Table of Contents

Table of Contents	1
License Agreement	3
Trademarks and Copyright	4
Creating A Form Based Application	5
Printing Hello World	5
Writing Code File	5
Procedural Project	6
Printing Hello World in Console	9
Printing Hello World in Desktop Form	
Creating a Complex Program	14
Creating Packages And Using Them In Your Applications	
Creating Databases And Tables	
Creating Table In A Desktop Form Project	
Using Visual Query(PQuery)	
Using A Database Project	41
Using Database Project	44
Using The PGird And PDBGird	45
Creating a PGird Object	45
Creating a PDBGird object	
Using The Report Builder	
Using PValue, PList, PValueList And PObject Objects	65
Using PList	65
Using PValue	75
Using PValueList	80
Using PObject	
Creating A Game Visually	
The Subtleties Of The Visual Programming	

Loops and Conditions: For Loop	101
Loops and Conditions: While Loop	110
Loops And Conditions: ForEach Loop	118
Loops and Conditions: If Condition	
Loops and Conditions: Else Condition	134
Loops and Condition: Else-If Condition	143
PVariable	152
PMethod	157
Creating Your Own Components And Distributing Them	163
PPL Source Code	163
XML Definition Files	169
Generate the Component Package	170
More In-Depth Object Binding	171
PDirectory, PFile And PFileList Example	177
Displaying the number of files available in a directory with PDirectory Object	177
Getting the file size of a file by using PFile object	
Using PFileList to display the constituents of a Folder	
Presource	195
Additional Support	196
Appendix A: Figures	197

License Agreement

This document and its contents are furnished "as is" for informational purposes only, and are subject to change without notice. ArianeSoft does not represent or warrant that any product or business plans **Expr**essed or implied will be fulfilled in any way. Any actions taken by the user of this document in response to the document or its contents will be solely at the risk of the user.

ARIANESOFT MAKES NO WARRANTIES, **EXPR**ESSED OR IMPLIED, WITH RESPECT TO THIS DOCUMENT OR ITS CONTENTS, AND HEREBY **EXPR**ESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR USE OR NON-INFRINGEMENT. IN NO EVENT SHALL PROOFHQ BE HELD LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING FROM THE USE OF ANY PORTION OF THE INFORMATION.

Copyright © 2009 by ArianeSoft . All rights reserved.

This document may not be reproduced, photocopied, displayed, transmitted or otherwise copied, in whole or in part, in any form or by any means now known or later developed, such as electronic, optical or mechanical means, without the written agreement of ArianeSoft . Any unauthorized use may be a violation of domestic or international law.

Trademarks and Copyright ArianeSoft , Pocket Programming Language, and the ArianeSoft logo are trademarks of ArianeSoft .

All other product or company names mentioned are used for identification purposes only, and may be trademarks of their respective owner.

Creating A Form Based Application

Creating applications with PIDE is an easy as well as efficient process. PIDE implements visual programming and allows its users to create better applications with minimal programming. Visual programming not only makes it easier to create programs just by using a few simple drag and drop operations but also allows creation of faster and much robust applications.

Whether you are a novice programmer or someone who is well qualified in creating computer applications, you will be more than happy to create applications with PIDE. Visual programming is an effective way to create computer applications without much of an effort. Using the already created templates for easy creation of application programs, PIDE users can easily create applications within seconds. In the first section of this tutorial, we will see how easy it is to create easy as well as complex applications with PIDE.

Printing Hello World

Writing Code File

By using the code file project in PIDE2, users will gain the required experience and knowhow of coding in PIDE. This experience can be transferred to other types of projects in PIDE which will enhance the type of projects that a user can accomplish through PIDE.

Start by creating a new Code File project. To do so, start PIDE and press Ctrl+N and select Code File Project from the "New Project Type.." window. Alternatively, you can also go to File> New and select Code File from the "New Project Type.." window.

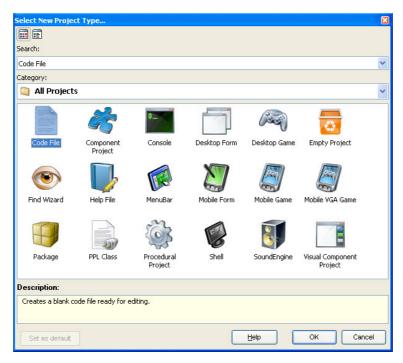


Figure 1: Creating a new project

• For creating a procedural project, users will need to use different methods and functions present in PIDE2. For printing a "Hello World" example in the code file project, use the ShowMessage() method.

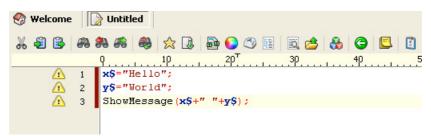


Figure 2: Code

- In the above code, we have used two variables namely variable **x** and variable **y**. Both of these variables have been assigned with string values namely, "**Hello**" and "**World**". The a **ShowMessage()** method allows a user to print a message within a standard dialog box and takes a string or a reference of a string to be printed in it. In the above example, we have included variable **x** as well as variable **y** with a string of a single space (+" "+) in between them to print **Hello World**.
- Press F5 or Go to Run-> Run to execute the code. After the code is executed, you will see 'Hello World' written in a alert message box.



Figure 3: Output

Procedural Project

Using procedural project creates a code based project where a programmer can write his own code (manually) in the **MainLibrary** object, he/she can add methods (which are procedures and functions only). But it is almost the same as writing a code file. Given below are the steps to create a procedural project in PIDE:

• Open PIDE2 and create a new **Procedural** project. To do this, press **Ctrl+N** from your keyboard and select **Procedural** in the **Select New Project File Type..** Window.

ocedural Proje	ct					
tegory:						
All Project	cts					Ę
Haran Markan Ma	Č	\$		S		
Code File	Component Project	Console	Database	Database Project	Desktop Form	
æ	4	۲				
Desktop Game	Empty Project	Find Wizard	Help File	MenuBar	Mobile Form	
		P		a la		
Mobile Game	Mobile VGA Game	Package	PPL Class	Procedural Project	Report	
escription:						
	oject allows you to imp it was possible with P		directly into a ne	w project. It also allow	vs you to create simp	le (non-form)

Figure 4: New Procedural project

- Just like any other programming language, we can add procedures in PIDE. Now we will work in the main library. For our *Hello World* message, we will add the **ShowMessage** command but use it within a **Proc Main** parent this time.
- To write our code, double click the **MainLibrary**. This will open a code editor that can be used to write code in it.

💏 Components	🏚 🚷 Welcome 📗	Untitled	Properties	
Application	🔺 🎲 Project Manager		🛒 Properties 🛷	P Events
🧰 Folder	₩ 43 🗟 × 4	ala II	MainLibrary : PLib	orary
🍪 NameSpace	00 42 134 ~ 0			
PComment	📄 🖌 🥠 💹 👧 🕀) 🖻 🖻	🞼 🗢 🖛 🖡	+ =
🖳 PConsole			Prope	erty Value
PLibrary		[🚵 🛷	- Object	
🔁 Class	Project	V	Name:	MainLibrary
PComponent	📳 MainLibrary	V	ClassName:	PLibrary
APEvent			E Library	
🔅 PMethod			Description:	
Robject			NameSpace:	
PProperty				



• In the code editor, write the **ShowMessage** command as in the screenshot and save the file.

Project Manager 🔀 MainLibrary	🥳 Properties 🐼	Events
ፚ 42 🕞 88 88 88 89 ☆ 🗔	🗿 MainLibrary : PLibr	rary
🔤 💿 🕾 🔄 🚵 🗞 😋 트 🗵	🞼 🕹 🕸 🖛 📥	+ -
Q1ρ	Prope	rty Value
1日 proc Main	- Object	
2 ShowMessage("Hello World!");	Name:	MainLibrary
3 end;	ClassName:	PLibrary
	- Library	
	Description:	
	NameSpace:	

Figure 6: Hello World Code

• Now, we can run our project by pressing **F5** or by going to **Run** in the main menu and choosing the **Run** option.

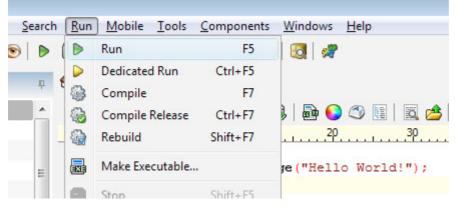


Figure 7: Run Hello World

Figure 8: Run the project

• After the project is executed you can see the results!

	9	.10
1Ę	proc Main	n
2	Show	Message("Hello World!");
3	end;	
		Message
		OK

Figure 9: Output



Enclosing strings within quotes is a common way of determining strings in PIDE and this distinguishes variables from strings.

Printing Hello World in Console

Printing text in console is really simple. Follow the steps given below to print "Hello World" in PIDE console window.

• Start by creating a new **Console** project. To do so, start PIDE and press Ctrl+N and select **Console** Project from the "**New Project Type.**" window. Alternatively, you can also go to **File> New** and select **Console** from the "**New Project Type.**" window.

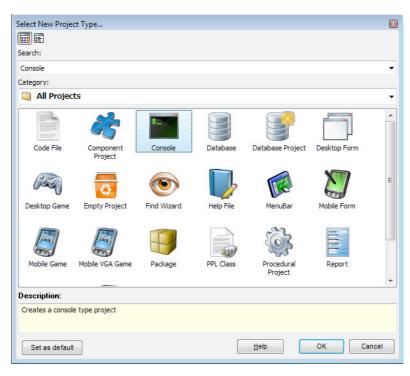


Figure 10: Create a new project

• Once the new project opens, double click the **MyApplication** object to create a new **OnStart** event. Right-click the **MyApplication** object and select **Events**, finally, choose **OnStart**.



Figure 11: OnStart event

• Click the **OnStart** procedure and press **Ctrl+Space** key to bring the code-completion window. Select **Writeln** in the **Code Completion** window.

🊜 🗐 🖹 🗶 🖂 😭 🕈 🎝 🖉 🖉		
🗞 🚷 🖹 🖹 🗳 🥎 🖹 🔮 🧕 🛷		
🙀 Project	v	
🖂 🏈 MyApplication	14	
🖉 proc OnStart (Sender\$)	04	۲
Defa PMyApplication	V 🧼	2
😳 writeln		

Figure 12: Code Completion window

• Once selected, click or press enter key on the **Writeln** object and change its **Expr** property to "**Hello World**" including the double quotes.

😵 Welcome 🗽 Untitled		Properties	₽ ×
🍪 Project Manager		📝 Properties 谷 Events	
X 48 🚯 × 42 🕈 🦊 🖌 🛷 🞯 🚷 🚷 🖻 🗐 🧐 🚳 🛷		🎲 writeln 1	-
C Project	ø		
E 🦑 MyApplication	14	Property Value	
E 🛷 proc OnStart (Sender\$)	/	- Parameters	
🎲 writeln("Hello World");	- <u>, , , , , , , , , , , , , , , , , , ,</u>	Hello World	f=
DefaultConsole	V 🧼 🖏	= Status	
		Operation: ProcCall	
		Source: writeln	

Figure 13: Expr property

• Press F5 to run your project.

Console
File Edit
Welcome to the PPL Console
Hello World

Figure 14: Output



Almost all objects in PIDE can be double clicked to generate default event. For example, PButton creates an OnClick event when double clicked while MyApplication created OnStart.

Printing Hello World in Desktop Form

Desktop Form project can be used by users of PIDE to create a form like interface for their applications. **Desktop Form** makes it utmost easy to create applications that can run in the windows based environment. Making the full use of visual programming, users can create applications with ease. In our **Desktop Form**, we will be creating a button that would print "**Hello World**" in a label, when clicked.

Create a new Desktop Form by selecting Desktop Form option in the "Select new project type..." window. For doing this, you can press the Ctrl+N key or Go to File>New. In the Select New project Type... window that appears, select Desktop Form project.

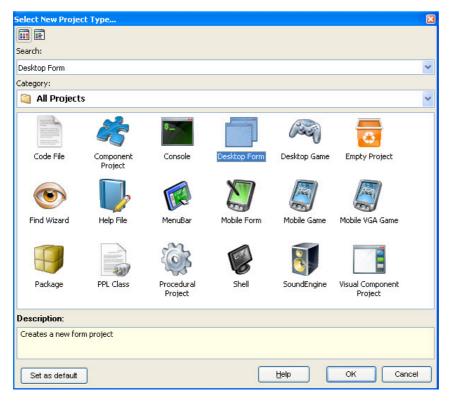


Figure 15: New Project

• Now, in the **Desktop Form**, right click the **Default Form** object to select **Edit Form** or **Press F12** while it is selected or double-click the **DefaultForm** object to bring the **Form Edit** screen.

% 🗐 🗟 🗙	🕸 🏦 🦊 🖌	🤣 🛃 🚷 d	8 🖻 🗄 🖉	ò 🗈 🖻 🛿
🍪 Project				
🧭 MyApplication	1			
😑 📄 DefaultForm				
🗆 🕎 Menut ⊿	f Edit	•	🔁 Edit	Ctrl+Enter
🖃 🗐 Fi 🔳		•	🔄 Edit source code	Shift+Enter
	P Events	•	🗾 Edit Form	F12
			🔲 Edit Help	
	, Cut	Ctrl+X		
	Сору	Ctrl+C		
Ê	Paste	Ctrl+V		
×	Delete	Ctrl+Del		
Ę	Rename	F2		
1	Move up	Alt+Up		
4	• Move Down	Alt+Down		
4	Batch operations	Shift+Ctrl+O		

Figure 16: Right click to Edit Form

• In the edit form view, create a **PButton** object by double clicking it from the **Components Panel** and rename the **PButton** to "Click" by changing its **Caption** property.

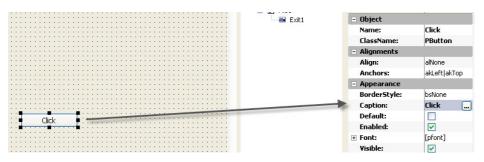


Figure 17: Caption Property

• Right click the button and use the Edit option to open DefaultForm in the code view.

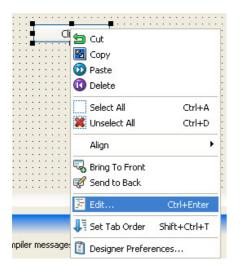


Figure 18: Right click to open DefaultForm

• Create a procedure pertaining to the **OnClick** event of **Click** button and include **ShowMessage**("**Hello World**") within its body. Right-click the **DefaultForm** object and select **Events**, finally, choose **OnClick**.

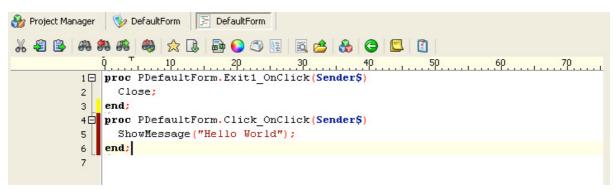


Figure 19: Code

• Press F5 to run the project and view the results. Use the "Click" button to view "Hello World" message.



It is beneficial to learn shortcuts in PIDE as they save time and effort. Quick one's like Ctrl+N for a new project and Ctrl+I for marking an IF statement is a nifty addition to PIDE's environment that goes a long way improving programming experience.

Creating a Complex Program

For creating a complex program, we will create a windows form which will allow a user to enter his details and display the details in a pre-formatted manner after the form is submitted. In this example, we will use form elements like **PButtons, PLabel, PEdit**.

• Create a new **Desktop Form** project by going to **File> New> Desktop Project**. Alternatively, you can also press **Ctrl+N** to create a new **Desktop Form** project.

Select New Proje	ct Type					×
Search:						
Desktop Form						~
Category:						
📋 All Project	s					*
Code File	Component	Console	Database	Database Project	Desktop Form	
Desktop Game	Project	Find Wizard	Help File	MenuBar	Mobile Form	
Mobile Game	Mobile VGA Game		PPL Class	Ś		
		Package		Procedural Project	Report	
Shell	SoundEngine	Visual Component Project				
Description:						
Creates a new fo	rm project					
Set as default				Help	ок	Cancel

Figure 20: New Project

• In **Desktop Form** project, Right click the "**Default Form**" and select "**Edit**" Option. Alternatively, you can also double-click the **Default Form** option to open **DefaultForm** in edit mode.

roject Manager				a. 🖦 🗅 🝘 🗌	
🔁 💷 🔺	• · · · · · · · · · · · · · · · · · · ·	′ 🥨 🔟 👀	🊷 🖪 🖻 📕	🧐 🖪 📑 🥨	20 V
ig Project MyApplicatio	D				
				43	/ 🗸 🎝 🖏
E T Menu	[*] Edit	×.	🚰 Edit	Ctrl+Enter	
	Data	•	🔄 Edit source code	Shift+Enter	4
and the second se	P Events	•	Edit Form	F12	×
*	Cut	Ctrl+X	Edit Help		
) Сору	Ctrl+C			
	Paste	Ctrl+V			
×	Delete	Ctrl+Del			
Ę	Rename	F2			
1	Move up	Alt+Up			
4	Move Down	Alt+Down			
4	Batch operations	Shift+Ctrl+O			
e	View Binds	Shift+Ctrl+B			
6	Clear binds				
4	Auto Create	Shift+Ctrl+A			
6	SubClass	Shift+Ctrl+S			
1	Expand class	Shift+Ctrl+E			

Figure 21: Right click to Edit Form

- Once in the **DefaultForm** Edit view, you can easily create a form by taking form elements from the **Components Pane**l.
- Start by clicking the **PLabel** component from the Component Panel by using the area in the form where you want to display it. Along with the **PLabel** component, you can also include **PEdit** components by clicking the individual elements and then clicking the area in the form where you want to display them. We will also have a **PListBox** component in our form, so click on the **PListBox** component in the component pane and click on the area beside a label.

Components	🐢 🗞 Welcome 🛛 🗽 Untitled	
🗁 Standard Controls	🔺 🍪 Project Manager 🛛 😼 DefaultForm	
🧾 PBitmap		
PButton		
🔀 PCheckBox	DefaultForm	
E PComboBox		
式 PControl		
ab PEdit		
PGrid	Label1 Edit1	
PGroupBox	Edit2	
📳 PImageList		
Ala PLabel	Label3 Male 🗧	
目 PListBox		
FFF PListView		
🗱 PMemo		
💞 PPaintCanvas		
PPanel		
🖶 PPrinter	······································	
PProgressBar		

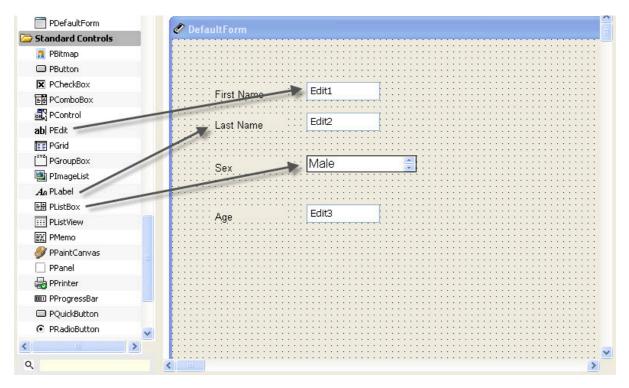
Figure 22: Place components

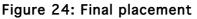
• To make our application more user friendly, we will change the Caption property of our **PLabel** components so that a user can recognize them. While we are at it, we can also create a caption or heading for form by changing the caption property of our form.

Welcome Untitled	Properties	
Project Manager 🛛 🔊 DefaultForm	🛃 Properties 🛷	
· • · · · · · · · · · · · · · · · · · ·	DefaultForm : PFo	rm
Ø MyForm	🞼 🕹 🖛 🖚 🖊	± =
	Proper	ty Value
	- Object	
	Name:	DefaultForm
First Name Edit1	ClassName:	PForm
	- Alignments	
Last Name ::: Edit2 ::::::::::::::::::	Align:	alNone
		The second second
Sex 🗰 Male 🚔 😳	Anchors:	akLeft akTop
	- Appearance	
	BorderIcons:	biSystemMenu biMinimize biMaximiz
	BorderStyle	bsSizeable
	Caption:	MyForm
	Color:	clBtnFace
	Enabled:	
	E Font:	[Tahoma,12,,clBlack,,]
	Name:	Tahoma
	Size:	12
		**
	Style:	
	Color:	CBlack

Figure 23: Change Caption property

• We will also create a PLabel, mark it with **Age** caption and correspond it with a **PEdit** object so that we can input the age of a person.





• Lastly, we will add a **PButton** and set its caption property to **Submit**. A click on this **PButton** will lead to the processing of the given information.

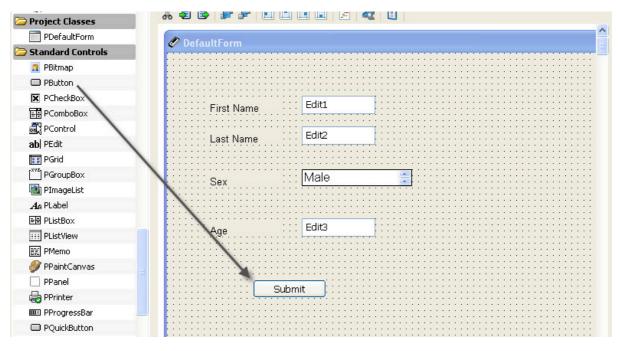


Figure 25: PButton

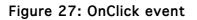
As the form is now complete, we will now look at how to submit various input values so that we can print the required output.

🍪 Project Manager 🛛 🏷 DefaultForm	
X 🕄 🖻 🗴 💐 🚹 🕹 🖉 🖓 🖓 🖓 🖓	b 🗈 🖻 🗳 🧐 🗈 🚱 📾 🛷
🍪 Project	v
🥙 MyApplication	v
DefaultForm	/ 🗸 🤣 🔊
🕀 🛅 MenuBar1	A
Ag Label1	v
Ag Label2	A
Ag Label3	v
Ag Label4	v
ab Edit1	v
ab Edit2	v
🔄 Script1	v
目標 ListBox1	v
ab Edit3	A
🍌 Memory1	A
At List1	A
Button1	04

Figure 26: Project Manager

• Double click the button object to create an **OnClick** event.





• Drop a **PVariable** object from the **Components Pane**l on the **OnClick** event. This will create the definition of a new variable. Drop this new variable to the **OnClick** event while holding the **Alt key** to create a new variable. i.e **Variable1\$.** Doing this would create a new code line using the variable so you can assign value to it and use it for various actions.

PProperty	ab Edit1	v
PVariable	ab Edit2	~
CodeFlow	🕞 Script1	v
🔄 Break	E骨 ListBox1	×
P Continue	ab Edit3	×
🔄 Exit	Memory1	~
🙆 For	At List1	✓
🖄 ForEach	B Button1	14
av∰ If	🖃 🕐 proc OnClick(Sender\$)	/ 🖉 🔲
🔄 Return	Local (Variable1\$) ;	/ 🖌 👋
🔄 Using	🕃 Variable1\$ =;	/ 🗸 🔍
S VarObject		



• Drop the **Edit1** to the **Expr** property of newly created variable and select the **Text** from the auto code complete box. Doing this will assign the text written in the **PEdit1** object to the variable.

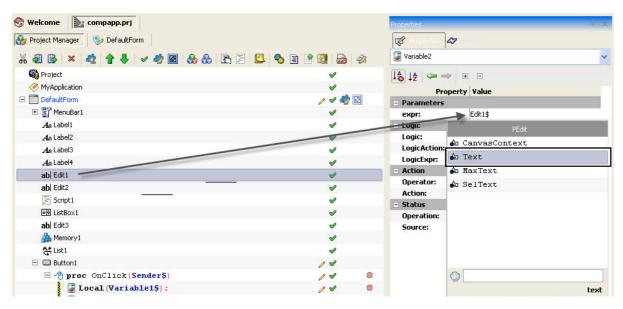


Figure 29: Drag PEdit to expr

• Do likewise for other **PEdit** objects also. After all the variables are created, make sure you have all the **PEdit** objects in the respective **Expr** property of the declared variables like you did with the first **PEdit** objects.

🖃 🔲 Submit1	a 1997	
🗆 🖑 proc OnClick(Sender\$)	✓	۲
Local (Variable1\$);	v	۲
<pre>Variable1\$ = Edit1.Text;</pre>	×	
Local (Variable2\$);	v	۲
Variable2\$ = Edit2.Text;	×	۲
Local (Variable3\$);	×	۲
Local (Variable4\$);	×	۲
<pre>Variable3\$ = Edit4.Text;</pre>	v	۲

- For the **list box**, drag a new **PVariable** to the **OnClick** event of the button and then drag the newly created Local variable again to the **OnClick** event while holding the **Alt key**. This would create a new variable.
- Drag the ListBox object to the Expr property of the newly created variable and select ItemText from the code complete window.



If anytime while programming, you feel that you are redoing some steps, you can also select the concurrent statements by pressing the Shift key and copy the statements. Then paste them to create more such statements.

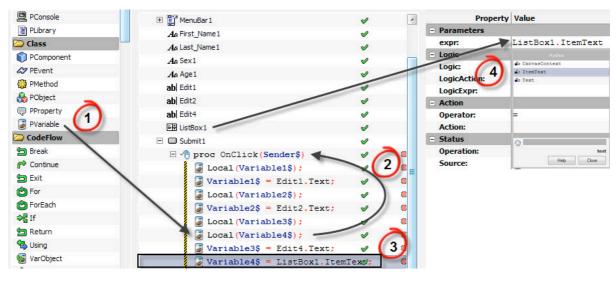


Figure 30: Steps from 1 to 4

- Now, we will have an **If** condition on the **PEdit** objects that inputs the age of a user. This **If** condition will display a custom message depending on the age entered by a user.
- For creating an If condition on PEdit4, drag and drop PEdit4 on OnClick proc. As the Code Completion window appears, select Text. This will create an Edit4.Text entry. Click on the entry and press Ctrl+I (This will make it an If condition).

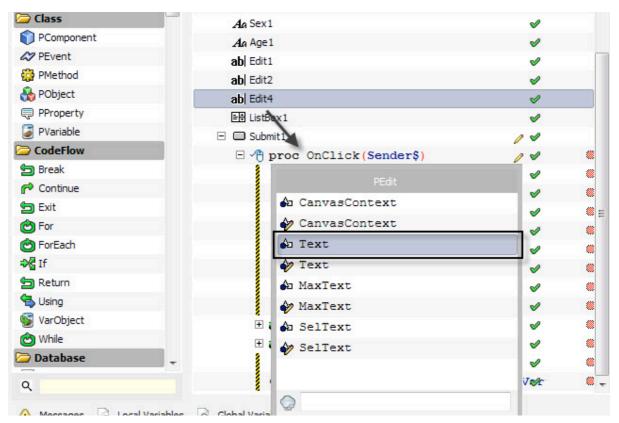


Figure 31: Drag operation

• Change the LogicExpr property of the If condition to <20. Doing this will check whether the age input by the user is less than 20.

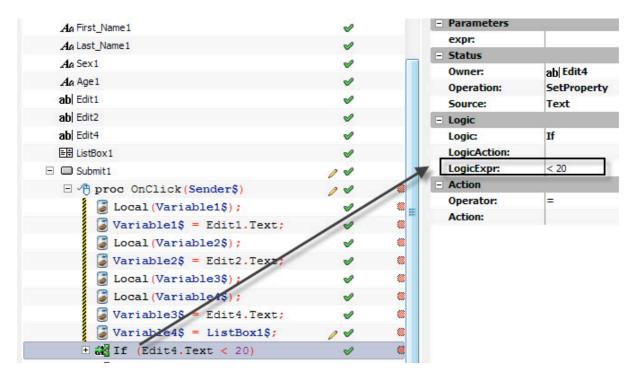


Figure 32: Change property

• If our **IF** condition holds true, we will print "*eligible for the program*". For doing this, create a new variable by dragging and dropping a **PVariable** to the **OnClick** Proc. Now drag and drop the newly created variable back to the **OnClick** procedure while holding the **Alt key**. Once a new variable is created, drag this variable on the **If** condition. In this variable change the **Expr** property to "*eligible for the program*". Doing this will enable you to display this message if the condition holds true.

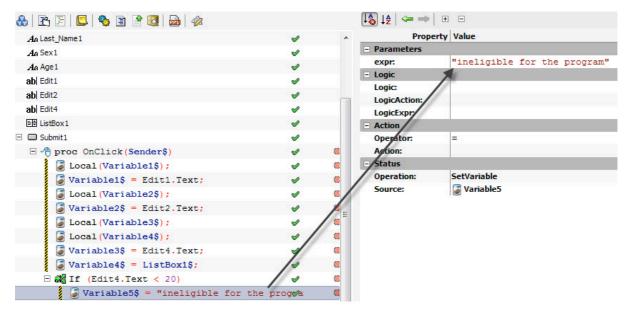


Figure 33: Change Expr property

• Just like the If condition we created in the step above, we will create one more If condition that would check if the age entered by a user is smaller than 20 and display a message accordingly.

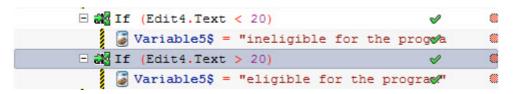


Figure 34: Condition

• Now that everything is done, all we need to do is create a dialog box that would print the message in a preformatted way. We would use ShowMessage () for this. Go to OnClick proc and press Ctrl+Space to open the Code Completion window. In this window, select ShowMessage(), and change its value to the variables you want to display. For our example, we will use (Variable1\$+" "+Variable2\$+" is a "+Variable4\$+" and is "+Variable5\$).

<pre></pre>	V	۲
PDefaultForm	ø	
	1	۲
ProcCall_ShowMessage	1	
🎲 ShowMessage (value\$ [, value\$	1	۲
	1	

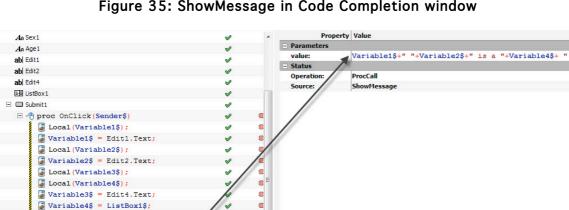


Figure 35: ShowMessage in Code Completion window

Figure 36: Change Value property

Now, run the project and see it in action.

M If (Edit4.Text < 20) Variable5\$ = "ineligible for

If (Edit4.Text > 20) Variable5\$ = "eligible f
Local (Variation)

Local (Variable5\$); 😳 ShowMessage (Variable1\$ /" "

File			
	First Name	Jhon	
	Last Name	Doe	
	Sex	Male	A
			Message
	Age	22	Jhon Doe is a Male and is eligible for the program
	S	ubmit	ОК

Figure 37: Output



In PIDE, you can have variables as well as strings in a same statement by using + sign and enclosing the strings with quotes. For example **"How are you today "+Name\$+" ?"** may be printed as **How are you today chuck?**

Creating Packages And Using Them In Your Applications.

Packages are important part of a PIDE project. Users can use PIDE to create applications with various sources like pictures, videos, graphics etc that are stored in a single file. A package will contain all the resources that a PPL application will contain. Given below are the steps to create a package so that it can be used in your PPL project.

• Create a new **Desktop Form** project by selecting it from the **Select New project Type...** window. For this press **Ctrl+N** key or Go to **Files>New**.

Select New Proje	ct Type					
Search:						
Desktop Form						~
Category:						
📔 All Project	s					*
Anna an Anna Anna Anna Martin an Anna Anna Anna Anna Anna Anna Anna Anna Anna Anna Anna	ŝ	\$		3		
Code File	Component Project	Console	Database	Database Project	Desktop Form	
æ	0	•		R		
Desktop Game	Empty Project	Find Wizard	Help File	MenuBar	Mobile Form	
Mobile Game	Mobile VGA Game	Package	PPL Class	Procedural	Report	
6	3			Project		
Shell	SoundEngine	Visual Component Project				
Description:						
Creates a new fo	rm project					
Set as default				Help	ок	Cancel

Figure 38: Create New Project

• Drag a **PPackage** object to the project area.

	🍓 Project
🗁 Database	
PDatabase	🥐 MyApplication
PDBCheckBox	DefaultForm
ab PDBEdit	🤪 Package1
📰 PDBGrid	*
🥵 PDBNavigator	
🗖 PField	
PQuery	
PTable	
🗁 Database Report	
PBand	
Ala PRDBText	
PReport	
PRShape	
Ala PRText	
🗁 File	
🤪 PPackage	
📦 PPackageFile	
lessource	
۹	

Figure 39: Drag and Drop PPackage

• Give a target location in the PackageFilename property of PPackage object

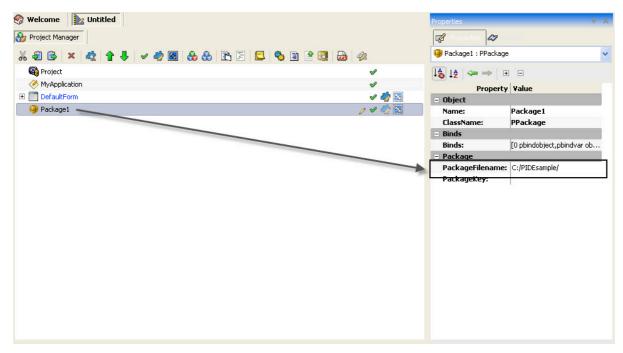


Figure 40: Change Property

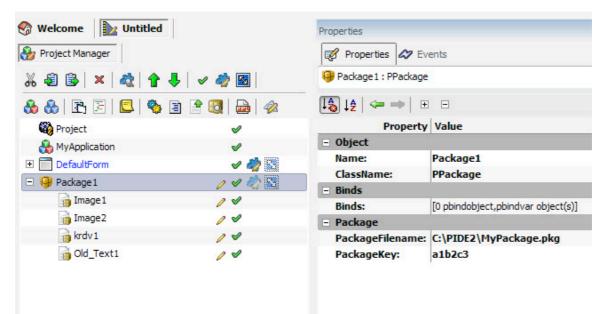
• Specify a package encryption key in the PackageKey property under package section

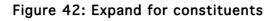
• Drag a file from the windows explorer to the **PPackage** object and select the component type as **PPackageFile**. Likewise, drag and drop as many files as you wish to the **PPackage** object and choose the **PPackageFile** option in the component type drop down box.

▼ 49 Search	Image: PIDE - Untitled * File Run Tools Components Image: PiDE - Untitled * Image: PiDE - Untitled *	: Windows Help ▶ ▶ ▶ ▶	
w 🛃 Print 🛋 E-mail 😰 Sharing Settings Size Rating	Burn Image: Components Image: Components Image: Components Image: Components Image: Components <tr< th=""><th></th><th></th></tr<>		
Desert Landscape Dock Fe	PBand	st eList	✓ ✓ ✓ ♣2 🖸
ers Garden Green Sea Turtle Humpb	Prodog PPacka Polalog Prontbalog Posta Polalog Posta Pontbalog Psurfa PopenDialog Psvetbalog	igeFile urce	
	PPackage		

Figure 41: Drop Files

• Using the "+" sign will expand the package and show all the constituents of the package.





• After all the files have been added to the package, a user needs to compile the project. Go to the **Run** menu and click on the compile option. Alternatively, users can also use the **F7** key to compile the project. After a project is compiled, the package file of the project is recreated at the location where it was specified in the **PackageFilename** property.

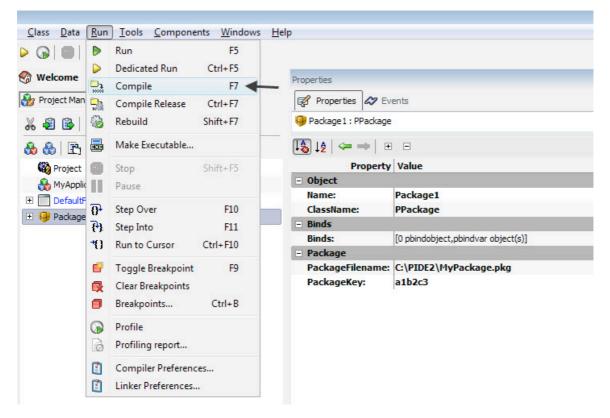


Figure 43: Run the Project

Creating Databases And Tables

Creating databases and tables with PIDE is a really easy job. A couple of drag and drop operations is all you need to create your database driven applications.

There are three options to build a database with PIDE, users can either use a database, a database project or a **Desktop Form** with a database attached to it.

Database - This project comes just with a database, tables and fields. Using this database, you can add a database to your existing projects.

Database Project – This type of PIDE project comes complete with a database, tables, query as well as a report. It also includes a **Default Form** object so that users can start right away with their database enabled forms.

Creating Table In A Desktop Form Project

A **Desktop Form** is one of the most useable applications of PIDE. With a **Desktop Form**, not only does it becomes very easy for a person to navigate through the application, it is easier to understand and also serves the exact purpose why a **Desktop Form** is made in the first place. By combining a **Desktop Form** with database, programmers are able to create applications that can have dynamic nature and can work really well with data.

Given below is the procedure to create a table within a **Desktop Form** and use it to store values.

• Create a New Project in PIDE by selecting File>New from the file menu and select **Desktop Form** from the **Select New Project Type...** Alternatively, you can also press **Ctrl+N** to being the **Select New Project Type...** window.

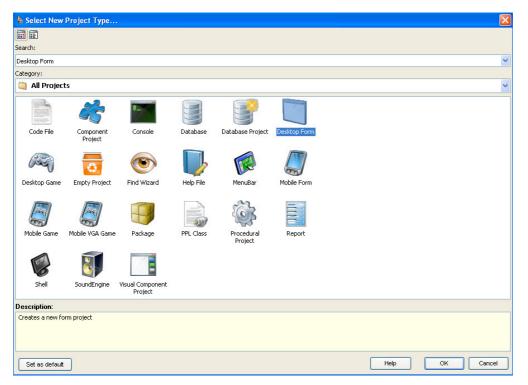


Figure 44: Creating new project

- Select a Desktop Form project and open it in the Project Manager view
- Once a new **Desktop Form** project opens, drag **PDatabase Object** from the **Components Panel** to the **Project Manager**

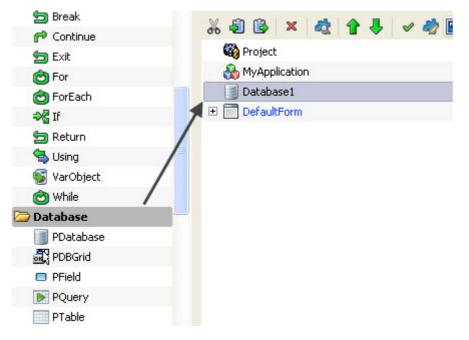


Figure 45: Drag a database

• Drag a **PTable Object** from **Components Panel** to the newly created **PDatabase Object**

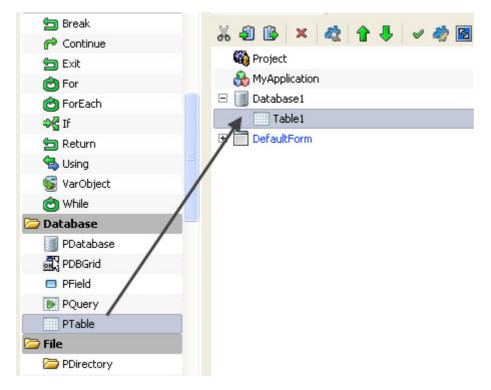


Figure 46: Drag a PTable

• Click on the **PDatabase Object** and set its filename property to a location where you would like to store the database you are creating. Remember to save this file with a **.db or .sdb extension**

% 🗐 🗟 🗙 d	2 1 4 🗸	🤣 🛃 🚷 👌	b 🖻 🗄 🕻	S 🛾 🕹	🔇 🖻 🛷			Database1 : PDa	tabase 🔽
🆓 Project						v		1a 12 🦛 🔿	÷ =
💑 MyApplication						v			erty Value
🖃 🧾 Database1						/ 🗸 🧞 📓		- Object	
Table1						0 4		Name:	Database1
🗉 📄 DefaultForm	Select File					? 🗙		ClassName:	PDatabase
22.004	Select File							- Binds	
	Look in:	🛛 😼 My Compute	r	~	G 🔰 📂 🛄 -			Binds:	[0 pbindobjec
		31/2 Floppy (A						- Database	
	2	Strain Disk (C:)						Filename:	Databas 🛄
	My Recent	Second Disk (D:)							
	Documents	🚞 Shared Docum	nents						/
		🚞 Administrator'	s Documents						/
									/
	Desktop							/	/
								/	
								/	
	My Documents							/	
								/	
							/		
	My Computer								
	(File name:	Database		~	Open	11		
					100 million (100 m				₽ X
🗥 Warnings (0) 🛛 📮	My Network	Files of type:	Default files (*.d	b;*.sdb)	*	Cancel			
acter in the lease									

Figure 47: Specify a filename

• Now, in the TableName Property of PTable Object, give a name to your table

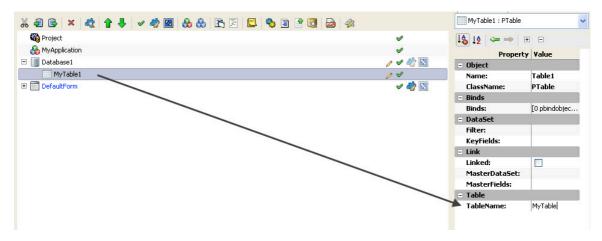
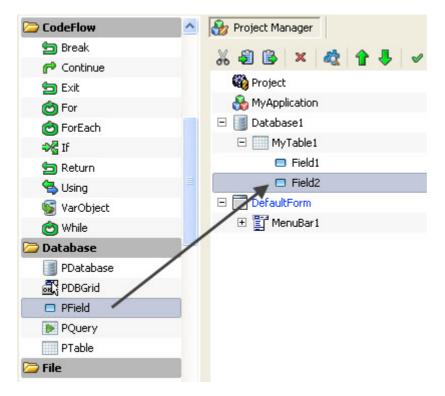
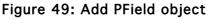


Figure 48: Give TableName

• Drag some **PField Objects** to the table object to add fields to it.





• Click all the **PField Objects** and change their **FieldName**, **FieldSize** and **FieldType Property** according to your needs.

16 😫 🖹 × 💐 🕇 💺 🗸 🧳 📓 🚷 🚷 🖹 🗐 🧐 🗃 🧐 👼 🍕	8	MyField2 : PField	
🙀 Project	v	18 12 🦛 🔿	÷ =
💑 MyApplication	v		erty Value
🗄 🧾 Database1	/ 🗸 🧼 🔀	= Object	
🖂 🥅 MyTable1	14	Name:	MyField2
Field1	00	ClassName:	PField
MyField2	04	- Binds	
DefaultForm	🗸 🤣 😒	Binds:	[O pbindobjec
🗉 🛅 MenuBar1		🖃 Field	
		FieldName:	MyField2
		FieldSize:	60
		FieldType:	ftText

Figure 50: Specify Property

• In the data menu, **Click** on **Create Database** to create your database file in the location specified earlier in the **Filename** Property. Alternatively, you can also right click the **PDatabase** Object and select **Create Database** from **Data** to create a database. Doing this would create a physical database that is blank and contains all the tables and fields.

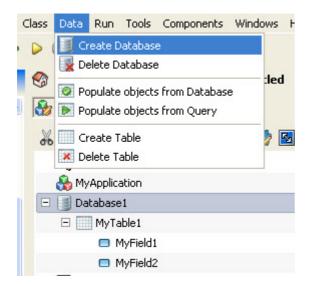


Figure 51: Creating Database

• Once you have created a database, you can double click the **PTable Object** to edit the field values.

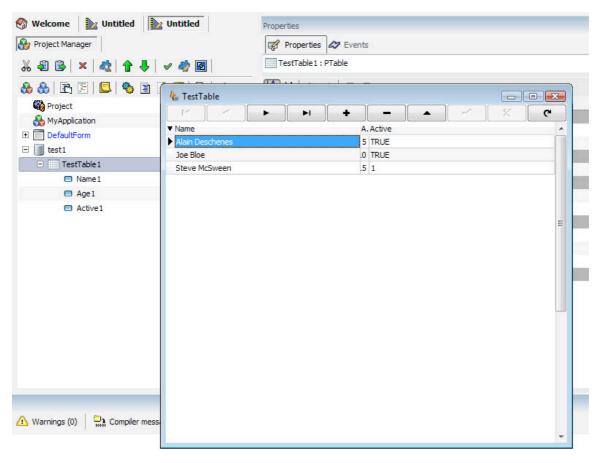


Figure 52: Viewing Table

• If you want to build your SQL query visually, you can also drag a **PQuery Object** from the **Components Panel** into the **Project Manager** or skip the next few steps to print everything on the form by default through **PDBGird**.

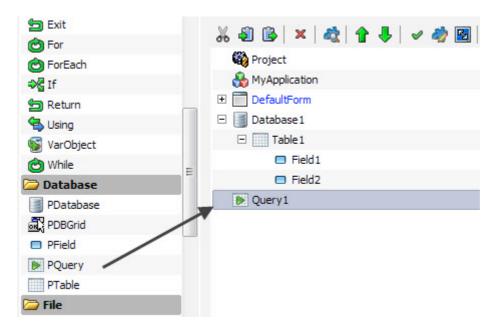


Figure 53: Drag PQuery

• After dragging and dropping the **PQuery** to the **Project Manager**, go to the **Properties Panel** and change the **Database Property** to a database that is already created.

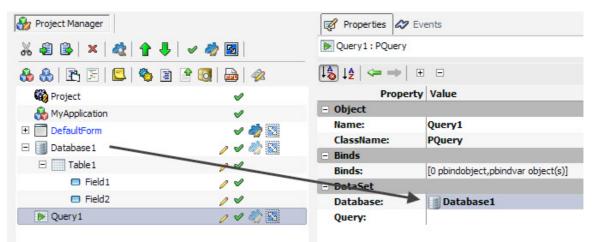


Figure 54: Change Database property

• Now double click the **PQuery Object** to create a **SQL query visually**. Refer to the next section(Using Visual Query)for in-depth knowledge on using the visual query editor for writing visual SQL queries.

Visual Query			
Tables	Builder SQL Result		
TestTable	Select 🔹	Query 1	
	Query 1		
		✓ TestTable	
		✓ Name ✓ Age	
		✓ Active	
		• III •	
		Criteria Selection Grouping criteria Sorting	
		All of the following are met	
		1. <u>TestTable.Name</u> LIKE 'Alain Deschenes'	

Figure 55: Visual query editor

• The database along with table and the query that we need to perform has been created. Now we can use other objects like **PDBGird** to display the table the way we want.

 Seturn Sing VarObject While 	Que
🗁 Database 🧮	□ 🕎
PDatabase	Ξ
R PDBGrid	
PField	
PQuery	
PTable	

Figure 56: Insert PDBGird

• After creating a database, you can drag the **PTable** to a **PForm**. The drag operation will initialize the **SmartMove** feature which will ask you about the action that you would like to accomplish. Here, Select '**Add Grid to Control**' to create a table with a grid view in your form.

🍪 Project	v
A MyApplication	
()) Hymphication	v
🗆 📑 Database1 🥒	/ 🖌 🧼 😒
🖻 🥅 MyTable1 🥔	? I
MyField1	9 I
	? I
	r 🖌 🦏 🔝
🖂 🕎 MenuBari	V
🖃 통 File1	ø
🖻 Exiti	V

Figure 57: Drag table to form

🌡 🗐 📴 🗶 🔌 🎓 🦊 🖌 🥔 🕅 🚷 🊷 🖹 🖹 🛄 🥎 🖹 🗐 题 🥔	v
	v
💑 MyApplication	s an
🛛 📑 Database1	/ 🖉 🚵
E MyTable1	04
MyField1	0 1
MyField2	1 1
DefaultForm	/ 🗸 🦓 🔀
Select smart move to apply	v
Add grid to control	V
Cancel	v

Figure 58: Smart move



SmartMove feature in PIDE is very helpful in selecting the actions that would be performed. Likewise, code completion window is a unique feature that would allow you to better program while following visual programming.

Using Visual Query(PQuery)

With **PQuery**, users can easily change the way their database is handled. Users can easily create **Select**, **Insert** as well as **Delete** and **Update** SQL statements to manipulate their database the way they want. Users need to run these SQL statements in your form and **PQuery** is an object that will help them doing this.

In the example given below, we will use the database created in the above example and work upon it to create an SQL query visually. Follow the given steps:

• Drag a PQuery object present in the Components Panel to the Project Manager.

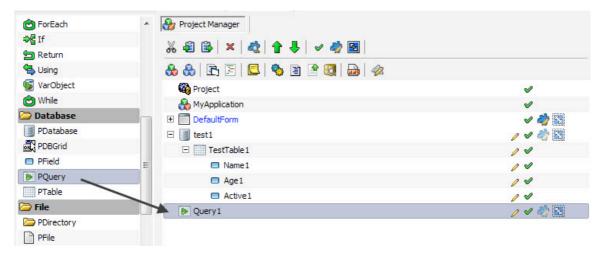


Figure 59: Drag PQuery

• Select your database name in the **Database Property** of **PQuery**. To do this, select the **PQuery**, and select the database name i.e test1, in the Database property.

😵 Welcome 🛛 🗽 Untitled	Properties Properties Properties Properties Query1: PQuery			
🍪 Project Manager				
🊜 🗐 📴 🗶 💐 👚 🖊 🛩 🤣 🔯 🛛				
🚴 🏡 🖹 🗐 📕 🧠 🗃 🔮 🔯 💩 🌾	8	🔩 🞼 🖛 🔿	+ -	
Project	v	Prope	erty Value	
MyApplication	v	- Object		
DefaultForm	🗸 🥠 🔝	Name:	Query1	
		ClassName:	PQuery	
E isti	/ / //	- Binds		
TestTable 1	0 4	Binds:	[0 pbindobject,pbindvar object(s)]	
Name1		DataSet		
Age1	11	Database:	test1	
Active1	14	Query:	None	
Query1	y1 🧪 🛷 🖏		test1	
			N	

Figure 60: Select appropriate database name

• Double click the **PQuery Object** to open visual query editor.

🎲 Project Manager	Visual Query			
🔏 🗟 🗟 🗙 🛛				
🚷 🚷 🖻 🗄 🚺	Tables	Builder SQL		_
🚳 Project	TestTable	Select 👻	G Query 1	
MyApplication		Query 1	_	
DefaultForm			-	
test1 TestTable1	_			
Name1				
Age1				
Ctive1				
Query1				
				-
			< III +	
			Criteria Selection Grouping criteria Sorting	
			All of the following are met	

Figure 61: Double click PQuery

• At the top, visual query editor has four buttons that are used to perform tasks like **new**, open, save, and run.

Visual Query	
Tables	Builder SQL
TestTable	Select Query 1 Query 1

Figure 62: The four buttons of visual query editor

• While the **New Button** creates a new query, the **Save Button** is used to save the existing query in a .SQL format. All the SQL queries saved in .SQL format can be opened in the visual query editor through the **Open Button**. Lastly, the **Run Button** is used to execute an SQL query.

Visual Query	
🗋 🗁 🔚 🛷	
Tables	suider SQL
TestTable	Select Query 1
	Query 1
🍇 Save SQL File	
● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●	Local Disk (C:) PIDE2
File nam	e: 🔹
Save as typ	e: SQL Files (*.sql)
Srowse Folders	Save Cancel
	Criteria Selection Grouping criteria Sorting

Figure 63: Save file button

- The main view of **visual query editor** consists of a **Tables** pane and three tabs. The tables pane contains of the names of all the tables present in the database that was associated with a **PQuery Object.**
- The first tab is the **Builder Tab** that allows users to build SQL query visually.
- The second tab is the **SQL Tab** that allows a user to write SQL query or modify a SQL query created with the help of builder tab.

Visual Query	
🗋 🗁 🔚 🛷	
Tables	Builder SQL Result
TestTable	SELECT TestTable.Name, TestTable.Age, TestTable.Active FROM TestTable

Figure 64: The SQI Tab

• The third tab is the **Results Tab** that shows the result of a query. After a query is created, it should be executed by pressing the run button. The result of the query is displayed in the **Result Tab**.

Visual Query					
Tables	Builder SQD Result				
TestTable	r c	20	•	M	۲ ۲
	Name				*
	Alain Deschenes				
	Joe Bloe				
	Steve McSween				

Figure 65: Results Tab

• Users of PIDE2 can use the builder tab present in visual query editor to build Select, Insert, Update and Delete SQL queries.

Visual Query		
Tables	Builder SQL Result	
TestTable	Select	Query 1
	Select Insert	
	Update Delete	✓ TestTable 📃 🗙
	Delete	✓ Name
		Age Active

Figure 66: Selecting Query Type

• The Criteria, Selection, Grouping criteria and Sorting Tabs can be used to build advanced SQL queries.

Visual Query		E
Tables	Builder SQL Result	
TestTable	Select 🔹	Query 1
	Query 1	✓ TestTable ✓ Name ✓ Age ✓ Active
		Criteria Selection Grouping criteria Sorting

Figure 67: Creating Advanced Query

Using A Database Project

A database project is something that contains just the basic ingredients for creating a database driven application. There are basically two ways of using a Database project.

Method 1: For creating customized applications and starting just with a database, its table and fields.

• In PIDE, Press **Ctrl+N** or Go to **File>New** and Click on **Database** to create a project with just a database, its table and two fields.

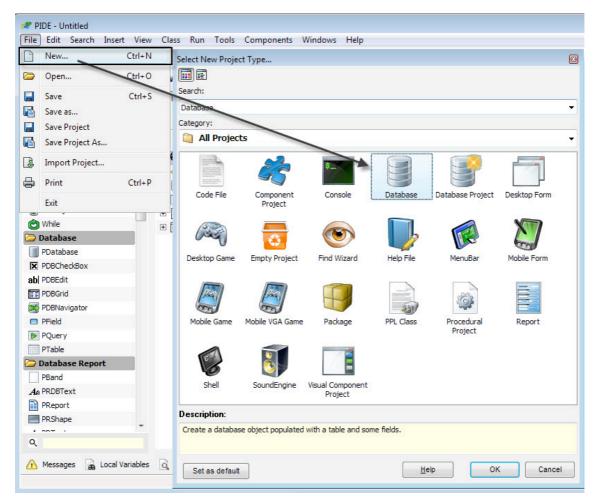


Figure 68: New Project

• Now that you have a database for your database driven application, you can easily add other objects like form or create a game if you like.

🗁 File	 Project Manager 	
🤪 PPackage		
pPackageFile	X 🕄 🗟 × 💐 🕈 🖊 🗸 🏘	
Interest PResource	🚷 🚷 🖻 🗐 📕 🎭 🗃 🔮 🚳 📠	<i></i>
🗁 Forms	🗆 📑 Database 1	11
PForm	Table 1	14
🗁 Menus	Field1	14
🖹 PMenu	Field2	14
PMenuBar	Form1	0 4 10 13
PMenuItem		2 - W CIIIII

Figure 69: Drag a PForm

Method2: Adding a database and all the related components to an existing project.

 Many times, we start with a project and then want to add database functionality to it. Instead of adding individual components like a database, its table and related fields, PIDE provides you to add all that in a single go with the help of **Database** project. While developing a **Desktop Form** project/**Desktop Game** project or any other such project, press **Ctrl+N** or go to **File>New** and select **Database** project to automatically create a Database and all its related elements in a single go!

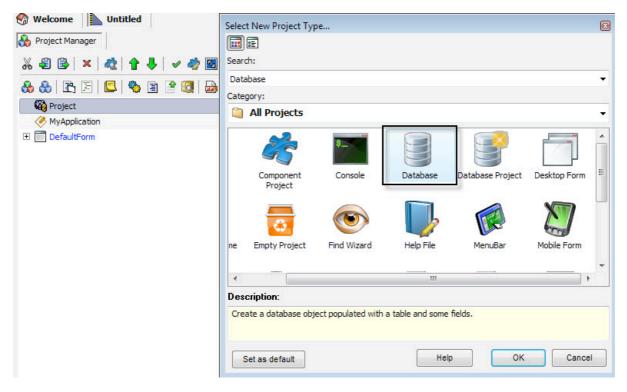


Figure 70: Create new project

🎲 Project Manager	
🊜 🅄 📴 🗙 💐 😭 🛉 🖊 🗸 🛷 🚳	
🗞 🚷 🖹 🗐 📕 🧠 🖹 😤 🗔 💩 🛷	
🏟 Project	v
🖻 🧾 Database 1	04
E Table 1	00
Field1	14
Field2	14
MyApplication	v
DefaultForm	🗸 🥠 🖏



Using Database Project

A Database Project is the premier choice if you want to create a database driven Windows or Windows Mobile based application. Just by a single click of a button, users can gave access to all that they need to create a database driven application. A database project provides its users with Database, **PQuery** object, **PReport** Object, as well as all the basic necessities of a database like a table and fields.

• For creating a database project, Press Ctrl+N and simply select Database Project from the "Select New Project Type" window.

Select	t New Project Typ	e			
	E				
Searc	:h:				
Data	abase Project				-
Categ	gory:				
	All Projects				-
	Component Project	Console	Database	Database Project	Î
	0				
1e	Empty Project	Find Wizard	Help File	MenuBar	Mobile Form
•					•
Dese	cription:				
	ates a full database Itabase	project including t	he following obje	ects:	
S	Set as default		He	lp ОК	Cancel

Figure 72: Create new project

Welcome Untitled	
Project Manager	
X 🗐 😫 × 🎕 🕈 🦊 🗸 🤣 🖪	
🗞 🚷 🖻 🗐 🧐 🗐 🔮 🚳 🛷	
🚱 Project	ø
MyApplication	ø
🛨 📑 Database 1	v
💓 Query1	V 🧼 😒
🗄 📄 Report1	🗸 🧄 😒
DefaultForm	V 🤣 🙁

Figure 73: Database Project

Using The PGird And PDBGird

PGird and **PDBGird** are objects in PIDE that are used to impart a grid type layout to a certain text. While they both are used to perform more or less the same thing, their application defers as **PGird** is more focused on the layout of a text whereas the layout of a **PDBGird** is related to the layout of table elements present in a database.

PDBGird is a very important object that is used for visually aligning table elements in an easily understandable format. Given below are the steps that are required to create a **PDBGird** as well as **PGird** objects in a PIDE application.

Creating a PGird Object

Creating a **PGird** Object is a simple to use procedure that both easy and effective one. With the help of a **PGird**, we can add grid functionality to the application and use it for efficient layout of data.

Given below are the steps to create a **PGird** Object in a database form:

- Start by creating a new **Desktop Form**
- Now drag and drop a **PGird** object on the **Default Form**
- For filling out the values, you can use the **Text** property of **PGird**.
- That's it! Press F5 to run your application.

While we have created a **PGird** in the above steps, this **PGird** will do nothing because it has not been programmed to do so. In the example below, we will create a **PGird** that mimics the actions of a spreadsheet that can calculate the average of various columns with numbers.

To explain **PGird**, we will create an application that would allow a user to enter details about some students and certain values based on the column headings and get a sum total.

Follow the steps given below:

• Open PIDE2 and create a new **Desktop Form** project from the Select **New Project Type...** window. Using **Ctrl+N** is the keyboard shortcut for displaying **Select New Project Type...** window; alternatively you can also use **File Menu>New**.

sktop Form						
tegory:						
All Projec	cts					
A second se	Ê	\$		3		
Code File	Component Project	Console	Database	Database Project	Desktop Form	
(Reg)	0	()			Σ	
Desktop Game	Empty Project	Find Wizard	Help File	MenuBar	Mobile Form	
Mobile Game	Mobile VGA Game	Package	PPL Class	Procedural Project	Report	
scription:						

Figure 74: Create new project

• In the **Project Manager**, double click the **DefaultForm** object to open the **Form Editor**.

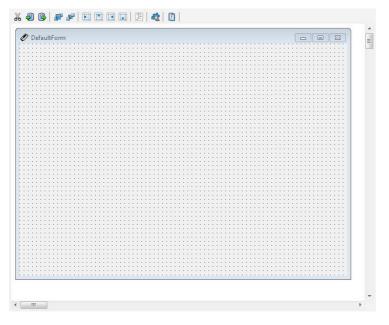


Figure 75: Form Editor

• We need to place a **PGird** on the Form. For doing this, click on the **PGird** object in the **Components Pane** and then click at the place where you want to place the **PGird** object. You can set the **PGird** according to your preference by dragging the borders to the desired size.

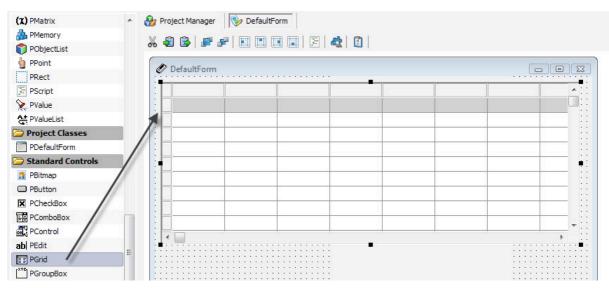


Figure 76: Drag PGrid

• Now, use the **Text** property of **PGird** to place appropriate heading for columns.

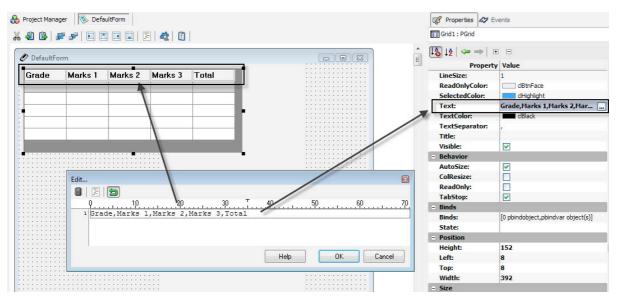


Figure 77: Text Property

• After filling out the columns in the text property of **PGird**, go back to the **Project Manager**.

😵 Project Manager 🛛 🦻 DefaultForm	
X 42 😫 × 42 🕈 🦊 🖌 🖋 📓 🗞 🚷 F: 🗐 🗳 🥸 🖹 😤 🧕	🔤 🛷
🙀 Project	v
K MyApplication	v
E DefaultForm	/ 🗸 🦏 📉
⊕ 🖺 MenuBar1	v
📰 Grid 1	14

Figure 78: Project Manager

Note: You can also use the **ColCount, RowCount** and the **ExtendedLastColumn** property to have more control over the layout of your **PGird**.

• In the **Project Manager**, click the **PGird** object and go to the associated events by clicking the **Events** pane.

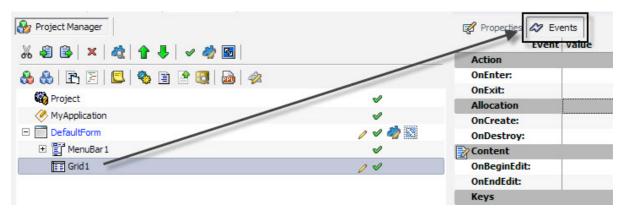


Figure 79: Events Pane

• In the Events pane, double click **OnEndEdit** event available under the Content column to create an **OnEndEdit** event.

🦻 Project Manager	Properties 22 Events
‰ € 🕃 × 🖧 ↑ 🗣 🗸 🛷 📓	Action
🗞 🚷 🖹 🗏 🖳 🧠 🖹 😤 🕄 🞰	OnEnter:
W Project	OnExit:
-	Allocation
MyApplication	OnCreate:
DefaultForm	
🕀 📑 MenuBar 1	✓ Content
- III Grid 1	⊘ ✓ OnBeginEdit:
proc OnEndEdit (Sender\$,	1\$, Row\$, Cance2\$ OnEndEdit: Grid1.OnEndEdit
	Keys
*	UnKeyDown:
	OnKeyPress:
	OnKeyUp:

Figure 80: Create event

• Drag a **PVariable** to the **OnEndEdit** event and name the local variable appropriately.

🗁 Class	 Project Manager 		
PComponent			
AP PEvent	- <u>- ~ 4</u> 🖹 🕒 × 🗠 🛧 🛧 🛩 🛷 🖾		
😳 PMethod	🔥 🚷 🖻 🔄 🕒 🥎 🖹 🔗 😡 🛷		
💑 PObject	= 🙀 Project	v	
PProperty	MyApplication	4	
🥃 PVariable 🔪		-	🤣 😒
CodeFlow	El MenuBar 1	4	
🔄 Break	E E File1	v	
P Continue	Grid1	14	
👆 Else	E proc OnEndEdit (Sender\$, Col\$, Rows	-	۲
🕪 Else If	<pre>Local (RowTotal\$);</pre>	14	۲
🕤 Exit		K (10) T (3)	

Figure 81: Drag PVariable

• Now, drag a **For** loop on the **OnEndEdit** event and drag the **PGird** object to the Finish property of **For** loop. This would result in a code complete window.

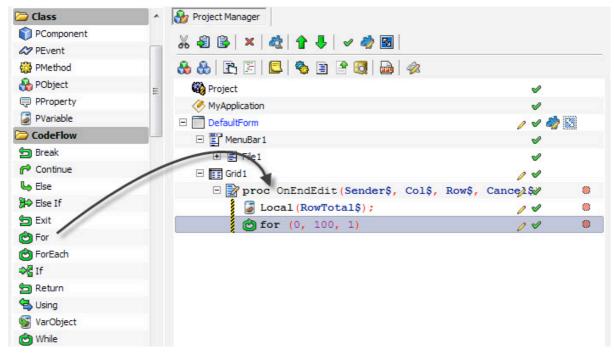


Figure 82: Drag For Object

೫ 🕄 📴 × 🎕 🕇 🦊 🛩 🤣 📓		C For 1		
🗞 🚷 🖹 🗐 📕 🧐 🗃 🔮 🔯 🛷			• ⊕ ⊡	
🍪 Project	~		operty Value	
MyApplication	~	- Parameters		
DefaultForm	/ 🗸 🦓 🕅	Finish:	100	
🗄 📳 MenuBar 1	× ~	Start:	PGrid	
Grid1		Steps: Variable:	🎝 Binds	a
🗆 📝 proc OnEndEdit (Sender\$, Col\$,	Rows, Cance &	•	🏂 Creating	E
Local (Variable1\$);	11	۲	Son Destroying	
😋 for (0, 100, 1)	14		% Updating	
			a State	
			🎝 Name	
			lowner 🕹	->
			la Components	=>
			🎝 Params	
			🚱 ID	
				~
			Help	Close

Figure 83: Set For properties

• Select **RowCount** in the code complete window.

- Parameters	
Finish:	100
Start:	best.
Steps:	PGrid T
Variable:	A RowCount

• Select the For loop again and specify a variable in it. In our example we will use y\$.

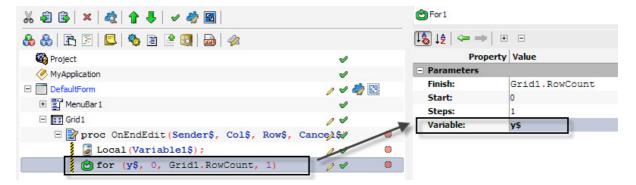


Figure 84: Change Property

• Alt + Drag the local variable to the For loop so that it comes below it. If the local variable is not below the For loop, drag it there.

X 🗐 🗟 × 🎕 🕈 🦊 🗸 🦓 🖪		
🗞 🚷 🖹 🗏 🔩 🔮 🖹 🔮 🔯 💩 🛷		
🙀 Project	V	
Key MyApplication	\$	
🗉 🛅 DefaultForm	14	🤣 🔊
🗉 🛅 MenuBar 1	V	
🖃 🛅 Grid 1	14	
🗆 📝 proc OnEndEdit (Sender\$, Col\$, Row\$,	Cance 2 St	۲
📓 Local (RowTotal\$) ;		
E 🙆 for (y\$, 0, Grid1.RowCount, 1) 🚽		
<pre>RowTotal\$ =;</pre>	14	۲

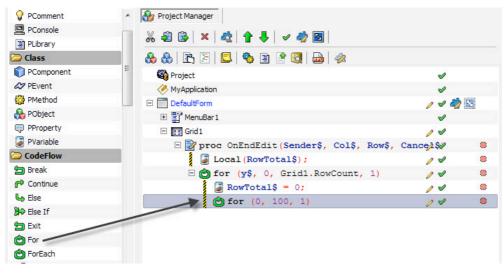
Figure 85: Alt + Drop

• Change the **Expr** property of the dragged variable to **0**. This would set the counter at **0**.

🖃 🛅 Grid 1			Logic	
	0 4		Logic:	
🖃 📝 proc OnEndEdit (Sender\$, Col\$, Row\$,	Cance 💱	۲	LogicAction:	1
Local (RowTotal\$);	14	۲	LogicExpr:	
🗆 😋 for (y\$, 0, Grid1.RowCount, 1)	/ 1	۲	- Parameters	
<pre>RowTotal\$ = 0;</pre>	04	۲	expr:	0
			Owner:	RowTotal
			- Status	
			Operation:	SetVariable
			Source:	RowTotal

Figure 86: Change Expr

• Drag another For loop form the Components Panel on the For loop available on the Project Manager.





• In the Finish property of the **For** loop, write (**Grid1.ColCount -1**). This should be enclosed within brackets so as to specify that it is not an Expression. Alternatively, you can also drag **PGird** object, select **ColCount** option and add **-1** to it later.

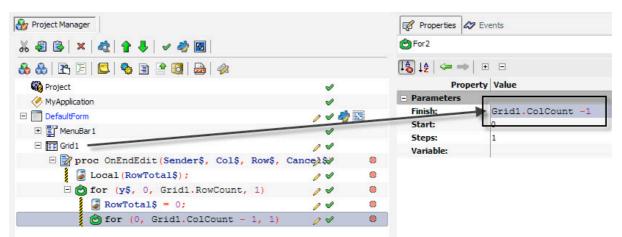


Figure 88: Change Property

• Tweak the Start property of this For loop to 2 and set the variable property to x\$.

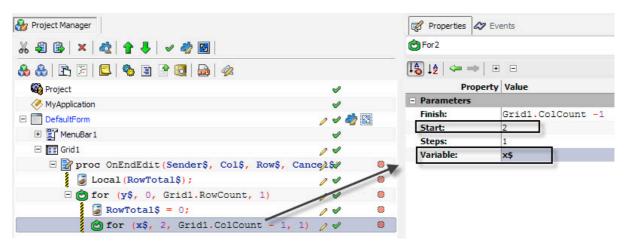


Figure 89: Change Property

• Drag the **PGird** object to the **For** loop. This would result in a **Code Completion** window.

🗉 🔄 MenuBar 1	v	
🖃 🥅 Grid 1	04	
🖃 📝 proc OnEndEdit (Sender\$, Col\$, Row\$, 🛛	Cance 🎗 💱	۲
<pre>Local (RowTotal\$);</pre>	14	۲
E of for (y\$, 0, Grid1.RowCount, 1)	14	۲
RowTotal\$ = 0;	14	۲
for (x\$, 2, Grid1.ColCount - 1,	1) 🥒 🖌	۲
PGrid		
<pre>Grid1\$</pre>	~	
🎝 Binds 📫		
🎸 Binds		
🍓 BeginCreating		
🎨 EndCreating		
🏂 Creating		
🏂 Destroying		
🎨 BeginUpdating		
🍓 EndUpdating		
🏂 Updating		
Global Variables 😨 💍		

Figure 90: Code Completion window

• Select Get(col\$,row\$) from the Code Completion window.

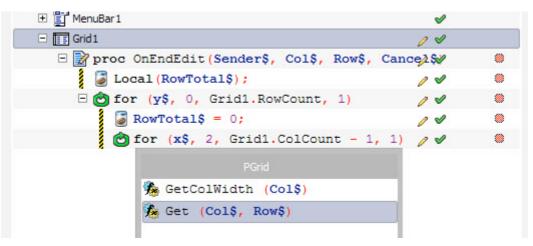


Figure 91: Select option

• Select the new Grid1.Get(col\$,row\$) and set its col as well as row property to x\$ and y\$ respectively.

688×1021↑€ √028				
b 🚷 🖹 🖹 🗳 🇞 🖹 🔮 🧕 🛷		+ ⊗ + ž	<=⇒ €	1
Contract Project	V	· · · · · · · · · · · · · · · · · · ·	Property	Value
MyApplication	v	- Paran	leters	
DefaultForm	/ 🗸 🦏 📓	Col:		×\$
🛨 🖺 MenuBar 1	V	KOW:		y\$
□ 🖬 Grid1	14 1	- Action		
E proc OnEndEdit (Sender\$, Col\$, Row\$,		Action		
	cancers	Opera	tor:	=
Local (RowTotal\$);		E Logic		
🗆 🙆 for (y\$, 0, Grid1.RowCount, 1)	114	Logic:		
RowTotal\$ = 0;	14	Logic/	Action:	
E of for (x\$, 2, Grid1.ColCoupt - 1,	1) 🦉 🖌	LogicE	xpr:	
Grid1.Get(x\$, y\$);	04	🔵 📃 Result	έ	
		Result	i:	
		- Status	5	
		Opera	tion:	MethodFunc
		Owner	c	Grid1
		Sourc	e:	Get

Figure 92: Change property

• Now, change the **Operator** property of **Grid.Get(x\$,y\$)** to += and drag the **RowTotal\$** variable to the result property. This would increment and store the result to this variable.

Project Manager			😴 Properties 🐼	Events
🐰 🗐 😫 🗙 🔩 🕈 🦊 🛩 🤣 🛃			Grid1.Get1	
🗞 🗞 🖹 🗐 🧐 🖹 🕈 🔞 💩 🛷			🚺 🕹 🖛 🛶	• =
🙀 Project	V		Proper	ty Value
MyApplication	v		- Action	
DefaultForm	I 🛷		Action:	
± ∰ MenuBar 1	v	Contracts (Operator:	+=
-	-		- Logic	
	v		Logic:	
🖃 📝 proc OnEndEdit(Sender\$, Col\$, Row\$, Can	cel\$⁄	۲	LogicAction:	
Local (RowTotal\$);	V	۲	LogicExpr:	
🗆 🕑 for (y\$, 1, Grid1.RowCount, 1)	V	۲	- Parameters	
RowTotal\$ = 0;	1	۲	Col:	x\$
G of for (x\$, 2, Grid1.ColCount -1, 1)	v	۲	Row:	y\$
RowTotal\$ += Grid1.Get(x\$, y\$);			- Result	
2	-		Result:	RowTotal\$
			Status	
			Operation:	MethodFunc
			Owner:	Grid1
			Source:	Get

Figure 93: Drag RowTotal\$

• Drag the Grid1 object to the parent For loop and select the Set(col\$, row\$,value\$) option. This would result in a Grid1. Set(col\$, row\$,value\$) code.

🎲 Project Manager		
೫ £ B × & ↑ ↓ ✓ 🧳 🖻		
🗞 🚷 🖹 🗏 🖳 🍫 🖹 🔮 😡 🚕		
🍪 Project	v	
Application	v	
DefaultForm	/ 🗸 🏹	
⊞ MenuBar1	v	
🗆 🛅 Grid1	14	
🗆 📝 proc OnEndEdit (Sender\$, Col\$, Row\$,	Cance 🎗 💱	۲
📓 Local (RowTotal\$) ;	14	۲
🗉 🙆 for (y\$, 0, Grid1.RowCount, 1)	14	۲
PGrid	14	۲
	1) 🥒 🖌	۲
SetFocus	14	۲
SetRect (r\$)		
🍓 SetBounds (1\$, t\$, w\$, h\$)		
🍓 SetColWidth (Col\$, Width\$)		
Set (Col\$, Row\$, Value\$)		
	1	

Figure 94: Drag and Drop

• Select Grid1. Set(col\$,row\$,value\$) and drag the PGird object to the Col property and select ColCount from the Code Completion box that appears.

s 🗐 🔒 🗙 💐 🛊 🎝 🦊 🗸 🋷 🦓 🙆			Grid1.Set1	
8 8 2 5 5 2 4 8 2 8 2 8			🕼 14 🖛 🔿 🗉	Ξ
Project	v		Property	Value
MyApplication	V		- Parameters	
DefaultForm	1 1 2		and the second se	Col
🛨 📳 MenuBar 1	1	and an other states	Row:	PGrid
- 📰 Grid1	04		Value:	A ColCount
E proc OnEndEdit (Sender\$, Col\$, Row\$, Cand	ce2\$/		Operation:	
<pre>Local (RowTotal\$);</pre>	11	×	Owner:	
🗆 😋 for (y\$, 0, Grid1.RowCount, 1)	10	/	Source:	1
RowTotal\$ = 0;	10/	۲		
🗆 🙆 for (x\$, 2, Grid1.ColCount - 1, 1)	10	۲		
RowTotal\$ += Get(x\$, y\$);	11	۲		
Grid1.Set(Col, Row, Value);	00	۲		

Figure 95: Drag operation

• Change the **Row** property to **y**\$

Project	v	Prop	erty Value
MyApplication	V	- Parameters	
DefaultForm	/ / 🎝 🖏	Col·	Grid1 ColCount
	2	Row:	y\$
		Value:	Value
	/ 1	- Status	
E proc OnEndEdit (Sender\$, Col\$, Row\$, C	ance2\$	Operation:	MethodProc
📓 Local (RowTotal\$) ;	14 1	Owner:	Grid1
🗆 😋 for (y\$, 0, Grid1.RowCount, 1)	/ /	Source:	Set
RowTotal\$ = 0;	/ / *)	
🗆 🙆 for (x\$, 2, Grid1.ColCount - 1,	1) / / (•	
RowTotal\$ += Get(x\$, y\$);	/ () () () () () () () () () () (
Grid1.Set(Grid1.ColCount, Row, V	alue 🔊 📈 🛛 🕷	•	

Figure 96: Change Row property

Change the value property of Grid1.Set(Grid.ColCount,y\$,value\$) to RowTotal\$ / (Grid1.ColCount - 2) + "%". This will divide the value in the RowTotal variable with that of the total number of columns with numerical values in our grid.

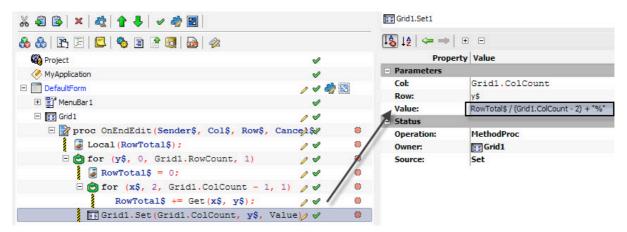
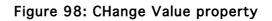


Figure 97: Change Property

• We will also need to round off the values and for that we will use the **RoundEx** method. To round the value to nearest two decimals, we will enclose the **value** property with **RoundEx(value,2)**+ "%".

🗞 🖛 🛶 🗉	Ξ
Property	Value
- Parameters	
Col:	Grid1.ColCount
Row:	vš
Value:	RoundEx(RowTotal\$ / (Grid1.ColCount - 2), 2) + "%"
- Status	
Operation:	MethodProc
Owner:	Grid1
Source:	Set



• Save your project by pressing Ctrl+S or go to the File Menu>Save. Give an appropriate name in the Save As.. window and press Save to save the file.

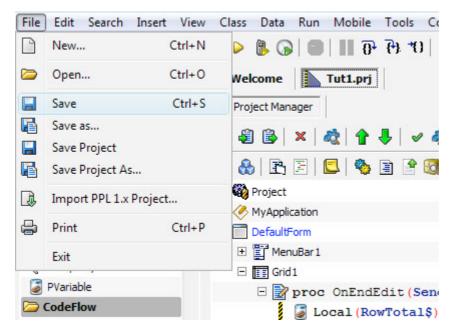


Figure 99: Save Project

• Run the project by pressing **F5** on the keyboard.

Grade	Marks 1	Marks 2	Marks 3	Total
1	80	70	60	70%
2	77	66	55	66%
3	87	65	68	73.33%
4	54	72	94	73.33%
5	62	55	71	62.67%

Figure 100: Output

Creating a PDBGird object

PIDE Users can use a **PDBGird** object to add grid access to a table and its columns. **PDBGird** not only allows a user to determine the layout of the data contained in a table, it also allows for easier readability of the data for a user. Not only is a user capable of showing data present in the table as a grid, he/she also gains the capability to display headings, column headings and manipulate table data with the help of **PQuery** object.

The **PQuery** object can be used to create queries that run on a table, these queries can be attached with **PDBGird** so that data returned by a query is displayed in a **PDBGird** as a programmer wants. Follow the steps given below to create a **PDBGird** object:

• Start with creating a **Desktop Form** project. For doing this, press **Ctrl+N** or go to **File>New** and select **Desktop Form** project from the **Select New Project Type..** window.

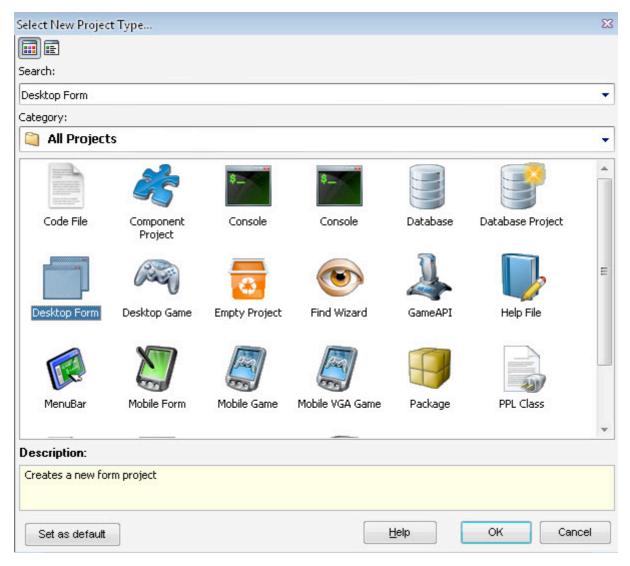


Figure 101: Create new project

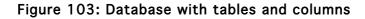
• Create a new database and then attach a table to it. Refer to tutorial given above for creating a table with data. In this example, we will use the same database created before in the earlier example.

‰43 B≥ × 43; 含 4 √ 42 B	
🗞 🚷 🖻 🗏 🗳 🎭 🖹 🖻 😡 📾 🛷 🤜 🗸	🔁 Database
Compared Project	File Edit View Favorites Too
MyApplication Image: DefaultForm	🚱 Back 👻 🌍 👻 🏂 🔎
	Address 🛅 D:\Database
Choose component type for "test.db" Image: PDatabase Image: PFile Image: PFile Image: PFile Image: PMemory Image: PResource	test.db Data Base File 2 KB

Figure 102: Drag Database

• Importing our database to the project will result in placement of the database on the project.

🏟 Project	v
🔗 MyApplication	v
🗄 💼 DefaultForm	🗸 🤣 🔛
= 📑 testi	04
🖂 🥅 TestTable1	04
Name1	/ 1
🗖 Age1	04
Active1	04



• **PDBGird** provides users with the facility to print elements in a table on screen in a grid like format. For printing our table in the database, we will drag the table on the DesktopForm object and select **Add Grid to control** from the context menu that appears.

🗞 🚷 🖻 🖻 🚨 🎭 🖹 🔮 😡 🥔 🕶 🔹	
🍪 Project	v
🥙 MyApplication 🛌	v
🕀 📄 DefaultForm	🗸 🥠 😒
🗆 🧾 test1	14
TestTable1	04
Name1	14
Age1	04
Active1	04

Figure 104: Drop database to Default Form

🏟 Project	V
🥙 MyApplication	v
DefaultForm	🗸 🥠 😒
Select smart move to apply	14
🗆 🔲 Add grid to control	14
Cancel	14
Age1	14
Active1	04

Figure 105: Smart Move

• Once **PDBGird** is available on the form, you can edit its properties to make it more suited for your application.



Figure 106: Change properties

• That's it! Save your project by pressing Ctrl+S or go to File>Save As.. to save the file on the hard disk.

File	Edit Searc	h Insert	View C	
3	New	Ct	rl+N	
D	Open	Ct	rl+O	
	Save	C	trl+S	
P	Save as			
	Save Project			
ß	Save Project	As		
ł	Import PPL	1.x Project		
4	Print	Ct	trl+P	
	Exit			

Figure 107: Save project

• Press **F5** to run the program.

e			
Alain Desc	5	TRUE	
Joe Bloe	10	TRUE	
Steve Mc	15		1
			•

Figure 108: Output

Using The Report Builder

Report builder is a tool that allows a user to create reports based on the data present in a database. After a database is created, we can easily create a report and then use that report to display data in a summarized form. The Report Builder contains various bands that are used to display data in the desired format. Given below are the various bands used with a report builder:

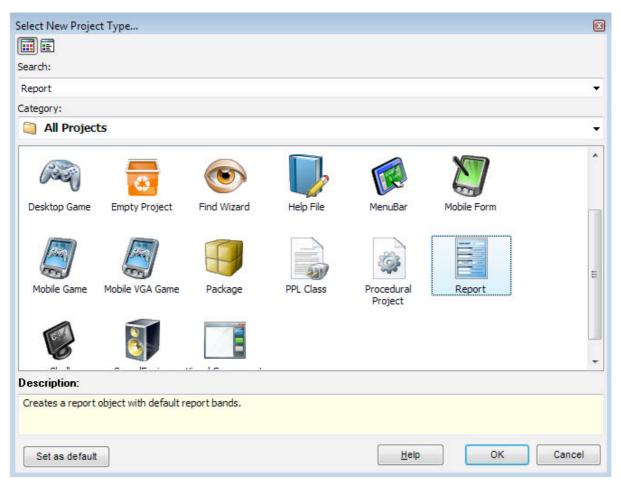
- **PageHeaderBand**: This is the band that describes the page Header for a report
- HeaderBand: This Band determines the Header section of the report
- **DetailBand**: Shows details
- FooterBand: As the name suggests, this band is present at the footer of the report
- PageFooterBand: Use this band for describing the page footer in a report

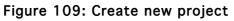
There are two ways to create a report builder:

Method1: Creating a standalone report builder and using it to create further projects.

Start PIDE and press Ctrl+N or go to File>New to create a new project.

• Now in the create Select New Project Type window, click on the Report icon.





• This will create an empty project with just the report and its associated bands.

Project Manager	
* 4 🖻 🔁 🗙 🔩 🕈 🖊 🖌 🛷 🥙 题 🗞 🗞 🖻 🖻 😓 🧠 🗃 🔮 🧕 🛷	
E Report1	/ 🗸 🧞 😒
PageHeaderBand 1	/ 1
HeaderBand 1	14
DetailBand 1	/ 1
FooterBand 1	14
PageFooterBand1	1 1

Figure 110: PReport

Now, you can drag various objects from the **Components Panel** and use them in conjunction with the report builder to create a solution with custom reports.

Method2: If you are using a current project and need to add a report to it, you can press **Ctrl+N** and select the **Report** option in the **Select New Project Type** window to create a report object with the existing project.

b 🚷 🖻 🗐 🧐 🗿 🖻 🔞 🥔 🛷	
Project	v
Report1	/ 🗸 🥠 🔀
PageHeaderBand 1	14
HeaderBand 1	11
DetailBand 1	11
FooterBand1	14
PageFooterBand1	14
K MyApplication	v
1 DefaultForm	V 🤣 🕄

Figure 111: PReport in existing project

Method3: Using Database Project. With database project, users will be able to create automatically generated copy of reports in their database driven forms. This method requires minimum configuration and is best suited for creating one click database applications.

X 🕄 📴 🗙 🙈 🎓 🖊 🗸 🛷 🌌	
🗞 🚷 🖹 🖹 📕 🍫 🖹 🔮 🔕 📾 🛷	
🍪 Project	V
MyApplication	v
🛨 🧾 Database1	v
💌 Query1	🗸 🧄 🔽
🗆 📄 Report1	🗸 🧄 😒
HeaderBand	v
🗆 📃 DetailBand	V
Ag Field 1	v
FooterBand	v
DefaultForm	V 🦓 🖏

Figure 112: PReport with Database

Using PValue, PList, PValueList And PObject Objects

Using PList

A **PList** is an object that can be used to hold a number of string or numerical values. Users can use **PList** to insert, append as well as delete values as they like. Apart from this, users can also use **PList** object to search, sort as well as get the values as text format. Given below is an example that would allow you to understand **PList** in a better way. In this example, we will sort five values according to their alphabetical order:

• Open PIDE2 and create a new Console project from the Select New Project Type... window. Using Ctrl+N is the keyboard shortcut for displaying Select New Project Type... window; alternatively you can also use File Menu>New.

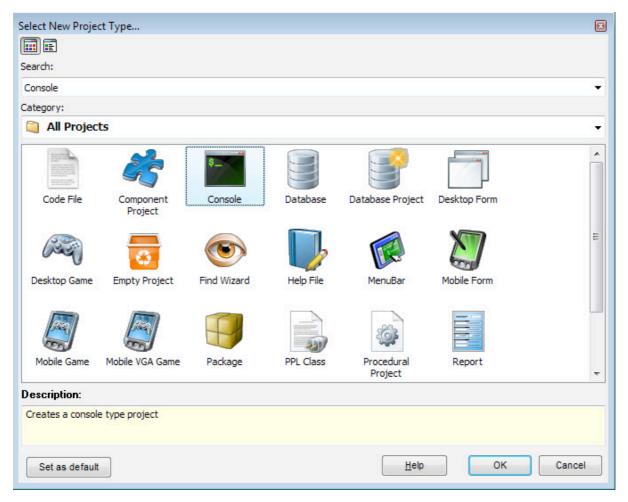


Figure 113: Create new project

• In the **Project Manager**, drag a **PList** object from the **Components Panel** and place it above the project **DefaultConsole**.

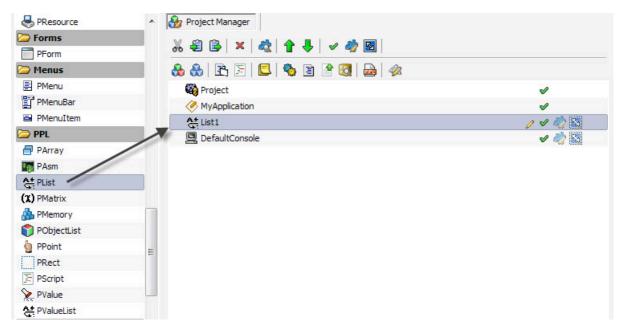


Figure 114: Drag PList

• Now we will create an **OnCreate** event on **DefaultConsole**. We need to create an event that will trigger an action after our objects have been created thus, we will use an **OnCreate** event. **OnCreate** event can be created by two ways; either by double clicking the **DefaultConsole** object or by selecting **OnCreate** in the Events section of the right click context menu of **DefaultConsole**.

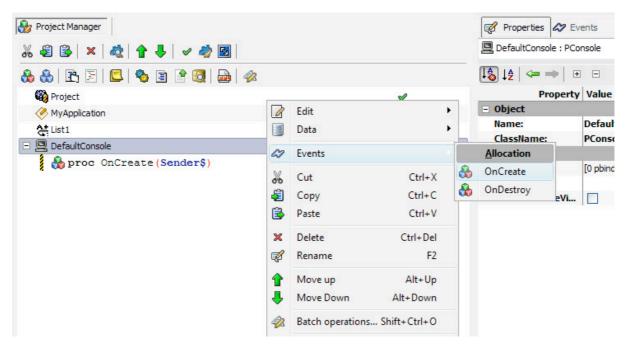


Figure 115: Create Event

• Now drag the **PList** object on **OnCreate** event. This will lead to a **Code Completion** window.

🎲 Project Manager	
‰ 43 🚱 × 42 含 ♣ < 42 🐼 🔤	
🗞 🚷 🖹 🖹 📕 🍓 🖹 🔮 🚳 📾 🛷	
Project	v
MyApplication	v
At List1	/ 🗸 🖏 🔀
DefaultConsole Sender\$)	
PList	
§ List1\$	
🖨 Binds 🔿 😑	
🎸 Binds	
🍓 BeginCreating	
🎨 EndCreating	
🏂 Creating	
🏂 Destroying	
BeginUpdating	
EndUpdating	
5 Updating	
Help Close	

Figure 116: Drag PList

• Select Add property from the Code Completion window.

At List1	/ 🗸 🏹 📓
DefaultConsole	/ 🗸 🚵 😒
🚷 proc OnCreate(Sender\$)	/ 🖌 (
PList	
🍫 Update (event\$, index\$) 🔶	
🍫 Clear	
🍫 AssignList (1\$)	
🍫 Assign	
🇞 AddList (1\$)	
🇞 Last	
🍫 Add 🚽	
<pre> Update (event\$, index\$) </pre>	
🎝 Value 📫	
♦ Value	
Help Close	

Figure 117: Select Add in Code Complete window

• Click List.Add() and write a value in its Expr property.

Project Manager		🜠 Properties 📣	2 Events
X 🖨 😰 × 🎕 🔒 🖊 🗸 🤣 🖾		Eist1.Add1	
8: & E: E 💷 🍫 🖹 🔮 🧕 🛷		[😓 👌 🖛 👄	• •
Project	v	Prope	erty Value
MyApplication	×	- Parameters	
At List1	1200	expr:	First
	1000	- Status	
		Operation:	MethodProc
E 🙀 proc OnCreate (Sender\$)	/ 🗸 🕚	Owner:	AtList1
At List1.Add ("First");	/ 🖌 🔍	Source:	Add

Figure 118: CHange Property

• Similarly, drag **PList** object on the **OnCreate** event and select the **Add** property from the **Code Completion** window. Writing another value in the **Expr** property of this **List.Add()** element will result in creation of one more value in the **PList** object.

🎲 Project Manager		🛒 Properties 🛷	P Events
🊜 🗐 📴 🛛 🗙 🔩 🎓 🦊 🖌 🛩 🤣 🔯		List1.Add2	
🗞 🚷 🖻 🗐 🗳 🧠 🖹 😤 🥘 🛷		🕼 🞼 🦛 👄	•
🍪 Project	v	Prope	erty Value
MyApplication	v	 Parameters 	
At List1	/ • •	expr:	Second
	100	- Status	
		Operation:	MethodProc
🖃 🂑 proc OnCreate (Sender\$)		Owner:	솭List1
List1.Add("First");	/ 🖌 🛛 🗎	Source:	Add
<pre>List1.Add("Second");</pre>	/ 🗸 🕚		

Figure 119: Change Expr Property

• We can create as many values in the **PList** object as we want.

🎲 Project Manager		
🊜 🤹 🔄 🗙 🚓 🛧 🦊 🛩 🤣 🔟		
🗞 🚷 🖹 🖺 🍫 🖹 🔮 🔞 🦗		
🙀 Project	v	
Average MyApplication	V	
List1	/ 🗸 🚵 😒	
🖃 📃 DefaultConsole	/ 🗸 🦓 😒	
🗆 🚷 proc OnCreate (Sender\$)		
<pre>At List1.Add("First");</pre>	/ /	
<pre>List1.Add("Second");</pre>	/ /	
<pre>List1.Add("Third");</pre>	/ /	
<pre>List1.Add("Fourth");</pre>	/ /	
<pre>List1.Add("Fifth");</pre>	/ /	

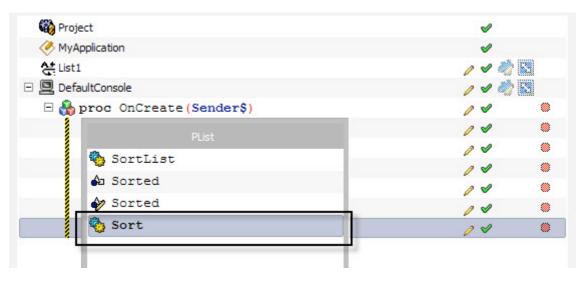
Figure 120: Add List

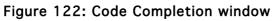
• Now we will sort the values stored in our **PList** object. For doing so, drag the **PList** object on the **OnCreate** event while holding the **Alt** key on the keyboard. This will allow us to use the object when the **DefaultConsole** is created. The **Alt+drag** operation will result in a **Code Completion** window.

🚷 🖻 🖻 🗳 🛙	1 🕒 🔯 📾 🛷		
Project	With Alt Key	V	
MyApplication	With Alt Key	v	
At List1		0 4 4	
		/ 🗸 🖏	
E 🍓 proc OnCreate (Sen orf)	/ /	
	PList	14	
List1\$ A Binds Binds	*	04	
a Binds		14	
♦ Binds	E	14	۲
BeginCreat:	ing	/ 1	۲
	2013		
S EndCreating	3		
🏇 Creating			
🏂 Destroying			
🍓 BeginUpdat:	ing		
🍓 EndUpdating	a.		
🏂 Updating			
A	· ·		

Figure 121: Drag with ALT key

• In the **Code Completion** window, select Sort property.





Project	II.	
MyApplication	v	
List1	/ 🗸 🦓 🖁	1
🗆 📃 DefaultConsole	/ 🗸 🦓 🖁	1
🗆 🚷 proc OnCreate (Sender\$)	04	۲
List1.Add("First");	14	۲
List1.Add("Second");	00	۲
<pre>List1.Add("Third");</pre>	14	۲
List1.Add("Fourth");	04	۲
List1.Add("Fifth");	14	۲
😂 List1 = new PList	0 4	۲
At List1.Sort();	04	۲

Figure 123: Sort list

- Now we need to print our **PList**. For printing the values, we will use the help of **PrintLine** command.
- Click **OnCreate** event and press **Ctrl+Space** bar to bring **Code Completion** window.

🎲 Project Manager	
೫ € 🗟 × & ↑ 🗣 ✓ 🛷 🛃	
🗞 🚷 🖹 🗐 🧠 🖹 🔮 🔞 💩 🛷	
🖏 Project	v
A MyApplication	v
At ValueList1	/ 🗸 🖏 😒
🗆 📃 DefaultConsole	/ 🗸 🖏 🔀
🗏 🂑 proc OnCreate (Sender\$)	/ 🗸 🛛 🍽
PDefaultConsole	/ 🗸 🛛 🔍
<pre>© DefaultConsole\$</pre>	/ 🖌 🕚
PApplication	
PObject	
© PBindObject	
© PBindVar	
A PCustomList	
At PList	
PArray	
(X) PMatrix	
PObjectList	

Figure 124: Code Complete window

• Select **PrintLine** object and drag the **PList** object to the **Value** property of **PrintLine**.

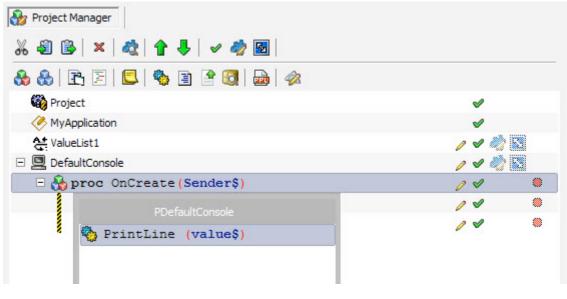


Figure 125: The value property

🎲 Project Manager			📝 Properties 🛷 E	Events
೫ 🕄 📴 🗙 💐 🛊 🖊 🛩 🤣 🔟			DefaultConsole.Prin	tLine 1
🗞 🚷 🖻 🗏 😫 🧐 🗃 🔮 🔕 📾 🛷			🕼 🞼 🖛 👄	• -
Project	<i>s</i>		Propert	y Value
MyApplication	v		- Parameters	
At List1			value:	value
DefaultConsole	0 4 🦏		Status	PConsole
🗆 🚷 proc OnCreate (Sender\$)	14	۲	Operation: Owner:	🎝 Binds 🗢
List1.Add("First");	v	۲	Source:	🐕 Creating
List1.Add("Second");	V	~		Sestroying
<pre>List1.Add("Third");</pre>				Debologing Mating
<pre>List1.Add("Fourth");</pre>	-	۲		
<pre>List1.Add("Fifth");</pre>	1	۲		la State
At List1.Sort();	1	۲		🎝 Name
PrintLine (value);	04	۲		🎝 Handle
. —				♠ CommandLineVisible
				🎝 Text
				♠ Value
				0
				Help Close

Figure 126: Change the value property

• The drag operation above will result in a **Code Completion** widow; select the **Text** command in it. Doing this will pass the text of **PValueList** to the **PrintLine** method which will then print it on the screen.

😂 📴 🗙 💐 😭 🦆 😺 🖉		9	DefaultConsole.PrintL	ine 1
🚷 🖹 🗏 🗳 🖹 🔮 🔯 💩 🛷		18	↓ ≜ 🦛 👄 🕀	Ξ
🙀 Project	v	<u></u>	Property	Value
MyApplication	v		Parameters	
tist1	V 🦄	23		Value
DefaultConsole	/ 1	1. A.	Status	PList
🗆 💑 proc OnCreate(Sender\$)	14		Operation: Owner:	🏟 Text
<pre>List1.Add("First");</pre>	v		Source:	L.
<pre>List1.Add("Second");</pre>		۲		1
<pre>List1.Add("Third");</pre>	/ *	۲		
<pre>List1.Add("Fourth");</pre>	/ /	۲		
<pre>List1.Add("Fifth");</pre>	v	۲		
<pre>List1.Sort();</pre>	v	۲		
PrintLine (Value);	14			

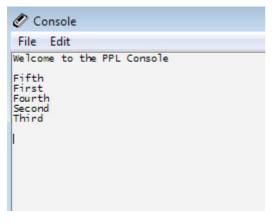
Figure 127: Add Value

• Save your project by pressing **Ctrl+S** or go to the **File Menu>Save**. Give an appropriate name in the **Save As..** window and press **Save** to save the file.

7	New	Ctrl+N	🕨 🖪 🞯 🛛 🔀 🖓 😗 🕲 🖉		
>	Open	Ctrl+0	Welcome Tut2.prj		
	Save	Ctrl+S	Project Manager		
	Save as Save Project		\$ \$ × \$ \$ 1 € \$ 8 8		
6	Save Project As		₩ Project	v	
J.	Import PPL 1.x Pro	oject	MyApplication	×	
₽	Print	Ctrl+P	At List1	v 🔌	
	Exit		DefaultConsole	V 🤌	
-			🗆 🗄 🌺 proc OnCreate (Sender\$)	s second	۲
1	PAsm		<pre>List1.Add("First");</pre>	\$	۲
4	PList		<pre>List1.Add("Second");</pre>	v	۲
(X)	PMatrix		<pre>At List1.Add("Third");</pre>	v	۲
۵	PMemory		At List1.Add("Fourth");	V	۲
٢	PObjectList		At List1.Add("Fifth");	V	۲
5	PPoint	=	At List1.Sort();	v	۲
	PRect		PrintLine (List1.Text);	1	۲
17-1	PScript				

Figure 128Save Project

• After saving the project, press F5 or run the project to see the output.





Note: By placing the **PValueList** before **DefaultConsole**, we are assuring that **PValueList** is created before **DefaultConsole** when the project compilation happens.

Using PValue

Just like a **PVariable**, **PValue** object can contain either a numerical value or a string value. Using **PValue** is a great way to use methods like **Uppercase**, **Random** etc to your application. Given below is an example that will show you how to use **PValue** as a container for various values and tweaking them with its help. In the example given below we will convert a string of alphabets in lowercase to uppercase with the help of **PValue**:

• Open PIDE2 and create a new **Console** project from the **Select New Project Type...** window. Using **Ctrl+N** is the keyboard shortcut for displaying **Select New Project Type...** window; alternatively you can also use **File Menu>New** also.

lect New Proje	ct Type						
arch:							
Console							
ategory:	2070						
All Projec	ts						9
Code File	Component Project	Console	Database	Database Project	Desktop Form		
Desktop Game	Empty Project	Find Wizard	Help File	MenuBar	Mobile Form		:
		P					
Mobile Game	Mobile VGA Game	Package	PPL Class	Procedural Project	Report		
escription:							
Creates a consol	e type project						
0-1				Help	ок	Canc	al
Set as default				Teib	UK	Canc	CI

Figure 130: Create new project

• Create an **OnCreate** event on **DefaultConsole**. We need to create an event that will trigger an action after our objects have been created thus, we will use an **OnCreate** event. **OnCreate** event can be created by two ways; either by double clicking the **DefaultConsole** object or by selecting **OnCreate** in the Events section of the right click context menu of **DefaultConsole**.

& 🚷 🖹 🗐 📕 🧐 🖉 📾	1 🎪 🤤	•		
🏟 Project				s and a second s
MyApplication				v
DefaultConsole Boroc OnCreate(Sender\$)		Edit Data Events	•	 ✓ ✓ ✓ ✓ ▲llocation
	*	Cut Copy Paste	CUITA	OnCreate OnDestroy

Figure 131: OnCreate event

• After an **OnCreate** event is created, drag a **PValue** object on it. This will create a **PValue** object declaration and also trigger a **Code Complete** menu.

At PList	🚷 🚷 🖹	🔄 🔼 🥎 🖹 🔮 🔕 📾 🛷 🧐 🔻					
(X) PMatrix	Project						
A PMemory		MyApplication					
PObjectList	E Default						
PPoint		roc OnCreate (Sender\$)					
PRect		Value1 = new PValue					
E PScript							
> PValue	7	PValue					
A PValueList	_	🎝 Binds 🗢 🔺					
Standard Controls		🎸 Binds 🗉					
PButton	_	BeginCreating					
X PCheckBox		C EndCreating					
PComboBox		S Creating					
PControl		A Destroying					
ab PEdit							
EE PGrid		BeginUpdating					
PGroupBox	=	SendUpdating					
PImageList		🏇 Updating					
Ag PLabel		🎝 State					
E PListBox		A. a					
III PListView							
E PMemo		Help Close					
PPaintCanvas							

Figure 132: PValue

• In the **Code Complete** menu, select value. This would create a **value1.value=**; statement.

🐣 🚷 🖹	E 🖪 🍫 🖹 🔮 🐯 📾 🛷 🧐 🗸
Project	
🔗 MyApplic	ation
🗆 📃 DefaultO	Console
🗆 🚷 pro	oc OnCreate(Sender\$)
8	Value1 = new PValue
	PValue
	🖧 Value
	le Value
	🎨 SaveToFile (Filename\$)
	🏇 Divide (value\$)

Figure 133: Value property

• Click on the newly created Value1.value=; statement and change its Expr property to "this all is in small but not for long".

26 🕄 😂 🗙 🙀 🖨 🔿 🕈 🦆 🛩 🤣 🛃		Xalue 1. Value 1	
🗞 🏡 🖹 🗐 🧐 🖹 🔮 🕄 🞰 🛷 🎯 🗸		[🕹 🗢 👄 🗉	
Project	1	Property	Value
MyApplication	×	- Parameters	
	/ / 🖏 🖾	expr:	"this all is in small but not for long"
		Action	
<pre> By proc OnCreate(Sender\$) Xalue1 = new PValue Value1.Value = "this all is in small but not for </pre>		Action:	
valuel = new Pvalue	/ 🖉 🔍	Operator:	=
Value1.Value = "this all is in small but not for	long"; / 🖉 🛛 🔍	- Logic	

Figure 134: Change Expr property

• Now, click on the **OnCreate** event and press **Ctrl+Space** bar to trigger code complete menu. Select **ShowMessage**() from it. This will be used to display the string in **Uppercase** in a message box.

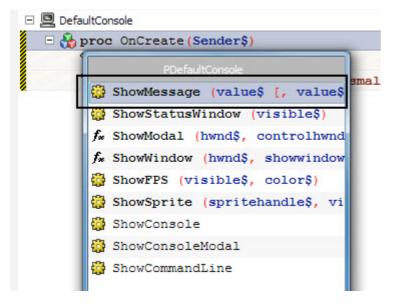


Figure 135: Select ShowMessage

• Drag the Value1 object to the value property of ShowMessage and select UpperCase from the code complete menu that appears.

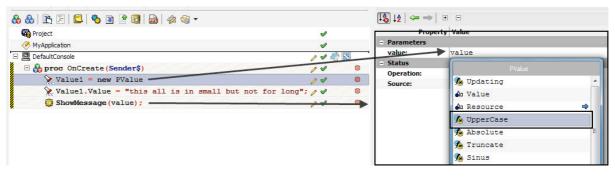


Figure 136: Change UpperCase from Code Completion

• That's it! You are ready with an application that converts lowercase letters to uppercase by using **PValue**. Save your project by pressing **Ctrl+S** or go to the File **Menu>Save**. Give an appropriate name in the **Save As..** window and press **Save** to save the file.

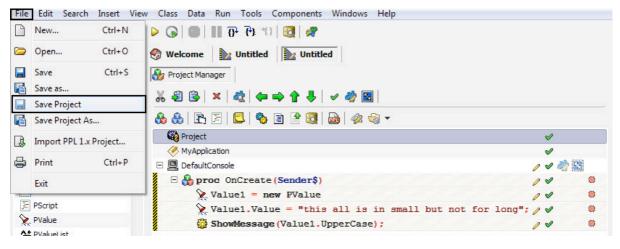


Figure 137: Save Project

• Press F5 to run the project and see the letters in uppercase.

File Ed	lit
elcome	to the PPL Console
ſ	Message 🗾
	THIS ALL IS IN SMALL BUT NOT FOR LONG
	ОК

Figure 138: Output

Using PValueList

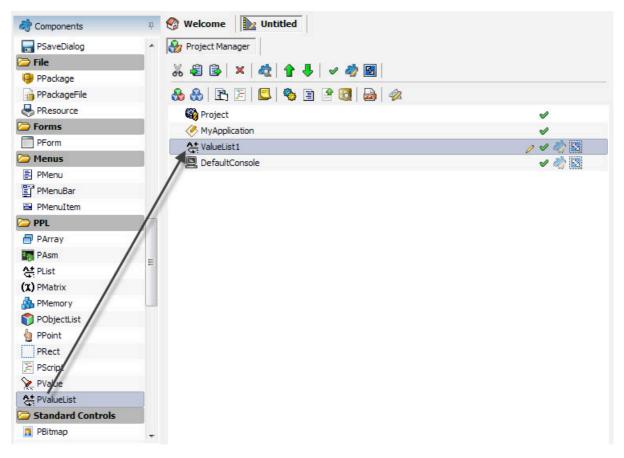
Like the name suggests, a **PValueList** is a **PList** object that contains **PValue** objects in it. Given below is an example that will convert two **PValueList** values to uppercase.

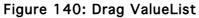
• Open PIDE2 and create a new Console project from the Select New Project Type... window. Using Ctrl+N is the keyboard shortcut for displaying Select New Project Type... window; alternatively you can also use File Menu>New also.

elect New Proje	ct Type					
earch:						
Console						
ategory:						
alegory.	ts					9
Code File	Component Project	Console	Database	Database Project	Desktop Form	
Desktop Game	Empty Project	Find Wizard	Help File	MenuBar	Mobile Form	:
Mobile Game	Mobile VGA Game	B	PPL Class	Procedural		
	Mobile VGA Game	Package	PPL Class	Project	Report	
escription:						
Creates a consol	e type project					
						- Consul
Set as default				<u>H</u> elp	ок	Cancel

Figure 139: Create New Project

• In the **Project Manager**, drag a **PValueList** object from the **Components Panel** and place it above the project **DefaultConsole**.





• Create an **OnCreate** event on **DefaultConsole**. We need to create an event that will trigger an action after our objects have been created thus, we will use an **OnCreate** event. **OnCreate** event can be created by two ways; either by double clicking the **DefaultConsole** object or by selecting **OnCreate** in the Events section of the right click context menu of **DefaultConsole**.

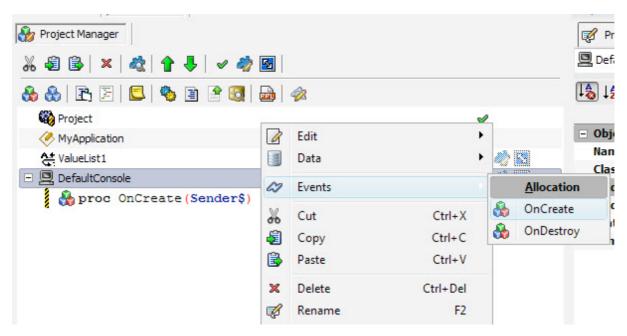


Figure 141: Create OnCreate Event

• Drop the **PValueList** object on the **OnCreate** Object. Doing this will allow us to specify actions on the creation of **DefaultConsole**.

* 🕄 🕻	3 × 🖧 🕇 🖊 🗸 🧳 🖸	
88	R 🗉 🗖 🍫 🖻 🔮 🚳 🍻	
Pro Pro	ject	A
🖉 My/	Application	v
🚑 Valu	ueList1	/ 🗸 🥠 🔣
🗆 📃 Def	faultConsole	/ 🗸 🦓 🔯
8	proc OnCreate (Sender\$)	/ 🖉 🗶
	PValueList	
	ValueList1\$ ^	
	🍫 BeginCreating	
	🎨 EndCreating 😑	
	% Creating	
	beginUpdating	
	bindUpdating	
	🧏 Updating	
	🚱 State	
	♦ State	
	🎝 Name	
	A	
	Help Close	

Figure 142: Code Completion window

• The Drop operation will result in auto **Code Completion** box. Select **Add** in it as we have to add values in it.

🎲 Project Manager	
🚜 🤹 😫 💐 😭 🛉 🖊 🖌 🛩 🤣 🛃 🛛	
🗞 🚷 🖹 🗐 🔩 🎭 🖹 🔮 🥘 💩 🛷	
📸 Project	v
MyApplication	V
A+ ValueList1	/ 🗸 🖏 🔀
🖂 🛄 DefaultConsole	/ 🗸 🖏 🔀
🍓 proc OnCreate (Sender\$)	/ 🗸 👋
PValueList	
🇞 AddList (1\$)	
🎨 Add	
Add (value\$)	

Figure 143: Select Add

• Enter the Value you want to give to your first value in the **PValueList**. To do this, select the newly created **PValueList.Add** object and change its **Value** property to the test you want it to show.

Welcome 📄 Untitled		Properties	
Project Manager		🥳 Properties 🖧	2 Events
‰ ቆ 😰 × & 含 ♣ ✓ 🤣 🛐		At ValueList1.Add1	
🗞 🚷 🖻 🗐 🧠 🗃 🔮 🚳 💩 🛷		[👌 🕹 🛉 📥	+ -
Project	v	Prope	erty Value
MyApplication	1	- Parameters	
At ValueList1	/ 4 🦚 🔀	value:	CD's
DefaultConsole	1200	Status	
E 🏠 proc OnCreate (Sender\$)		Operation:	MethodProc
		Owner:	섵 ValueList1
At ValueList1.Add("CD's");	2 🖉 🧶	Source:	Add

Figure 144: Change Value Property

- The above steps mark the creation of a list value in our **PValue**Lise object. Now we will learn how to add one more object in our list.
- To add another object in the **PValueList**, drag the **PValueList** object to the **OnCreate** event and select Add from the **Code Completion** window. Change the text of the **value** property to create yourself another list value.

🎲 Project Manager		🛒 Properties 🛷	' Events
X 🗐 🗟 🗙 🏘 🛊 🗣 🗸 🛷 🦓 🖻		AlueList1.Add2	
🗞 🚷 🖻 🗐 🗳 🎕 🖹 😤 🥘 🛷		🔩 🞼 🖛 🖚 🕸	+ -
Project	v	Prope	rty Value
MyApplication	V	- Parameters	
At ValueList1	1100	value:	DVD's
DefaultConsole	1000	- Status	
		Operation:	MethodProc
🗆 🂑 proc OnCreate (Sender\$)		Owner:	At ValueList1
<pre>ValueList1.Add("CD's");</pre>	/ /	Source:	Add
At ValueList1.Add("DVD's");	/ 🗸 🔍	Jources	l'ind

Figure 145: Change Value Property

• Again, follow the same procedure of dragging and dropping the ValueList object on the OnCreate procedure and selecting Add from the Code Completion window. One can create as many list values as he or she likes by following the steps given above. After all the other values have been created, we will have to make them in Uppercase with the use of Uppercase property.

🇞 Project Manager		
೫ 🕄 🗟 × 🎕 🕈 🖊 🗸 🤣 🔄		
🗞 🚷 🖹 🗏 🔩 🔮 🔮 🚱 🞰 🛷		
🙀 Project	V	
Application	v	
At ValueList1	V 🤌	83
🖃 🖳 DefaultConsole	/ 🗸 🔌	83
🗆 🚯 proc OnCreate (Sender\$)	14	۲
<pre> ValueList1.Add("CD's"); </pre>	v	۲
<pre>ValueList1.Add("DVD's");</pre>	v	۲
<pre> ValueList1.Add("Mp3 Players"); </pre>	v	۲
<pre>ValueList1.Add("Music cassettes");</pre>	v	۲
<pre>ValueList1.Add("Digital Music");</pre>	v	۲

Figure 146: Project View

• A list contains various items and this for making each and every item in uppercase, we will have to convert each and every item to Uppercase. For doing this, we will have to use the **ForEach** loop. This loop will perform a specified action for each and every item in the **PValueList**. Drag the **ForEach** loop from **Components Pane** to the **Project Manager**.

PComponent	*	🎲 Project Manager			
APEvent					
😳 PMethod		೫ ₽ ₽ × 00 1 0 0 0 0 0 0 0			
💑 PObject		🚷 🚷 🖹 📃 🎭 🖹 🔮 🔯 🞰 🛷			
📮 PProperty		Project		1	
PVariable	E	MyApplication		1	
CodeFlow	7	At ValueList1		 N N 	1
🔄 Break	- 33		0	v 🧼 🛛	
P Continue		E 🎇 proc OnCreate (Sender\$)		 ✓ 	ч ())
lese Else		<pre>{ ValueList1.Add("CD's");</pre>		1	۲
🕽 Else If		<pre>A ValueList1.Add("DVD's");</pre>		1	۲
🖆 Exit		<pre>A ValueList1.Add("Mp3 Players");</pre>		1	۲
🖄 For		<pre>At ValueList1.Add("Music cassettes");</pre>		1	۲
🖄 ForEach —		<pre>A ValueList1.Add("Digital Music");</pre>		1	۲
₽¶ If		foreach (())	1	v	۲

Figure 147: Drag ForEach

• We will specify the **PValueList** list to the **ForEach** loop through the **Expr** property. Drag the **PValueList** object to the **Expr** property.

₩ 월 B• × 42 ↑ ↓ <i>∨ 4</i> 2 ⊠	ForEach1			
🗞 🚷 🖻 🖻 🗳 🧐 🖻 🤔 📾 🛷			🔩 🞼 🗠 🖚	
🍓 Project	v		-	erty Value
🥙 MyApplication	v		- Parameters	*
A+ ValueList1	/ / 🦏		ежри	
🗆 📃 DefaultConsole	/ 🗸 🦏		Output:	() /alueList
🗆 🌺 proc OnCreate(Sender\$)	04	-		🎝 Binds 🗢
<pre> ValueList1.Add("CD's"); </pre>		۲		🏂 Creating
<pre>ValueList1.Add("DVD's");</pre>	-	۲		h Destroying
<pre>At ValueList1.Add("Mp3 Players");</pre>	4	۲		S. Updating
<pre>ValueList1.Add("Music cassattes");</pre>	V	۲		A State
<pre>ValueList1.Add("Digital Music");</pre>	v	۲		
🙆 foreach (())	14	۲		An Name
PrintLine (ValueList1.Text);	v	۲		litems
				🏂 IndexOf (value\$)
				🏇 Index
				🏂 Next
				A

Figure 148: Drag PValueList

• In the Code complete window, select Items.

🎲 Project Manager			👩 Properties 🖌	> Events	
🊜 🗐 📴 🗙 💐 🛊 🌲 🖌 🛷 🦓 🛃			O ForEach1		
🗞 🚷 🖻 🗏 🔍 🥱 🗉 🔮 🔞 📾 🛷			🕼 🕹 🦛 🔿	•	
Project	<i>s</i>		Prop	erty Value	
MyApplication	×		- Parameters		
Att ValueList1	0 4 🦏	2	expr:	ValueList1.Items	
E DefaultConsole	1 4 🦓	3111100	Output:	PValueList	
🗆 💑 proc OnCreate(Sender\$)	14	۲		i Items	
<pre>ValueList1.Add("CD's");</pre>	v	۲		A Items	=>
<pre>At ValueList1.Add("DVD's");</pre>	v	۲			
<pre> ValueList1.Add("Mp3 Players"); </pre>	V	۲			
<pre> ValueList1.Add("Music cassettes"); </pre>	v	۲			
<pre>ValueList1.Add("Digital Music");</pre>	v	۲			
offoreach (ValueList1.Items)	11	۲			
PrintLine (ValueList1.Text);	V	۲			

Figure 149: Select Items

• Drop the **PValueList** object to the **ForEach** loop. This will bring a **Code Completion** window.

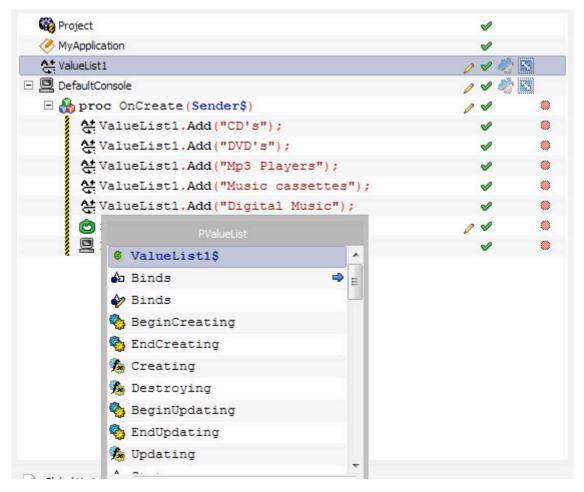


Figure 150: Code completion window

• Scroll to Items property from the list and click on the green arrow pointing towards right. This will again bring a **Code Completion** window.

At ValueList1	04	🧼 🔁
🗉 🛄 DefaultConsole	14	N 🔊
🗆 🚷 proc OnCreate(Sender\$)	14	
<pre> ValueList1.Add("CD's"); </pre>	V	۲
<pre>ValueList1.Add("DVD's");</pre>	v	۲
<pre>ValueList1.Add("Mp3 Players");</pre>	V	۲
<pre>ValueList1.Add("Music cassettes");</pre>	v	۲
<pre>ValueList1.Add("Digital Music");</pre>	V	۲
C PValueList	14	۲
A Items	s second	۲
♦ Items	-	
🎝 Items 🔿		

Figure 151: Select Items

• Select the Uppercase property in the new Code Completion window.



Figure 152: Select Uppercase

- Now we will have to display our **PValueList** values in the screen. For printing the values, we will use the help of **PrintLine** command.
- Click **OnCreate** event and press **Ctrl+Space** bar to bring **Code Completion** window.

🎲 Project Manager	
‰ 🕄 📴 🗙 🎎 🔒 🖊 🖌 🛷 🦓 🔯	
💑 🚷 🖻 🗐 🧠 🖹 🔮 🔯 🞰 🛷	
🙀 Project	V
MyApplication	v
🔆 ValueList1	/ 🗸 🚵 😒
🖃 🛄 DefaultConsole	/ 🗸 🦓 🔀
🗏 🏠 proc OnCreate(Sender\$)	/ 🗸 👋
PDefaultConsole	/ 🖉 👋
PDefaultConsole	/ 🗸 🔲
PApplication	
💑 PObject	
CO PBindObject	
© PBindVar	
💑 PCustomList	
At PList	
🗗 PArray	
(X) PMatrix	
PObjectList	
<u>H</u> elp <u>C</u> lose	

Figure 153: Code Complete Window

• Select **PrintLine** object and drag the **PValueList** object to the **Value** property of **PrintLine** object.

🧞 Project Manager		
& & E E E & & E & & A		
🏟 Project	v	
MyApplication	v	
At ValueList1	14	1
DefaultConsole	14	1
B 🔂 proc OnCreate (Sender\$)	04	۲
PDefaultConsole	14	۲
PrintLine (value\$)	11	



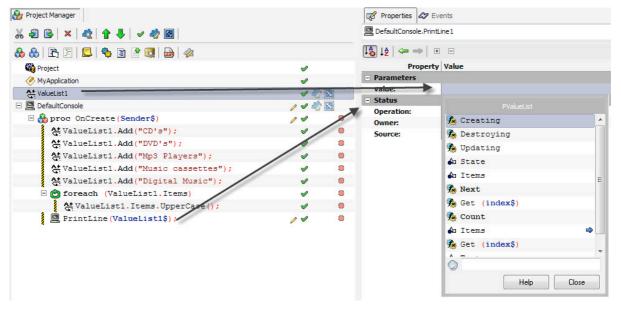


Figure 155: Change Value property

• The drag operation above will result in a **Code Completion** widow; select the **Text** command in it. Doing this will pass the text of **PValueList** to the **PrintLine** object which will then print it on the screen.

🎲 Project Manager			Properties 🛷 E	ivents
ፚ፝ቘ፝ 🗟 🗙 💐 🛧 🛊 🕹 🖌 🛷 🖓 🔯			DefaultConsole.Prin	tLine 1
🗞 🚷 🖻 🗏 📕 🧠 🖹 🔮 🚳 📾 🛷			🕼 14 🖛 👄 1	. □
Project	v		Propert	y Value
MyApplication	v		- Parameters	
At ValueList1	~			ValueList1.Text
DefaultConsole	040		- Status	PConsole
Approc OnCreate (Sender\$)	14		Operation:	la Text
At ValueList1.Add("CD's");	4		Owner:	el lext
At ValueList1.Add("DVD's");	1		Source:	
At ValueList1.Add("Mp3 Players");	1	۲		
<pre> ValueList1.Add("Music cassettes"); </pre>	/ .			
<pre>ValueList1.Add("Digital Music");</pre>	v	۲		
E 🙆 foreach (ValueList1.Items)	v	۲		
ValueList1.Items.UpperCase();	v	۲		
PrintLine (ValueList1\$);	14	۲		

Figure 156Change the Calue property

• Save your project by pressing Ctrl+S or go to the File Menu>Save. Give an appropriate name in the Save As.. window and press Save to save the file.

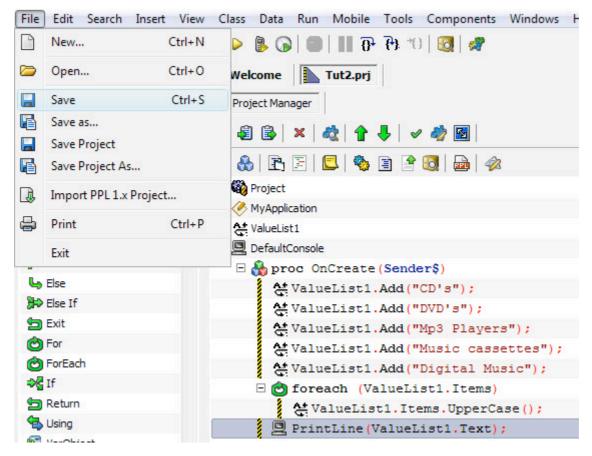


Figure 157: Save project

• After saving the project, press F5 or run the project to see the output.

🖉 Console	
File Edit	
Welcome to the PPL Console	
CD'S DVD'S MP3 PLAYERS MUSIC CASSETTES DIGITAL MUSIC	

Figure 158: Output

Note: By placing the **PValueLise** before **DefaultConsole**, we are assuring that **PValueList** is created before **DefaultConsole** when the project compilation happens.

Using PObject

Being a base class of all the objects in PCl (PPL Control Library), **PObject** is an important object but its use is more or less restricted to being a base class for other objects only. **PObject** can also be used to create a custom class component **PObject** as a basis. Given below is an example that will allow you to create a custom class that will have Username and Password as properties and will also have VerifyPassword as a method.

• Create a new component project. For doing this, click on the File>New menu item or press Ctrl+N on the keyboard to bring the Select New Project Type... window. Now select Component Project in this window and press Ok.

Select New Projec	t Type					
Search:						
Component Projec	t					•
Category:						
All Project	s					•
Code File	Component Project	Console	Console	Database	Database Project	-
Desktop Form	Desktop Game	Empty Project	Find Wizard	Game API	Help File	111
MenuBar	Mobile Game	Mobile VGA Game	Package	PPL Class	Procedural Project	
Description: Creates a new cor environement.	nponent project. 1	The component will be	visible in the Cor	mponents palette		-
Set as default				Help	OK Car	ncel

Figure 159: Create new project

• Delete the MyComponent object as we need to create our own class.

Project MyComponent	
	▼ √ M
Confirm	
Are you sure you want to d	felete the object(s)?
	Yes No

Figure 160: Delete Object

• Now drag a **PObject** object to the **Project Manager** and name it MyClass. For doing this press **F2** on the keyboard while the **PObject** is still selected and change the name of the object.

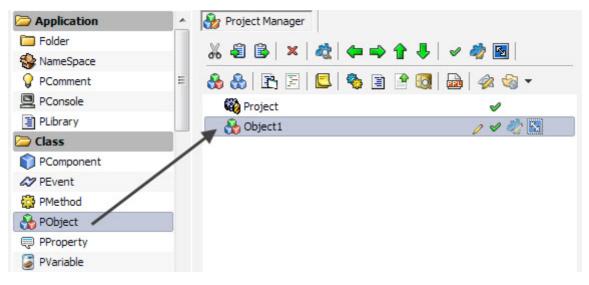


Figure 161: Drag PObject



Figure 162: Rename

• Now drag two **PProperty** objects to the **MyClass** object so that they become its child objects.

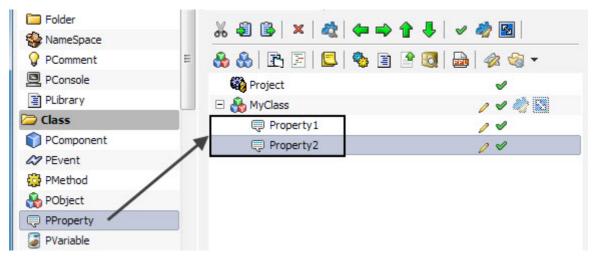


Figure 163: Drag PProperty to Project Manager

• Rename the first **PProperty** object to Username and the other to Password. Press **F2** on the keyboard to do so.

🄏 🤹 🗟 🗙 🖧 🖛 🔿 🤆	🎓 🦊 🖌 🧳 🜆
🚷 🚷 🖹 🗏 🖳 🎨 📓	🖹 🔯 🚘 🛷 🍕 🗸
Project	ø
🗆 💑 MyClass	/ 🗸 🖏 😒
🤤 Username	04
Password	14

Figure 164: Change PProperty Name

• After renaming both the **PProperty** objects, drag a **PMethod** object to the **Project Manager** and rename it as VerifyPassword.

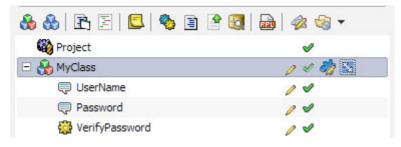


Figure 165: Drag PMethod and rename it

• After all the components of the class are declared, click on the **Class** menu item and select **Create Class**.



Figure 166: Create Class

• Clicking on the **Create Class** menu item will create a **MyClass** custom object in the **Components Panel** that can be used in the project. This component can be used in your code and will automatically have username as well as password as its property. The VerifyPassword method can be used to verify the password entered.

Creating A Game Visually

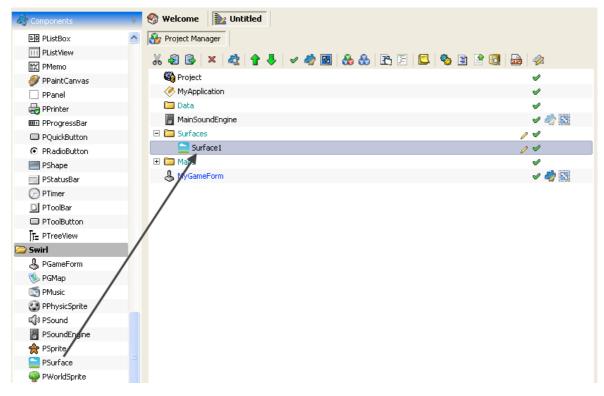
PIDE can be used to create a game visually. Creating a game without using any coding means that you will be able to create as well as run the game just by using drag and drop features of PIDE. Given below are the steps used to create a game visually through PIDE:

• Start by creating a new project and select a **Desktop Game** project in the **Select New Project Type..**

Select New Proje	ct Type					×
Search:						
Desktop Game						~
Category:						
🗎 All Project	s					*
Code File	Component		Console	Database	Database Project	
Desktop Form	Project	Project	Find Wizard	Help File	MenuBar	
Mobile Form	Mobile Game	Mobile VGA Game	Package	PPL Class	Procedural	
	6				Project	
Report	Shell	SoundEngine	Visual Component Project			
Description: Create a desktop	game					
Set as default			He		OK Cano	el

Figure 167: Create New project

• Once a new **Desktop Game** project is created, add a **PSurface** object to it by dragging and dropping the **PSurface** object to the surfaces folder. A **PSurface** is an object that contains a bitmap that is attached to a sprite.





• Double click the **PSurface** object and choose an image from your computer. Once you have chosen image is included in the bitmap, close the **surface editor**.

Surface1 - Surface		×	Object		
🗋 🗁 📑 🗳 🖻	× 🗉 🗓 🛃 🥖 📀		Name:	Surface1	
			ClassName:	PSurface	
			Appearance		
			ColorMask:	clFuchsia	
o 🔻 Open					? 🔀
Look ir	n: 🛅 Icons	🔽 🧿 🖻 💌	.	(48×48)	<u>a</u>
	address_book2	T barcode	Tox		
	airmail	The baseball	box_edit		
My Recent	armclock	battery	box into		
Documents	antenna		box_new		
	application	blackboard	box_out		
	application connection	The bolt	box_software		
Dealthan	application edit	bomb	box tall		Contraction of the local distribution of the
Desktop	application_enterprise_view	book_blue	box view		
	application_new	veries	brain		
		book_blue_new	branch	ALC: NOT THE OWNER.	
	application_server	book blue_view	branch_element		
My Documents	application_view	book bookmark	branch view		
	astrologer	bookmark blue	The brickwall		
	atom		📅 brickwall_warnin		- I
	band_aid	books blue edit	The brief case		
My Computer		\	_		
			2		
	File name: bomb	×	Open		
My Network	Files of type: All (*.png;*.gif;	.*.ipg;*.ipeg;*.bmp)	Cancel		

Figure 169: Surface Editor

• Expand the maps folder and click the **PGameMap** Object to open the **Game Map**.

• Create a **Sprite** object by double clicking the **PSprite** object on the **Game Map**. A **Sprite** is used to display the surface has a lot of attributes that can be used to manipulate how objects in game behave.

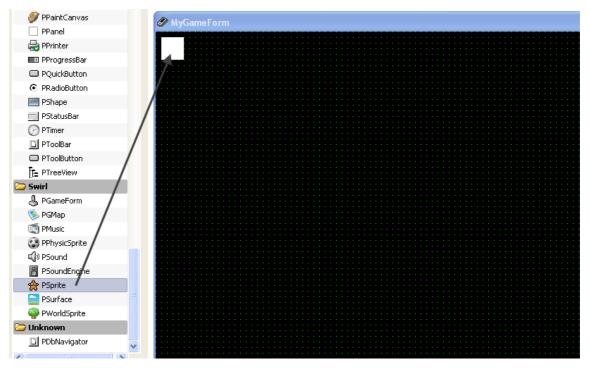


Figure 170: Drag PSprite

• While the **Sprite** object is still selected, go to its properties and specify the surface you created earlier in the **Surface** property.

K 4 🗟 😹 F F 🗉 🗉 🖼 🗐 🖉	🚖 Sprite1 : PSprite		
🖉 MyGameForm	1 🕹 🕹 🖛 🔿	+ =	
	Propert	ty Value	
r 🙀	🖃 Light		
	Light:	-1	
	LightRadius:		
	- Offset		
	AutoOffSetX:		
	AutoOffsetY:		
	OffsetX:		
	OffsetY:		
	Position		
	AutoScrollX:		
	AutoScrollY:		
	FixedX:		
	FixedY:		
	Height:	32	
	LayerIndex:		
	Left:	8	
	Top:	8	
	Width:	32	
	ZOrder:		
	- Surface		
	AlphaSurface:		
		-	
	AlphaWhite:		
	Index:		
	Surface:	Surface1	
	- Tiling	None	
/102/10.00 fps	🗸 TileX: 💙		
	TileY:	🚞 Surface1	

Figure 171: Select Surface Property

• Now we will create an event that would allow us to perform actions according to mouse movements. For implementing a move movement event, right click on the **GameForm** object and select **OnMouseMove** event from the **Events** tab.

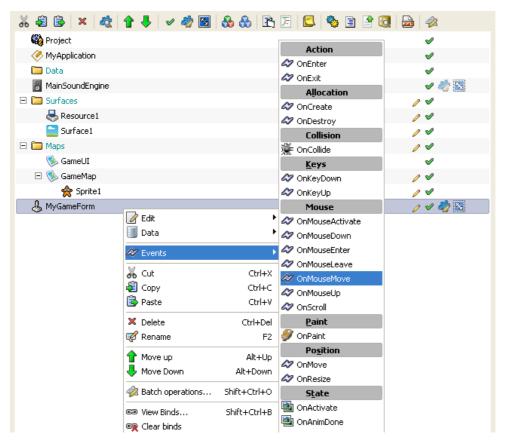


Figure 172: The events tab

• Drag the **PSprite** object present in your project and drop it on the newly created event. This will open the **auto insert** box that would allow you to choose a method from the list. Here you can write "**Move**" to choose the **Move** method.

X 🕄 🕒 🗙 🔌 🛉 🦊 🖌 🖉 🦓 💹 🚷 🚷 🗈 💷 🧐 😫	ol 🔁 🖗
🆓 Project	v
MyApplication	v
🛅 Data	v
B MainSoundEngine	a 🧄 🖉
🖃 🧰 Surfaces	/ 1
esource1	/ 1
🔁 Surface 1	/ 1
🖂 🧰 Maps	/ 1
🗞 GameUI	v
🗆 🌭 GameMap	/ 1
😤 Sprite1	04
🖻 🕹 MyGameForm	/ 🗸 🦓 🔯
👚 proc OnMouseMove(Sender\$, x\$, y\$)	/ 🖉 🛛 🔍
PSprite	
Printe	
🍫 Move (X\$, Y\$, Relative\$)	
Nove (X\$, Y\$, Relative\$)	
Move (X\$, Y\$, Relative\$)	

Figure 173: Code complete

• After the method is created, click on it and change its X as well as Y property to point to \$x and \$y.

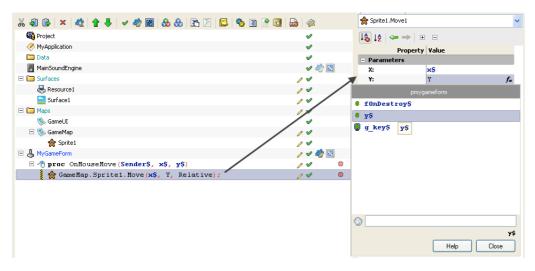


Figure 174: Change Property

• Once you have set the \$x and \$y variables, run the project to see your game in action.

The Subtleties Of The Visual Programming.

Loops and Conditions: For Loop

Having loops or iterations is very important of any programming language. Loops not only allow a programmer to perform a complex recursive task in an efficient manner, it also allows him/her to use other parts of programming in an efficient manner to result in a highly complex application.

Given below is an example that will allow you to process a set of instructions to a pre-set number of times if a condition is deemed true. In the given example, we will to print the factorial of a number entered by the user.

 Start by creating a new Desktop Form project. To do so, start PIDE and press Ctrl+N and select Desktop Form Project from the "New Project Type.." window. Alternatively, you can also go to File> New and select Desktop Form from the "New Project Type.." window.

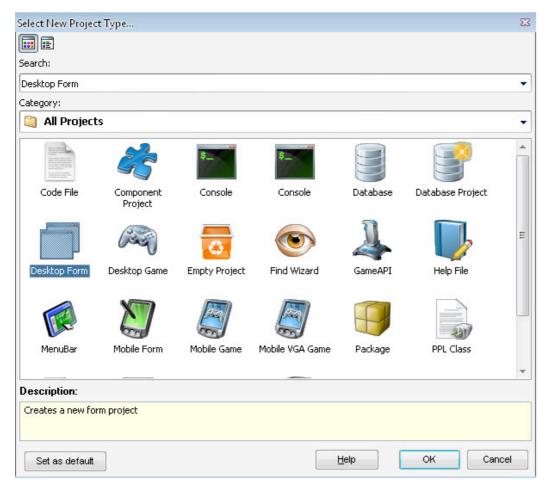


Figure 175: Create New Project

• In the project, double click the **Default Form** object to enter the **Form editor**.

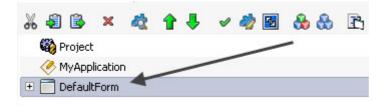


Figure 176: Double click

• In the **Default Form** Editor, we will place one **PEdit** box along with one **PButton**. While the **PEdit** boxes will contain the number we want to use for finding the factorial, **PButton** will be used for the initiation for all the actions we will have on the number. On the **Components Pane**, look for **PEdit** object and click on it; then, click on the form to place the **PEdit** object on the form. After placing a **PButton** just like other **PEdit** object, you are ready to start visual programming.

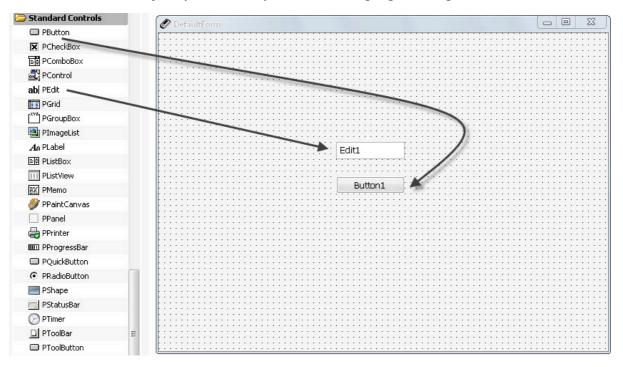


Figure 177: Place components in the form

• Once you have all the objects in place, click the **Project Manager** on the top left corner of the screen and return to it. In the **Project Manager**, click on the **PButton** so as to create an **OnClick** event. This event will be used to activate all the actions that we will be using once our **PButton** is clicked.

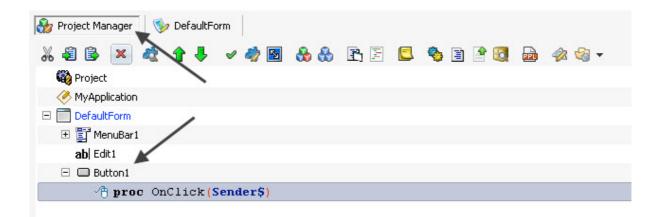


Figure 178: Project manager

• Like all the programming languages that require us to create variables so as to hold values and enable us to manipulate them, PIDE also does the same by providing the capability to create variables by first defining them and later declaring them. For doing this, drag a **PVariable** to the **OnClick** event. This will create a variable definition.

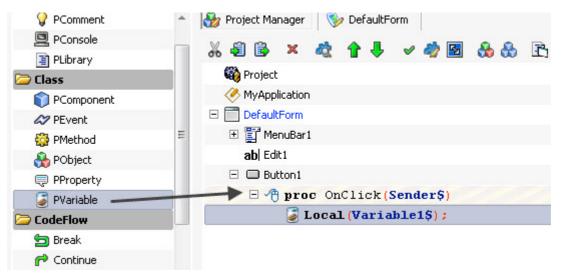


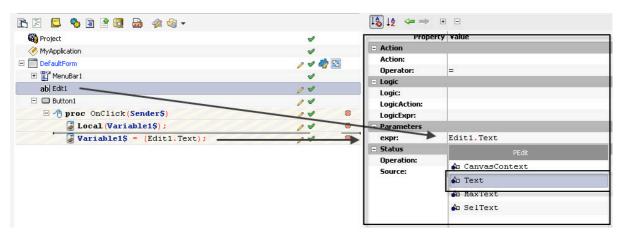
Figure 179: Drag PVariable

• Once the variable is declared, drag our newly created variable back to the **OnClick** event while keeping the **ALT** key pressed on the keyboard. This will create a variable that is ready to hold a value.

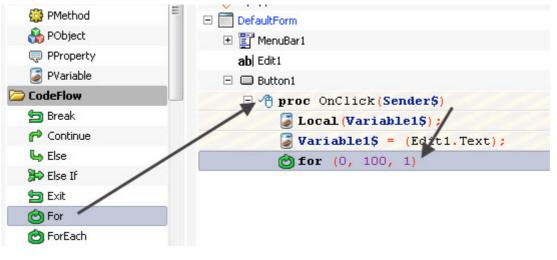
🍪 Project
K MyApplication
🖃 🧰 DefaultForm
🕀 📰 MenuBar1
ab Edit1
🖃 🗔 Button1
🗏 🖑 proc OnClick(Sender\$) 🔶
Local (Variable1\$) ;
<pre> Ø Variable1\$ = (); </pre>

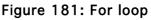
Figure 180: Drag with ALT key pressed

- Drag the **PEdit1** object to the **Expr** property of our newly created variable. This will initiate a code complete window.
- Select the Text property in the code complete window. This would give the value of the text entered by a user in our **PEdit** box to this variable.



• Drag a **For** loop on the **OnCreate** object. This would create a **For** loop that can be used to run a set of programming instructions specific number of times to get the factorial.





• Our For loop will need us to run the loop based on some conditions. Drag a new **PVariable** on the **OnCreate** event.

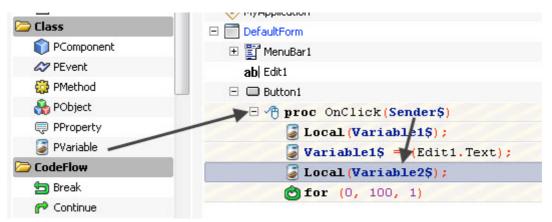


Figure 182: Drag PVariable on OnCreate

• Drag the newly created variable declaration back to the **OnClick** object while holding the **ALT** key. Once the variable is created change its **Expr** property to 1 so that our variable has a value 1.

🍪 Project	V		Property	Value
MyApplication	v		Action	
	/ 1 🎝		Action:	
🛨 📰 MenuBar1	v		Operator:	=
-	-		- Logic	
ab Edit1	0 4		Logic:	
🖃 🖵 Button1	14		LogicAction:	
🖃 🕂 proc OnClick(Sender\$)	14	۲	LogicExpr:	
Local ariable1\$);	14	۲	- Parameters	
<pre>Variable1\$ = (Edit1.Text);</pre>	14	۲	expr:	1
Local (Variable2\$) ;		۲	Status	
<pre> Variable2\$ = (1); </pre>	04	۲	Operation:	Set¥ariable
of for (0, 100, 1)	14	۲	Source:	🅃 ¥ariable2

Figure 183: Drag to OnClick

• Now, in the **For** loop, drag the newly created variable to the start property. This will ensure that our loop starts at the value contained in this variable i.e 1.

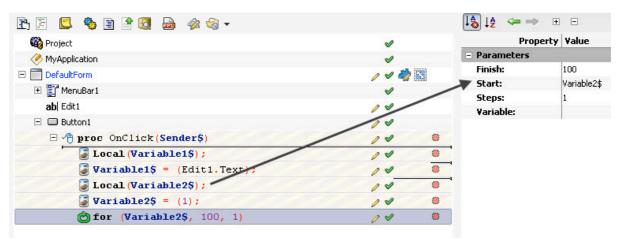


Figure 184: Drag variable to start property

• Drag Variable1\$ to the Finish property of our **For** loop. This will ensure that our loop plays the required number of times we want i.e equal to the number that is input in the **PEdit** box.

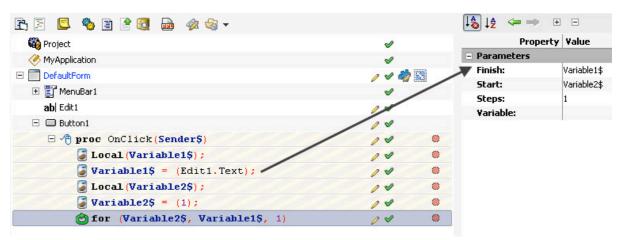


Figure 185: Drag Variable 1\$ to Finish

- Our For loop will run 1 step at a time so we will keep the Step property to 1.
- Drag a **PVariable** to **OnClick** event and drag the variable definition back to the **OnClick** object while keeping the **ALT** key pressed. This will create a new variable.

쳘 Class	E ToefaultForm	/ 🗸 🦏 📉
📦 PComponent	🛨 🕎 MenuBar1	2
🛷 PEvent	ab Edit 1	14
🍪 PMethod		14
💑 PObject	<pre>proc OnClick(Sender\$)</pre>	14 0
💭 PProperty	<pre>Local (Variable1\$);</pre>	14 0
🥃 PVariable	Variable1\$ = (Edit1.Text);	10 0
CodeFlow	Local (Variable2\$);	10 0
🔄 Break	<pre>Variable2\$ = (1);</pre>	/ *
P Continue	<pre>Local(Variable3\$);</pre>	/ *
👆 Else	<pre>Variable35 = ();</pre>	/ / .
🗯 Else If	C for (Variable2\$, Variable1\$, 1)	/ / 0
🔄 Exit		

Figure 186: Create New Variable

• Once this variable is created, drag its definition one more time to the **OnClick** event while holding the **ALT** key. This will create one more instance of this same variable. Change the **Expr** properly of one of this variable to 1.

	× -		Logic:	
🖃 🖵 Button1	/ 1		LogicAction:	
Proc OnClick (Sender\$)	14	۲	LogicExpr:	
Lool (Variable1\$);	14		- Parameters	
<pre>Wariable1\$ = (Edit1.Text);</pre>	14	۲	expr:	1
Local (Variable2\$);	14	۲	- Status	4
<pre> Variable2\$ = (1); </pre>	14		Operation:	SetVariable
Local (Variable3\$) ;	00	۲	Source:	/ 🕃 Variable3
<pre>Variable3\$ = (1);</pre>	04	۲		/
<pre>Variable3\$ = ();</pre>	14	۲		
for (Variable2\$, Variable1\$, 1)	14	۲		

Figure 187: Change Expr property

• Drag the other variable on the **For** loop.

🖃 🔲 Button1	/ 1	
🖃 🖑 proc OnClick(Sender\$)	11	۲
Local (Variable1\$);	14	۲
<pre> Wariable1\$ = (Edit1.Text); </pre>	14	۲
Local (Variable2\$);	14	۲
<pre> Variable2\$ = (1); </pre>	14	۲
🔪 🥃 Local (Variable3\$) ;	14	۲
Variable3\$ = (1);	14	۲
🗆 🔂 for (Variable2\$, Variable1\$, 1)	11	۲
<pre> Variable3\$ = (); </pre>	04	۲

Figure 188: Drag variable on For Loop

• Change the **Expr** property of our new variable (Variable3\$*Variable2\$). Doing this would ensure that variable3\$ will contain multiples of all the numbers from 1 (variable2\$) to the number that is entered in the **PEdit** box(variable1\$).

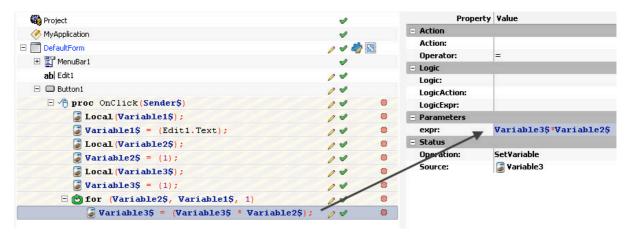


Figure 189: Change Expr property

• Click on the **OnClick** event and press **Ctrl+Space bar** to initialize a **code complete** window. Select **ShowMessage** property from the window.

3 🛅 DefaultForm	0	V	🧳 🕙
🛨 📑 MenuBar1		V	
ab Edit1	0	V	
🖃 🔲 Button1	0	V	
🖃 🖑 proc OnClick(Sender\$)	0	V	۲
PDefaultForm	1	V	۲
😳 ShowMessage (value\$ [, value\$	0	V	۲
	0	V	۲
	0	V	۲
	0	V	۲
	1	V	۲
	0	V	۲
(ariable2\$)	: 0	1	۲

Figure 190: Code complete window

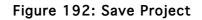
• Change the Value property of ShowMessage object to Variable3\$.

🖹 🖹 📮 🍫 🗎 🔮 🐯 📾 🛷 🧠 🗸		🞼 🖛 🖛 🔿	+ =	
🏟 Project	<i>s</i>	Prope	Property Value	
MyApplication	ø	- Parameters		
DefaultForm	/ / 2	value:	Variable3\$	
	2	- Status		
ab Edit1	11	Operation:	ProcCall	
	14	Source:	ShowMessage	
□ √↑ proc OnClick(Sender\$)	/* *	/		
<pre>Local (Variable1\$);</pre>				
	12 /			
<pre> Variable1\$ = (Edit1.Text); </pre>	/*/ *			
Local (Variable2\$);	1/ 0			
<pre> Variable2\$ = (1); </pre>	// *			
Local (Variable3\$) ;	/ /			
<pre> Variable3\$ = (1); </pre>	/ 0			
🗆 😋 for (Variable2\$, Variable1\$, 1)	/ *			
<pre>Variable3\$ = (Variable3\$ * Variable2\$);</pre>	/ .			
🚱 ShowMessage (Variable3\$) ;	/ *			

Figure 191: Change Value Property

• Press Ctrl+S to save the program. Alternatively, you can also go to File>Save As to save your project to wherever you want.

File	Edit Search	Insert View	Cla		
3	New	Ctrl+N			
0	Open	Ctrl+O			
	Save	Ctrl+S			
B	Save as				
	Save Project				
ß	Save Project As				
ł	Import PPL 1.x Project				
4	Print	Ctrl+P			
	Exit				



• Now, press F5 and check your program for results!

🖉 DefaultForm			
File	5 Button1	Message K 6 OK	

Figure 193: Output

Loops and Conditions: While Loop

Quite similar to **For** object, the While object is used to perform a series of repetitive actions on a statement based on a condition but unlike the **For** statement, the **While** statement does not need as many parameters. A **While** condition will continue the iteration process while a certain condition is true. For example, whilst the **For** statement decides to process a set of instructions or not unless a condition is met and the corresponding start and end values are set, a while condition on the other hand will process the instructions enclosed within the while statement irrespective of the starting or the ending point. **While** condition is especially useful where a user is not familiar with the amount of loops certain set of instructions have to make.

Given below is a **While** condition that will allow you to process a condition without the starting or the ending position set. The example given below will display a set of lines that are equal to the number input by a user. The important thing to note here is that these lines are generated dynamically.

Start by creating a new Desktop Form project. To do so, start PIDE and press Ctrl+N and select Desktop Form Project from the "New Project Type.." window.
 Alternatively, you can also go to File> New and select Desktop Form from the "New Project Type.." window.

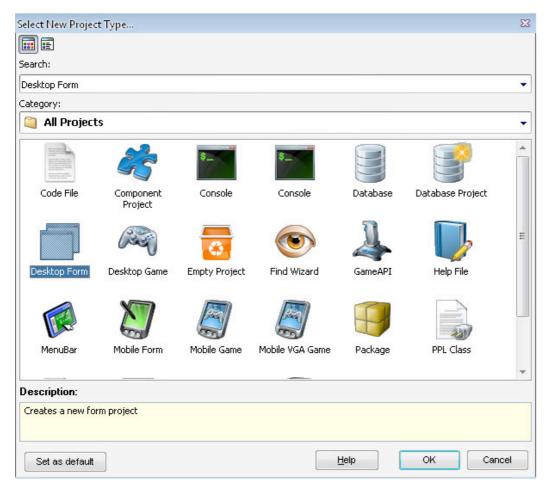
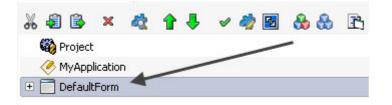
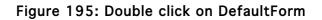


Figure 194: Create new project

• In the project, double click the **Default Form** object to enter the Form editor.





• In the **Default Form** Editor, we will place one **PEdit** box along with one **PButton** and a **PLabel** on the form editor. While the **PEdit** box will contain the number we want to use for printing the text, **PButton** will be used for the initiation for all the actions we will have on the number and **PLabel** will be used to print the content. On the **Components Pane**, look for **PEdit** object and click on it; now, click on the form to place the **PEdit** object on the form. After placing a **PButton** and **PLabel** just like the other **PEdit** object, you are ready to start visual programming. Use the figure given below to arrange the components.

Edit1		 	
Di utter d		 	
: Button1		 	
	.	 	
· · · · · · · Label1			

Figure 196: Component layout

• Click on the **Project Manager** button available on the top left of the screen and double click the **PButton** to create an **OnClick** event. This **OnClick** event will be used to initialize the various actions we will have on the data entered in the **PEdit** box.

🍪 Project Manager 🛛 🦻 DefaultForm	
% 43 🖹 × 🍢 ↑ 🖡 🖌 🛷 🖓 🖻	
👶 💩 🖹 🖻 🔕 🆓 🖹 🖹 🕲 📾 🛷 🌚 🗸	
🏟 Project 💦	v
Key MyApplication	v
DefaultForm	/ 🗸 🦓 📉
🕀 📑 MenuBar1	v
ab Edit1	14
Ala Labeli	14
🖃 🗔 Button1 🕊	14
<pre> proc OnClick(Sender\$)</pre>	/ 🗸 👋

Figure 197: Project manager

• Drag a **PVariable** from the component panel on the **OnClick** event. This will create a variable declaration for our variable.

📋 PLibrary		4
🗁 Class		/ 🗸 🎝 🔛
🗊 PComponent	⊡ III MenuBar1	
🛷 PEvent	ab Edit1	14
😳 PMethod	Ag Label1	
💑 PObject	🖃 🗔 Button1	14
💭 PProperty	proc OnClick(Sender\$)	/ / .
🥃 PVariable 🛛 🚽 🖉	<pre>Local(Variable1\$);</pre>	2 2 8
🗁 CodeFlow		

Figure 198: Drag PVariable to OnClick

• Drag the newly created variable declaration on the **OnClick** event while holding the **ALT** key the keyboard. This would create a new variable that we can use to hold values.

🍪 Project	v
Key MyApplication	v
🖃 🛅 DefaultForm	/ 🗸 🦓 🔣
🕀 📴 MenuBar1	v
ab Edit1	14
Ag Label1	14
🗆 🗔 Button1	14
🗆 🐴 proc OnClick(Sender\$)	/ 🖉 🛸
Local (Variable1\$) ;	/ 🖉 🛸
<pre> Variable1\$ = (); </pre>	/ 🖉 🗶

Figure 199: Drop Variable1\$ to OnClick

• Drag the **PEdit** object to the **Expr** property of our newly created variable and select **Text** property in the code complete window. This would assign the value entered in the **PEdit** box to this variable.

Project	v	Proper	rty Value
MyApplication	v	- Action	
DefaultForm	/ 🗸 🦓 🔯	Action:	
🛨 🖺 MenuBar1	4	Operator:	=
ab Edit1	04	E Logic	
		Logic:	
Aa Label1	14	LogicAction:	
😑 📼 Button1	20	LogicExpr:	
🖃 🖑 proc OnClick (Sender\$)	14	Parameters	
Local (Variable1\$);	/ / *	expr:	Edit1.Text
<pre> Variable1\$ = (Edit1.Text); </pre>		- Status	PEdit
		Operation: Source:	🎝 CanvasContext
		Source:	🎝 Text
			\Lambda MaxText
			🎝 SelText

Figure 200: Drag PEdit1 object to Expr property of Variable 1\$

• Drag a While object from the **Components Panel** to the **OnCreate** event.

🙆 For	🛨 🧾 MenuBar1	v
🖄 ForEach	≡ abj Edit1	14
📲 If	An Label1	14
🔄 Return	E Button1	14
👆 Using	<pre>proc OnClick(Sender\$)</pre>	/ / .
😼 VarObject	<pre>Local (Variable1\$);</pre>	/ / .
🖄 While 🛛 🖊	Variable1\$ = Edit1\$;	/ / 0
🗁 Database	🖄 while ()	/ 🖉 🗶
📑 PDatabase		



• Go to the While statement and change its Expr property to (Variable1\$>=1)



Figure 202: Change the exper property

• Drag a **PVariable** to the **OnCreate** event so as to create a declaration for a new variable and drop this newly created variable declaration onto the **OnClick** event while holding the **ALT** key on the keyboard. This will create a new variable.

🔁 Class	🖃 🥅 DefaultForm	/ 🗸 🦂 📉
📦 PComponent		v
APEvent	ab Edit1	14
😳 PMethod	Ag Label1	14
🚷 PObject	Button1	14
🤤 PProperty		/ *
🥃 PVariable 🗕	<pre>Local (Variable1\$);</pre>	/ *
CodeFlow	Variable1\$ = Edit1\$;	/ *
🔄 Break	<pre>Local(Variable2\$);</pre>	/ / 0
P Continue	<pre> Variable2\$ = (); </pre>	/ 🖉 🐞
👆 Else	while (Variable1\$ >= 1)	/ *
🗯 Else If		

Figure 203: Drag PVariable to Project Manager

• Drag this variable to while object.

× -	
/ 1	
/ 1	
14	۲
14	۲
14	۲
14	۲
14	۲
00	۲

Figure 204: Drag Variable

• While the variable is still selected, change its **Expr** property to **Variable2\$+" Number is repeated "+Variable1\$+" times. "** This statement will assign a preformatted text to the variable.

MyApplication	v	Action	
DefaultForm	/ 🗸 🦂 🔀	Action:	
		Operator:	=
-		- Logic	
ab) Edit1	/ 1	Logic:	
Aa Label1	/ 1	LogicAction:	
E Button1	0 🖌	LogicExpr:	
Proc OnClick (Sender\$)	14	Parameters	
Local (Variable1\$);	14	🗧 🗶 expr:	Variable2\$+" Number is repeated "+Variable1\$+" times. "
<pre>Variable1\$ = (Edit1.Text);</pre>	14 /	 Status 	
Local (Variable25) ;	1.2	Operation:	SetVariable
B 😋 while (Variable1\$ >= 1)	100	Source:	🕃 ¥ariable2
Variable2\$ = (Variable2\$ + " Number	er is repaave		

Figure 205: Change Expr

• Now we will place a decrement that would decrease the Variable1\$ one at a time. For doing this, drag the declaration of variable1\$ on the OnCreate event while holding the ALT key on the keyboard.

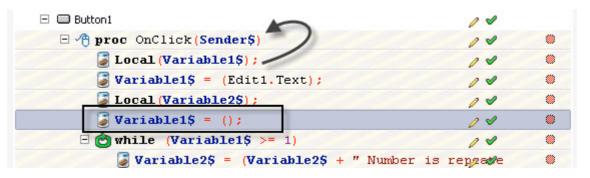


Figure 206: Drag Variable1\$ with ALT key

• Drag this variable to the While loop and change its Expr property to (Variable1\$-1). This statement will decrease our variable by 1 every time the loop plays. Alternatively, you can also just select -= in the Operator property and put 1 in the Expr property.

			LUGIC	
Aa Labeli	/ /		LogicAction:	
🖃 🗔 Button1	0 🖌		LogicExpr:	
🖃 🖑 proc OnClick (Sender\$)	14	۲	- Parameters	
Local (Variable1\$) ;	14	۲	expr:	Variable1\$-1
<pre> Wariable1\$ = (Edit1.Text); </pre>	14	۲	- Status	
Local (Variable2\$);	04	-	Operation:	Set¥ariable
E 🙆 while (Variable1\$ >= 1)			Source:	🥃 ¥ariable1
🕃 Variable2\$ = (Variable2\$ + " Number	is reparte	۲		
<pre>Variable1\$ = Variable1\$ - 1;</pre>	04	۲		

Figure 207: Change Expr property

• Now we have to print the preformatted text to the screen. For this we will use our **PLabel**. Drag the **PLabel** object to **OnClick** object and select the caption property in the **Code Completion** box that appears.

#Ph	70	
🏟 Project	×	
🥙 MyApplication	A	
E Contraction English	/ 🗸 🦏	83
🛨 🕎 MenuBar1	v	
ab Edit1	04	
Ag Label1	04	
🗉 🗔 Button1 🄪	04	
🖃 🐴 proc OnClick(Sender\$)	14	۲
PLabel	14	۲
🚱 Caption	14	۲
♦ Caption	14	۲
V caperon	/ /	۲
	04	۲
	'Number is repsade	۲
	04	۲

Figure 208: Drag PLabel

• Assign variable2\$ to the caption property of PLabel we have just created. For doing this, drag Variable2\$ to Expr property of the PLabel statement.

An Label1	/ 1		- Logic	
🖃 🗔 Button1	0 4		Logic:	
🖃 🖑 proc OnClick(Sender\$)	14	۲	LogicAction:	
Local (Variable1\$);	14	۲	LogicExpr:	
<pre>Wariable1\$ = (Edit1.Text);</pre>	14	۲	- Parameters	
Local (Variable2\$);	14	۲	expr:	¥ariable2\$
- 🙆 while (Variable1\$ >= 1)	04		Status	44
🖉 Variable2\$ = (Variable2\$ + " Number i	a remette		Operation:	SetProperty
	o prove	~~~	Owner:	A_Label1
Variable1\$ = Variable1\$ - 1;	14		Source:	Caption
As Label1.Caption = Variable2\$;	14	۲	Jourcer	Laption

Figure 209: Drag Variable2\$ to expr property

• Press Ctrl+S to save the program. Alternatively, you can also go to File>Save As to save your project to wherever you want.

File	Edit	Search	Insert	View	Cli	
	New			Ctrl+N		
	Open		1	Ctrl+0		
	Save			Ctrl+S		
B	Save as	5				
	Save Project					
B	Save Project As					
R	Import	PPL 1.x	Project.			
4	Print			Ctrl+P		
	Exit					

Figure 210: Save project

• Now, press F5 and check your program for results!

File	aultForm
	5
	Button1
	0 Number is repeated 5 times. Number is repeated 4 times. Number is repeated 3 times. Number is repeated 2 times. Number is repeated 1 times.

Figure 211: Output

Loops And Conditions: ForEach Loop

A very useful object used for performing iterations, **ForEach** object is like a shorthand for specifying the For object but for each item in another object with individual items. It is specially used where a list of items has to be iterated against an action.

Given below is an example that will allow you to see the usage of **ForEach** in PIDE for setting iterations. The given example scans each item available in a list (**PValueList**) and shows it in a message box individually.

 Create a new Desktop Form project by pressing the Ctrl+N key on the keyboard or going for the File>New option. In the Select New Project Type.. window, select Desktop Form project.

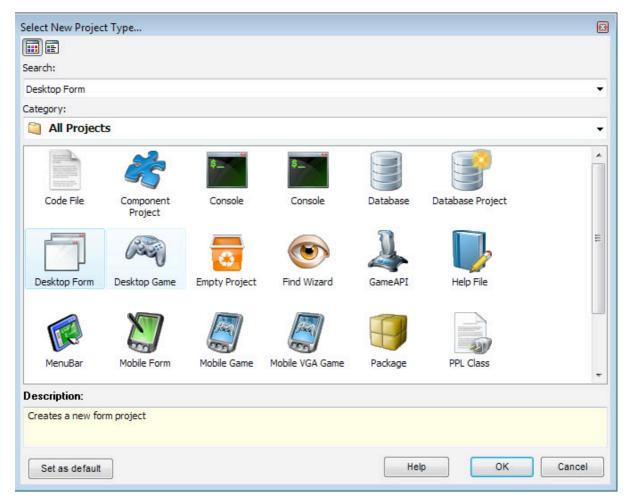


Figure 212: Create new project

• Right click on the **DefaultForm** and select **OnCreate** from the Events menu. This event would allow you to trigger actions when this form is created.

‰ 48 🔒 × 42 ⇐ ➡ 含 🕹 ✔ 4	🌶 😼	🚷 🚷 🖻 🗐 🗐 🧠 🚱		OnExit
Project				Allocation
MyApplication			8	OnCreate
DefaultForm		F 10	1	OnDestroy
<pre>proc OnCreate(Sender\$)</pre>		Edit		<u>K</u> eys
🗆 📑 MenuBar 1		Data	3	OnKeyDown
🗆 🖹 File1	00	Events	0	OnKeyPress
Exit1	00	Cut Ctrl+X	3	OnKeyUp
	5	Copy Ctrl+C		Mouse
	3	Paste Ctrl+V	Ð	OnMouseActivate

Figure 213: Create OnCreate object

• Add a **PValueList** to the project. For doing so, drag a **PValueList** on the **OnCreate** event.

🍞 PObjectList	DefaultForm	1 4 3	3
🖢 PPoint	🖃 🖶 proc OnCreate (Sender\$)	11	۲
PRect	At ValueList1 = new PValueList	04	۲
PScript	🗆 🕎 MenuBar 1	v	
📡 PValue	🗆 🖹 File1	1	
At PValueList	🖻 Exit1	v	
Project Classes			
PDefaultForm			

Figure 214: Drag PValueList on OnCreate event

• Now we will fill the items in the **PValueList**. To do this, double click the Value property of **PValueList** object and write down the list like in the figure below.

2	E 🏠 proc OnCreate (Sender\$)	/ 🖌 👋	- value	
	At ValueList1 = new PVa		Items: Sorted:	[0 pvalue object(s)]
Č.	🖃 🕎 MenuBar 1	1	Value:	-
	🖃 🗐 File1	1	vulue.	
	🖻 Exit1	4		
	Edit Q. T. 10 1 Hi 2 Hello 3 Bonjour 4 Adios 5 Caio		40	

Figure 215: Fill items

• Now that the **PValueList** has been created, we need to join it with a **ForEach** object that will run a condition for each item of the list. Drag a **ForEach** object below the **PValueList** object.

🌈 Continue	🔄 🔄 🚷 🖹 🖹 📕 🧐 🗃 🖻 🔯 📾 🛷 🍕			
👆 Else	Project		v	
🗭 Else If	MyApplication		1	
🔁 Exit	DefaultForm	0	🗸 🥠 😒	
😇 For	🗏 🛛 🚷 proc OnCreate (Sender\$)	0		۲
🖄 ForEach 🛛 🚬	At ValueList1 = new PValueList	0		۲
😂 If	j foreach ()	0		۲
🔁 Return	E MenuBar 1		v	
🔄 Using	E E File1		1	
S VarObject	Exit1		<i>v</i>	
🙆 While			50	
Database				

Figure 216: Drag ForEach object

• Drag the **PValueList** object we created earlier to the **ForEach** object's **Expr** property and select Items from the **Code Complete** window.

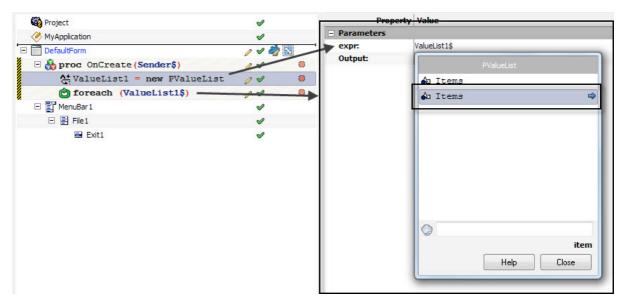


Figure 217: Select Items

• The Output property of the **ForEach** is the output variable where you want to store the current item. Since PPL uses a very relax type-checking, you can assign any object type to a variable as type-checking is done at runtime, not at compile time. For adding a variable to the output property, drag a **VarObject** component from **Component Pane** under **PValueList** object so that it is created before **ForEach**. The **VarObject** is used to tell the compiler which variable class it will hold.

P Continue	🔄 🔄 🚷 🖹 🗐 🛄 🥎 🖹 🔮 🔯 🎰 🛷 🍕	•	
🕒 Else	Project	v	
🗭 Else If	MyApplication	v	
🔄 Exit	DefaultForm	1 1 2	
🖄 For	E Aproc OnCreate (Sender\$)	14	
🖄 ForEach	At ValueList1 = new PValueList	14	۲
🖓 If	→ → i\$ > PObject	04	۲
🔄 Return	foreach (ValueList1\$)	11	۲
👆 Using 🥢	🗆 🕎 MenuBar 1	1	
😼 VarObject 🦯	E File1	v	
🖄 While	Exit1	v	
🔁 Database			
PDatabase			

Figure 218: VarObject

• We now need to tell the visual code generator what class type object the output variable is going to contain. Select **PValue** from the **ClassName** property of the newly created **VarObject**.

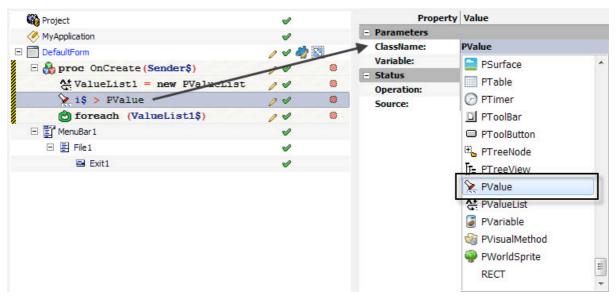
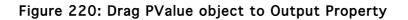


Figure 219: Select PValue

• Now drag the **PValue** statement to the Output property of the **ForEach** object.

			and the second
Project	\$		Property Value
MyApplication	1	- Parame	ters
DefaultForm	1 1 2	expr:	ValueList1\$
Greate (Sender\$)	14	Output:	i\$
ValueList1 = new PValueList	00		
📡 i\$ > PValue 🥌	11		
🙆 foreach (ValueList1\$, i\$)	04	•	
🖃 📴 MenuBar 1	V		
🖂 🖺 File1	ø		
Exit1	V		



• For the output, drag the **PValue** statement on the **ForEach** object while holding **ALT** key on the keyboard. In the **Code complete** window that appears, select **Show** to print the values on the screen.

	ValueList1 = new PValueList i\$ > PValue	~	(
	foreach (ValueList1\$, i\$) 🗲	20	(
E Men	PValue		
	\Lambda AsInt	^ [
	🎸 AsInt		
	\Lambda AsFloat		
	♦ AsFloat		
	👍 AsBool		
	♦ AsBool		
	😼 TeList (Delimitor\$)		
	🎨 Show	=	
	😼 Paste		
	0		
	Help Close	s	

Figure 221: Select Show

• Press Ctrl+S to save the program. Alternatively, you can also go to File>Save As to save your project to wherever you want.

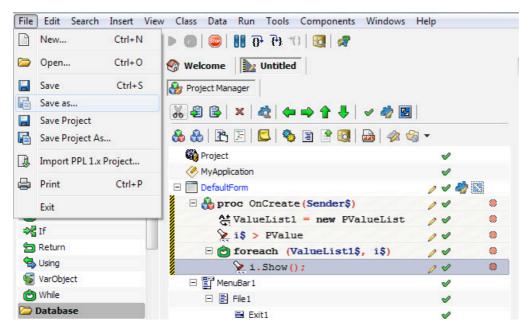


Figure 222: Save project

• Now, press **F5** and check your program for output. You will see different message boxes, each representing each list item available on the **PValueList**.



Figure 223: Output

Loops and Conditions: If Condition

Usage of conditional statements is crucial for any program that requires a higher degree of processing and logic. PIDE provides the **IF** object as the very basic of making conditional decisions. With the use of **IF** condition, users get the facility to evaluate a specific condition and then specify its result after analysis of the outcome of their condition. In the example given below we will create a program to find the subtraction of two numbers. This program will allow a user to input two numbers and alert him/her if the first variable is lower than the second variable.

In this program, we will make use of two **PEdit** objects to enter two numbers for subtraction, One PLabel for the subtraction sign and one **PButton** to act as a trigger for the actions on the program.

 Start by creating a new Desktop Form project. To do so, start PIDE and press Ctrl+N and select Desktop Form Project from the "New Project Type.." window. Alternatively, you can also go to File> New and select Desktop Form from the "New Project Type.." window.

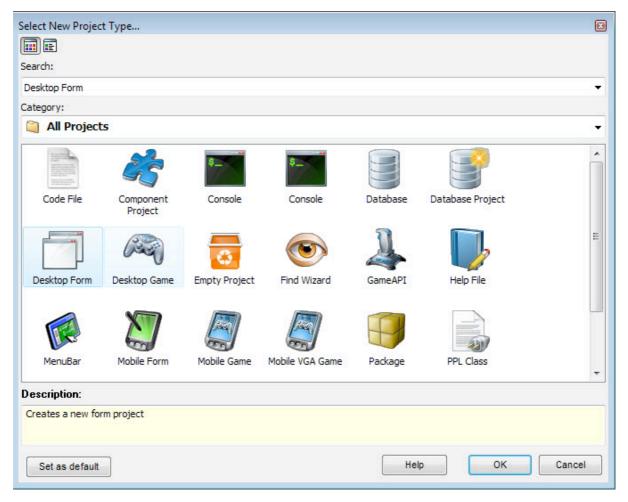


Figure 224: Create new project

• In the project, double click the **Default Form** object to enter the Form editor.

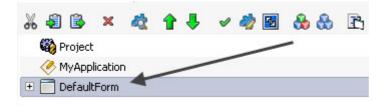


Figure 225: Double click

In the Default Form Editor, we will place two PEdit objects along with one PLabel and one PButton. While the PEdit boxes will contain the numbers to subtract, the PLabel will be used to hold the subtract sign, PButton will be used for the initiation for all the actions we will have on the numbers. On the Components Pane, look for PEdit object and click on it; then, click on the form to place the PEdit object on the form. Similarly, put another PEdit on the form like shown in the figure. Now, find a PLabel on the Components Pane and place it in between the two PEdit boxes. After placing a PButton just like other components, you are ready to configure your components.

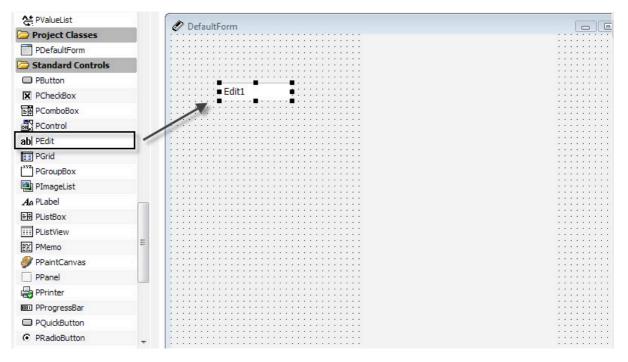


Figure 226: Place PEdit

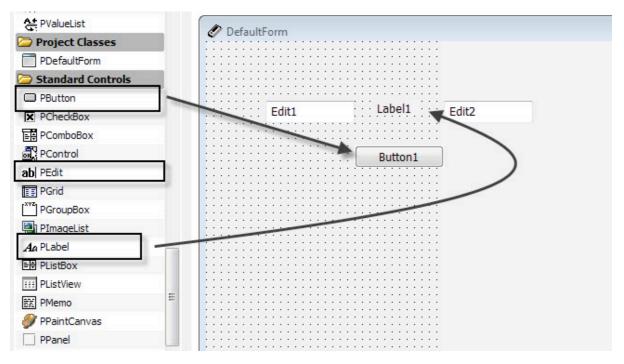


Figure 227: Place Components on Desktop form

• Click on the **PLabel** and change its caption property to represent subtraction sign by placing a – symbol there.

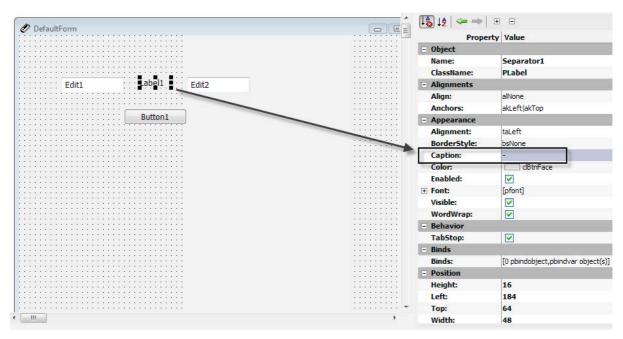


Figure 228: Place PLabel on Desktop Form

- You can also change the caption property in the **PButton** to say "Subtract"
- Go back to the **Project Manager** by clicking the **Project Manager** button at the upper left corner.

🍪 Project Manager 🛛 🎯 DefaultForm	🥳 Properties 🛷	P Events
X 24 F F E E E E F A I	Subtract1 : PButt	con
	🔩 🕼 🖛 🖚	+ -
	Prope	erty Value
	- Object	- 100
	Name:	Subtract1
	ClassName:	PButton
Edit1 Edit2	- Alignments	
	Align:	alNone
······	Anchors:	akLeft akTop
Button1	- Appearance	
	BorderStyle:	bsNone
	Caption:	Subtract
	Color:	-

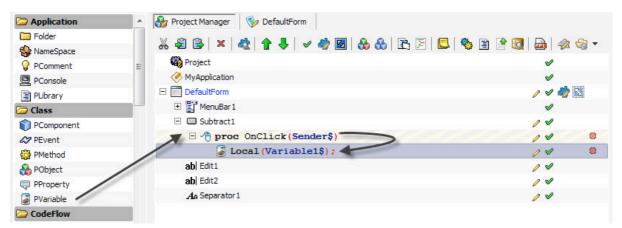
Figure 229: Project Manager

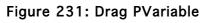
• In the **Project Manager**, double click the **PButton** to create an **OnClick** event. This event will be used to trigger actions once this button is clicked.

🚜 49 🗟 🗙 42 1 🖊 🗸 🤣 📓 88 88 13 🗐 😂 🥸	🖹 🔮 🔯 🞰 🛷 🍕 🗸
🙀 Project	v
MyApplication	V
DefaultForm	/ 🗸 🦓 🔝
🕀 🕎 MenuBar 1	v
🗆 🗔 Subtract1 🥌	/ 1
<pre> proc OnClick(Sender\$) </pre>	/ 🖉 🔍
ab Edit1	/ 1
ab Edit2	14
Ag Separator 1	14

Figure 230: OnClick event

• Drag a **PVariable** onto the **OnClick** event. This will create variable definition that can be used to program actions in the program.





• Once you have declared a variable, you need to assign it to something. This is accomplished in PIDE by dragging the variable declaration and dropping it once more on the **OnClick** event while holding the **ALT** key on the keyboard.



Figure 232: Alt+drag on OnClick

• Click the newly made variable and drag the **PEdit** box on its **Expr** property. In the **Code Completion** box that appears, click **Text** option to select **Edit1.Text**

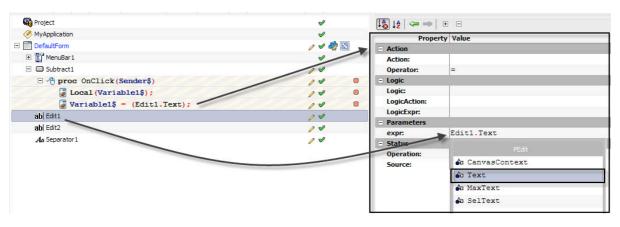


Figure 233: Drag PEdit to expr

• The variable created above will now contain the text entered in the **PEdit1** box. Just like the couple of steps given above, we will create a variable for the other **PEdit** box.

🧰 Folder			Variable2.Variable2_1
Se NameSpace	₩ 🕹 🕒 × 🎕 🕇 🦊 🗸 🏘 🗟 🚷 🖻 🗄 🧐 🗎	3 🔍 🕮 🎋 🍕 🔭	
PComment	E Project	v	[12]↓2 <= → ⊕ ⊟
PConsole	Key MyApplication	v	Property Value
PLibrary	DefaultForm	/ 🗸 🦏 🔯	- Action
Class	🛨 📑 MenuBar 1	v	Action:
PComponent	🖂 🗖 Subtract1	/ ×	Operator: =
AP PEvent	proc OnClick (Sender\$)	0 20 0	E Logic
🙀 PMethod	/ S Local (Variable1\$);	/10 .	Logic:
A PObject	<pre>Variable1\$ = (Edit1.Text);</pre>	/ /	LogicAction:
PProperty	Local (Variable2\$);	/ /	LogicExpr:
PVariable	<pre>Variable2\$ = (Edit2.Text);</pre>	04 .	- Parameters expr: Edit2.Text
CodeFlow	ab Edit1	/ /	expr: Edit2.Text
🔄 Break	ab Edit2	14	Operation: SetVariable
Continue	Aa Separator 1	0 4	Source: 🛛 🕃 Variable2



- After creating variables to store the value of both our **PEdit** objects we will check that the value of our first number is smaller than the value of the second number. To accomplish this, we will use an **IF** condition. This condition will check if the value of the first variable is larger than the second variable, it will display a message alerting a user to input a smaller value. For creating the **IF** condition, start by dragging and dropping the **PEdit1** box object to the **OnClick** object.
- The above action will initiate **Code Completion** window. Select the **Text** option in this window.

🧞 Project Manag	ger 🛛 🦻 DefaultForm	
* 🗿 🚯 🗴	< 🗞 🛧 🦊 🖌 🥠 🔤 🗞 🚷	🖹 🖹 📕 🍓 🖹 🔮 🔞 📾 🛷 🎯 🗸
🚳 Project		4
🧭 MyApplica	tion	v
🗉 🛅 DefaultFo	rm	/ 🗸 🦏 🔯
🕀 📑 Menul	Bar 1	✓
🗆 🗖 Subtra	act1	14
P B P	roc OnClick(Sender\$)	/ * *
6	PEdit	/ *
	CanvasContext	/ / .
1	✤ CanvasContext	
ab Edit1		
	lext	14
Aa Sepa	🔄 MaxText	14
	♦ MaxText	
	A SelText	
	♦ SelText	

Figure 235: Select Text option

• Select the newly created statement and press **Ctrl+I** while it is selected. This will change this statement to an **IF** statement.

🖃 🔲 Subtract1	04
E 🕂 proc OnClick (Sender\$)	/ 🗸 👋
Local (Variable1\$);	/ 🗸 👋
<pre>Variable1\$ = (Edit1.Text);</pre>	/ 🗸 🔘
Local (Variable2\$);	/ 🗸 👋
<pre>Variable2\$ = (Edit2.Text);</pre>	/ 🗸 🔘
a If (Edit1.Text ==)	/ 🗸 🛸
ab Edit1	14

Figure 236: Press Ctrl+i for IF statement

• Now, while the **IF** statement is still selected, change its **LogicExpr** property to **<Variable2\$.** Doing this will check if the value entered in the **PEdit1** box is smaller than the value that is stored in **variable2** that holds the value stored in the **PEdit2** object.

□ / proc OnClick(Sender\$)	14	۲	Action:	
Local (Variable1\$);	14	۲	Operator:	=
<pre>Variable1\$ = (Edit1.Text);</pre>	04	۲	E Logic	
<pre>Local (Variable2\$);</pre>	04	۲	Logic:	If
<pre>Variable2\$ = (Edit2.Text);</pre>	14	۲	LogicAction:	41-11-11-11
If (Edit1.Text < Variable2\$)	04	۲	LogicExpr: – Parameters	<variable2\$< td=""></variable2\$<>
ab Edit1	14		expr:	
ab Edit2	0 4		- Status	
An Separator 1	14		Operation:	SetProperty

Figure 237: Change LogicExpr property

• Drag a PVariabe to the **OnCreate** event to create a variable declaration and then drag the variable to the **OnCreate** event while holding the **ALT** key to create a new variable. Once this variable is created, drag and drop it on the **IF** statement so that it comes under it. A sure sign to make sure that you have put this variable under the **IF** statement is to check for the small - expansion mark on the left of the **IF** statement.

🗄 🗔 Subtract1	0 4
E 🖓 proc OnClick(Sender\$)	/ 🗸 🕷
Local (Variable1\$);	/ 🗸 🕷
<pre>Variable1\$ = (Edit1.Text);</pre>	/ /
Local (Variable2\$);	/ /
<pre>Variable2\$ = (Edit2.Text);</pre>	/ 🗸 📲
Local (Variable3\$);	/ 🗸 🕷
<pre> If (Edit1.Text < Variable2\$) </pre>	/ 🗸 🕷
<pre> Variable3\$ = (); </pre>	/ 🗸 🕷
ab Edit1	14
ab Edit2	00
Ag Separator 1	04

Figure 238: Create new variable

• Once you have this new variable in place. You need to change its **Expr** property to print "*Sorry! Number1 cannot be smaller than Number 2*". This statement will be executed if the first number is found to be smaller than the second number.

	vw	TIME:	- ACLION		
E MenuBar 1	v		Action:		
🛛 🔲 Subtract1	/ /		Operator:	=	
- / proc OnClick(Sender\$)	14	۲	- Logic		
Local (Variable1\$);	14	۲	Logic:		
<pre>Variable1\$ = (Edit1.Text);</pre>	14	۲	LogicAction:		
Local (Variable2\$);	14	۲	LogicExpr:		
<pre>Variable2\$ = (Edit2.Text);</pre>	04	۲	- Parameters		
Local (Variable3\$);	14	-	- Status	"Sorry! Number	1 cannot be smaller
If (Edit1.Text < Variable2\$)	12		Operation:	SetVariable	
Variable3\$ = ("Sorry! Number1 cannot be	smaller that	۲	Source:	Variable3	
ab Edit1	14				
ab Edit2	14				
An Separator 1	0 1				

Figure 239: Change expr property

• For ensuring that the button press generates the subtraction of the two numbers, we will drag the **variable3\$** declaration again to the **OnCreate** event. This will create another variable with the same name. Go to the **Expr** property of this variable and input **Variable1\$-Variable2\$**. Our newly created variable will hold the value

subtraction results of the two numbers. Drag this variable above the **IF** statement so that it does not interferes with its value.



Figure 240: Change Expr property

S rocar (Aariantezs);	/ *	
<pre>Variable2\$ = (Edit2.Text);</pre>	14	۲
Local (Variable3\$);	11	۲
🛹 🅃 Variable3\$ = (Variable1\$ - Variable2\$);	00	۲
If (Edit1.Text < Variable2\$)	14	۲
Variable3\$ = ("Sorry! Number1 cannot be smaller	r thad	۲
ab Edit1	14	
ab Edit2	14	
Ag Separator 1	11	

Figure 241: Drag and drop above IF

• Click on the **OnClick** event and press **Ctrl+Space** to initiate **Code Completion** window. Select **ShowMessage**() from it.

🖃 🔲 Subtract1	/ 1
<pre></pre>	/ 🖌 🕚
PDefaultForm	/ 🖉 🕚
	/ 🖉 👋
😳 ShowMessage (value\$ [, value\$	/ / .
	/ *

Figure 242: ShowMessage

• In the value property of **ShowMessage**, write **Variable3**\$. This will print the value that is contained within Variable3. If the condition of the first number being smaller than the second number is found to be true, Variable3 will have the alert but if it is not found to be true, variable3 will contain the result of subtraction of the first and the second number.

A the American Section				
MyApplication	~		Property	Value
DefaultForm	/ 🗸 🦓		- Parameters	
🗉 🕎 MenuBar 1	V		value:	Variable3\$
🖻 🗖 Subtract1	14	/	- Status	
□ 🖑 proc OnClick(Sender\$)	11	1	Operation:	ProcCall
Local (Variable1\$);	14/	•	Source:	ShowMessage
<pre>Variable1\$ = (Edit1.Text);</pre>	10	۲		
Local (Variable2\$);	14	۲		
<pre>Variable2\$ = (Edit2.Text);</pre>	11	۲		
Local (Variable3\$);	11	۲		
Variable3\$ = (Variable1\$ - Variable2\$);	11	۲		
<pre>If (Edit1.Text < Variable2\$)</pre>	00	۲		
Variable3\$ = ("Sorry! Number1 cannot be smaller	thad	۲		
ShowMessage(Variable3\$);	14	۲		

Figure 243: Change Value property

• After confirming that your program matches the below given screenshot, save your program and run it.

% 48 🚯 × 42 1 🗣 🗸 42 🐼 🖪 & & E E 🧐 🗃 🔗 (3 🔂 🛷	-
🙀 Project	V	
MyApplication	V	
E DefaultForm	V 🏹	83
🛨 🛅 MenuBar 1	v	
🖃 🔲 Subtract1	ø	
E / proc OnClick (Sender\$)	v	۲
Local (Variable1\$);	V	۲
Local (Variable2\$);	V	۲
Local (Variable3\$);	V	۲
<pre>Variable1\$ = (Edit1.Text);</pre>	V	۲
<pre>Variable2\$ = (Edit2.Text);</pre>	V	۲
Variable3\$ = (Variable1\$ - Variable2\$);	V	۲
🗆 📸 If (Edit1.Text < Variable2\$)	V	۲
Variable3\$ = ("Sorry! Number1 cannot be smalle	r thad	۲
ShowMessage (Variable3\$);	V	۲
ab Edit1	v	
ab Edit2	V	
Ag Separator 1	V	

Figure 244: Project Screenshot

• Press **Ctrl+S** to save the program. Alternatively, you can also go to **File>Save** As to save your project to wherever you want.

File	Edit Search	Insert View Cla
3	New	Ctrl+N
	Open	Ctrl+O
	Save	Ctrl+S
B	Save as	
	Save Project	
ß	Save Project	As
ł	Import PPL 1	x Project
4	Print	Ctrl+P
	Exit	

Figure 245: Save your project

• Now, press F5 and check your program for results!

Defaulti	Form	
	1	- 4
		Subtract
		Message
		Sorry! Number1 cannot be smaller than Number 2
		ОК

Figure 246: Output

Loops and Conditions: Else Condition

The **Else** object is used to specify an alternate result or outcome of the **IF** condition. By using a combination of **IF** and **ELSE** object, users can completely evaluate their results in event any event.

Possibly one of the easiest to use components of PIDE, the **ELSE** statement is something that is only useable with the **IF** condition. To explain fully, a statement in the **IF** case is executed only if a condition is evaluated to be true but the else case is executed when the condition in an **IF** case if evaluated to be false. Rather than having two **IF** cases that would govern the execution or non execution of a statement, it is far better a programming practice to use in **ELSE** condition.

The example given below inputs two integers and gives an alert if the first integer is greater than the second integer or otherwise. In this example, an **IF** case will evaluate the condition and an else case will return the output if that condition is not fulfilled.

• Start with creating a **Desktop Form** Project. To do so, press **Ctrl+N** or go to **File>New** and select **Desktop Form** project in the **Select New Project Type..** window.

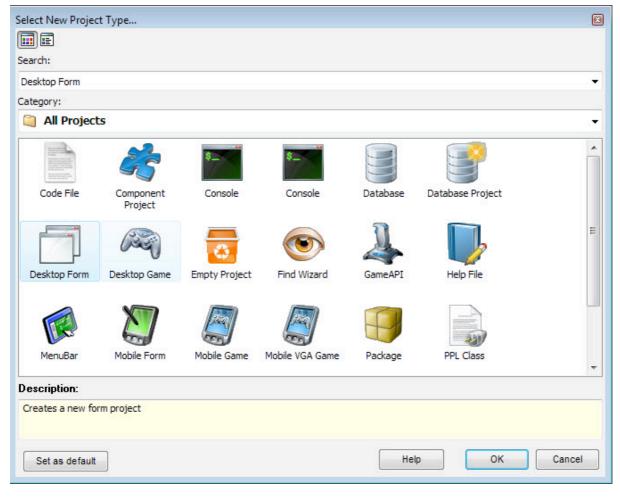


Figure 247: Create new project

• In the **Default Form Project Manager**, double click the **Default Form** to initialize the Form editor and place two **PEdit** boxes with three **PLabels** and one **PButton**; like shown in the figure below. To create the aforesaid components on the **Form editor**, click on the component you are looking for in the **Components Pane**. After selecting the component, click on a place on the form editor to place that component there. Refer to the screenshot below to configure your components.

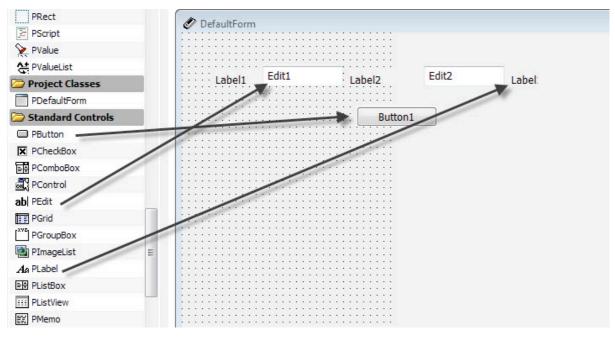


Figure 248: Place coponents

• After you have all the components placed on the form, you will be required to change their **Caption** property to something that makes sense. To change the **Caption** property of a component, click on the component and go to its properties in the properties panel on the right. In properties, find the **Caption** property and change it to anything you want. We have changed our **PLabel** to following:

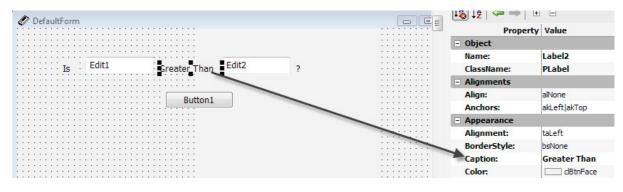


Figure 249: Change Caption property

• Once you have made the required changes to the form, revert back to the **Project Manager** by clicking the **Project Manager** button on the top left of the screen.

Noject Manager 📎 DefaultForm	
X 48 🕼 🕞 🖃 🔲 📰 💷 🗐 🖂 🚳 🕄	
DefaultForm	
$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{i=1}^{n} \sum_{i$	
Is Edit1 Greater Than Edit2	?
Button1	

Figure 250: Project Manager

• In the **Project Manager**, select the **PButton** and double click it to create an **OnClick** object. This **OnClick** event will be used to process actions on the click of the button.

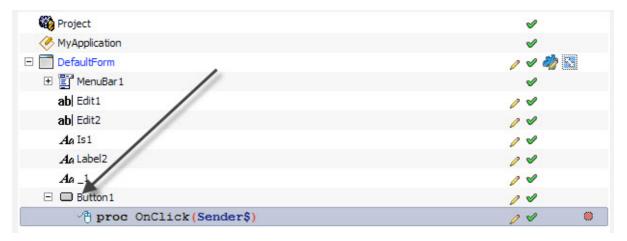


Figure 251: Create OnClick event

• Now, we will be required to create variables that would be used to store and process values for the program. Like every other programming language, PIDE also requires you to declare variables before storing values in them. Drag a **PVariable** to the **OnClick** event to create a variable definition.

🔁 Folder		¾ 월 🚱 × 💐 含 🖡 🗸 🛷 🕅 & & E 🖂 🧐	
SameSpace			🚊 📑 🔍 🕮 🌾 🍕 🔭
PComment	E	🍓 Project	v
PConsole		MyApplication	V
PLibrary		DefaultForm	/ 🗸 🦓 🚷
Class		🗉 🕎 MenuBar 1	v
PComponent		ab Edit1	0 1
A PEvent		ab Edit2	00
😳 PMethod		Aa Is1	14
A PObject		Aa Label2	00
PProperty		A a_1	0 4
PVariable		🖃 🗖 Button 1	00
CodeFlow	And Personal Property lies in the lease of the	<pre></pre>	/ / .
🔄 Break		Local (Variable1\$);	/ 🖉 🔍
Continue			

Figure 252: Drag PVariable

• Now, to create a variable, drag the newly created variable definition back to the **OnClick** event while holding the **ALT** key. This would create an empty variable ready to hold information.

🔏 🗐 📴 🗙 🎝 🔒 🖊 🖌 🧳 🐻 🗞 🚷 🗈 🖻	🕒 🧇 🖹 🔮 🔞 🖬 🖉
🍪 Project	v
Key MyApplication	v
DefaultForm	/ 🗸 🦏 🔊
🕀 📳 MenuBar 1	v
ab Edit1	14
ab Edit2	14
Aa Is1	14
Ag Label2	/ 1
A a _1	14
🖃 🔲 Button 1	/ 1
proc OnClick (Sender\$)	/ 🗸 🔘
<pre>Local (Variable1\$);</pre>	/ 🗸 🔘
<pre>>> 3 Variable1\$ = ();</pre>	/ 🗸 🔘

Figure 253: Create a new variable

• While the new variable is still selected, drag the **PEdit1** object to its **Expr** property. This will initiate the code complete window.

		EINTELLA NUME
DefaultForm	/ 🗸 🤣 🔛 📃 Actio	n
🗄 🖺 MenuBar1	✓ Action	on:
ab Edit1	Ø ♥ Oper	rator: =
ab Edit2	∕ ✓ 📃 Logic	c
Ag Is1	∕ ✓ Logic	
An Label2	Logic	CAction:
Aa_1		cExpr:
Button1		meters
- / proc OnClick (Sender\$)	expr	
<pre>Local (Variable1\$);</pre>	- Statt	PEdit
<pre> Variable1\$ = (); </pre>	/ V Sour	A
		🎝 Text
		A MaxText
		A SelText

Figure 254: Drag PEdit to Expr property

• Select 'Text' value from this window so that it now contains Edit1.text.

Aa Is1	/ V Logic:
An Label2	ℓ ♥ LogicAction:
Aa_1	LogicExpr:
□ Button1	- Parameters
	expr: Edit1.Text
	- Status
Local (Variable1\$);	● Operation: SetVariable
<pre>Variable1\$ = (Edit1.Text);</pre>	🖉 🖉 Source: 🖉 Variable1

Figure 255: Change Expr Property

• Just like the last variable, create another variable by first declaring its definition and then assigning value to it.

Application	*	Project Manager Sy DefaultForm Sy	
Folder NameSpace		🐰 🗐 🕒 × 🎕 🕈 🦊 🗸 🤣 🗐 & & 🖻 🗐 🥸 🖹	🔮 🔯 📾 🛷 🍕 🕶
PComment	E	🚳 Project	v
PConsole		MyApplication	v
E PLibrary			/ 🗸 🦏 🔀
Class		🗄 🕎 MenuBar 1	v
PComponent		ab Edit1	/ 1
AP PEvent		ab Edit2	/ 1
🙀 PMethod		Aa Is1	/ 1
🚯 PObject		Ag Label2	14
PProperty		Aa_1	/ 1
PVariable		E Button1	0 1
CodeFlow		<pre></pre>	/ /
🔄 Break		Local (Variable1\$);	/ / @
P Continue		Variable13 - (Editi Text);	/ *
les Else		Local (Variable2\$);	/ /
🕽 Else If		<pre> Variable2\$ = (); </pre>	/ 🖉 🔍

Figure 256: Create variable

• In this variable, drag **PEdit2** to the **Expr** property and select the **Text** property so that this variable contains the value of second input box.

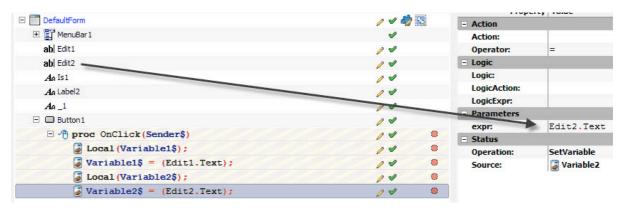


Figure 257: Change Expr property

• Once both the variables for the values are created, we now have to work upon the **IF** condition that would check if a number is greater than the other or not. For doing this, drag the **PEdit1** object to the **OnClick** event; this initiates a code complete window. In the code complete window, select the **Text** property.

🚳 Projec	ct	✓
K MyAp	plication	A
🖃 📄 Defau	lltForm	/ 🗸 🦓 🔯
🕀 📑 M	lenuBar 1	A
ab) Er	dit1	04
ab E	dit2	04
Aa Is	sh	14
Aa La	abel 2	14
Aa_	1	1 1
E 🔲 Bi	uttor	14
EM	<pre>proc OnClick(Sender\$)</pre>	/ 🖉 🛸
	PEdit	/ 🖉 🕷
	CanvasContext	/ 🖉 🛸
		/ 🖉 🔍
	Y CanvasContext	/ 🖉 🕚
	🎝 Text	
	🎸 Text	
	🚱 MaxText	

Figure 258: Select Text property

• Click on the newly created statement, and press Ctrl + I to change it to an IF statement.

Aa_1	04
Button1	14
<pre></pre>	/ 🖉 👒
Local (Variable1\$);	/ 🖉 🕷
<pre>Variable1\$ = (Edit1.Text);</pre>	/ 🖉 👒
Local (Variable2\$);	/ 🖉 👒
<pre>Variable2\$ = (Edit2.Text);</pre>	/ 🖉 🕷
If (Edit1.Text ==)	/ 🖉 🕚

Figure 259: Press Ctrl+I to create IF statement

• Click the newly created IF statement and go to its **LogicExpr** properly. Here, write **<Variable2\$.** This would trigger the **IF** statement if the text in **PEdit1** is smaller than text in the **Variable2\$**.

Ag 1	14		Logic:	If
			LogicAction:	
Button1	/ 1		LogicExpr:	<variable2\$< td=""></variable2\$<>
<pre></pre>	/ 1	۲	- Parameters	
Local (Variable1\$);	14	۲	expr:	
<pre>Variable1\$ = (Edit1.Text);</pre>	14	۲	- Status	
Local (Variable2\$);	14	۲	Operation:	SetProperty
<pre>Variable2\$ = (Edit2.Text);</pre>	14	۲	Owner:	ab Edit1
If (Edit1.Text(< Variable2\$))	04	۲	Source:	Text

Figure 260: Change LogicExpr property

• Click on the **IF** statement and press **Ctrl+Space** bar. In the **Code Completion** box that appears, select **ShowMessage** option.



Figure 261: ShowMessage

• In the value property of **ShowMessage**, enter "*Number 1 is smaller than Number 2*"

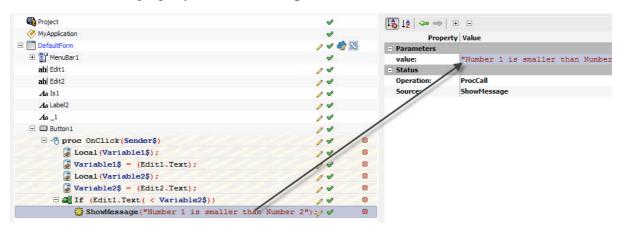


Figure 262: Change Value property

• Now, drag an ELSE object to the **OnClick** object and press **Ctrl+Space** Bar while it is selected. Select **ShowMessage** form the **Code Completion** window.

		-
3		
🥃 1	PDefaultForm	
🥃 i 🗧	🐉 ShowMessage (value\$ [, value\$	
: 🎥 🗆		
		r than Number 2");
L 9 (

Figure 263: Use ShowMessage

• Select the newly created **ShowMessage** object and change its **Value** property to *"Number 1 is greater than Number 2"*

b Project Manager 🦻 DefaultForm			Properties A	AND PRODUCT IN
🚜 🗐 🚱 🛛 🗶 😭 🗣 🖊 🖌 🖋 😼 🚷 🔀 🗄 🔚 🔜 🧐 🖹 😤 🔞	1 🔤 🛷	• 🌚 🔹	ShowMessage2	
🙀 Project	1		[🕹 🞼 🖛 👄	+ -
Key MyApplication	V		Prop	erty Value
E DefaultForm	/ 🗸 🆏		- Parameters	
🛨 🕎 MenuBar 1	V		value:	"Number1 is greater than Number2"
ab Edit1	14		Status	
ab Edit2	14		Operation:	ProcCall
Aa Is1	14		Source:	ShowMessage
Aa Label2	00			
Aa_1	14			
🖃 🔲 Button 1	14			
□ / proc OnClick (Sender\$)	11	۲		
Local (Variable1\$);	14	۲		
<pre>Variable1\$ = (Edit1.Text);</pre>	14	۲		
Local (Variable2\$);	10	۲		
<pre>Variable2\$ = (Edit2.Text);</pre>	10	۲		
<pre> If (Edit1.Text(< Variable2\$)) </pre>	11	۲		
ShowMessage ("Number 1 is smaller than Number 2")	:00	۲		
🗆 👆 else	10	۲		
ShowMessage("Number1 is greater than Number2");	00	۲		

Figure 264: Change value property

• The application is complete! Save the application by pressing **Ctrl+S** or going to **File>Save As..** and save it in a folder of choice.

File	Edit	Search	Insert	View	Cla
	New			Ctrl+N	
	Open			Ctrl+O	
	Save			Ctrl+S	
	Save as				
	Save Pr	oject			
B	Save Pr	oject As			
ß	Import	PPL 1.x	Project.		
-	Print			Ctrl+P	
	Exit				

Figure 265: Save Project

• Press **F5** to run the project and see the output:

DefaultForm File					
Is	3	Greater Than Button1	2	?	
		Buttoni			
			Message		×
			Number 1 is	greater than Nun	nber 2
					ОК

Figure 266: Run Project

Loops and Condition: Else-If Condition

In the above sections, we have seen that a certain condition can be evaluated with the help of **IF** condition and if the condition is not true, we can use the **ELSE** condition to take care of the other situation also. While a combination of **IF** and **ELSE** works mose of the times, we would still require a conditional statement that can be used to specify more than one outcomes of a condition.

There are many real world situations where one needs to have more than one outcome or choice for one statement. The given example allows a user to input an abbreviation of 'days of the week' and returns its full form. Here we will simply check for the various outcomes of a single user entry and output the result according to that.

 Start with creating a Desktop Form Project. To do so, press Ctrl+N or go to File>New and select Desktop Form project in the Select New Project Type.. window.

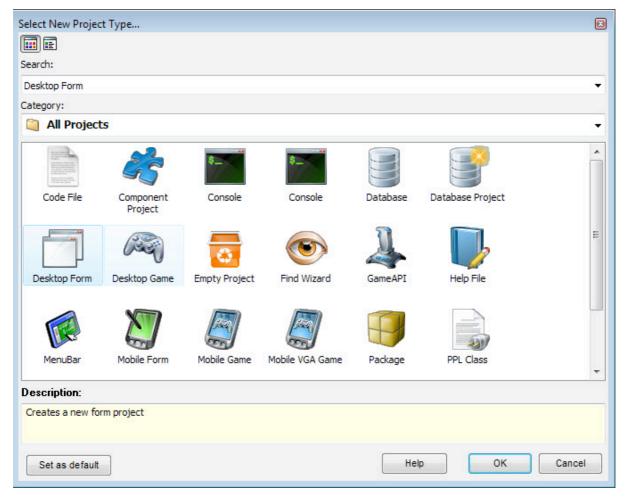


Figure 267: Create new project

• In the **Default Form Project Manager**, double click the **Default Form** to initialize the Form editor and place one **PEdit** object with one **PLabel** and one **PButton**; like shown in the figure below. To create the aforesaid components on the **Form editor**, click on the component you are looking for in the **Components Pane**. After selecting

the component, click on a place on the form editor to place that component there. Refer to the screenshot below to configure your components.

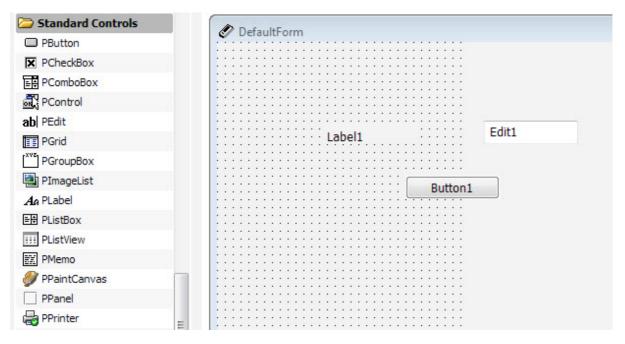


Figure 268: Place Components

• For viewing purposes, we will now change the **Caption** property of **PLabel1** to **Enter an abbreviation of any day in a week**

🖉 DefaultForm		3 +z		
······································		Proper	rty Value	
			ruide	
	E	Object		
		Name:	_Enter_an_abbreviation_of_any_day_in_a	
		ClassName:	PLabel	
		- Alignments		
Enter an abbreviation of any day in a week Edit1			17	
cifier all abbreviation of any day in a week		Align:	alNone	
·····				
		Anchors:	akLeft akTop	
· · · · · · · · · · · · · · · · · · ·		MARKED STREET, STREET, STR		
Button1		- Appearance		
		Alignment:	taLeft	
			the second se	
		BorderStyle:	bsNone	
	and a second	Caption:	Enter an abbreviation of any day in a week	
		1210 CO. 10		
		Color:	dBtnFace	

Figure 269: Change Caption

• Return to the **Project Manager** and double click the **PButton** to create an **OnClick** event that will initiate the actions that are to be performed.

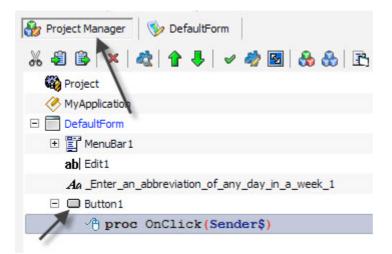


Figure 270: Create OnClick event

• Now, we will be required to create variables that would be used to store and process values for the program. Like every other programming language, PIDE also requires you to declare variables before storing values in them. Drag a **PVariable** to the **OnClick** event to create a variable definition.

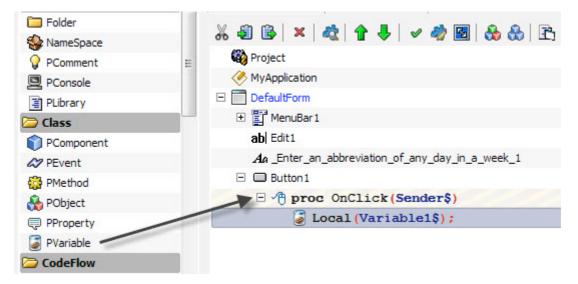


Figure 271: Drag PVariable to OnClick

• To create the variable, drag and drop the variable definition to the **OnClick** event while holding the **ALT** key on the keyboard. This will create a variable ready to hold values.

🍓 Project
MyApplication
DefaultForm
🛨 🕎 MenuBar 1
ab Edit1
An _Enter_an_abbreviation_of_any_day_in_a_week_1
E Button1
proc OnClick (Sender\$)
<pre>Local (Variable1\$);</pre>
<pre> Variable1\$ = (); </pre>

Figure 272: Drag variable

• Drag the **PEdit1** object to the **Expr** property of this newly created variable to associate it with the **PEdit** box. This action will initiate a **Code Completion** window; select Text option in the window as it will ensure that out variable holds the value that is being input in the **PEdit** Input box.

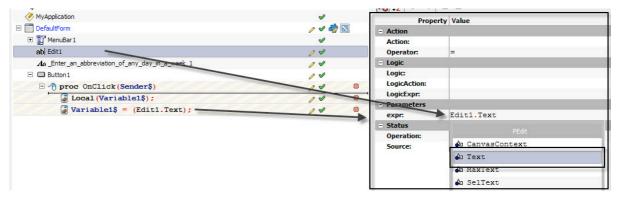


Figure 273: Drag PEdit to expr property

• Now to check the entered value in the **PEdit** box we will have to have an **IF** condition. To do this, drag the **PEdit1** object on the **OnCreate** event and select **Text** property in the **Code Completion** window that appears.

🔏 🗐 🗟 🔺 💐 😭 🖊 🖌 🖋 🦓 🛐 🚷 🚷 🖹 🗐 🖺
🚳 Project
MyApplication
DefaultForm
🛨 🛅 MenuBar 1
ab Edit1
Ag_Enter_an_abbreviation_of_any_day_in_a_week_1
🖃 🖵 Button L
<pre></pre>
PEdit
🖨 CanvasContext
Y CanvasContext
la Text
Text
🖨 Maxiext

Figure 274: Drag PEdit and select Text

• Click on the newly created statement and press **Ctrl+I** while it is selected. Doing this would change the statement to an **IF** condition.

🍪 Project Manager 🛛 🦻 DefaultForm	
‰ 43 🚱 × 42 ↑ ↓ ✓ 49 🖻 & & ₽ ጅ ⊑	🍫 🖹 🔮 🔯 🔜 🛷 🍕 🗸
🍪 Project	v
A MyApplication	v
🖃 🛅 DefaultForm	/ 🖌 🦓 🔛
🗉 📴 MenuBar 1	v
ab Edit1	04
Ag_Enter_an_abbreviation_of_any_day_in_a_week_1	04
🗆 🗖 Button1	04
E 🖓 proc OnClick (Sender\$)	/ 🖉 👋
<pre>Local (Variable1\$);</pre>	/ 🖌 🔘
<pre>Variable1\$ = (Edit1.Text);</pre>	/ 🖉 👋
👬 If (Edit1.Text ==)	/ 🖌 🔘

Figure 275: Press Ctrl+I for IF statement

• While it is still selected, go to the **LogicExpr** property of our **IF** statement and write *"mon"*.

🖃 🧰 DefaultForm	/ 🗸 🎝 📉	Parameters		
. ■ E MenuBar1	ø	expr:		
ab Edit1	04	- Action		
As _Enter_an_abbreviation_of_any_day_in_a_week_1	04	Action:		
🖃 🚍 Button1	04	Operator:	=	
E A proc OnClick (Sender\$)	/ *	E Logic		
<pre>Local (Variable1\$);</pre>	/ * *	Logic:	If	
<pre>Variable1\$ = (Edit1.Text);</pre>	/ * *	LogicAction:		
If (Edit1.Text == "mon")	/ / 8	LogicExpr:	=="mon"	
	/ • •	Parameters		
		expr:		
		- Status		
		Operation:	SetProperty	
		Owner:	ab Edit1	
		Source:	Text	

Figure 276: Set LogicExpr

• While the **IF** statement is still selected, press **Ctrl+Space** Bar on the keyboard. This will bring the code complete window. Select **ShowMessage** from it.

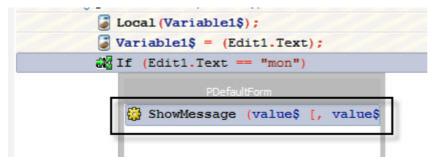


Figure 277: ShowMessage

• Change the Value property of the **ShowMessage** statement to "Monday". This will be displayed if the IF conditions finds "mon" written in the **PEdit** box.



Figure 278: Change value property

• Now we will program for all the other days of the week. Drag an ELSE IF object from the Components Pane to the OnClick event. This will create an empty Else IF statement. In its Expr property, write (Variable1\$ == "tue"). As the Variable 1\$ already holds the data in the PEdit box, if its value is found to be same as "tue" it becomes certain that the abbreviation is for Tuesday.

Application	-			🕼 Properties 🛷 Events
E Folder		🔏 🕹 🕒 × 🕸 🕈 🦊 🗸 🛷 🖾 🗞 🚷 🖻 🖻 🗳 🚳 🗃 🖉 🗔 📾	A	Dese_If1
SameSpace			1 4 Va +	
PComment	=	Company Project	1	[🎝 🛔 🗢 👄 📄 🗉 🗉
PConsole		V	1	Property Value
PLibrary		E DefaultForm	/ 🧳 🔯	- Parameters
Class		🗄 🖺 MenuBar1	1	expr: Variable1\$=="tue"
PComponent		ab Edit1	/ *	
AP PEvent		Aa_Enter_an_abbreviation_of_any_day_in_a_week_1	/	
😳 PMethod		E 🗖 Button1		
👫 PObject		Proc OnClick(Sender\$)	/	
PProperty		Local (Variable1\$);	/	
PVariable		Variable1\$ = (Edit1.Text);	/	
CodeFlow		🗄 📸 If (Edit1.Text == "mon")	/	
🔄 Break	_/	🔂 ShowMessage ("Monday") ;	/ 🔍	
P Continue	//	Delse if (Variable1\$ == "tue")	/	
👆 Else				
🕽 Else If				
🗐 Exit				

Figure 279: Create Elself Object

• While it is still selected, press **Ctrl+Space** bar on the keyboard to initialize code complete window. In this window, select **ShowMessage** for raising an alert.

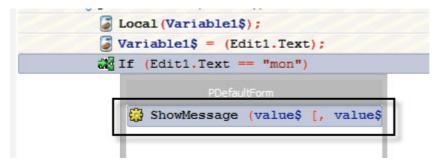


Figure 280: ShowMessage

• In the Value Property of ShowMessage, write "Tuesday".



Figure 281: Change value property

• Just like the above **Else IF** statement, we can create statements for all the days of the week.

MyApplication	v	^ [🚯 🐙 🖛 🗉	
E DefaultForm	a 🎻 😒	Pr	operty Value
🕀 📳 MenuBar 1	v	- Parameters	Contraction of the second s
ab Edit1	V	expr:	Variable1\$ == "sun"
Aa _Enter_an_abbreviation_of_any_day_in_a_week_1	v		
E Button1	v		
<pre></pre>	V 4		
Local (Variable1\$);	 ✓ 		
<pre>Variable1\$ = (Edit1.Text);</pre>	V 4		
E in (Edit1.Text == "mon")	V 4		
ShowMessage ("Monday");	V (
<pre> Boundary else if (Variable1\$ == "tue") </pre>	V (
ShowMessage ("Tuesday");	V (=	
<pre> Book else if (Variable1\$ == "wed") </pre>	 ✓ 		
ShowMessage ("Wednesday");	 ✓ 		
<pre>Belse if (Variable1\$ == "Thr")</pre>	 ✓ 		
🚱 ShowMessage ("Thursday");	V (
<pre>B lse if (Variable1\$ == "fri")</pre>	V (
ShowMessage ("Friday");	V (
<pre>Belse if (Variable1\$ == "sat")</pre>	V (
🎲 ShowMessage("Saturday");	V (
<pre>Black if (Variable1\$ == "sun")</pre>	✓		
ShowMessage ("Sunday");	V (100	

Figure 282: Project screenshot

• The application is complete! Save the application by pressing Ctrl+S or going to File>Save As.. and save it in a folder of choice.

File	Edit Sear	ch Insert	View	Cla				
	New		Ctrl+N					
	Open		Ctrl+O					
	Save	Ctrl+S						
P	Save as							
	Save Projec	:t						
P	Save Projec	t As						
ß	Import PPL	1.x Project						
9	Print		Ctrl+P					
	Exit							

Figure 283: Save Project

• Press **F5** to run the project and see the output:

🖉 DefaultForm	
File	
Enter an abbreviation of any day in a week	
	ОК

Figure 284: Output

PVariable

A **PVariable** in PIDE can be used to add a variable to an application. This variable can be used to act as a standard variable with different methods and properties attached to it. In this simple example given below, we will use **PVariable** to declare a variable and then use that variable to store values. The variable will then transfer its value to another variable which will then display it on the screen.

 Start with creating a Desktop Form Project. To do so, press Ctrl+N or go to File>New and select Desktop Form project in the Select New Project Type.. window.

esktop Form						
tegory:						
All Project	ts					
Code File	ŝ	Console	Console	Database	Production	
Code File	Component Project	Console	Console	Database	Database Project	
T	Reg	3	•			
Desktop Form	Desktop Game	Empty Project	Find Wizard	GameAPI	Help File	
R				P		
MenuBar	Mobile Form	Mobile Game	Mobile VGA Game	Package	PPL Class	
escription:						
reates a new fo	rm project					

Figure 285: Create New Project

• In the **Default Form Project Manager**, double click the **Default Form** to initialize the Form editor and place one **PEdit** object with one PLabel and one **PButton**; like shown in the figure below. To create the aforesaid components on the Form editor, click on the component you are looking for in the **Components Pane**. After selecting the component, click on a place on the **Form Editor** to place that component there. Refer to the screenshot below to configure your components.

	I	2	De	fa	ul	tF	0	m	n																		
							•						•	•			• •										
•	•	•	• •	•	•	• •	•	•	•	•	•	•	•	•	• •	•	• •	•	•	•	•	•	•	• •	•	· ·	
•	•		• •																					:::			
	:			:					:														1				
:	:	:	: :	:	:		:	:	:	:	:	:	:	:		:	:	•	:	•	:	:		Ed	it	1	
•	•	•	• •	•	•	• •	•	•	•	•	•	•	•	•	• •	•	• •	•	•	•	•	•		• •	•		
	•		• •																					<u> </u>	-		
	:																								E	Button1	
•	•	•	• •	•	•	• •	•	•	•	•	•	•	•	•	• •	•	• •	•	•	•	•	•	•		•	• •	
•	•		• •														• •	•	•	•	•	•	•	• •		• •	
1			: :											:							:						
																								ah	el	1	
																									÷.	÷.	
•	•	•	• •	•	•	• •	•	•	•	•	•	•	•	•	• •	•	• •	•	•	•	•	•			•		
•	•	•	• •	•	•	• •	•	•	•	•	•	•	•	•	• •	•	• •	•	•	•	•	•	•	• •	•	· ·	
•	:	•	• •	•	•	• •	•	•	•	•	•	•	•	•			• •		•	•	•	•	'	• •		<u>.</u>	

Figure 286: Place components

• No we can go back to our **Project Manager**. Click the **Project Manager** button on the top left and double click the **PButton** to create an **OnCreate** event. This event will be our basis for all the actions because they will only be started when the button is clicked.

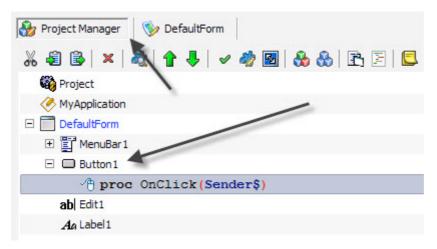


Figure 287: Double click for OnClick

• All programming languages are using variables to store data; these variables can then be used to manipulate data with the help of various programming tools. For creating a variable in PIDE we first need to declare it. For doing so, we will use the **PVariable** object form the component pane. Drag a **PVariable** from the component pane to the **OnClick** event. This will create a definition for a new variable.

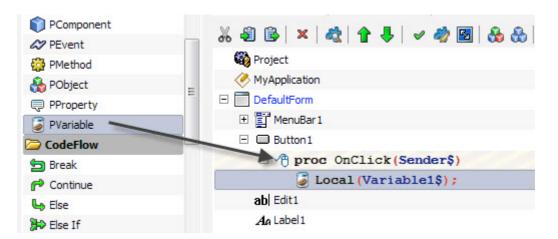


Figure 288: Drag PVariable

• Now, to create a variable, we will simply drag our newly created variable definition back on the **OnClick** procedure while pressing the **ALT** key. This will create a new variable ready to hold any value.

🍪 Project
MyApplication
DefaultForm
🛨 🛅 MenuBar 1
Button1
<pre>proc OnClick(Sender\$)</pre>
<pre>Local (Variable1\$);</pre>
<pre>Variable1\$ = ();</pre>
ab Edit1
Aa Label1

Figure 289: Create Variable

• The newly created variable does not contain any value as for now. Drag the **PEdit1** object to the **Expr** property of Variable1\$ and select the Text property in the **Code Completion** box that appears.

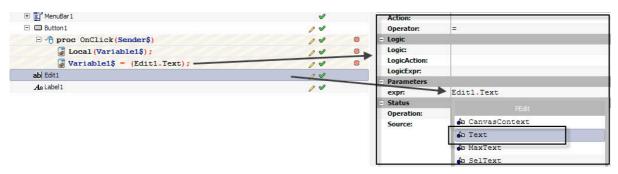


Figure 290: select Text in source

• Now Variable1\$ holds the value entered in the PEdit box. We now have to transfer that value to PLabel1. For doing this, we will have to change the Caption property of

PLabel to be equal to the value contained in the **Variable1**. Drag the PLabel1 object to **OnCreate** event and select Caption form the code complete window.

🙀 Project	v
X MyApplication	v
DefaultForm	/ 🗸 🦏 🔯
🕀 📴 MenuBar 1	v
E Button1	04
Proc OnClick(Sender\$)	/ 🗸 👋
Local (Variable1\$;	/ 🗸 👒
<pre>Variable1\$ = (Edit1.Text);</pre>	/ 🖉 👋
An Label1.Caption =;	/ 🗸 👋
ab Edit1	04
Ag Label1	04

Figure 291: Drag Plabel1 to OnCreate

• All that remains now is to assign Variable1\$ to the Caption of PLabel1. Drag Variable to the Expr property PLabel object.

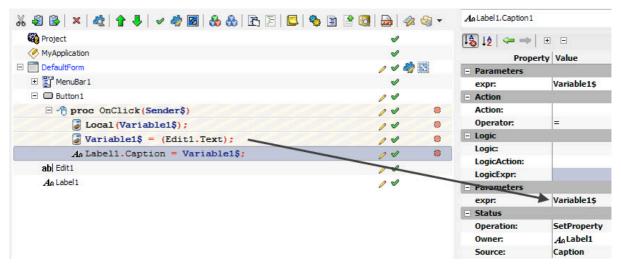


Figure 292: Drag Variable to Expr property

• Doing the above steps, we have given the value contained in a **PEdit** box to a PLabel. Save your project by pressing **Ctrl+S** key on the keyboard or go to **File>Save As..** to save it to your desired place.

File	Edit Searc	h Insert	View	Cla
2	New		Ctrl+N	
	Open		Ctrl+0	
	Save		Ctrl+S	
B	Save as			
	Save Project			
P	Save Project As			
ł	Import PPL 1.x Project			
4	Print		Ctrl+P	
	Exit			

Figure 293: Save Project

• Press **F5** to run the project and see the output:

🥙 DefaultForm	
File	
	PIDE Application
	Button1
	PIDE Application

Figure 294: Project Output

PMethod

A **PMethod** is used to add a method to a program. Methods in programming are used to add varied functionality to an application. Methods in an application do not only allow a programmer to add additional functionality, it also allows him/her to make necessary adjustments according to the programming structure. Given below is an example that would illustrate how to use **PMethod** to add two numbers and print their sum on a message box.

 Start with creating a Desktop Form Project. To do so, press Ctrl+N or go to File>New and select Desktop Form project in the Select New Project Type.. window.

esktop Form						
tegory:						
All Project	ts					
Annual Annua	ŝ	\$	\$		S	
Code File	Component Project	Console	Console	Database	Database Project	
	(Reg)	0	1			
Desktop Form	Desktop Game	Empty Project	Find Wizard	GameAPI	Help File	
R				P		
MenuBar	Mobile Form	Mobile Game	Mobile VGA Game	Package	PPL Class	
scription:						
eates a new fo	rm project					

Figure 295: Create New Project

• In the **Default Form Project Manager**, double click the **Default Form** to initialize the Form editor and place two **PEdit** objects with one **PButton**; like shown in the figure below. To create the aforesaid components on the Form editor, click on the component you are looking for in the **Components Pane**. After selecting the component, click on a place on the form editor to place that component there. Refer to the screenshot below to configure your components.

Project Classes	# ¥ \$ ₽ ₽ ₽ ₽ ₽	🖅 🕰 🛄
PDefaultForm		
🗁 Standard Controls	DefaultForm	
PButton		
PCheckBox		\
E PComboBox	Edit1	Edit2
PControl		
ab PEdit		
PGrid		Button1
[^{Xvz}] PGroupBox		
PImageList		
Ag PLabel		
目 PListBox		
HI PListView		
PMemo		
PPaintCanvas		

Figure 296: Place components

- After placing the **PEdit** objects and the **PButton** object on the form, click on the **Project Manager** button to go back to the **Project Manager**. You would notice there are 2 **PEdit** objects and 1 **PButton** object added to the **Project Manager**.
- Once in the **Project Manager**, drag a **PMethod** object and drop it outside the **DefaultForm** code.

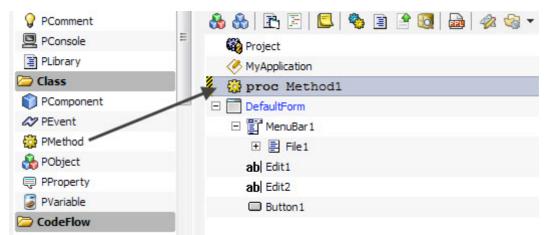


Figure 297: Drag PMethod

• Click on the **PMethod** object and change its **Parameters** property to **x\$,y\$**. This would specify that this method would take two variables, namely x and y.



Figure 298: Change Parameters property

• Now, while the **PMethod** object is still selected, press **Ctrl+Space** to bring the **Code Completion** window.

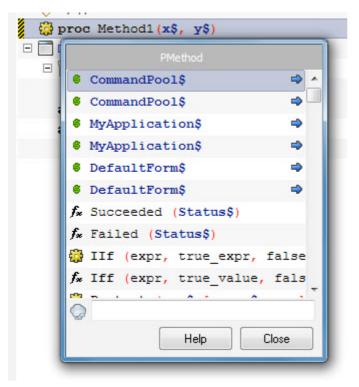


Figure 299: ShowMessage

• In the Code Completion window, select ShowMessage.

🧭 My	Application
😳 pr	roc Method1(x\$, y\$)
	PMethod
-	🐉 ShowMessage (value\$ [, value\$
Ę	🐉 ShowStatusWindow (visible\$)
	🖈 ShowModal (hwnd\$, controlhwno
j	🖡 ShowWindow (hwnd\$, showwindow
Ę	🐉 ShowFPS (visible\$, color\$)
Į	🐉 ShowSprite (spritehandle\$, vi

Figure 300: ShowMessage

• Select the **ShowMessage** object and write x\$+y\$ in its **Expr** property. This would print the addition of both variables in a message window.



Figure 301: Change Expr property

• Now, double click on the **PButton** object to create an **OnClick** event. This event will trigger the addition of both numbers.

🍪 Project	A
Key MyApplication	v
E proc Method1(x\$, y\$) ShowMessage(x\$ + y\$);	/ 4
ShowMessage (x\$ + y\$);	/ 4
DefaultForm	/ 🗸 🦏 🔛
🗆 📳 MenuBar 1	v
E File1	v
ab Edit1	04
ab) Edit2	04
🖃 🔲 Button 1	14
proc OnClick(Sender\$)	00

Figure 302: Double Click for OnClick event

• Drag the **PMethod** object to the **OnClick** event.

🍓 Project	v
Key MyApplication	v
🗄 🎲 proc Method1 (x\$, y\$)	/ 🖉 🐞
ShowMessage (x\$ + y\$);	/ 🖉 🐞
DefaultForm	/ 🗸 🦓 🔝
🗆 🕎 MenuBar 1	v
🗉 🖹 File1	v
ab Edit1	14
ab Edit2	14
🗆 🗖 Button 1	14
🗄 🕂 proc OnClick(Sender\$)	/ 🖉 👒
Method1 (x, y);	/ 🖌 👋

Figure 303: Drag PMethod

• While the **PMethod** object is still selected, drag and drop the **PEdit1** and **PEdit2** objects to the x and y property of **PMethod**. This would trigger a **Code Completion** window. Select the text property in them. Doing so will allow x and y parameters to have the values input by the user in **PEdit** objects.

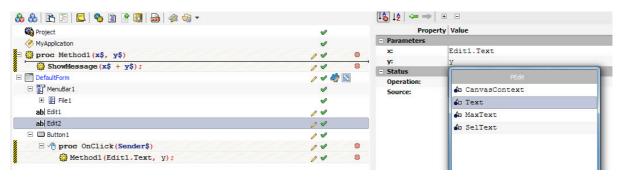


Figure 304: Drag PEdit objects to x,y property

• Save the application by pressing Ctrl+S on the keyboard or go to File>Save Project menu item to save the application.

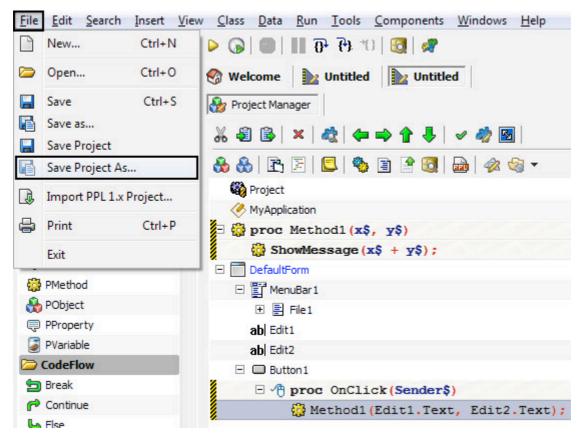
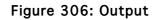


Figure 305: Save Project

• Press **F5** to see the result! This application takes in two numbers and prints the addition by using a **PMethod** object.

DefaultForm File	n		
	10	30	
	[Button1	Message
			40
			ОК



Creating Your Own Components And Distributing Them

Creating applications is all about creating programs that can work like you want them to be. Just like every other programming language, PIDE too allows you to build your own components and reuse them whenever and where ever you feel like. Apart from the many uses of creating your own components in PIDE, you can also create your own components and provide them to other PIDE developers for inclusion in their library. There are a certain number of steps and procedures involved in development of components in PIDE; here we will know the method to create your own components in PIDE as well as distributing them.

Components are individual packages that have to be installed into a PIDE system for it to be used while programming. Once installed, programmers can use the components in their programming needs.

Creating a component in PIDE basically requires three things, they are:

- **PPL source code files** This is where you create the component and save it as a PPL source code so that it can be acted upon during the program execution b PIDE.
- XML definition files These are required by PIDE in order to complete the component packaging. You do not have to create the XML files by yourself; once a PPL source code is created, PIDE can be used to create a XML file to use in component creation.
- **Help files** While not always necessary, you would need help files for your components if you want to distribute them or want to use the component's help for future reference.

Now that you know the things you need to create a component in PIDE, we will start with each component in detail.

PPL Source Code

There are two ways to create a .ppl file or application; wether you can go with a manual ppl coding or use visual coding to create equally stable applications within no time. For our example we will use visual coding.

In this section, we will learn about creating PIDE classes that can be used to deliver specific functions in a program. These classes would contain their own properties, methods, logic etc. The **DbNavigator** class example is the perfect example of a class that can be created with the help of PIDE programming and then used with other projects.

The DbNavigator class uses a toolbar interface and allows a user to navigate through the fields of a table easily. Given below are the steps that would allow you to create a DbNavigator class in the **Components Pane**l.

• Start by creating a new **Component Project** in PIDE2 and deleting the existing components class as we will be creating that on our own. Drag a **Pcontrol class** to the project and rename it as DbNavigator to make it look like a different class.

🌍 PObjectList	₩ 43 🚯 × 42 (4= => 🕆 🖡
🖕 PPoint	
PRect	🙀 Project
F PScript	DbNavigator
📡 PValue	×
A PValueList	
🗁 Standard Controls	
PButton	
PCheckBox	
E PComboBox	
PControl	
ab PEdit	
PGrid	
PGroupBox	
📳 PImageList	
Aa PLabel	

Figure 307: Drag a Pcontrol to the Project Manager

• The DbNavigator will work by going to the previous or the next data element and this requires a dataset to work with. For providing the dataset to our **DbNavigator Class**, we will include a property by dragging and dropping a **PProperty** to the **DbNavigator Class**.

🗞 Components	P	😵 Welcome 🛛 🗽 Untitled
Application	*	🎲 Project Manager
🛅 Folder	1	% € 🗟 × 🔩 ↑ 💺 🗸 🏘 🖪 🚷 🗄 🖹
Se NameSpace		
💡 PComment		🖓 Project
PConsole	Ε	🖃 🛄 DbNavigator
PLibrary		🤤 Dataset
🚮 PThread		
Class		
PComponent	/	
AP PEvent		
😳 PMethod		
🚯 PObject		
📮 PProperty		
PVariable		

Figure 308: Drag PProperty to DbNavigator

• After dropping the **PProperty**, rename it as Dataset and set its **Type Property** to **Control**.

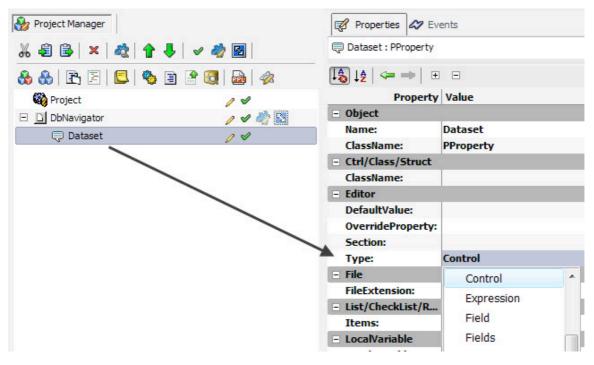


Figure 309: Set the Type property

• After change the Type property, select PDataSet in the ClassName Property.

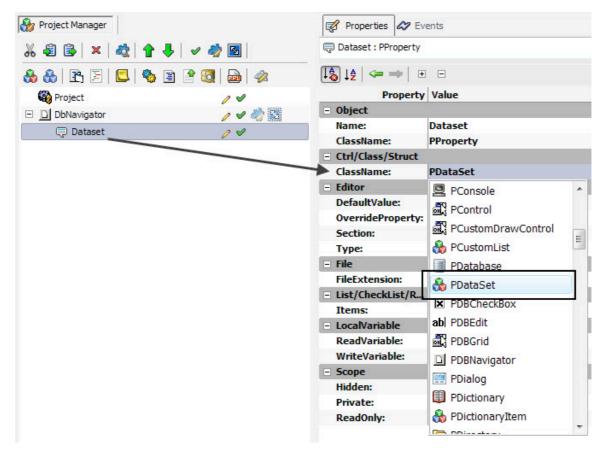


Figure 310: Set the ClassName property

• Because we want to create a navigator, we will also have to create some buttons for performing tasks. For having buttons in the **DbNavigator**, drag 4 **PToolButton** objects to the DbNavigator and name them **First**, **Last**, **Prior and Next**.

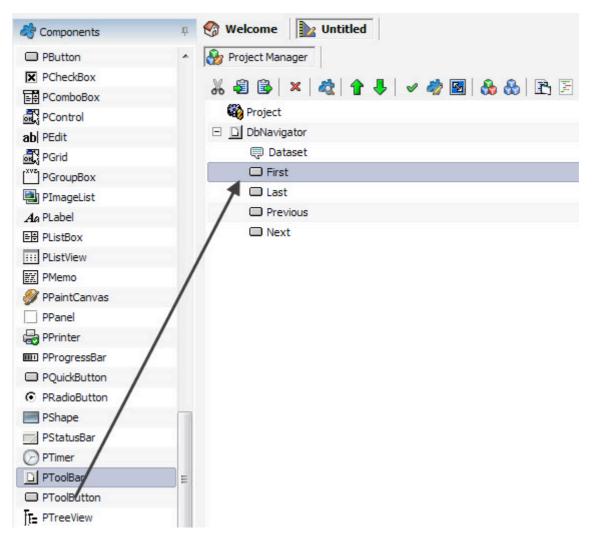


Figure 311: Drag four PToolButton Objects

• After creating the buttons, create events for the buttons by double clicking on the buttons.

) 49 🚱 × 42 🕈 🦊 🗸 42 🐼 🔂 3 🔂 2 5 5 5 5 5 5 5 5 5	
🙀 Project	14
D DbNavigator	/ 🗸 🖏 🔀
💭 Dataset	14
🖃 🖵 First	04
<pre> / proc OnClick(Sender\$) </pre>	/ 🗸 🔘
🗆 🗖 Last	14
<pre> proc OnClick(Sender\$) </pre>	/ 🖌 🖉
Previous	14
<pre> proc OnClick(Sender\$)</pre>	/ 🗸 🔘
🗆 🗖 Next	14
<pre>/ proc OnClick(Sender\$)</pre>	/ /

Figure 312: Double click the buttons to create events

• Now drag the **Dataset Property** to the First button's event, type 'first' and select the **first message** in the context menu that appears. Similarly, drag **Dataset Property** to the events of other buttons and select the appropriate messages.

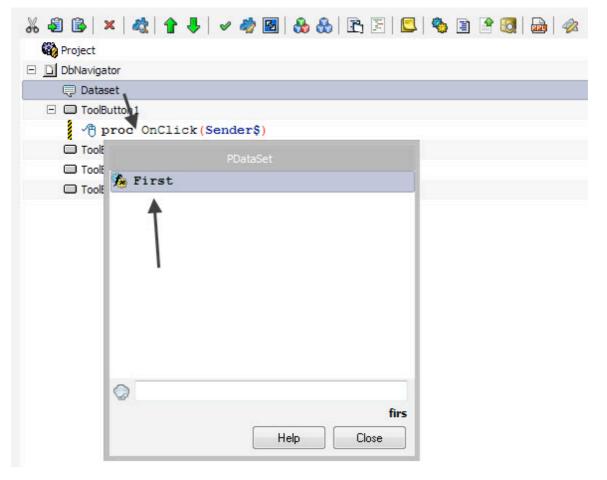


Figure 313: Select a message

🗞 🚷 🖻 🔄 🖳 🧠 🖹 🔮 🔯 💩 🛷		
🍪 Project	14	
DbNavigator	/ 🗸 🦓	23
💭 Dataset	14	
🖂 🗔 First	14	
□ 🖓 proc OnClick(Sender\$)	14	۲
💭 Dataset.First();	14	۲
🗆 🗖 Last	14	
□ 🖑 proc OnClick(Sender\$)	14	۲
💭 Dataset.Last();	11	۲
🖃 🔲 Prior	14	
□ 🖓 proc OnClick(Sender\$)	14	۲
💭 Dataset.Prior();	14	۲
Next	14	
🖃 🖑 proc OnClick(Sender\$)	14	۲
📮 Dataset.Next();	04	: ()

Figure 314: Select messages for all buttons

- The class component is almost complete. You can add images, sounds and other objects to the class to enhance its functionality and look. After you are completed with the **DbNavigator**, save the file with the desired name by pressing **Ctrl+S** or by using the **Save As..** Option form the file menu.
- After you have saved the file, some changes have to be made in the DbNavigator properties for it to become a Class Component. In the DbNavigator properties, check the AutoCreate Property, specify the category in which DbNavigator will be included in by writing it in the Category Property, write PForm in the ChildOf Property to determine where this class will work on. PForm as well as Ppanel can be placed on almost all visual codes. After checking the HasOwner, HasParent and Visual Property, save the file again. Users can also tweak other properties like the library Property and the Icon Property if they are using other options. HasOwner means the object will be created and owned by the root object (like form, panel, etc...). Simply speaking, selecting this property specifies that this object has a parent object that would contain it.

HasParent is the same as **HasOwner** and ties the control to the parent control. If the parent control is moved, it would also move the child control which has been specified with the **HasParent** property. **HasParent** property can be very useful in in many functions. For example, if you hide the parent control, all child controls will hide as well, etc...

- Behavior	
TabStop:	
- Binds	
Binds:	[0 pbindobject,pbindvar object(s)]
Editor	
AsFolder:	
AutoCreate:	
AutoGenerate:	
Category:	Database
ChildOf:	PForm
DefaultEvent:	
DefaultProperty:	
DragExts:	
DragProperty:	
HasOwner:	
HasParent:	
Icon:	component.bmp
Library:	pd
Rename:	
Sort:	
Visible:	
Visual:	
- Position	

Figure 315: The property panel

• After tweaking the properties of the class, go to the components menu and select **Install components from the current project**. This will create a DbNavigator in the **Components Panel** under the database column.

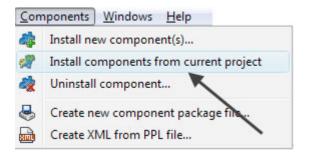


Figure 316: Components Panel

• To use the newly created **DbNavigator Class**, create a **New Project** and add a database along with the tables and fields to it.

Drag the **DbNavigator Class** to the **Project Manager** and change the **Dataset Property** of DbNavigator to the **table** in the database. This will provide the data available in the fields of the given table to the DbNavigator.

XML Definition Files

PIDE includes a built-in function that will automatically generate a XML file from a .ppl file. When a component is created, PIDE automatically creates the XML definition that is needed for its creation.

Generate the Component Package

Once you have the .xml file definition, the help(.hlp) file(optional) and the .ppl source code, you can generate a component package that can then be installed in other PIDE installations. To generate the package,

• Select the Create Component Package Option from the Components Menu. This will open the Select Component Definition File... Window.

New Component Package		×
Component package filename:		ОК
	ì	Cancel
Included library files:		
	0	
	۰	

Figure 317: creating a new component package

The Select Component Definition File Window allows you to locate the XML definition file associated with the .ppl source code.

When you select the .xml definition file, the **Select Help Definition File... Window** will open allowing you to locate the help file for the component.

When you select the help file, the **Save Component Package Where... Window** will open allowing you to locate a location to save the component package file.

This will create a **ZIP** file containing the **PPL**, **XML** and **HLP** files to distribute. If you or someone else would want to install that component package file that is just created in PIDE, select **Install New Components**, not Install Components from current project.

More In-Depth Object Binding

Object binding allows a programmer to bind two objects in PIDE with each other, with the use of object binding, object properties can be bound by properties or by variables so as to provide more control over the program and provide interactivity.

The example given below illustrates how the text property of a **PEdit** object can be bind with the caption property of a **PButton**.

 Start with creating a Desktop Form Project. To do so, press Ctrl+N or go to File>New and select Desktop Form project in the Select New Project Type.. window.

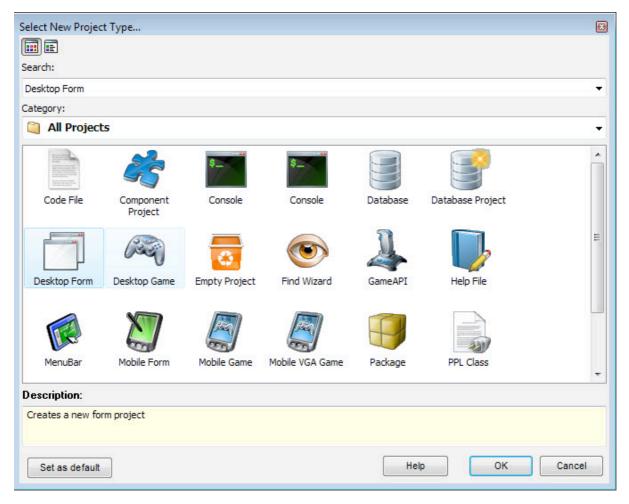


Figure 318: Create New Project

• In the **Default Form Project Manager**, double click the **Default Form** to initialize the Form editor and place one **PEdit** object with one **PButton**; like shown in the figure below. To create the previously mentioned components on the Form editor, click on the component you are looking for in the **Components Pane**. After selecting the component, click on a place on the form editor to place that component there. Refer to the screenshot below to configure your components.

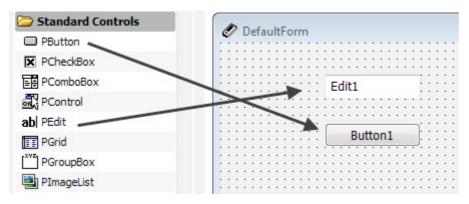


Figure 319: Place components

• For binding the property values of two objects with each other, **right click** on the source object and select **View Binds** or press **Shift+Ctrl+B** to open the **Edit Binds window**.

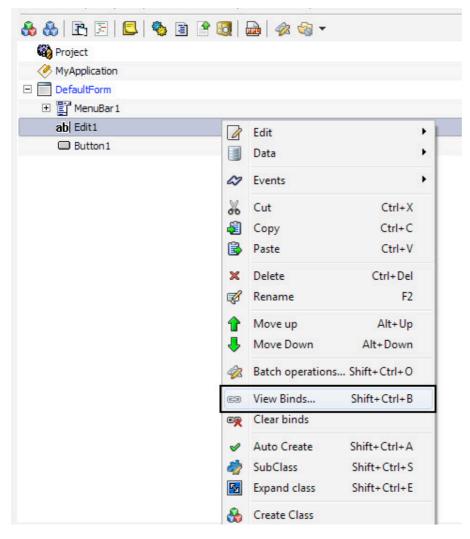


Figure 320: View Binds option

• In the Edit Binds window, select the New Button to open a 'New Component Class' menu and select pbindobject. The pbindvar option is used to link a variable with the object property.

Font.Binds <pbindobject,pbindvar collection=""></pbindobject,pbindvar>	E
Choose component class:	
© pbindobject	
© pbindvar	
He	əlp

Figure 321: choosing components class

• Property panel for object binding is divided in two parts i.e. the **source** and the **target**. For binding two objects with each other, fill in all the property values in the given properties.

		/ 🗸 🦓	Object	5
		V	Name:	pbindobject[0]
		v	 ClassName:	pbindobiect
	ls <pbindobject,pb< td=""><td>indvar collection></td><td> Source SourceOptions: SourceProperty: </td><td>boAuto boRead boWrite boDe</td></pbindobject,pb<>	indvar collection>	 Source SourceOptions: SourceProperty: 	boAuto boRead boWrite boDe
Index	Name	ClassName	 Target TargetObject: 	-
ce ()	pbindobject[0]	pbindobject	TargetOptions:	boAuto boRead boWrite

Figure 322: Source and Target sections

• In the **Property Panel**, select the **SourceOptions** property and check or uncheck the property you want. The **SourceOptions** property consists of four choices, namely,

boAuto – Specifies whether property will change automatically or not

boDefault – Specifies whether this is the default value for the target or not

boRead - Specifies whether it will read from the target or not

boWrite – Specifies that the target value will be written to the source property value

• The **SourceProperty** property specifies the source property that will be linked with the target property.

- Object	10 10		
Name:	pbindobject[0]		
ClassName:	pbindobject		
- Source	an an ann		
SourceOptions:	boAuto boRead boW	rite boDe	
SourceProperty:	Text	ę	
 Target 	Align	^	
TargetObject:	Anchors		
TargetOptions: TargetProperty:	Binds		
Targetrioperty.	BorderStyle		
	Color		
	Enabled	=	
	Font	-	
	Height		
	Left		
	ReadOnly		
	State		
	TabStop		
	Text		
		-	

Figure 323: SourceProperty

• The TargetObject property allows a user to specify the **Target Object** that will be linked with the **Source Object**.

Property	Value	
Object		
Name:	pbindobject[0]	
ClassName:	pbindobject	
- Source		
SourceOptions:	boAuto boDefault boWrite	
SourceProperty:	Text	
Target		
TargetObject:	Button1	f*
TargetOptions:	None	
TargetProperty:	DefaultForm	
	DefaultForm.Button1	
	ab DefaultForm.Edit1	
	DefaultForm.Exit1	
	E DefaultForm.File1	
	🖺 DefaultForm.MenuBar1	
	MyApplication	

Figure 324: Selectiong a TargetObject

• The **TargetOptions** property allows a user to specify the options that will determine the object binding options for the **Target Object**. Just like SourceOptions, these options govern how the **Target Object** will function with **source object**.

Property	Value	
 Object 		
Name:	pbindobject[0]	
ClassName:	pbindobject	
- Source		
SourceOptions:	boAuto boDefault boWrit	
SourceProperty:	Text	
 Target 		
TargetObject:	Button1	
TargetOptions:	boAuto boWrite	
TargetProperty:	Jo boAuto	
	boDefault	
	boRead	
	✓ boWrite	

Figure 325: Target options

• The **TargetProperty** property specifies the property of the target object that is linked with the property of the **source object**.

Property	y Value		
 Object 			
Name:	pbindobject[0]	pbindobject[0]	
ClassName:	pbindobject		
- Source	1		
SourceOptions:		boAuto boDefault boWrite	
SourceProperty:	Text		
Target			
TargetObject:	Button1		
TargetOptions: TargetProperty:	boAuto boWrite Caption		
rargetrioperty.			
	None	<u><u></u></u>	
	Align		
	Anchors		
	Binds		
	BorderStyle		
	Caption	E	
	Color		
	Default		
	Enabled		
	Font		
	Height		
	Left		
	TabStop		
	T		

Figure 326: Target property

• After all the object properties are filled, you can close the object binds window and run your project to see the results.

ile	
i like PPL	
i like PPL	

Figure 327: PEdit text binded with PButton object

PDirectory, PFile And PFileList Example

File handling is an integral part of any comprehensive application. Believe it or not, anything from a small text editor application like notepad to an hefty one like PIDE itself needs to save data to and Load Data from files and that's where file handling comes in. Being a useful programming ability, PIDE too has the ability to open, save and close files as well as folders with the help of **PDirectory**, **PFile** and **PFileList** objects play an important role in it.

In this section of the manual, we will get to know about such objects and what PIDE users can do with them.

Displaying the number of files available in a directory with PDirectory Object

• Start by creating a new **Desktop Form** project. For doing this, press **Ctrl+N** or Go to **File>New** and select **Desktop Form** from **Select New Project Type...** window.

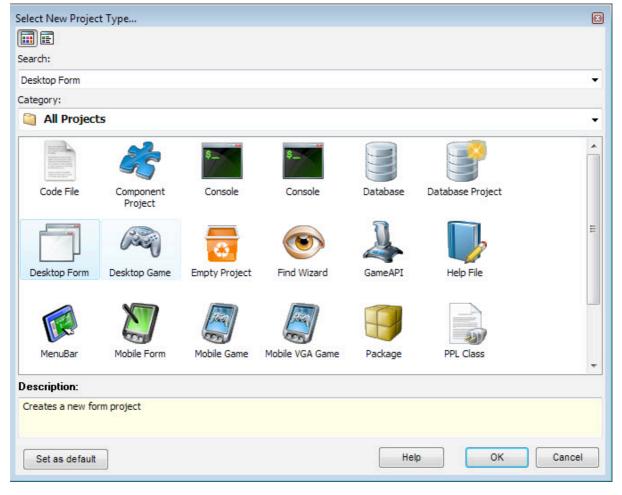
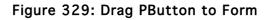


Figure 328: Create new project

• Drag a **PButton** object to the **Project Manager** and double click it to create an **OnClick** event. This event will help us create actions once the button is clicked.

🚵 PMemory	🚷 🚷 🖹 🗐 🛄 🎭 🖹 🔮 🔇	📠 🛷 🁒 🔻
🌍 PObjectList	🖓 Project	v
PPoint	MyApplication	v
PRect	🖃 🥅 DefaultForm	/ 🗸 🦏 🔊
PScript	🖃 🕎 MenuBar 1	v
Yelue	🕀 🖶 File1	v
PValueList	Button1	04
Project Classes PDefaultForm		
Standard Controls		
PButton		
PCheckBox		



🍪 Project		V	
MyApplication		V	
DefaultForm	0	I 🧳	3
🖻 📴 MenuBar 1		v	
∃ E File1		<	
🖃 🗔 Button 1	0	v	
<pre> proc OnClick(Sender\$) </pre>	0	v	۲

Figure 330: Create OnClick event

• Drag a **PDirectory** object from the File section of **Components Pane** to the **OnClick** event. This would declare a new **PDirectory** object.

PColorDialog	🐘 🚷 🕃 🖹 🔲 🎭 🖹 🔗 🧕	📄 🏟 🎲 🔻
🐖 PDialog	Roject	<i>v</i>
A PFontDialog	MyApplication	v
🔄 POpenDialog	DefaultForm	/ 🗸 🤣 🔀
🔚 PSaveDialog	🖃 📳 MenuBar 1	v
쳘 File	E E File1	v
Directory	🖃 🗔 Button 1	14
PFile	proc OnClick (Sender	r\$) 🖉 🖉
🔄 PFileList	Directory1 = new	PDirectory 🖉 🖉 🔍
🤪 PPackage	×	

Figure 331: Drag PDirectory

• Change the **PathName** property of **PDirectory** to point to the directory you want. For our example, we will point it to the default PPL folder.

% € 😫 × 🖧 🖛 🔿 🕇	L 🖌 🤣 🐼		Directory1 : PDirector	у
		•	I∰ ↓≜ <= ⇒ ±	
🍪 Project		v	Property	Value
MyApplication		1	- Binds	
		/ 🗸 🦏 📓	Binds:	[0 pbindobject,pbindvar object(s)]
🖂 📳 MenuBar 1		2	Directory	
⊡ I File1		1	AutoScan:	
		-	Extension:	*,*
		/ *	PathName:	
□ √ proc OnClick(Send □ Directory1 = ne			Recursive:	
	Select Directory Directory Name: C:\Program Files\PPL2 Directories: C:\ Program Files PPL2 help Icons PCL Project Templates		ArianeSoft blog.URL ArianeSoft Youtube cha ArianeSoft.ca.url	

Figure 332: Change PathName

• The Extension property allows us to select the files we want to grab with the help of **PDirectory** object. While you can keep the Extension Property pointing to *.* to account for every file, you can also change this property to account for specific type of files. In our example we will change this to *.url. This would ensure that only files with .url extension are taken by the **PDirectory**.

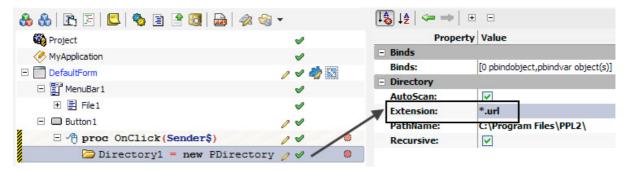


Figure 333: Change Extension property

• While the **PDirectory** object is still selected, press **Ctrl+Space** to bring code complete window and select **ShowMessage** from it.

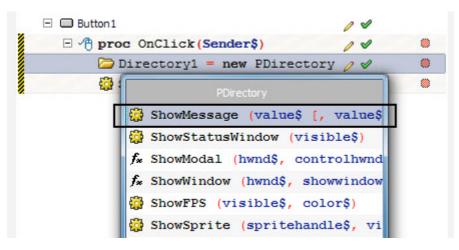


Figure 334: ShowMessage

• While **ShowMessage** object is still selected, drag the **PDirectory** object to its **Value** property. This would initiate a **code complete** window.

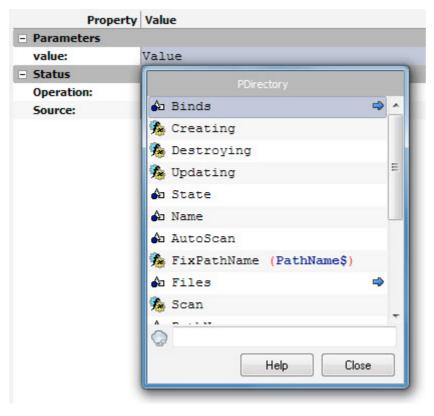


Figure 335: Code Complete window

• In the code complete window, go to Files and click the arrow beside it.

 Parameters 	D:				
value:	Directory 1\$	_			
- Status	PDirectory				
Operation:	🖨 Binds 🔿				
Source:					
	Screating				
	🏂 Destroying				
	🏂 Updating				
	la State				
	🎝 Name	_			
	🗛 AutoScan				
	🏇 FixPathName (PathName\$) 🚽				
	🎝 Files 🗢				
	🏂 Scan				
	A	-			
	Help Close	٦			

Figure 336: Click the arrow

• Now in the code complete window, select **Count** property. After you have selected **Count** property, the Value property of **ShowMessage** should look like **Directory1.Files.Count**

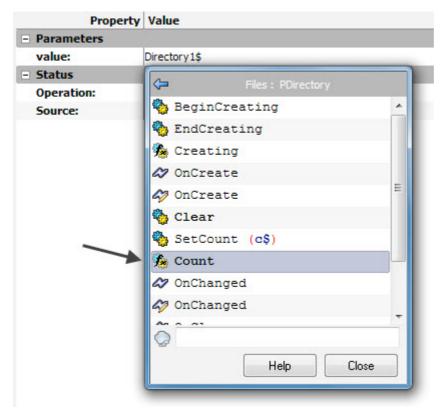


Figure 337: Select Count Property

• That's it! The application is complete. Now you can save the project by going to File>Save Project and save it anywhere on the computer.

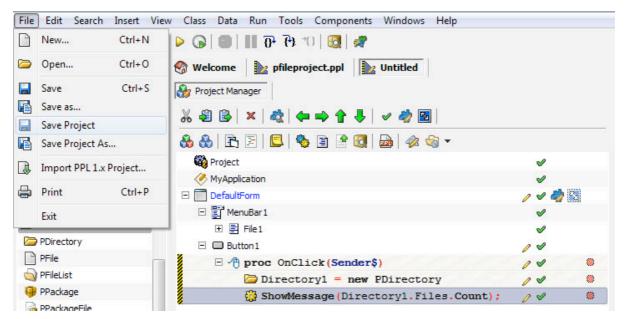


Figure 338: Save Project

• Press **F5** to see the result!

e	
Button1	Message
	5
	ОК

Figure 339: Output

Getting the file size of a file by using PFile object

• Start by creating a new **Desktop Form** project. For doing this, press **Ctrl+N** or Go to **File>New** and select **Desktop Form** from **Select New Project Type...** window.

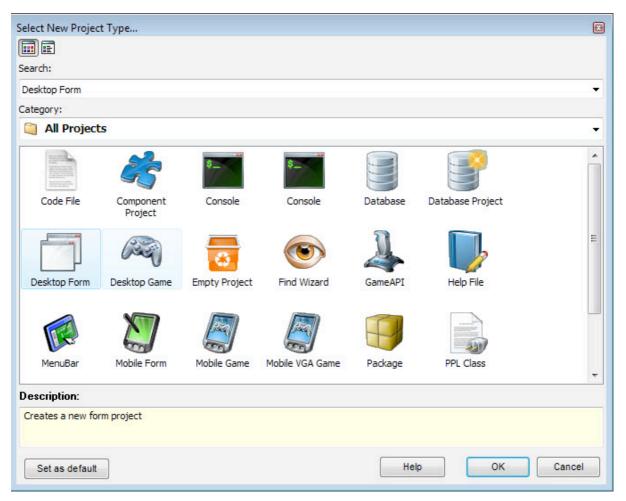


Figure 340: Create New Project

• Drag a **PButton** object on the **Default Form** and double click it to create an **OnClick** event which would be used to initiate all the actions on our application. Alternatively, for creating an **OnClick** event you can also right click the **PButton** object and select **OnClick** from the **Events** sub menu.

೫ £ 🗟 × 🎎 🖛 🔿 🕈 🖊 🗸 🐗	2 😼				Action
🐣 🚷 🖹 🗐 🖳 🍫 🖹 🔮 🧕 4	2	+		~	OnEnter
Project			00	~	OnExit
MyApplication			1		A <u>l</u> location
DefaultForm			 Image: A state of the state of	۵	OnCreate
🗉 🕎 MenuBar 1			v	-	OnDestroy
🗉 🗖 Button 1 🚩	0	Edit	/	•	<u>K</u> eys
<pre> proc OnClick(Sender\$) </pre>		Data	/	. 3	OnKeyDown
	-			3	OnKeyPress
	07	Events 🚩		3	OnKeyUp
	26	Cut	Ctrl+X		Mouse
	3	Сору	Ctrl+C	1	OnClick
	3	Paste	Ctrl+V	18	OnMouseActivate

Figure 341: Create OnClick event

• Drag an **PEdit** object to the project above the **PButton** object you just created. This **PEdit** box will help us point to the path of the file we want to the size of.

🗁 Project Classes	🐁 🚷 🖻 🔄 🗳 🎭 🖹 🔮 😡 🛷 🧠 🔻	
PDefaultForm	Project	×
🗁 Standard Controls	MyApplication	V
PButton	DefaultForm	/ / 🤣 🖏
X PCheckBox	🖃 🖺 MenuBar 1	v
PComboBox	ab Edit1	04
PControl	Button 1	14
ab PEdit	<pre> proc OnClick(Sender\$) </pre>	/ *
Figure PGrid	* ************************************	-

Figure 342: Drag PEdit

• Now we would need to store the address of the file input by the user on the **PEdit** object in a variable so that it can be used. For doing this, drag a **PVariable** object to the **OnClick** event. This would create the definition of a new variable.

PConsole	- 🗞 🚷 🖻 🗄 📕 😓 🍫 🗃 🔮 👧	
PLibrary	E 🖓 Project	v
🔁 Class	MyApplication	v
🗊 PComponent	DefaultForm	/ / 2
AP PEvent	⊞ MenuBar1	v
🚱 PMethod	ab Edit1	14
🚯 PObject	D Button1	14
PProperty	proc OnClick(Sender\$)	/ *
PVariable	<pre>Local (Variable1\$);</pre>	0 4 8
CodeFlow		

Figure 343: Drag PVariable

• Drag the newly created Variable definition back to the **OnClick** procedure while holding the **ALT** key on the keyboard to create the variable.

🚯 Project	v	
Average MyApplication	v	
🖃 🧰 DefaultForm	/ 🗸 🦏	3
王 플 MenuBar1	v	
ab Edit1	14	
🗆 🗖 Button 1	14	
□ / proc OnClick(Sender\$)	14	۲
Local (Variable1\$);	14	۲
<pre> Variable1\$ = (); </pre>	04	۲

Figure 344: Create Variable

• Drag the **PEdit** object to the **Expr** property of the newly created variable. This would initiate a **Code Completion** window.

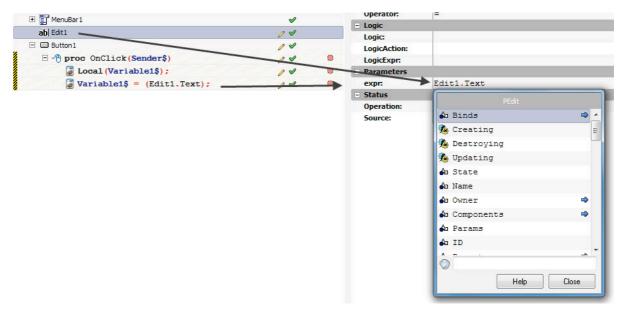


Figure 345: Drag PEdit1 to expr

• Select the **Text** property in the code complete window. This would send the text entered in the **PEdit** box to the variable.

- Parameters	
expr:	Edit1\$
 Status Operation: 	PEdit
Source:	🖓 CanvasContext
1	🎝 Text
-	🖓 MaxText
	🎝 SelText
	Close

Figure 346: Select Text

• Go to the **Components Panel** and drag **PFile** from the File section to the **OnClick** object.

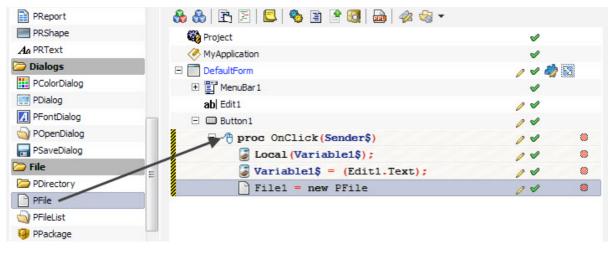


Figure 347: Drag PFile to OnClick

• Click on the **PFile** object and drag **Variable1\$** to its filename property. This would provide the path entered in the **PEdit** box to the **PFile** object.

🊜 🗐 🖹 🗙 💐 🖛 🔿 🕈 🦊 🗸 🤣		File1 : PFile	
🗞 🚷 🖻 🗐 🛄 🧠 🖹 🔮 🔕 👦 🛷 🌚 🗸		🞼 🕹 🕹 🖛 📥	.
🍪 Project	1	Prop	erty Value
MyApplication	×	E Binds	
	/ / 4	Binds:	[0 pbindobject
E		🖃 File	
	v	Filename:	Variable1\$
ab Edit1	14		
🖃 🔲 Button 1	14		
🗄 🖑 proc OnClick(Sender\$)	10		
<pre></pre>	14		
<pre>Variable1\$ = (Edit1.Text);</pre>	14		
File1 = new PFile	04		

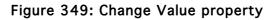
Figure 348: Drag variable to Filename

• Now, for displaying the file size, Press **Ctrl+Space** on the keyboard while **PFile** object is till selected. This would generate a code complete window. Select **ShowMessage** in it.



• Select the **ShowMessage** statement and drag **PFile** statement to its value property. In the **code complete** window that initiates, select the **Size** property. This would allocate the size of the File to the message box.

MyAppication DefaultForm DefaultForm abl Editi abl Editi Buttoni Proc OnClick (Sender\$) Local (Variable1\$); Variable1\$ = (Edit1.Term?; Parameters Variable1\$ = (Edit1.Term?; Variable1\$ = (Edit1.Term?; Variable1\$ = (Edit1.Term?; Variable1\$ = (Edit1.Term?;	Project	4		Property	Value
Defaultorm ✓ ✓ ✓ ✓	MyApplication	v		Parameters	
MenuBar1 abj Editi Button1 Proc OnClick(Sender\$) Operation: Source: Docal (Variable1\$); Variable1\$ = (Edit1.Text); Variable1\$ = w PFile	DefaultForm	/ 1 🎝	53	1	File1.Size
ab Edt1 Button1 Button1 Conclick(Sender\$) Local(Variable1\$); Variable1\$ = (Edit1.Texp); File1 = new PFile Variable1\$ = (Variable1\$); Variable1\$ = (Edit1.Texp); Variable1\$ = (Variable1\$); Variable1\$ = (Variable1\$); V	王 III MenuBar1				PFile
Button1 Button1 Proc OnClick(Sender\$) Local(Variable1\$); Variable1\$ = (Edit1.Text); File1 = new PFile Variable2	ab Edit1	04	-		🖡 Size
Image: Sender\$) Image: Sender\$) Image: Sender\$) Image: Local (Variable1\$); Image: Sender\$) Image: Sender\$) Image: Variable1\$ = (Edit1.Text); Image: Sender\$) Image: Sender\$) Image: File1 = new PFile Image: Sender\$) Image: Sender\$)	E Button1	14		Source.	
Local (Variable1\$); Image: Constraint of the second se	- 🖓 proc OnClick(Sender\$)	10	۲		
File1 = new PFile	Local (Variable1\$);	14	۲		- Excension
	<pre>Variable1\$ = (Edit1.Text);</pre>	14	۲		
ShowMessage (File1.Size);	File1 = new PFile	04	۲		
	ShowMessage (File1.Size);	200			



• That's it! The application is complete. Now you can save the project by going to File>Save Project and save it anywhere on the computer.

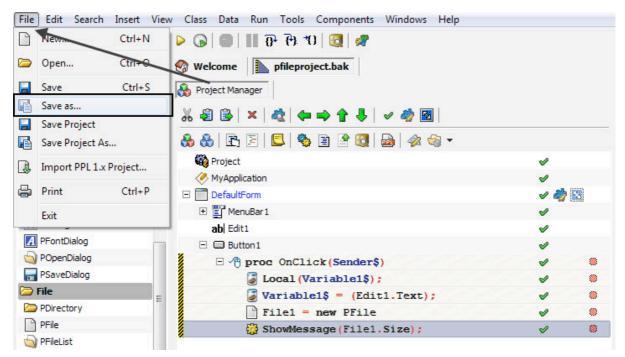


Figure 350: Save Project

• Press **F5** to see the result!

DefaultForm File		
\ppl\remote.ico		
Button1	<i>8</i> .	
	Message	X
	2550	
		ок

Figure 351: Output

Using PFileList to display the constituents of a Folder

PFileList is an object that is used inside the **PFolder** and contains a list of **PFile** items that the **PFolder** object contains. With the help of **PFileList**, actions can be performed on the while list of items. Given below is an example that would demonstrate the use of **PFileList** object to delete all the objects present in the **PFolder**. Using a **ForEach** object along with the file property of **PDirectory** we can get the **PFileList** object and work with various methods on the files; the example given below will demonstrate how to delete all the files in a directory:

• Create a new **Desktop Form** project in PIDE. For doing this, go to **File>New** or press **Ctrl+N** to get the **Select New Project Type...** window. In the **Select New Project Type..** window, select **Desktop Form** project and press **Ok**.

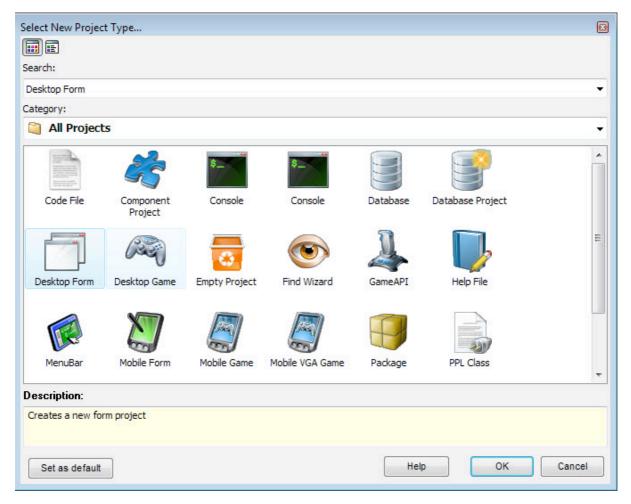
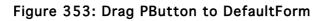


Figure 352: Create New Project

• After selecting the **Desktop Form**, Drag a **PButton** to the **Project Manager**, this will allow us to trigger the actions on the click on the button.

🝌 PMemory	🐣 🚷 🖹 🗐 🔲 🍫 🖹 🔮 🕄	! 🔂 🤣 🌚 🕶
🌍 PObjectList	Project	4
🖢 PPoint	MyApplication	v
PRect	E DefaultForm	/ 🗸 🦏 🖏
🔄 PScript	🖃 📰 MehuBar 1	v
📡 PValue	File1	1
A PValueList	Button 1	04
Project Classes	-	
PDefaultForm		
Standard Controls		
PButton		
PCheckBox		



• Double click the **PButton** to create an **OnClick** event. This event will allow us to raise all the actions we want.

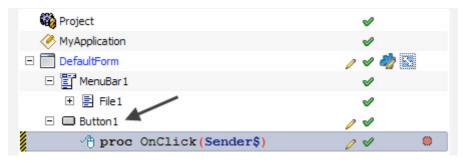


Figure 354: Create OnClick event

• Drag a **PDirectory** object from the File section of **Components Pane** to the **OnClick** event. This would declare a new **PDirectory** object.

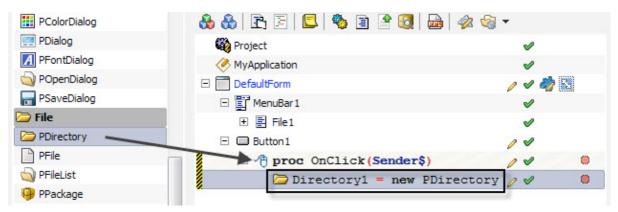


Figure 355: Drag PDirectory

• Change the **PathName** property of **PDirectory** to point to the directory you want. For our example, we will point it to a dummy folder that contains various files.

🕹 🍪 🖹 🖹 🗳 🕲 🔮 🕄	📠 🛷 🍓 🕶		🕼 🞼 🗢 🖚	•
Project		v	Prope	erty Value
MyApplication		v	- Binds	
DefaultForm		/	Binds:	[0 pbindobject,pbindvar object(s)]
E MenuBar1 (Select Directory	X	- Directory	
Button 1	Select Directory		Autoscali:	*.*
- mproc OnClick (Sende	Directory Name:		Extension: PathName:	1.7
<pre></pre>	C: \Users \Ashutosh \Desktop \PIDETest		Recursive:	
	Directories:	Eles: (*, *) Text Document - Copy (10).txt Text Document - Copy (2).txt Text Document - Copy (3).txt Text Document - Copy (4).txt Text Document - Copy (5).txt Text Document - Copy (6).txt Te		

Figure 356: Change PathName

• Since we are going to delete files, it would be clever to filter the files that we want to delete. In the Extension property of **PDirectory** object, set *.txt so that all the text files are targeted.

🔩 🞼 🖛 👄 🗎	+ -	
Proper	y Value	
= Binds		
Binds:	[0 pbindobject,pbindvar object	ct(s)]
- Directory		
AutoScan:		
Extension:	*.txt	
PathName:	C:\Users\Desktop\PIDET	est\
Recursive:		

Figure 357: Change Extension Property

• Now that the **PDirectory** object is configured, we can continue with the **PFileList** object that is represented by the **Files** method. For deleting all the files in a directory specified in the **PDirectory** object all we need to do is to call the **PFileList** object that will scan all the files in the **PDirectory** and delete all the files in there. For doing this, select the **PDirectory** object and press **Ctrl+Space** bar on the keyboard, this will bring the **Code Completion** menu.

	OnClick(Sender\$)	/ 1	(
Dir	ectory1 = new PDirectory	/ 1	(
	PDirectory		
6	Directory1\$		
	PDefaultForm		
	PApplication		
8	PObject		
C3	PBindObject		
63	PBindVar		
8	PCustomList		
<u>e</u>	PList		
8	PArray		
(x)	PMatrix		
6			
	<u>H</u> elp <u>C</u> lose		

Figure 358: Hit Ctrl+Space for Code Complete window

• Select **Files** menu item or write Files in the text box to go straight to it. Once on it, press the arrow beside it to go to a sub method.

Directory1 = new PDirectory PDirectory
€ fFiles\$
🎝 Files 🖨
f* FileSize (filename\$)
f. PackageFiles (packagehandle\$,
fx PackageFileSize (packagehand)
files
<u>H</u> elp <u>C</u> lose

Figure 359: Select Files

• In the **PDirectory**.**Files** menu, select **DeleteFiles** method. This method will delete all the files available in the directory that is specified in the **PDirectory**.

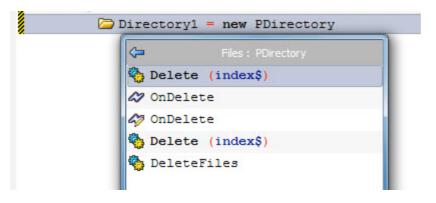


Figure 360: Select DeleteFiles

• After you have selected the **DeleteFiles** method, the **Project Manager** should look like the screenshot given below.

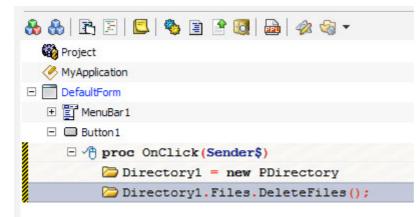


Figure 361: Project screenshot

• The application that can delete the text files from a directory is complete. Now you can save the project by going to **File>Save Project** and save it anywhere on the computer.

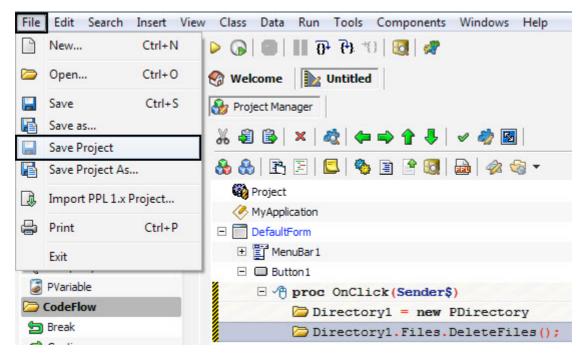


Figure 362: Save Project

• Press **F5** to see the result! Below screenshot shows that the folder that had many text files before became empty when we clicked the button.

Ø DefaultForm		
File Button1	PIDETest	→ 4y Search PIDET
	Organize Include in library Name Name	
	Desister	is folder is empty.

Figure 363: Output

Presource

Presource object can be used to store information from a file directly in the PIDE code. Dragging a file from your computer to the **Project Manager** shows **PResource** in the context window. By selecting **PResource** in the context window, we can mark an object as the resource that is needed for the execution of a program.

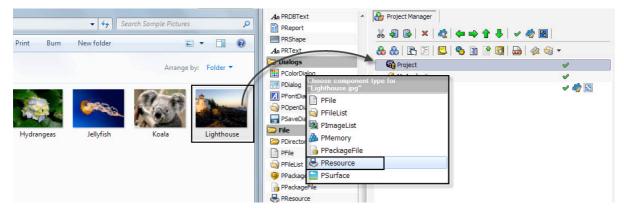


Figure 364: Drag and drop files and mark PResource

A popular example of using PResource would be of using it to store music as well as sound files that are used in applications for button clicks etc.

Additional Support

For additional support, including discussion boards, frequently asked questions, and a knowledge base, please visit the <u>http://www.arianesoft.ca/</u> website.

Know more about PIDE and what you can do from it, Visit <u>http://ppl-lang.com/</u>

Get all the support you want from your peers, visit <u>http://forum.arianesoft.ca/</u>

Watch n' learn! Visit the YouTube channel at http://www.youtube.com/arianesoft

Appendix A: Figures

Figure 1: Creating a new project	5
Figure 2: Code	6
Figure 3: Output	6
Figure 4: New Procedural project	7
Figure 5: Main Library	7
Figure 6: Hello World Code	8
Figure 7: Run Hello World	8
Figure 8: Run the project	8
Figure 9: Output	8
Figure 10: Create a new project	9
Figure 11: OnStart event	9
Figure 12: Code Completion window	10
Figure 13: Expr property	10
Figure 14: Output	10
Figure 15: New Project	11
Figure 16: Right click to Edit Form	12
Figure 17: Caption Property	12
Figure 18: Right click to open DefaultForm	12
Figure 19: Code	13
Figure 20: New Project	14
Figure 21: Right click to Edit Form	15
Figure 22: Place components	16
Figure 23: Change Caption property	16
Figure 24: Final placement	17
Figure 25: PButton	17
Figure 26: Project Manager	18

Figure 27: OnClick event	
Figure 28: Drag a PVariable	
Figure 29: Drag PEdit to expr	
Figure 30: Steps from 1 to 4	20
Figure 31: Drag operation	
Figure 32: Change property	21
Figure 33: Change Expr property	21
Figure 34: Condition	22
Figure 35: ShowMessage in Code Completion window	
Figure 36: Change Value property	22
Figure 37: Output	23
Figure 38: Create New Project	24
Figure 39: Drag and Drop PPackage	25
Figure 40: Change Property	25
Figure 41: Drop Files	
Figure 42: Expand for constituents	
Figure 43: Run the Project	27
Figure 44: Creating new project	
Figure 45: Drag a database	
Figure 46: Drag a PTable	
Figure 47: Specify a filename	
Figure 48: Give TableName	
Figure 49: Add PField object	
Figure 50: Specify Property	
Figure 51: Creating Database	
Figure 52: Viewing Table	
Figure 53: Drag PQuery	
Figure 54: Change Database property	
Figure 55: Visual query editor	

Figure 56: Insert PDBGird	
Figure 57: Drag table to form	
Figure 58: Smart move	
Figure 59: Drag PQuery	
Figure 60: Select appropriate database name	
Figure 61: Double click PQuery	
Figure 62: The four buttons of visual query editor	
Figure 63: Save file button	
Figure 64: The SQI Tab	
Figure 65: Results Tab	
Figure 66: Selecting Query Type	
Figure 67: Creating Advanced Query	40
Figure 68: New Project	41
Figure 69: Drag a PForm	42
Figure 70: Create new project	42
Figure 71: Get database	43
Figure 72: Create new project	44
Figure 73: Database Project	44
Figure 74: Create new project	46
Figure 75: Form Editor	46
Figure 76: Drag PGrid	47
Figure 77: Text Property	47
Figure 78: Project Manager	
Figure 79: Events Pane	48
Figure 80: Create event	
Figure 81: Drag PVariable	49
Figure 82: Drag For Object	49
Figure 83: Set For properties	
Figure 84: Change Property	

Figure 85: Alt + Drop	51
Figure 86: Change Expr	51
Figure 87: Drop a For loop	51
Figure 88: Change Property	
Figure 89: Change Property	
Figure 90: Code Completion window	53
Figure 91: Select option	53
Figure 92: Change property	54
Figure 93: Drag RowTotal\$	54
Figure 94: Drag and Drop	55
Figure 95: Drag operation	55
Figure 96: Change Row property	56
Figure 97: Change Property	
Figure 98: CHange Value property	
Figure 99: Save Project	57
Figure 100: Output	57
Figure 101: Create new project	
Figure 102: Drag Database	59
Figure 103: Database with tables and columns	59
Figure 104: Drop database to Default Form	60
Figure 105: Smart Move	60
Figure 106: Change properties	60
Figure 107: Save project	61
Figure 108: Output	61
Figure 109: Create new project	
Figure 110: PReport	63
Figure 111: PReport in existing project	63
Figure 112: PReport with Database	64
Figure 113: Create new project	65

Figure 114: Drag PList	
Figure 115: Create Event	
Figure 116: Drag PList	67
Figure 117: Select Add in Code Complete window	
Figure 118: CHange Property	68
Figure 119: Change Expr Property	69
Figure 120: Add List	69
Figure 121: Drag with ALT key	70
Figure 122: Code Completion window	70
Figure 123: Sort list	71
Figure 124: Code Complete window	72
Figure 125: The value property	72
Figure 126: Change the value property	73
Figure 127: Add Value	73
Figure 128Save Project	74
Figure 129: Output	74
Figure 130: Create new project	75
Figure 131: OnCreate event	76
Figure 132: PValue	76
Figure 133: Value property	77
Figure 134: Change Expr property	77
Figure 135: Select ShowMessage	77
Figure 136: Change UpperCase from Code Completion	
Figure 137: Save Project	
Figure 138: Output	79
Figure 139: Create New Project	80
Figure 140: Drag ValueList	
Figure 141: Create OnCreate Event	81
Figure 142: Code Completion window	

Figure 143: Select Add	83
Figure 144: Change Value Property	
Figure 145: Change Value Property	
Figure 146: Project View	
Figure 147: Drag ForEach	
Figure 148: Drag PValueList	
Figure 149: Select Items	
Figure 150: Code completion window	
Figure 151: Select Items	
Figure 152: Select Uppercase	
Figure 153: Code Complete Window	
Figure 154: Select PrintLine	
Figure 155: Change Value property	
Figure 156Change the Calue property	90
Figure 157: Save project	90
Figure 158: Output	91
Figure 159: Create new project	
Figure 160: Delete Object	
Figure 161: Drag PObject	
Figure 162: Rename	
Figure 163: Drag PProperty to Project Manager	94
Figure 164: Change PProperty Name	94
Figure 165: Drag PMethod and rename it	94
Figure 166: Create Class	95
Figure 167: Create New project	96
Figure 168: Drag Psurface	97
Figure 169: Surface Editor	97
Figure 170: Drag PSprite	
Figure 171: Select Surface Property	

Figure 172: The events tab	99
Figure 173: Code complete	
Figure 174: Change Property	
Figure 175: Create New Project	101
Figure 176: Double click	
Figure 177: Place components in the form	
Figure 178: Project manager	
Figure 179: Drag PVariable	
Figure 180: Drag with ALT key pressed	104
Figure 181: For loop	104
Figure 182: Drag PVariable on OnCreate	105
Figure 183: Drag to OnClick	105
Figure 184: Drag variable to start property	105
Figure 185: Drag Variable 1\$ to Finish	106
Figure 186: Create New Variable	106
Figure 187: Change Expr property	106
Figure 188: Drag variable on For Loop	107
Figure 189: Change Expr property	107
Figure 190: Code complete window	
Figure 191: Change Value Property	
Figure 192: Save Project	
Figure 193: Output	109
Figure 194: Create new project	110
Figure 195: Double click on DefaultForm	111
Figure 196: Component layout	111
Figure 197: Project manager	112
Figure 198: Drag PVariable to OnClick	112
Figure 199: Drop Variable1\$ to OnClick	112
Figure 200: Drag PEdit1 object to Expr property of Variable 1\$	113

Figure 201: Drag While to Project Manager	
Figure 202: Change the exper property	113
Figure 203: Drag PVariable to Project Manager	114
Figure 204: Drag Variable	114
Figure 205: Change Expr	114
Figure 206: Drag Variable1\$ with ALT key	115
Figure 207: Change Expr property	115
Figure 208: Drag PLabel	115
Figure 209: Drag Variable2\$ to expr property	116
Figure 210: Save project	116
Figure 211: Output	117
Figure 212: Create new project	118
Figure 213: Create OnCreate object	119
Figure 214: Drag PValueList on OnCreate event	119
Figure 215: Fill items	119
Figure 216: Drag ForEach object	
Figure 217: Select Items	
Figure 218: VarObject	121
Figure 219: Select PValue	121
Figure 220: Drag PValue object to Output Property	121
Figure 221: Select Show	
Figure 222: Save project	
Figure 223: Output	
Figure 224: Create new project	
Figure 225: Double click	
Figure 226: Place PEdit	
Figure 227: Place Components on Desktop form	126
Figure 228: Place PLabel on Desktop Form	126
Figure 229: Project Manager	127

Figure 230: OnClick event	
Figure 231: Drag PVariable	
Figure 232: Alt+drag on OnClick	
Figure 233: Drag PEdit to expr	
Figure 234: Create New variable	
Figure 235: Select Text option	
Figure 236: Press Ctrl+i for IF statement	
Figure 237: Change LogicExpr property	
Figure 238: Create new variable	
Figure 239: Change expr property	
Figure 240: Change Expr property	
Figure 241: Drag and drop above IF	
Figure 242: ShowMessage	
Figure 243: Change Value property	
Figure 244: Project Screenshot	
Figure 245: Save your project	
Figure 246: Output	
Figure 247: Create new project	
Figure 248: Place coponents	
Figure 249: Change Caption property	
Figure 250: Project Manager	
Figure 251: Create OnClick event	
Figure 252: Drag PVariable	
Figure 253: Create a new variable	
Figure 254: Drag PEdit to Expr property	
Figure 255: Change Expr Property	
Figure 256: Create variable	
Figure 257: Change Expr property	
Figure 258: Select Text property	

Figure 259: Press Ctrl+I to create IF statement	
Figure 260: Change LogicExpr property	
Figure 261: ShowMessage	
Figure 262: Change Value property	
Figure 263: Use ShowMessage	
Figure 264: Change value property	
Figure 265: Save Project	
Figure 266: Run Project	
Figure 267: Create new project	
Figure 268: Place Components	
Figure 269: Change Caption	
Figure 270: Create OnClick event	
Figure 271: Drag PVariable to OnClick	
Figure 272: Drag variable	
Figure 273: Drag PEdit to expr property	
Figure 274: Drag PEdit and select Text	
Figure 275: Press Ctrl+I for IF statement	
Figure 276: Set LogicExpr	
Figure 277: ShowMessage	
Figure 278: Change value property	
Figure 279: Create ElseIf Object	
Figure 280: ShowMessage	
Figure 281: Change value property	
Figure 282: Project screenshot	
Figure 283: Save Project	
Figure 284: Output	
Figure 285: Create New Project	
Figure 286: Place components	
Figure 287: Double click for OnClick	

Figure 288:	Drag PVariable	154
Figure 289:	Create Variable	154
Figure 290:	select Text in source	154
Figure 291:	Drag Plabel1 to OnCreate	155
Figure 292:	Drag Variable to Expr property	155
Figure 293:	Save Project	156
Figure 294:	Project Output	156
Figure 295:	Create New Project	157
Figure 296:	Place components	158
Figure 297:	Drag PMethod	158
Figure 298:	Change Parameters property	159
Figure 299:	ShowMessage	159
Figure 300:	ShowMessage	160
Figure 301:	Change Expr property	160
Figure 302:	Double Click for OnClick event	160
Figure 303:	Drag PMethod	161
Figure 304:	Drag PEdit objects to x,y property	161
Figure 305:	Save Project	162
Figure 306:	Output	162
Figure 307:	Drag a Pcontrol to the Project Manager	164
Figure 308:	Drag PProperty to DbNavigator	164
Figure 309:	Set the Type property	165
Figure 310:	Set the ClassName property	165
Figure 311:	Drag four PToolButton Objects	166
Figure 312:	Double click the buttons to create events	167
Figure 313:	Select a message	167
Figure 314:	Select messages for all buttons	168
Figure 315:	The property panel	169
Figure 316:	Components Panel	169

Figure 317: cre	eating a new component package	170
Figure 318: Cr	reate New Project	171
Figure 319: Pla	ace components	172
Figure 320: Vi	iew Binds option	172
Figure 321: ch	loosing components class	173
Figure 322: So	ource and Target sections	173
Figure 323: So	ourceProperty	174
Figure 324: Se	electiong a TargetObject	175
Figure 325: Ta	arget options	175
Figure 326: Ta	arget property	176
Figure 327: PE	Edit text binded with PButton object	176
Figure 328: Cr	reate new project	177
Figure 329: Dr	rag PButton to Form	178
Figure 330: Cr	reate OnClick event	178
Figure 331: Dr	rag PDirectory	178
Figure 332: Ch	nange PathName	179
Figure 333: Ch	nange Extension property	179
Figure 334: Sh	nowMessage	180
Figure 335: Co	ode Complete window	180
Figure 336: Cl	ick the arrow	181
Figure 337: Se	elect Count Property	181
Figure 338: Sa	ve Project	182
Figure 339: Ou	utput	182
Figure 340: Cr	reate New Project	183
Figure 341: Cr	reate OnClick event	184
Figure 342: Dr	rag PEdit	184
Figure 343: Dr	rag PVariable	184
Figure 344: Cr	reate Variable	185
Figure 345: Dr	rag PEdit1 to expr	185

Figure 346: Select Text	
Figure 347: Drag PFile to OnClick	
Figure 348: Drag variable to Filename	
Figure 349: Change Value property	
Figure 350: Save Project	
Figure 351: Output	
Figure 352: Create New Project	
Figure 353: Drag PButton to DefaultForm	
Figure 354: Create OnClick event	190
Figure 355: Drag PDirectory	
Figure 356: Change PathName	191
Figure 357: Change Extension Property	191
Figure 358: Hit Ctrl+Space for Code Complete window	
Figure 359: Select Files	
Figure 360: Select DeleteFiles	193
Figure 361: Project screenshot	
Figure 362: Save Project	194
Figure 363: Output	194
Figure 364: Drag and drop files and mark PResource	