A Demo Project Using PowerBuilder 2022 R3

TABLE OF CONTENTS

1. INTRODUCTION
   1. INTRODUCTION TO POWERBUILDER AND PROJECT
   2. FUNCTIONAL AND NON FUNCTION REQUIREMENTS
2. DATABASE IN SQL Central (SQL Anywhere 17)

2.1 CREATING DATABASE IN SQL CENTRAL

2.2 CONNECTING THE DATABASE TO LOCAL SYSTEM

2.3 CREATING TABLE CUSTOMERS

2.4 CREATING TABLE PRODUCTS

1. BUILDING A CLIENT SERVER APPLICATION

3.1 BUILDING A LOGIN WINDOW

3.2 CONNECTING TO THE DATABASE

* 1. CREATING ANCESTOR WINDOW

3.4 SETTING UP THE MENUS

3.5 BUILDING DATA WINDOW OBJECTS AND ATTACHING

3.6 RUNNING THE DEBUGGER

* 1. PREPARING THE APPLICATION FOR DEPLOYMENT

1. AUTOMATING BUILD AND DEPLOY

4.1 CREATE BRACHES SUCH AS MASTER, PRODUCTION, QA AND FEATURES WITH POWER BUILDER SOURCE CONTROL

4.2 PULLING CODE FROM GITHUB USING A BATCH SCRIPT

1. **INTRODUCTION** 
   1. **INTRODUCTION TO POWERBUILDER AND PROJECT**

PowerBuilder is an enterprise development tool that allows you to build many types of applications and components. It is one of a group of Appeon products that together provide the tools to develop client/server, multitier, and Internet applications. A PowerBuilder client application can contain:

* A user interface

Menus, windows, and window controls that users interact with to direct an application.

* Application processing logic

Event and function scripts in which you code business rules, validation rules, and other application processing. PowerBuilder allows you to code application processing logic as part of the user interface or in separate modules called custom class user objects.

**PowerBuilder applications are event driven**

In a client application, users control what happens by the actions they take. For example, when a user clicks a button, chooses an item from a menu, or enters data into a text box, one or more events are triggered. You write scripts that specify the processing that should happen when events are triggered.

Windows, controls, and other application components you create with PowerBuilder each have a set of predefined events. For example, each button has a Clicked event associated with it and each text box has a Modified event. Most of the time, the predefined events are all you need. However, in some situations, you may want to define your own events.

**PowerScript language**

You write scripts using PowerScript, the PowerBuilder language. Scripts consist of PowerScript commands, functions, and statements that perform processing in response to an event.

For example, the script for a button's Clicked event might retrieve and display information from the database; the script for a text box's Modified event might evaluate the data and perform processing based on the data.

The execution of an event script can also cause other events to be triggered. For example, the script for a Clicked event in a button might open another window, triggering the Open event in that window.

**PowerScript functions**

PowerScript provides a rich assortment of built-in functions that can act on the various components of your application. For example, there is a function to open a window, a function to close a window, a function to enable a button, a function to update the database, and so on.

You can also build your own functions to define processing unique to your application.

**Object-oriented programming with PowerBuilder**

Each menu or window you create with PowerBuilder is a self-contained module called an object. The basic building blocks of a PowerBuilder application are the objects you create. Each object contains the particular characteristics and behaviours (properties, events, and functions) that are appropriate to it. By taking advantage of object-oriented programming techniques such as encapsulation, inheritance, and polymorphism, you can get the most out of each object you create, making your work more reusable, extensible, and powerful.

**Installable cloud apps**

PowerBuilder provides the PowerServer Toolbar to help you deploy your PowerBuilder applications as installable cloud applications.

Installable cloud applications are deployed to the server and installed and run over the Internet. It is made up of the client side (most of which are the same as in PowerBuilder traditional client/server applications) and the server side (which are industry-standard REST APIs for accessing the database and services).

**Multitier applications**

PowerBuilder lets you build applications that run in a distributed computing environment. A multitier application lets you:

* Centralize business logic on servers, such as JBoss, WebLogic, WebSphere, or COM+
* Partition application functions between the client and the server, thereby reducing the client workload
* Build scalable applications that are easy to maintain

**Database connectivity**

PowerBuilder provides easy access to corporate information stored in a wide variety of databases. Data can be accessed through the PowerBuilder ODBC interface, or through a native or direct connection to a database.

PROJECT:

This project aims to create a simple database application with customers and products.

* 1. FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS

Functional Requirements  
 - User authentication via a login window.  
 - Create, read, update, delete operations for customers and products.  
 - Menu-driven navigation.  
 - Database integration for storing and retrieving data.  
  
 Non-Functional Requirements  
 - Responsive and user-friendly interface.  
 - Secure data handling.  
 - Efficient query processing.  
 - Scalable architecture for future enhancements.

1. DATABASE IN SQL Central (SQL Anywhere 17)

SQL Central is a graphical tool for managing your database servers, databases, and the objects they contain.

* 1. CREATING DATABASE IN SQL CENTRAL

Open SQL central in the local system > navigate to Tools > click on create database > create database on this computer > browse a folder for creating the database > provide the username and password > set the minimum password length and finish.

* 1. CONNECTING THE DATABASE TO THE LOCAL SYSTEM

Select connect SQL Anywhere 17 > give database username and password that is created > browse the file where database is present and connect.

* 1. CREATING THE TABLE CUSTOMERS

provide the column names such as id, first name, last name, city and state and mention their data types and sizes.

Queries:

INSERT INTO table customers (“id”,”first name”, “last name”, “city”, “state”) VALUES (‘provide the necessary values’)

* 1. CRAETING THE TABLE PRODUCTS

Provide the column names such as id, product name, price, availability

Queries:

INSERT INTO table products (“id”,”first name”, “last name”, “city”, “state”) VALUES (‘provide the necessary values’)

1. BUILDING A CLIENT SERVER APPLICATION

**Create a new workspace:** Create a new file and save

**Create a target:** Existing/Template application > give the name to sheets:sheet1: w\_customers, sheet2: w\_products > Template type: MDI > Give name for the sheets >give to do list and finish

[Specify an icon for the application](https://docs.appeon.com/pb2025/getting_started/XREF_76881_Specify_an_icon.html) if necessary.

Change the size of the main window as required.

3.1 BUILDING A LOGIN WINDOW

**Create a new window(w\_welcome):** File > new >window and click ok. Customize as required.

**Add controls to the window:** Insert > control > picture: static text(st\_userid,st\_password), single line edit(sle\_userid,sle\_password), command button(cb\_ok, cb\_cancel) is used for taking the inputs from user.

Change the tab order on the window: mention the tab order (sle\_passeord=20,cb\_ok=30,cb\_cancel=40)

Write the script to open the window(w\_genapp\_frame)

//\*-----------------------------------------------------------------\*/

//\* open: Create sheet manager and post event

//\*-----------------------------------------------------------------\*/

string ls\_sheets[]

/\* Create an instance of the sheet manager \*/

inv\_sheetmgt = Create n\_genapp\_sheetmanager

this.Post Event ue\_postopen ( )

//open login window

open(w\_welcome)

3.2 CONNECTING TO A DATABASE

**Look at the Demo Database:**

DB profile >ODBC > new > Userid and Password for profile, preview and test the connection to success

DB painter> click on view> objects >drag and drop the tables. Right click on the table to alter.

**Run the Connection Object wizard**: from to-do list>run connection object wizard for more connections>select the database and finish.

Declare a global variable:

n\_genapp\_connectservice and constructor () in down drops for connection information.

Function and of\_getconnectioinfo, of\_connectDB, of\_disconnectDB down drop to check if it is as intended.

Declare and Instance Variables down drop to check if it is as intended.

N\_genapp\_connectservice from second drop down and drag and drop- n\_genapp\_connectservice gnv\_connect and save.

**Modify the connection information:**

Function and of\_getconnectioninfo and comment the as\_userid, as\_dbpass, as\_dbparam

Add script for w\_welcome cb\_ok

// 1) Instantiate the Transaction object

//Local variable declarations

string ls\_database, ls\_userid, ls\_password

//Assignment statements

ls\_userid = Trim ( sle\_userid.text )

ls\_password = Trim ( sle\_password.text)

if ls\_userid = "" or ls\_password = "" then

MessageBox("Login Error", "User ID and Password cannot be Empty.")

return

end if

if ls\_userid="db1" and ls\_password ="Akshaya@123" then

ls\_database="ConnectString='DSN=akshaya\_db1;UID=db1;PWD=Akshaya@123'"

//Instantiate with user-entry values

SQLCA.userid = ls\_userid

SQLCA.dbpass = ls\_password

SQLCA.dbparm = ls\_database

gnv\_connect = CREATE n\_genapp\_connectservice

if gnv\_connect.of\_ConnectDB ( )= 0 then

bLoginSuccess = true

MessageBox("login success", "welcome," + ls\_userid)

open(w\_genapp\_frame)

close(w\_welcome)

else

MessageBox("login failed","unable to connect to database")

end if

else

MessageBox("Login Error","Invalid Username or Password")

end if

**Complete the login and logout scripts:**

Add script for cancel

HALT

Add script for application genapp close()

gnv\_connect.of\_DisconnectDB ( )

HALT

**Run the application**

3.3 CREATING ANCESTOR WINDOW

Add a library to the search path if required from the genapp target>right click, select the library and add to the search path.

**Create a new ancestor sheet window**: file >inherit > window (w\_genapp\_basesheet)>ok

**Create a new sheet window inheritance hierarchy**: follow the same above method for both customer (w\_customers) and products(w\_products).

**Add a Data Window control for the master Data Window:** file > new >standard visual type > Data Window > u\_dwstandard. Drag and drop the user object(u\_dwstandard) to w\_maste\_detail\_ancestor and add vscrollbar(dw\_master).

**Add a Data Window control for the detail Data Window**: .Drag and drop the user object(u\_dwstandard) to w\_maste\_detail\_ancestor(dw\_detail)

**View the scripts inherited from the user object**: dw\_detail, dberror, u\_dwstandard)

Add user events and event scripts:

W\_master\_detail\_ancestor and new event in the drop down

**Event name:** ue\_retrieve **Argument type:** integer

Script:

IF dw\_master.Retrieve() <> -1 THEN

dw\_master.SetFocus()

dw\_master.SetRowFocusIndicator(Hand!)

END IF

Event name: ue\_insert Argument type: integer

Script:

dw\_detail.Reset()

dw\_detail.InsertRow(0)

dw\_detail.SetFocus()

dw\_master.InsertRow(0)

**Event name:** ue\_update **Argument type:** interger

Script:

IF dw\_detail.update() =1 AND dw\_master.update() = 1 THEN

COMMIT using SQLCA;

MessageBox("Save","Save succeeded")

this.event ue\_retrieve()

ELSE

if sqlca.sqlcode <> 0 then

messagebox("error","ID already exists")

else

messagebox("error","update failed. sqlcode: " + string(sqlca.sqlcode))

end if

rollback using sqlca;

return

END IF

**Event name:** ue\_delete **Argument type:** integer

Script:

dw\_master.deleteRow(0)

messagebox("success","deletion successful")

messageBox("deletion","kindly update")

dw\_detail.Reset()

dw\_detail.InsertRow(0)

dw\_detail.SetFocus()

**Add scripts to retrieve data for the Data Window controls:**

Dw\_master, rowfocuschanged(), w\_master\_detail\_ancestor from drop down

Script

long ll\_itemnum

ll\_itemnum = this.object.data[currentrow, 1]

IF dw\_detail.Retrieve(ll\_itemnum) = -1 THEN

MessageBox("Retrieve","Retrieve error-detail")

END IF

**Adjust a runtime setting for sheet window size:**

n\_genapp\_sheetmanager > function, of\_opensheet from the down drop

change the cascaded! To original!

li\_rc = OpenSheet ( lw\_sheet, as\_sheetname, w\_genapp\_frame, 0, Original! )

3.4 SETTING UP THE MENUS

**Modify the frame menu:** m\_genapp\_frame in  WYSIWYG view> file > right click on new > edit to &report and check if the report is visible.

Inherit > menus (m\_genapp\_sheet)

**Create a new sheet menu**: edit in WYSIWYG view > right click >insert items in the end

|  |  |
| --- | --- |
| Menu item | MicroHelp Text |
| &Insert | Insert a row |
| Upd&ate | Update the database |
| &Delete | Delete the current row |

Add menu scripts to trigger user events: double click on the items in WYSIWYG view

|  |  |
| --- | --- |
| Menu name | Script for clicked event |
| m\_edit.m\_insert | w\_master\_detail\_ancestor lw\_activesheet  lw\_activesheet = w\_pbtutor\_frame.GetActiveSheet()  lw\_activesheet.EVENT ue\_insert() |
| m\_edit.m\_update | w\_master\_detail\_ancestor lw\_activesheet  lw\_activesheet=w\_pbtutor\_frame.GetActiveSheet()  lw\_activesheet.EVENT ue\_update() |
| m\_edit.m\_delete | w\_master\_detail\_ancestor lw\_activesheet  lw\_activesheet = w\_pbtutor\_frame.GetActiveSheet()  lw\_activesheet.EVENT ue\_delete() |

Attach the new menu and run the application: from w\_master\_detail\_ancestor> in properties attach m\_my\_sheet in the menus list box and run the application.

3.5 BUILDING DATA WINDOW OBJECTS AND ATTACHING

**Create and preview a new Data Window object:** file > new > Data Window tab > Tabular > Quick select > add the necessary table and columns (customers and products table created from SQL Anywhere) > finish and save (d\_custlist) and continue the same for product. (d\_prodlist)

Save the Data Window object

**Make cosmetic changes to the first Data Window object:** make necessary changes such as aligning the texts for both (d\_custlist and d\_prodlist)

**Create a second Data Window object:** file > new > Data Window tab > free form > SQL select > add the necessary table and columns (customers and products table created from SQL Anywhere) > finish and right click on the header area of the table > select all > design > retrieval arguments > give (id). Specify WHERE clause from ther where tab > select “customer”,”id”. Right click on the value > select argument as (:id). Return to the painter from meny > give the options required > finish > provide the id when prompted to specify retrieval arguments. Follow the same for products and save the Data Window (d\_customers, d\_products)

**Make cosmetic changes to the second Data Window object**: Arrange the columns as required.

**Attach the Data Window object to the master Data Window control**: dw\_master above in w\_customers > properties view > d\_custlist and d\_prodlist in the Data Window list box.

**Attach the Data Window object to the detail Data Window control:** dw\_detail below in w\_customers > properties view > d\_custlist and d\_prodlist in the Data Window list box.

**Run the application**

3.6 RUNNING THE DEBUGGER

**Add breakpoints in application scripts**: Debug button > double click on any lines as below

This.ToolBarSheetTitle = “MDI Application Toolbar”

gnv\_connect = CREATE n\_genapp\_connectservice (cb\_ok of welcome)

IF dw\_detail.Retrieve(ll\_itemnum) = -1 THEN(rowfocuschanged of dw\_master)

**Run in debug mode:** click on start button > step in > step over > step out > continue.

**Set a watch and a conditional breakpoint:** check everything in watch tab, local tab and continue. Again break points tab and exit the debug mode.

3.7 PREPARING THE APPLICATION FOR THE DEPLOYMENT

**Create the Project object:** file > new > project > application > required options > finish

**Create the executable file:** right click on genapp target > properties > deploy and check if everything is fine. Click on deploy to deploy the created application.

**Create a shortcut:** if required to the local system.

**Test the executable:** file to ensure that the application works as indended.

1. AUTOMATING THE BUILD AND DEPLOY

Use PowerBuilder’s PBAutoBuild220.exe to automate builds. Insatall the PBAutoBuild220.exe from the Power Builder Installer.

* 1. CREATE BRACHES SUCH AS MASTER, PRODUCTION, QA AND FEATURES WITH POWER BUILDER SOURCE CONTROL
* Open PowerBuilder and navigate to workspace and right click and Source Control.
* Select Git as the source control provider.
* Configure the repository settings:
* Enter the Git Repository URL.
* Provide authentication details if required.
* Choose a local working directory.
* Click Connect to establish the link with Git.
* Use Check-in/Check-out options within PowerBuilder to manage code changes.
* Add the features for the application in the feature branch and merge it with QA branch and test the application with build script by pulling from the QA branch.
* Once the QA branch works fine, merge it with the production branch and test the application with build script bt pulling from the production branch.
* Once everything is works as intended, merge it with master/main branch.

By integrating Git with PowerBuilder, developers can track changes, collaborate efficiently, and maintain a structured version history while automating the build and deployment process. Right click on the workspace to create the necessary branches.

4.2 PULLING CODE FROM GITHUB USING A BATCH SCRIPT

In this project, a batch script is used to automate pulling the latest code from a GitHub repository.   
  
SAMPLE BATCH SCRIPT  
1. Create a new batch script file (`Build.bat`).  
2. Add the following script to pull the latest code:

@echo off

cd path\to\git local repo

echo pulling the code from github...

git pull origin master

if %errorlevel% neq 0 (

echo pull got failed!

pause

)

echo Building power builder application...

"path\to\pbautobuild" /f "path\to\json file"

if %errorlevel% neq 0 (

echo power bulider Build failed!

pause

)

echo Build Successful. Deploying...

xcopy "path\of\exe file" "path\to\DeploymentFolder" /Y

Xcopy "C:\Users\Akshaya\Documents\Appeon\PowerBuilder 22.0\genapp.pbt" "C:\Users\Akshaya\Documents\Appeon\PowerBuilder 22.0\DeployedApp\_before\_merge" /Y

echo Deployment complete!

if %errorlevel% neq 0 (

echo power builder deployment failed!

pause

)

The above script can be modified for QA as well as Production   
Save the file and run it before building the application to ensure the latest code is used. This method helps keep the local workspace synchronized with the GitHub repository, minimizing conflicts and ensuring all changes are up to date.In this project, PowerBuilder's Source Control feature is used to connect to Git for version management.