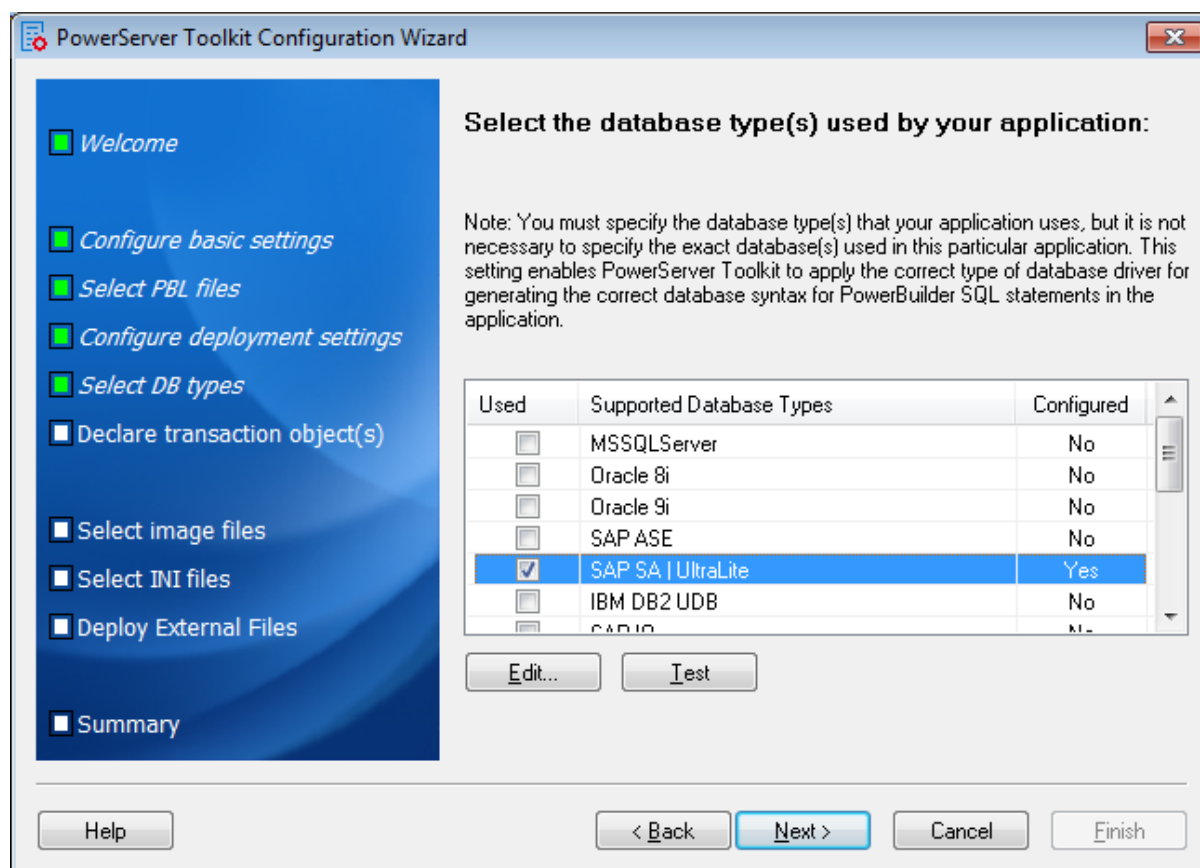
Step 3: Click **Next** to proceed.

Figure 6.21: Deployment Profile



6.3.4 Selecting DB Type(s)

Select the database type used by the application. In this tutorial, the database type is *SAP SA / UltraLite*. Select the Used column of *SAP SA / UltraLite*.

Figure 6.22: DB Type

If SAP SA | UltraLite is not configured (The Configured column is "No"), you should select it and click the **Edit** button to create a profile for it. The Database Type Profile Configuration window will be displayed.

Figure 6.23: Database Type Profile Configuration window

Step 1: Select the Outer Join Syntax that PowerBuilder will format from the drop-down list for Database in the Database Type Profile Configuration window.

Step 2: Specify the following settings for ODBC interface in the Database Type Profile Configuration window.

Table 6.4: Database type profile settings

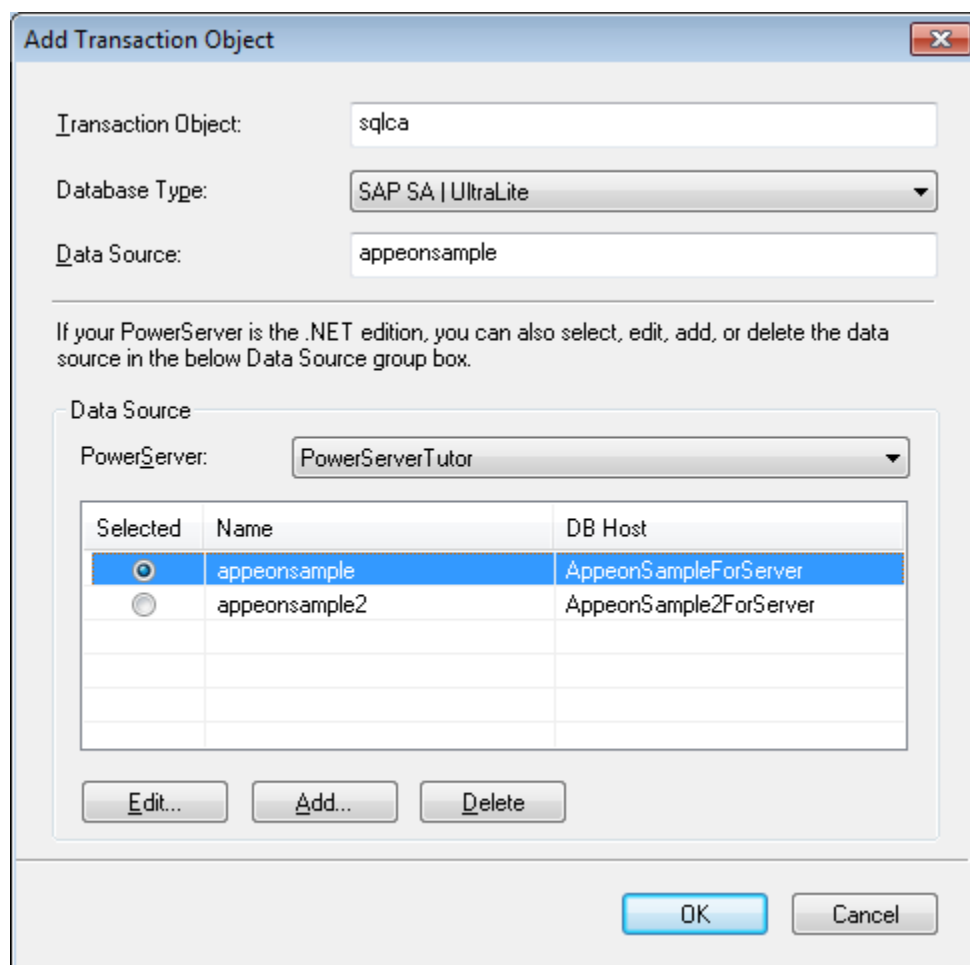
In this field	You should
ODBC Interface	Choose the ODBC Interface radio button.
Data Source	Choose ApeonSample from the dropdown list. If ApeonSample is not available, please choose the Apeon demo database (apeonsampleforserver or apeonsample2forserver), or any other SAP SQL Anywhere data source.
User ID	Leave this field blank.
Password	Leave this field blank.

Step 3: Click **Test** to verify that the connection is successful.

Step 4: Click **OK**.

6.3.5 Declaring transaction object(s)

When the tutorial PowerBuilder application is deployed to the Web, PowerServer handles the database connection using data sources rather than transaction objects defined in the

Figure 6.25: Data sources in the PowerServer

The following steps show you how to create a data source that links to the apeontutor ODBC data source using the ODBC driver.

Step 4: Click **Add** to create a data source. The Add dialog box is displayed.

Figure 6.26: Add a data source

The following table provides instructions for how to specify the data source settings.

Table 6.5: Instructions to specify data source settings

Settings	Instructions
Name	Input "appeontutor" as the name of the data source.
Driver	Select "ODBC Driver" as the driver type for the data source.
ODBC Data Source	Select "appeontutor" as the data source name. This will connect to the <i>appeontutor</i> ODBC data source that was created in Section 6.2.3, "Configuring ODBC data source" . If the database type is SAP SQL Anywhere and the database file resides in an NTFS folder, please grant the "NETWORK SERVICE" or "Everyone" user with full controls over that folder.

Settings	Instructions
User Name	Type "dba".
Password	Type "sql".

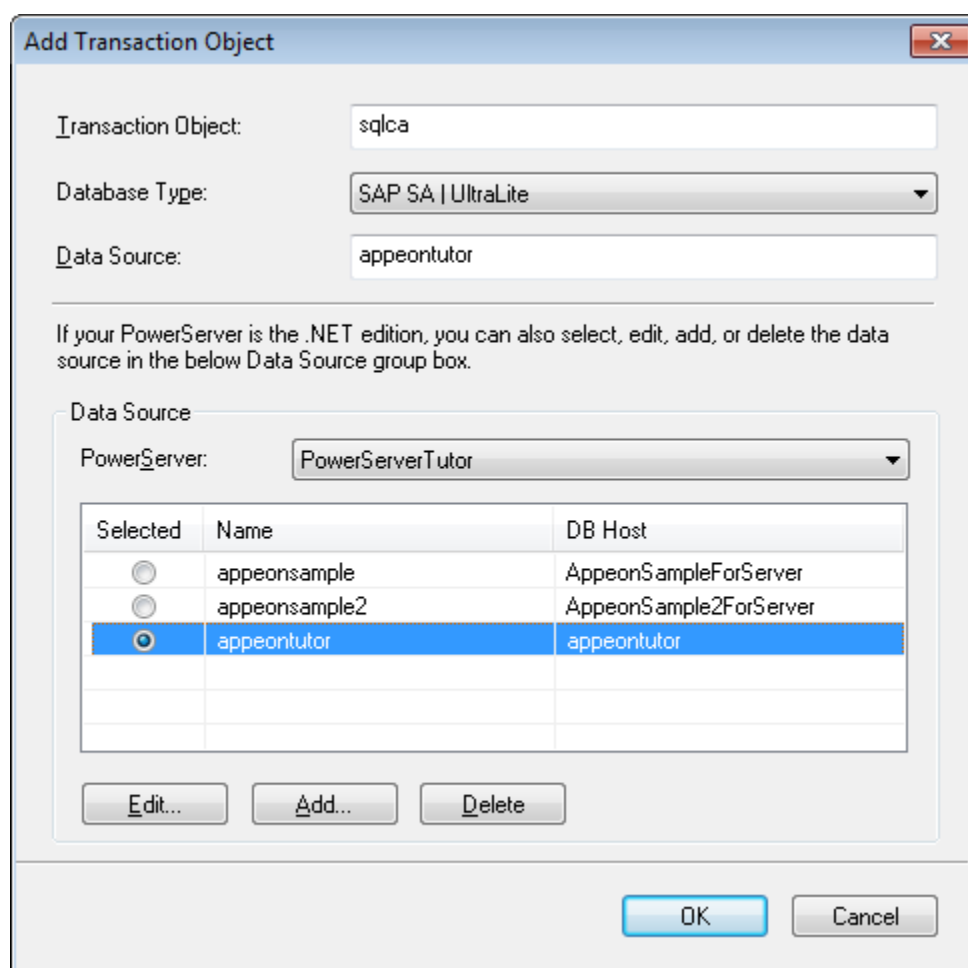
Step 5: Click **Test** to verify the connection to the *appeontutor* data source.

Step 6: Click **OK** to close the Add dialog box.

The *appeontutor* data source has been added.

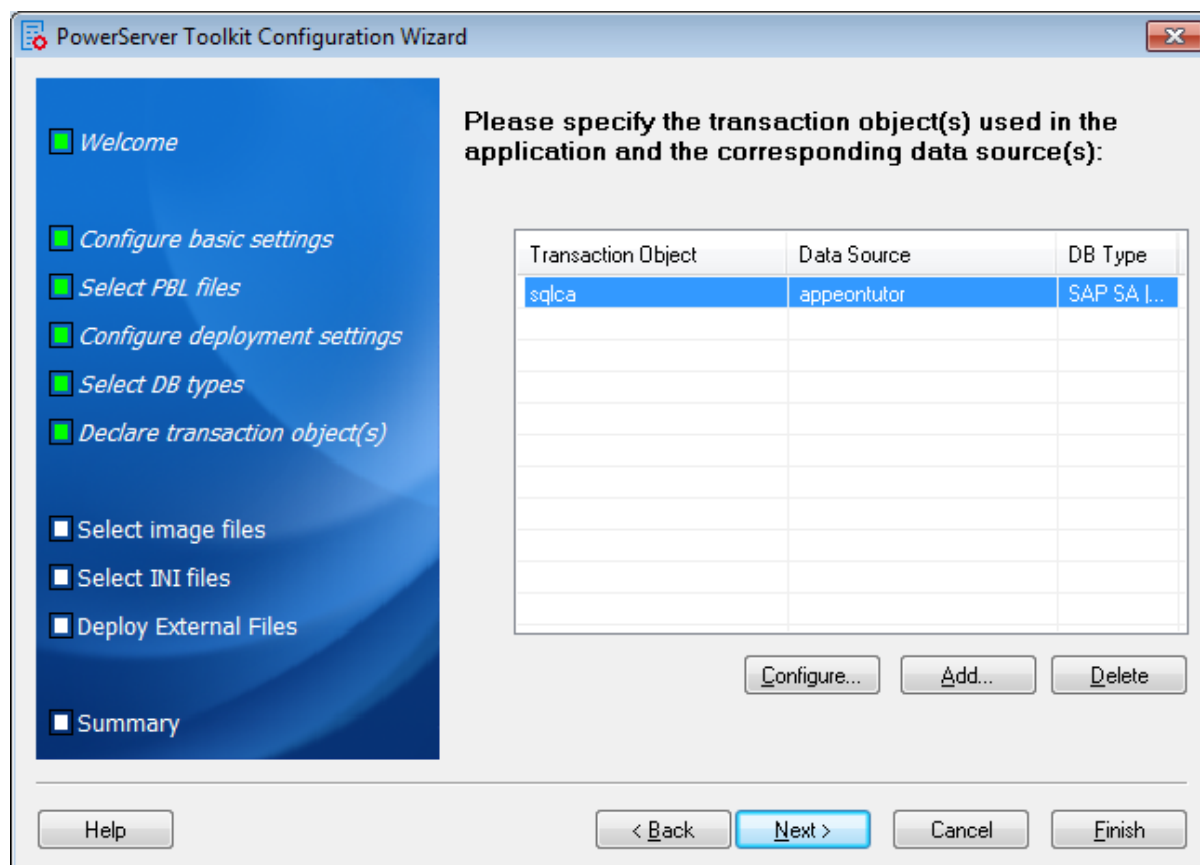
Step 7: Select the **Selected** button of the *appeontutor* data source.

Figure 6.27: appeontutor is added



Step 8: Click **OK** to close the Add Transaction Object dialog box.

The "sqlca" transaction object is added.

Figure 6.28: sqlca is added

Step 9: Click **Next** through the rest of settings, until you get to the **Summary** page.

Step 10: Click **Finish** on the **Summary** page to close the wizard.

The Apeon tutorial application is successfully configured.

The PowerServer Application Deployment Wizard will be launched, if you keep the "**Deploy the application now**" option on the **Summary** page as selected. Refer to [Section 6.5, "Deploying the Tutorial PowerBuilder Application"](#) for how to use it.

6.4 Analyzing Unsupported Features

This chapter shows you how to use the UFA tool to analyze the Apeon tutorial PowerBuilder application for unsupported features and optimize and full build the tutorial application.

In this chapter, you will learn how to:

- [Perform unsupported feature analysis on the tutorial application](#)
- [Optimize and full build the tutorial application](#)

6.4.1 Unsupported features analysis

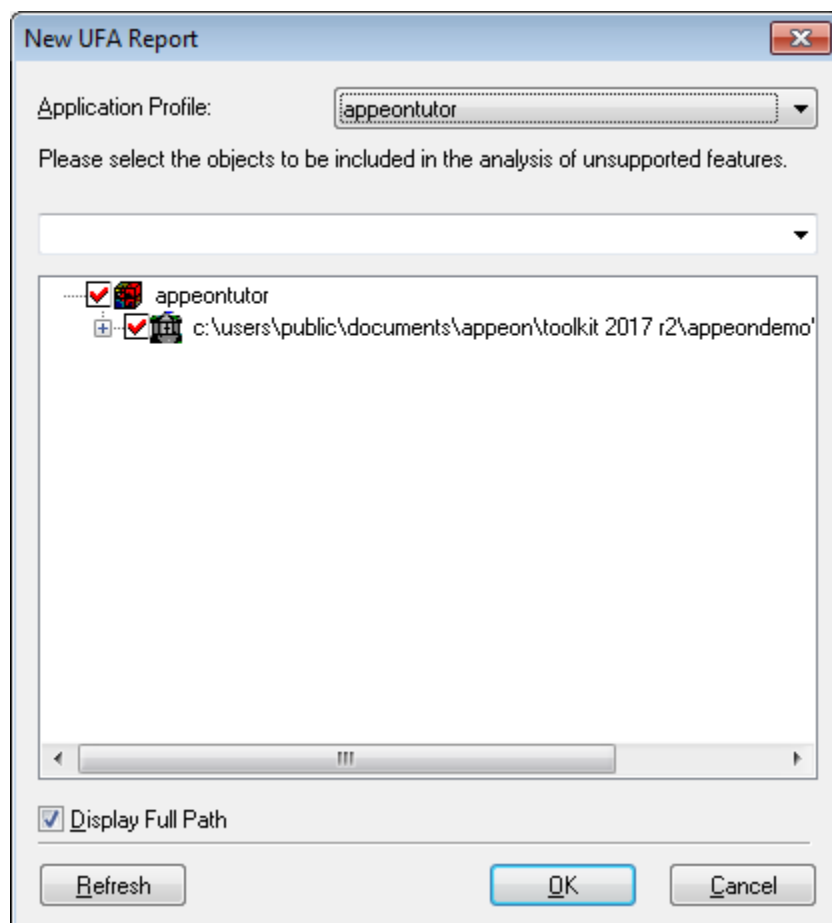
PowerServer provides an unsupported feature analysis (UFA) tool to handle the PowerServer unsupported source code in a PowerBuilder application.

To perform Unsupported Features Analysis to the tutorial application, perform the following steps:

Step 1: In the PowerBuilder IDE, click the **Analyze** button (🔍) on the PowerServer Toolkit. The Unsupported Feature Analysis Report window (UFA Report window) appears.

Step 2: Select the **File > New Report** menu in the UFA Report window. The New UFA Report dialog box will appear.

Figure 6.29: New UFA Report dialog box

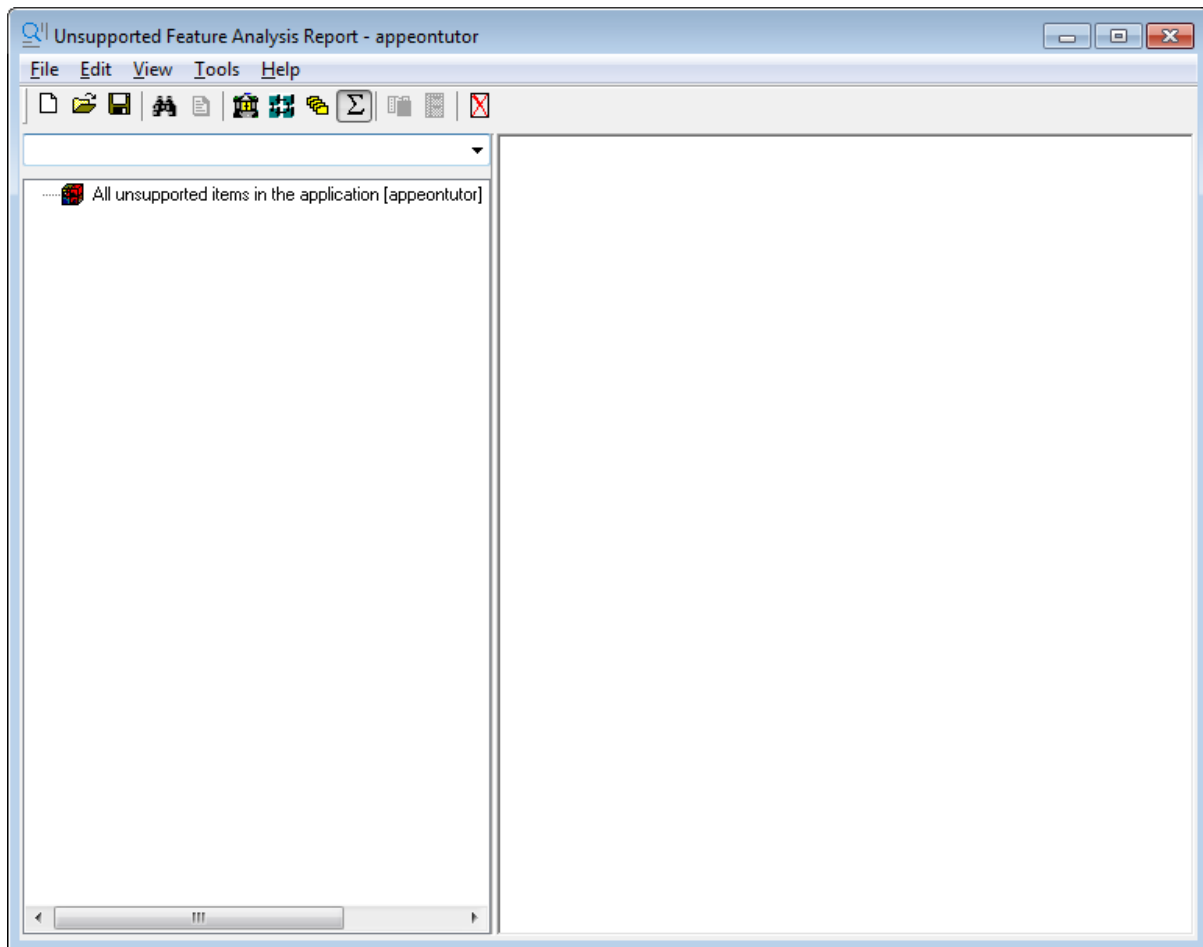


Step 3: Click the **OK** button to analyze the entire application.

The Unsupported Features Analysis Status dialog box is displayed.

After the feature analysis is completed, click the **Close** button and the UFA Report is loaded into the UFA Report Window. You can view the analysis result in the left treeview and modify the unsupported features by following the instructions in the Chapter 5, *Workarounds for Unsupported Features* in *Workarounds & APIs Guide*, which is also available at http://docs.apeon.com/apeon_online_help/workarounds_and_api_guide/. The Guide provides examples of some common unsupported features and ways to work around them.

In this tutorial, the application contains no unsupported features.

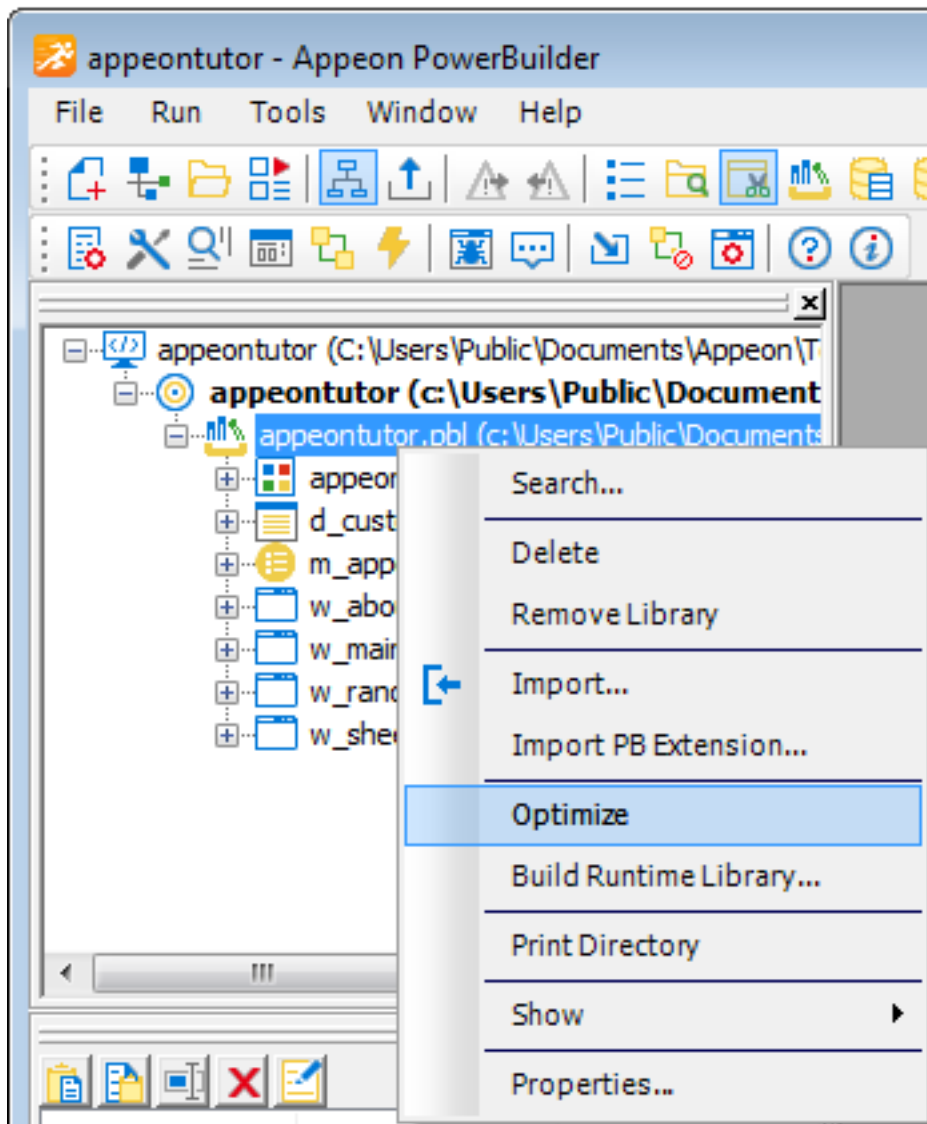
Figure 6.30: UFA Report window

6.4.2 Optimizing and full build

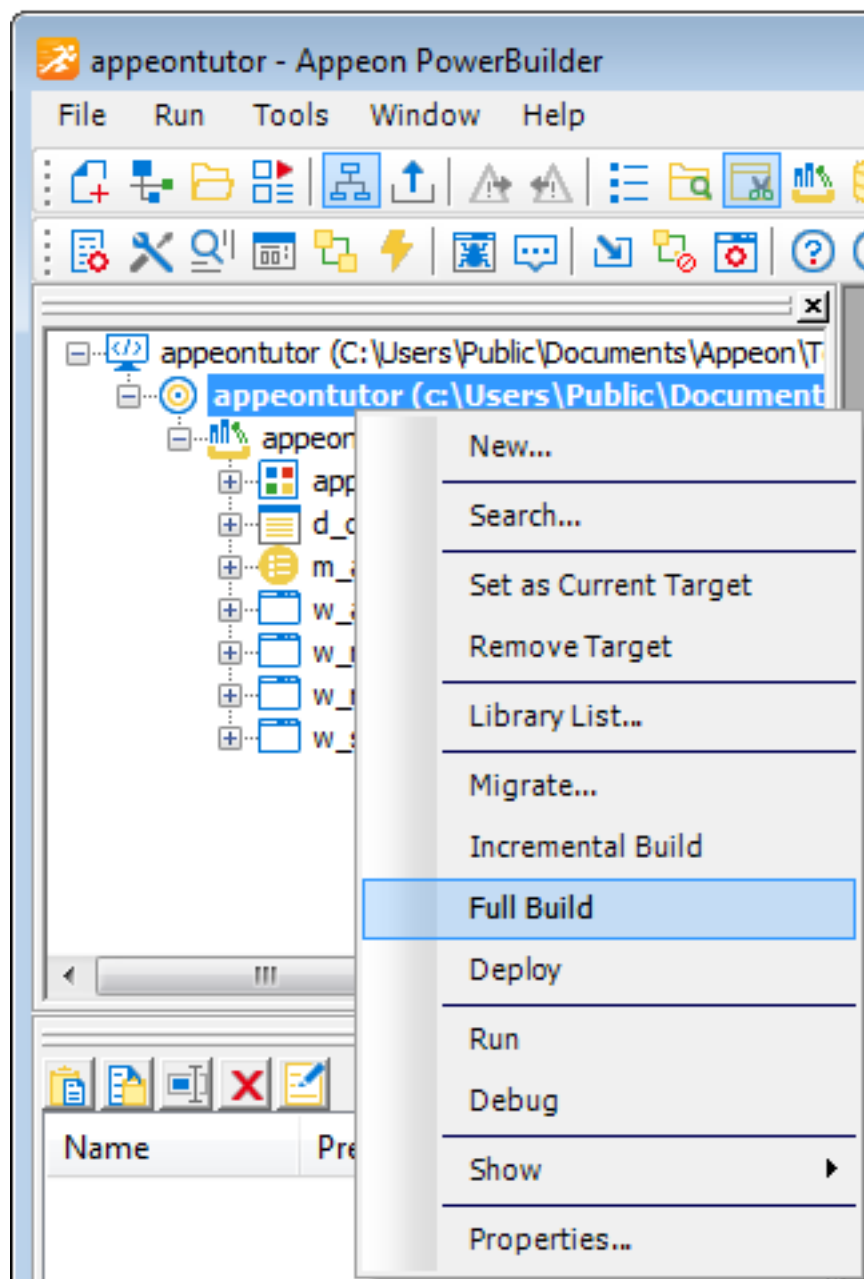
Before any PowerBuilder application is converted by PowerServer, ensure the PBLs have been optimized. You must also "full build" the PowerBuilder application successfully before PowerServer will be able to convert it.

To optimize and full build the tutorial application:

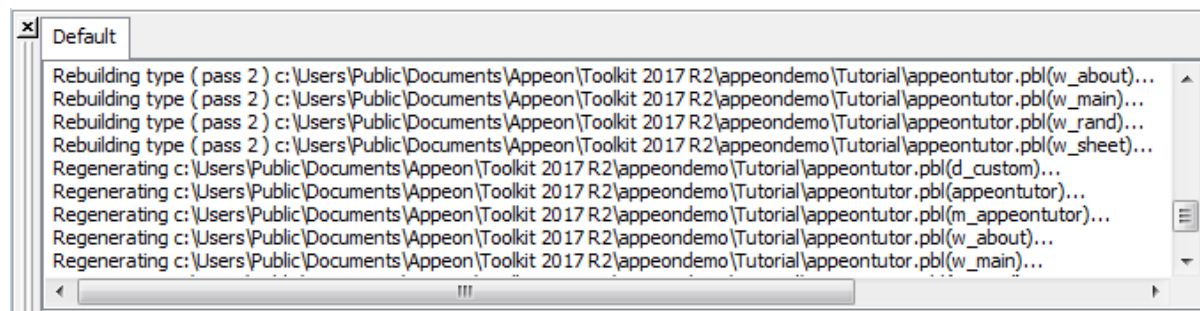
Step 1: Right click on *appeontutor.pbl* in the PowerBuilder system tree. Click **Optimize**.

Figure 6.31: Optimize

Step 2: Right click on the *appeontutor* Target in the system tree and select **Full Build** from the context menu.

Figure 6.32: Full Build

Step 3: The full build process begins. Some information is displayed in the Output window. Make sure that when the full build is complete, no error messages occur in the Output window.

Figure 6.33: Finished Full Build of target apeontutor

6.5 Deploying the Tutorial PowerBuilder Application

In the previous lessons, the Apeon tutorial PowerBuilder application has undergone careful analysis for unsupported features. Now you will launch the PowerServer Application Deployment Wizard to convert the tutorial application onto the Web automatically.

To deploy the tutorial application onto the Web automatically:

Step 1: Make sure PowerServer is started. For details, refer to [Section 6.3.3.1, “Starting PowerServer and Web server”](#).


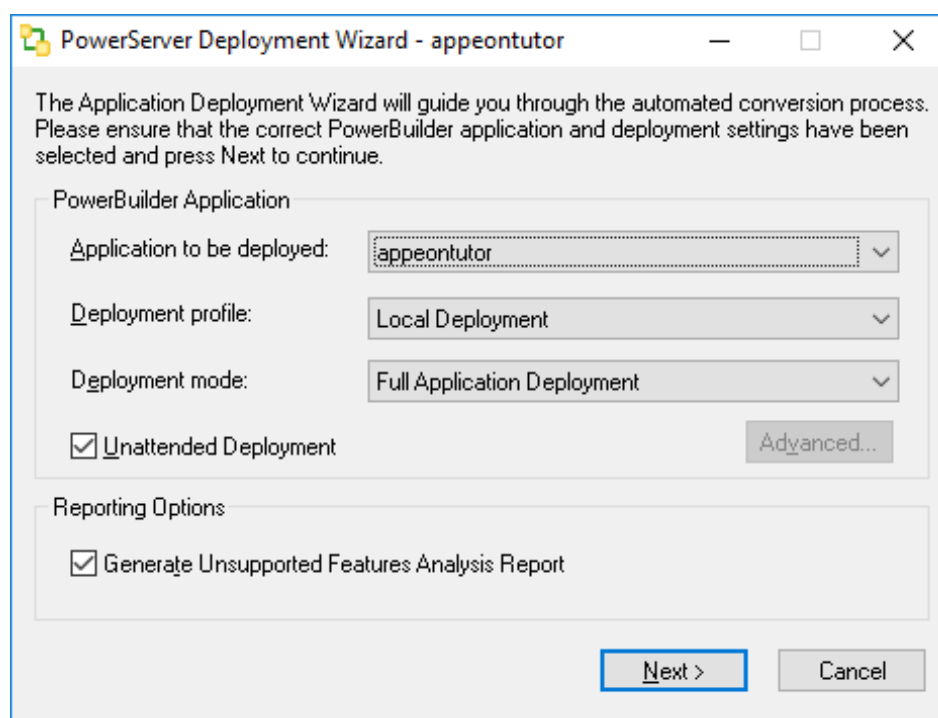
Step 2: Within the PowerBuilder IDE, click the **Deploy** button () in the PowerServer Toolkit. The Application Deployment Wizard starts.

Figure 6.34: PowerServer Application Deployment Wizard



The start page of the Wizard displays the following information:

- Application to be deployed - The current application (appeontutor) is selected by default from the list of applications that are defined in PowerServer Toolkit application profiles.
- Deployment profile - Make sure the deployment profile, *Local Deployment*, is selected.
- Deployment mode - Since it is the first time the tutorial application will be deployed, the *Full Application Deployment* mode will be selected by default. For subsequent deployments, the *Incremental Application Deployment* will be selected by default.
- Unattended Deployment - Check this option so that the entire deployment process is automatic.

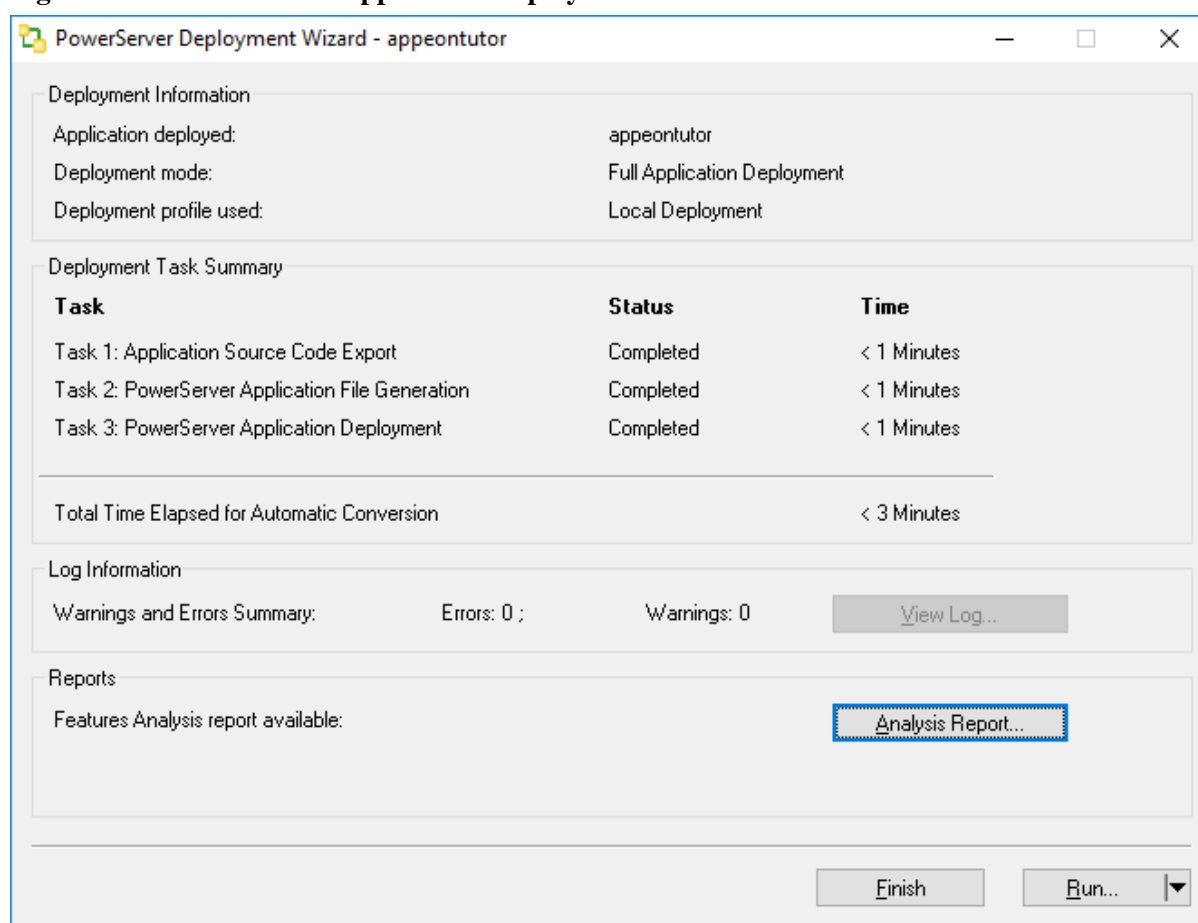
Step 3: Click **Next** to start the automatic Web deployment process. The PowerServer Application Deployment Wizard will perform three tasks:

1. PowerBuilder application source code export and analysis (Task 1: Application Source Code Export).
2. Parsing of the PowerBuilder source code (PBL files) into Web application files that are stored on the local machine (Task 2: PowerServer Application File Generation).
3. Transfer of local Web files to the Web Server by using either file copy or FTP, and uploads DataWindows to the PowerServer (Task 3: PowerServer Application Deployment).

When the deployment process is complete, the report dialog box is displayed.

Step 4: Click **Finish**.

Figure 6.35: PowerServer Application Deployment Wizard



6.6 Running the Web Application

The Apeon tutorial PowerBuilder application has been converted into a standard Web application with proper database connection which can now be run.

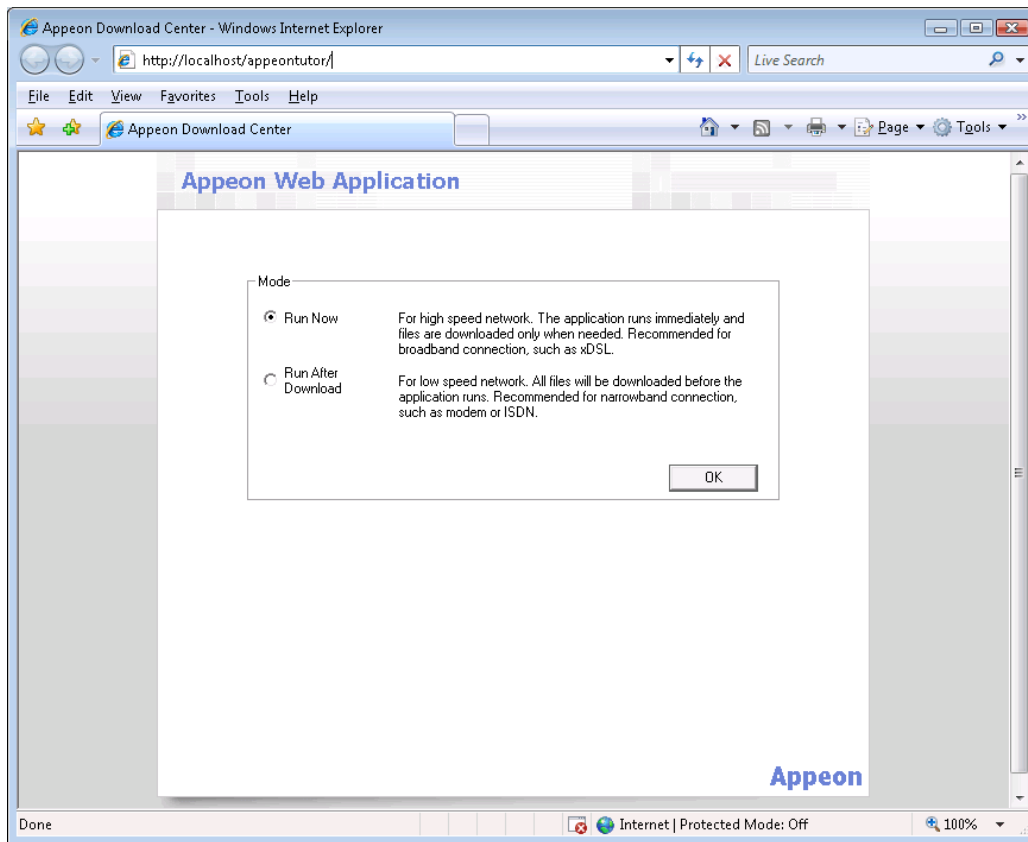
To launch the Web application after the tutorial application has been converted:

Step 1: Make sure PowerServer is running.

Step 2: Click the **Run** button (⚡) in the PowerServer Toolkit and select the *appeontutor* application in the popup window, or manually open a browser window and type `http://localhost/appeontutor/` into the Address bar and press **Enter**.

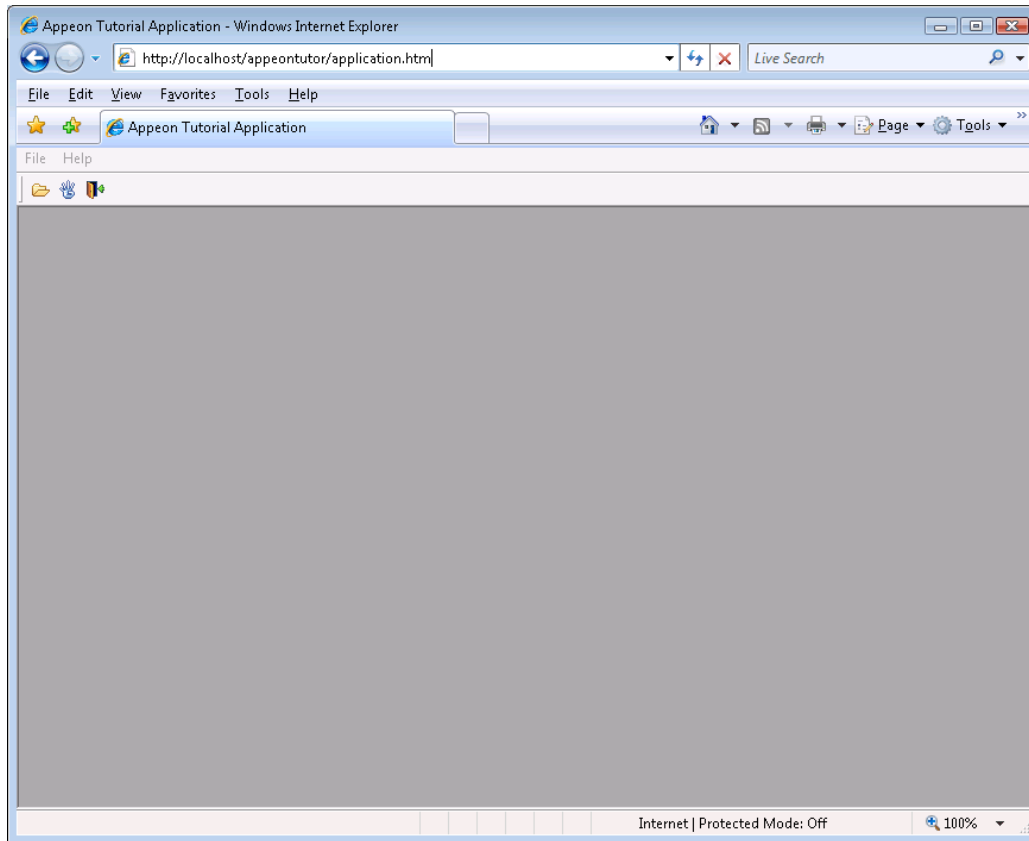
The Web application starts.

Figure 6.36: Application starts



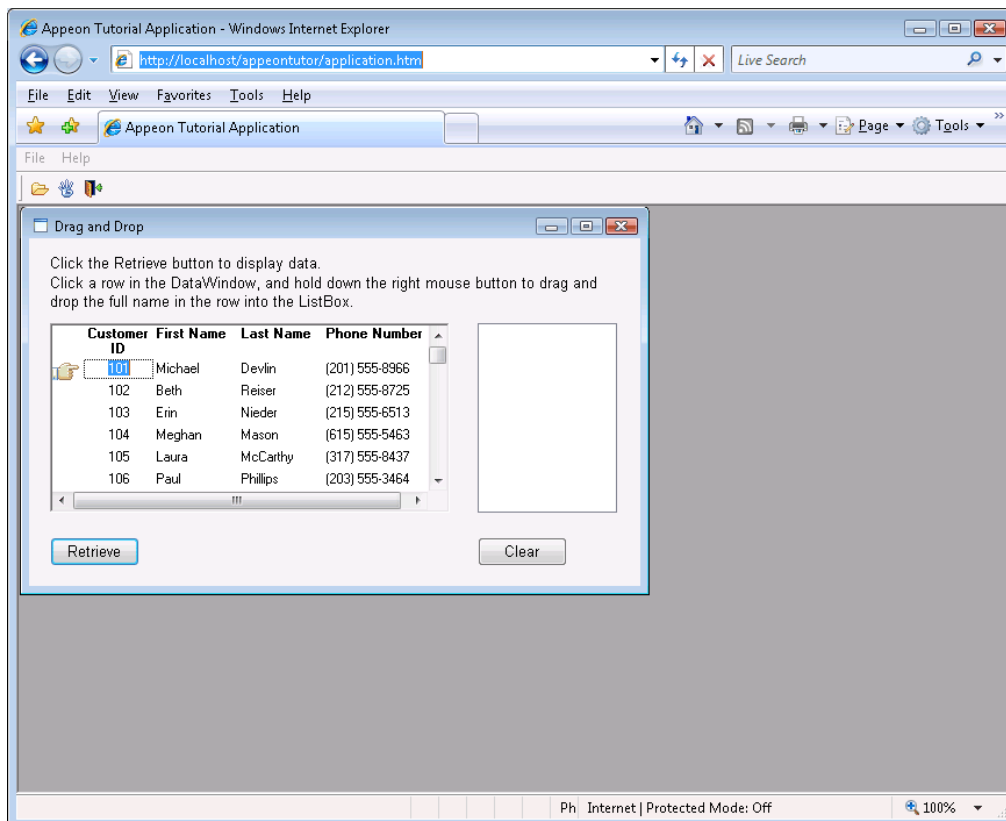
Click **OK** to run the Web application now.

Figure 6.37: Application starts



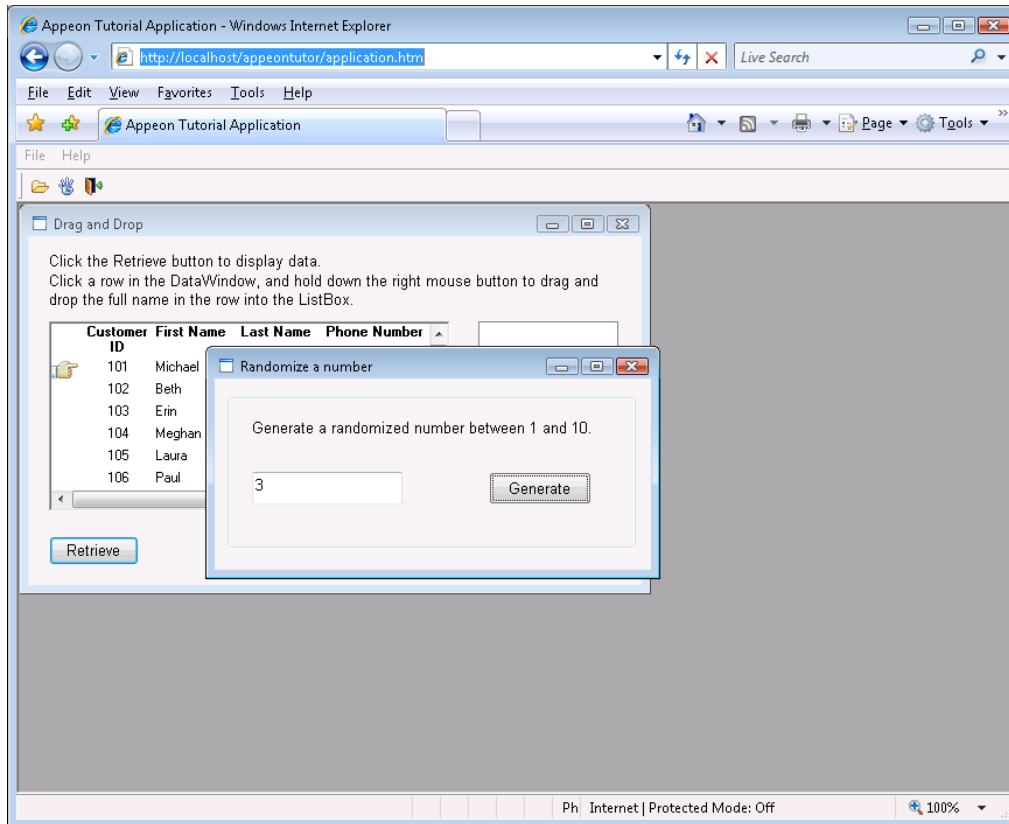
Step 3: Open the **Drag and Drop** window.

Figure 6.38: Drag and Drop



Step 4: Open the **Randomize a number** Window.

Figure 6.39: Randomize a number Window



Index

A

add a PowerServer profile, [56](#)
add a Web server profile, [59](#)
advantages of PowerServer for Web RAD
 best Web GUI, [1](#)
 fastest path to the Web, [1](#)
 leverage current organizational skills, [1](#)
 minimize business risks, [1](#)
analyze unsupported features, [69](#)
Apeon client functions, [23](#)
 description, [24](#)
 use, [26](#)
apply PowerServer open interfaces, [23](#)

B

basic principles for modularizing an application, [19](#)
basic requirements for rewriting complex applications, [19](#)
benefits of modularizing application, [19](#)

C

classify application into types, [17](#)
configure basic settings, [54](#)
configure deployment settings, [55](#)
configure ODBC data source, [48](#)
configure PowerServer Toolkit, [53](#)
convert the different application types, [18](#)
create a workspace, [44](#)

D

debug deployed applications, [14](#)
declare transaction object, [64](#)
define migration objective, [7](#)
define migration objective for non-PFC application, [7](#)
define migration objective for PFC application, [8](#)
deploy the Tutorial PowerBuilder Application, [74](#)
different application types, [18](#)

E

enhance an application with Web or PowerServer features, [21](#)
enhance the application with Web or PowerServer features, [13](#)

F

feature modification methods, [13](#)
fine-tune the runtime performance, [14](#)

G

getAllClients, [22](#)
getAllSessions, [22](#)
getSessionByID, [22](#)
getSessionCount, [22](#)
go live, [15](#)
go-live deployment, [14](#)

I

identify unsupported features, [12](#)
install required softwares, [5](#)
integrate PowerServer Web applications with JSP/ASP, [27](#)

K

killAllSessions, [23](#)
killSessions, [23](#)

L

leveraged functionality with server components, [2](#)
load an application in SAP Sybase Enterprise Portal, [26](#)
load the tutorial PBL file, [46](#)
load the Tutorial PowerBuilder Application, [44](#)

M

Migration Process, [4](#)
Migration Tutorial, [42](#)
modify unsupported features, [12](#)
modularization process, [19](#)
modularize application, [19](#)

O

optimize and full build, [71](#)

P

PFC modification
 differently behaved features, [38](#)
 unsupported calls, [36](#)
 unsupported Function not founds, [35](#)
 unsupported object event, [35](#)
 unsupported object functions, [34](#)
 unsupported object properties, [30](#)
 unsupported Shared variables, [35](#)

unsupported standard class objects, [29](#)
unsupported system functions, [35](#)
unsupported user objects, [28](#)

PowerServer open interfaces

apply, [23](#)
description, [22](#)
getAllClients, [22](#)
getAllSessions, [22](#)
getSessionByID, [22](#)
getSessionCount, [22](#)
killAllSessions, [23](#)
killSessions, [23](#)
rollbackAllTransactions, [23](#)

PowerServer Web RAD methodology, [2](#)
pre-configure for the Web applications, [11](#)
prepare the target application, [10](#)
process application based on migration objectives, [10](#)
Processing PFC Applications, [28](#)

R

rapidly build application with PowerServer, [17](#)
restrictions on supporting Enterprise Portal, [26](#)
rollbackAllTransactions, [23](#)
run the tutorial application, [50](#)
run the Web Application, [75](#)

S

select a deployment profile, [62](#)
select DB Type, [62](#)
select PBL file, [55](#)
Single sign-on, [27](#)
special deployment steps for distributed applications, [14](#)
special processing required for PFC applications, [10](#)
start PowerServer and Web server, [56](#)
support external resources, [17](#)

T

traditional approach to Web development, [1](#)
trial deployments and debugging, [13](#)

U

understand the general limitations of the PowerServer solution, [6](#)
unsupported features analysis, [69](#)

upgrade obsolete PBLs, [9](#)
upgrade PFC applications, [9](#)
upgrade the original application, [9](#)

W

Web RAD with PowerBuilder and PowerServer, [1](#)