

Table of Contents

Table of Contents.....	1
License Agreement.....	6
Trademarks and Copyright.....	7
About ArianeSoft.....	8
About the Pocket Programming Language (PPL).....	8
About PIDE.....	8
Values and Benefits.....	8
About This Manual.....	9
Organization.....	9
Conventions.....	9
Installation.....	10
Requirements.....	10
Understanding the Interface.....	11
Menu Bar.....	12
Action Buttons.....	12
Components.....	13
Work Area.....	14
Project Manager.....	15
Code View.....	17
Properties/Events.....	19
Properties.....	20
Property Editors.....	21
Code Navigator.....	41
Debugging Console.....	42

Tips on PIDE.....	44
Creating and Managing Projects.....	46
Creating a New Project.....	46
Opening a File into the Work Area.....	47
Saving a Project.....	49
Close an Open Project.....	50
Working with Code.....	51
About the Pocket Programming Language (PPL).....	51
Accessing the Code View.....	51
Commenting code.....	52
Formatting Code.....	52
Adding Objects.....	53
To add an application object to the code,.....	56
Collapsing/Expanding Nodes.....	56
Adding Color to Code.....	56
To add a RGB color to your code,.....	56
Adding Characters.....	58
Creating Notes.....	59
Edit a Note.....	61
Procedure List.....	61
Components.....	62
Navigating Code.....	62
The Design Environment.....	64
Drag and Drop Components.....	64
Auto Create.....	65
Sub-Class.....	65

Expand Class.....	66
Edit.....	66
Edit Code.....	66
Application Object Properties and Events.....	66
Properties Panel.....	67
Events Panel.....	69
Layout.....	71
Add Elements.....	73
Position Elements.....	74
Edit Control Code.....	75
Non Visual Elements.....	75
Events and Properties.....	76
Creating An Event.....	77
Adding Objects within Events.....	80
Testing Projects.....	83
Design-Time Testing.....	83
Breakpoints.....	83
Code Stepping.....	85
Run-time Testing.....	86
Profiling.....	87
Watches.....	87
Debug Console.....	88
Debug Tab.....	88
Errors Tab.....	88
Warnings Tab.....	89
Compiler Messages Tab.....	89

Component Library.....	90
What is a Component?.....	95
Class.....	95
Adding Custom Components.....	95
Managing the Component Library.....	96
Creating Your Own Components.....	97
XML Definition Files.....	97
Generate the Component Package.....	98
Creating classes from current project in components panel.....	99
Creating A SQLite Database In PIDE.....	107
Visual Query Editor.....	115
Select SQL Query and Visual Query Editor.....	121
Insert Query with Visual Query Editor.....	128
Update Query with Visual Query Editor.....	131
Deleting Operation with the help of Visual Query Editor.....	134
Help File Editor	136
Appearance in Help file.....	138
The Help Editor Interface.....	139
Editing .hlp files in PIDE2.....	140
Creating topics from files.....	141
Code Templates In Code Editor.....	144
Using the Default Code Template.....	144
Modifying a .ctp or code template file.....	147
Creating and using a Code Template file.....	149
The PIDE Shell.....	151
Batch Operations In PIDE.....	157

Inserting Files With PIDE.....160

Package Files With PIDE.....162

Code Collection Editor.....167

Object Binding With PIDE.....170

Packaging for Deployment.....175

Additional Support.....176

Appendix A: Available Application Objects.....177

Appendix B: Figures.....179

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 expressed 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, EXPRESSED OR IMPLIED, WITH RESPECT TO THIS DOCUMENT OR ITS CONTENTS, AND HEREBY EXPRESSLY 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 © 2008 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.

About ArianeSoft

ArianeSoft Inc. was formed in early 2006 by Alain Deschenes, a senior analyst and programmer working primarily in electronic data interchange (EDI). For many years, Alain worked on developing different programming languages. Leveraging his experience in the field of compilers and interpreters, he finally created what he knew would be embraced and welcomed by developers worldwide—PPL. Exhaustive

About the Pocket Programming Language (PPL)

Development on PPL started in early 2004. The first version, 1.0, was officially released in September of 2006. Since then, PPL has had great success through the Windows and Windows Mobile community. Many games for the Windows Mobile platform have been developed on the PPL platform and sold around the world.

About PIDE

PIDE (Pocket Programming Language Integrated Design Environment) provides a set of tools and workspace to create PPL-based projects designed to run on Windows Mobile, Windows Vista, and Windows XP operating systems.

Values and Benefits

PIDE provides customers with a number of valuable benefits:

- Run-time Execution—run your projects as you are building them, allowing you to see design-time modifications in real-time execution
- Component Library—a built-in component library allows you to quickly add application functionality to the design environment, generating applicable code behind the scenes
- IDE—PIDE leverages the conventions of other IDEs (i.e., VisualStudio, Eclipse, etc.) to create a comfortable and familiar work environment.

About This Manual

This manual will provide all the information you need to make the most out of PIDE.

Organization

This manual is organized accordingly:

- Installation of the software,
- Understanding the interface,
- Creating and Managing Projects,
- Working with Code,
- The Design Environment,
- Testing Projects,
- The Component Library, and
- Packaging for Deployment.

To find what you are looking for quickly, refer to the table of contents and search for a topic based upon a menu item.

Conventions

This guide uses the following conventions to highlight certain words and phrases that differentiate their meaning visually.

Table 1: Conventions used in this guide

This style...	Indicates	For example:
<i>italics</i>	Notes regarding special information about a feature	<i>Note: You can also...</i>
bold	A feature of the interface or functionality within PIDE	To open the project, click on the Open button.

Installation

PIDE installation is similar to other windows-based software. To start the installation process, double click on the PIDE setup icon. This will launch the installer.

[screen shot]

[AS: Will document completely when the build is stabilized and includes an installer]

Requirements

PIDE has the following system requirements:

- Windows 2000, Windows XP, Windows Vista compatible computer
- 512mb RAM
- 32mb disk space
- Internet connection

Understanding the Interface

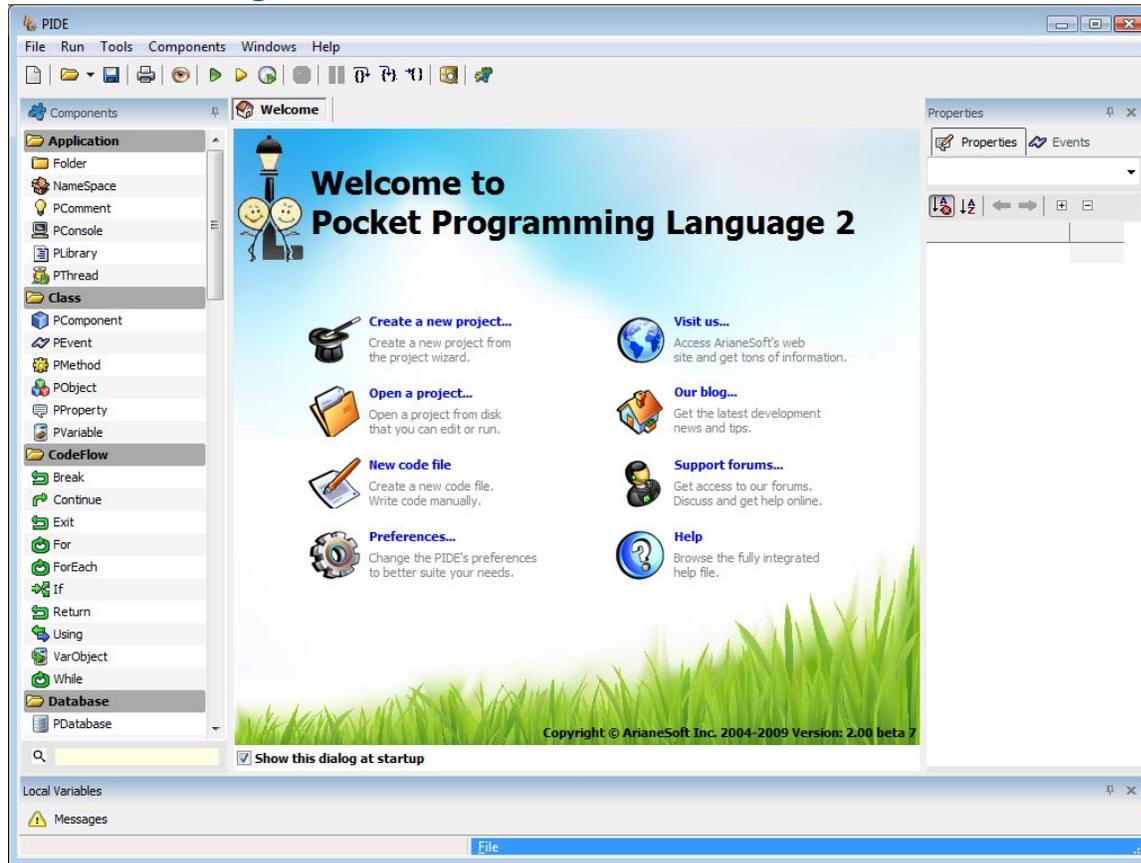


Figure 1: Main Interface

The PIDE interface is divided into six primary areas:

- Menu bar,
- Action buttons (project and code),
- Components,
- Work area,
- Properties/Events,
- Code Navigator, and
- Debugging Console.

Menu Bar

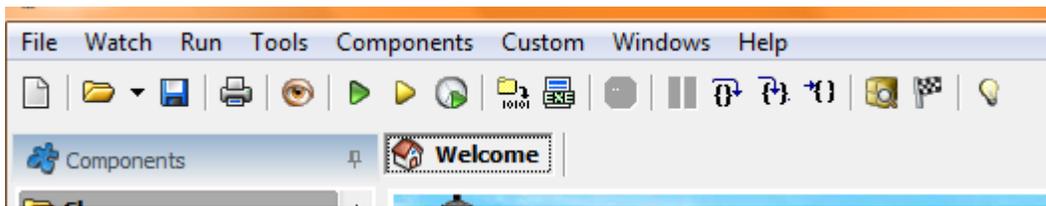


Figure 2: Menu Bar

The menu bar provides quick access to many of PIDE's features. The menus are categorized accordingly:

- File,
- Edit,
- Search,
- Insert,
- View,
- Project,
- Run,
- Tools,
- Components,
- Custom,
- Windows, and
- Help.

Action Buttons



Figure 3: Action Buttons

The action buttons provide quick access to key project- and code-based activities. These actions buttons will change depending upon the current view of the work area—project management or code view.

Components



Figure 4: Components

The components window, located on the left-hand side of the interface, provides quick and easy access to all of PIDE's components.

Work Area



Figure 5: Work area

The work area, located in the central part of the interface, provides access to the project's code as well as the design-time workspace (a visual representation of objects with the project and their layout within the project's interface).

There are two views within the Work Area:

- Project Manager, and
- Code View.

As you open application objects into code view, they will appear as buttons above the Project Action Button bar to allow you to easily navigate between code and the project

manager.

Project Manager

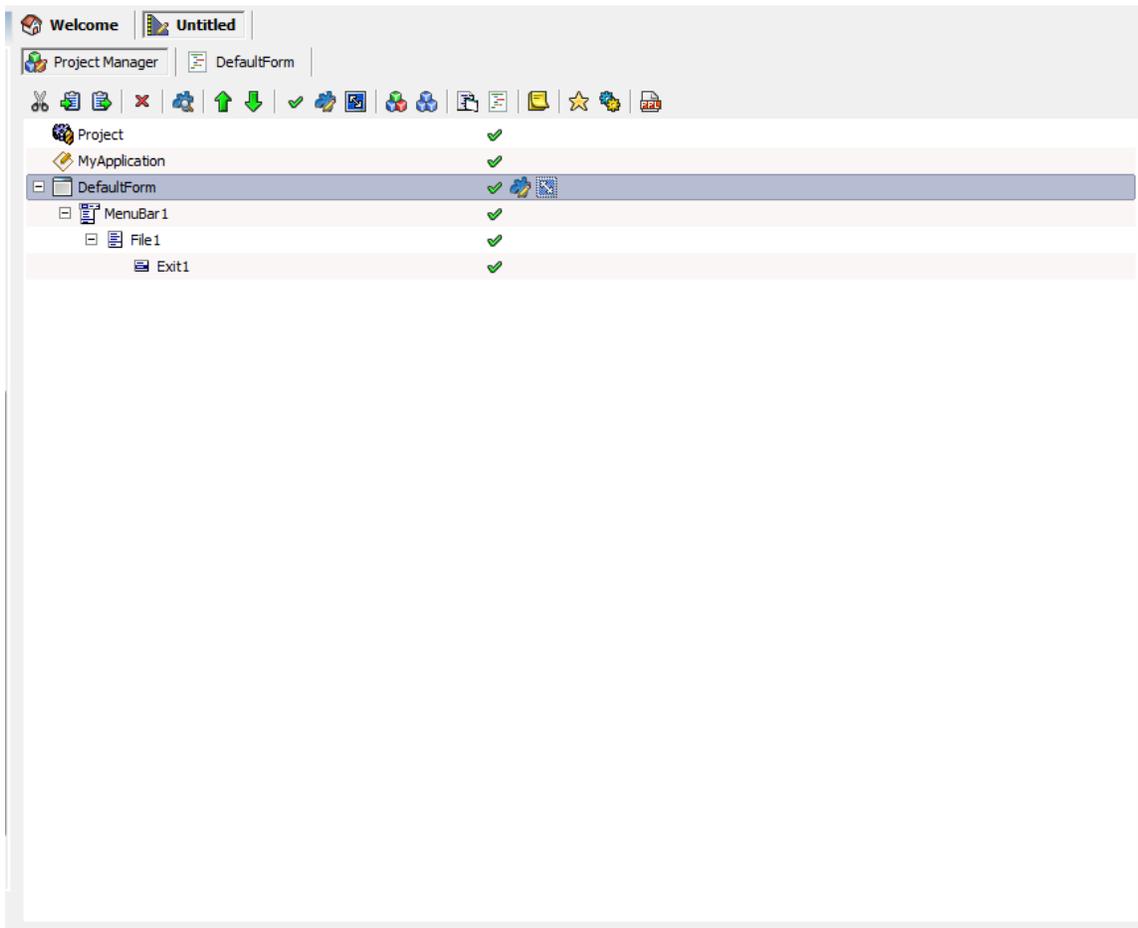


Figure 6: Project manager

The project manager view shows all of the objects within the project including:

- Project Name,
- Application Name,
- Application Objects (expandable to show sub objects), and
- Notes (expandable to show all notes)

To return to the project manager view, click on the **Project Manager Button** above the Project Action Button bar.

When in project manager view, the Action Buttons will change to reflect functionality related to working with projects. These buttons include:

Button	Function
Cut	Cut the selected item out of the project
Copy	Copy the selected item
Paste	Paste the most recently cut project item into the project at its current location
Delete	Delete the selected item from the project
Find object	Find an object within the project. Clicking this opens a secondary window.
Move up	Move the selected item up; only applicable to Notes and Application objects
Move down	Move the selected item down; only applicable to Notes and Application objects
Auto create	The object is automatically created and initialized. If you do not select it, you will need to create the object (i.e., instance a new copy of the object in memory such as <code>MyForm\$ = new PForm</code>)
Sub class	This creates a sub class that is added to the palette so that you can re-use it by dragging it into the visual editor
Expand class	Expand on a base class by adding code. All instances of the class used throughout your project will be automatically updated with the additions. Furthermore, adding the expanded class will include your additions.
Edit	Edit the selected object
Edit source code	Open the selected object into the code view window
Add note	Add a note to the selected object; this will open the Note Window
View project source	View the source code for the entire project

	as a hierarchical tree of class and sub classes.
--	--

Note: you can also access the functionality of the Project Action Buttons by selecting an item and right-clicking on it to display the context-sensitive menu.

Code View

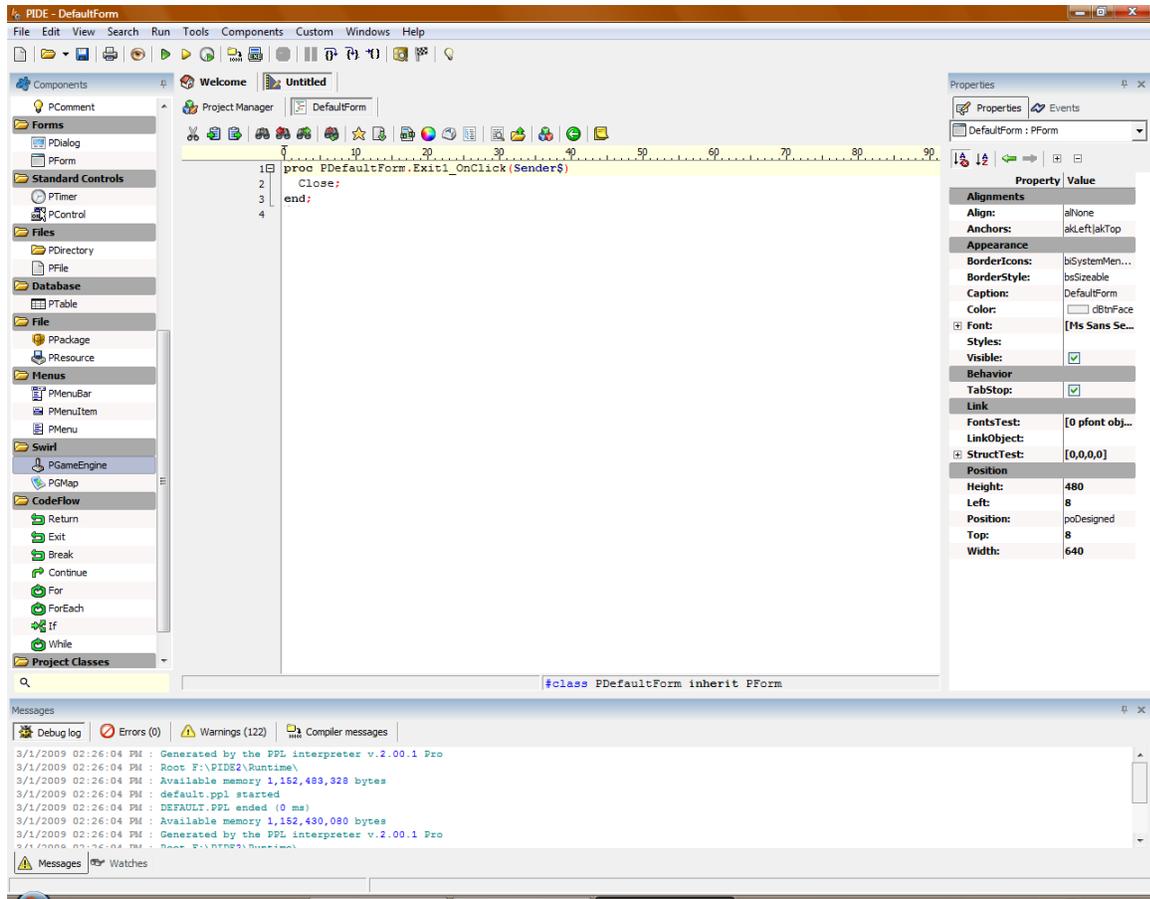


Figure 7: Code View

The code view shows the application (and all of its objects) as they appear in the Pocket Programming Language.

To access the code view, select an application object and then click on the Edit Source Code button in the Project Action button bar.

When in code view, the Action Buttons will change to reflect functionality related to working with the code. These buttons include:

Button	Function
--------	----------

Cut	Cut the selected item out of the project
Copy	Copy the selected item
Paste	Paste the most recently cut project item into the project at its current location
Find Text	Find text within the code
Find Previous Occurrence	Find the previous occurrence of a text search within the code
Find Next Occurrence	Find the next occurrence of a text search within the code
Replace Text	Replace text within the code
Comment Selected Code	Add a comment at the current point of the cursor within the code
Insert RGB Color	Add the selected RGB color to the code
Characters	Insert special characters into the code (at the point of the cursor)
Format Code	Automatically format the code (i.e., indenting)
Find Definition	Find the definition for the word on which the cursor is currently placed. The search is carried out in the project and the default and included library files.
Toggle Navigator	Open and close the code navigator
Collapse/Expand	Collapse and expand a class or sub-class
Add Note	Add a note to the code

Properties/Events

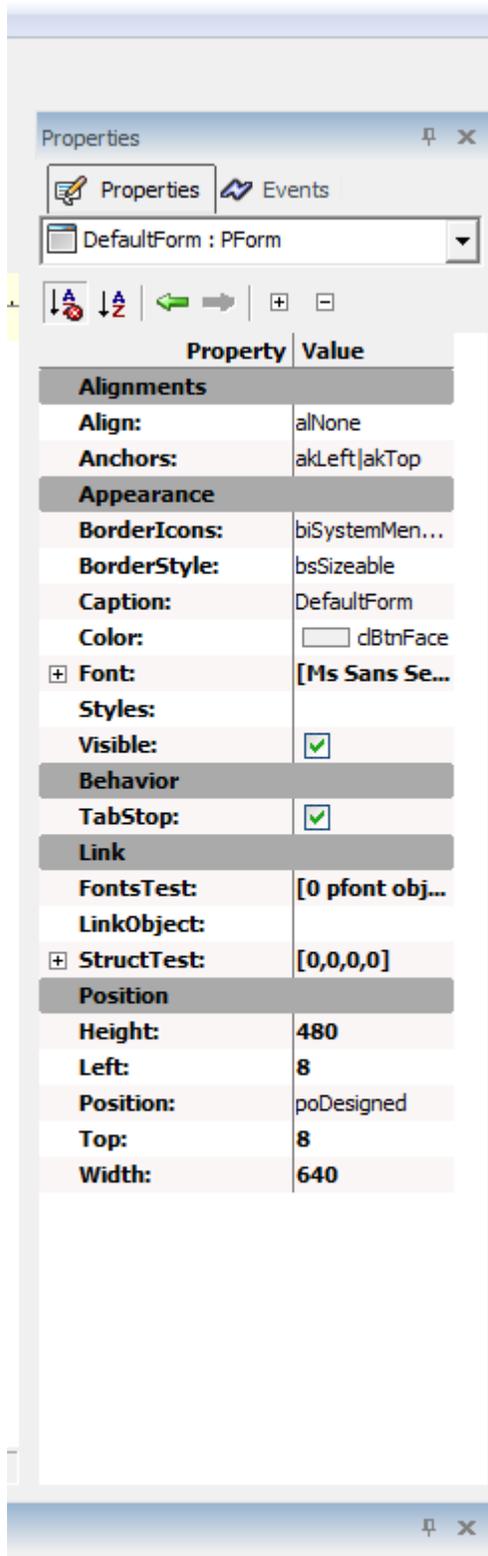


Figure 8: Properties/ Events tab

The Properties/Events window, located on the right-hand side of the interface, provides quick and easy access to a selected object's properties or events associated with that object.

Properties

An object's properties include parameters that determine how the object behaves within the code. Like any object that has properties, objects created in PIDE2 with the help of visual programming also have properties that can be used to enhance the functionality of those objects. The table given below contains a list of properties that are linked with visual objects and their explanation:

Property Name	Description
Expr	Specifies an expression
Result	The result property is used to assign the result of an object to a variable.
Owner	This property specifies the owner or the parent of the object.
Operation	Specifies the operation that will be performed by an object.
Source	This property determines the source object.
Logic	This property allows a user to implement logical statements like If or While to the object.
LogicAction	This property specifies the condition that the logic statements will check.
LogicExpr	This property specifies the expression to which logic applies to.
Action	This property determines the action that will be performed by the object.
Operator	This property specifies the operator that will be used to check the condition.

While properties give a lot of power in the hands of the programmer, the **Edit as Expression** feature of PIDE2 provides superior control over properties and their abilities. While working with a property, press **Ctrl+F5** key or use **Right click -> Edit as Expression** to open a code editor window for the property where you will be able to modify the code of the property by using PPL code. Once completed, this code will be executed with the property in the runtime.

*Note: Once a property is tweaked with the help of **Edit as Expression** feature, it is surrounded*

by brackets to specify that the property has been **edited** as an expression.

Property Editors

For increased customizability and flexibility in selecting properties, PIDE offers its users with property editors of different kinds. By using the various property editors available in PIDE, users will not only be able to fine tune their objects and their behavior, they will also be able to decide how their objects will look like. Given below are the various property editors and their description:

Matrix Editor – The matrix editor works with the **PMatrix Class** that follows the matrix variable. The matrix editor can be used to create or manipulate a matrix by including or excluding rows or columns from it. Users can gain easy access to the matrix editor by double clicking the **PMatrix Object**.

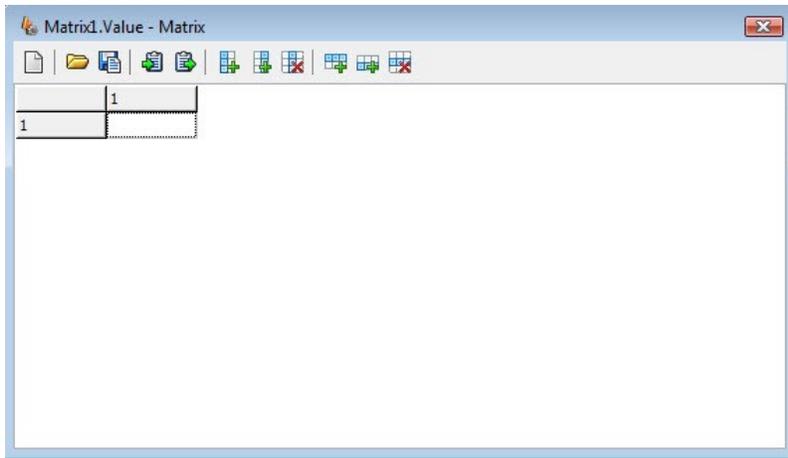


Figure 9: Matrix Property Editor

Apart from the usual **New, Open, Save, Copy** and **Paste** buttons; Matrix editor also consists of **Add columns before, Add columns after, Delete rows, Delete columns, Add row above** and **Add row below** buttons. As their name suggests, all the buttons are used to perform the things they are named after and do not need any previous experience of creating matrix variables with visual programming.

Color Editor – The color editor in PIDE is just like a color selection tool that can be used to impart a color to the object it is applied on. Users can gain easy access to the color editor by double clicking the **Color Property** in the **Properties Panel**. In a color editor, after selecting a color, press the **OK button** to conform changes.

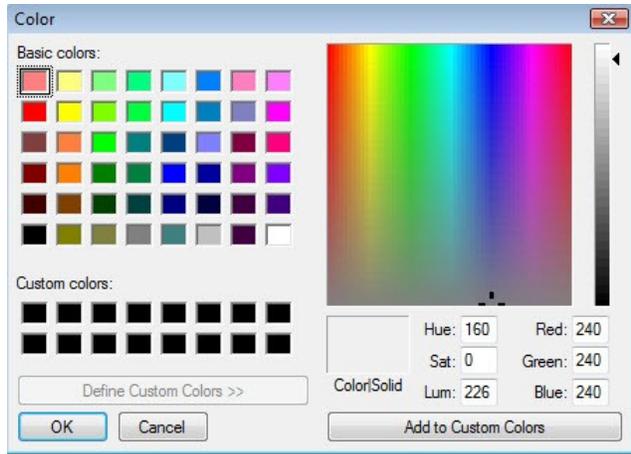


Figure 10: Color Property Editor

Surface Editor – Images are important for any computer program and the ability to include images in the software environment to provide an easy to use interface is what makes or breaks a computer software. PIDE is an excellent platform that is not only easy to use but also allows its users to create interactive programs with the use of images as well as sounds. The PImageList Object is used in PIDE to contain a list of images or surface images that can be used in an application. Just by including images in an ImageList, users can easily incorporate images for buttons and other elements of an application.

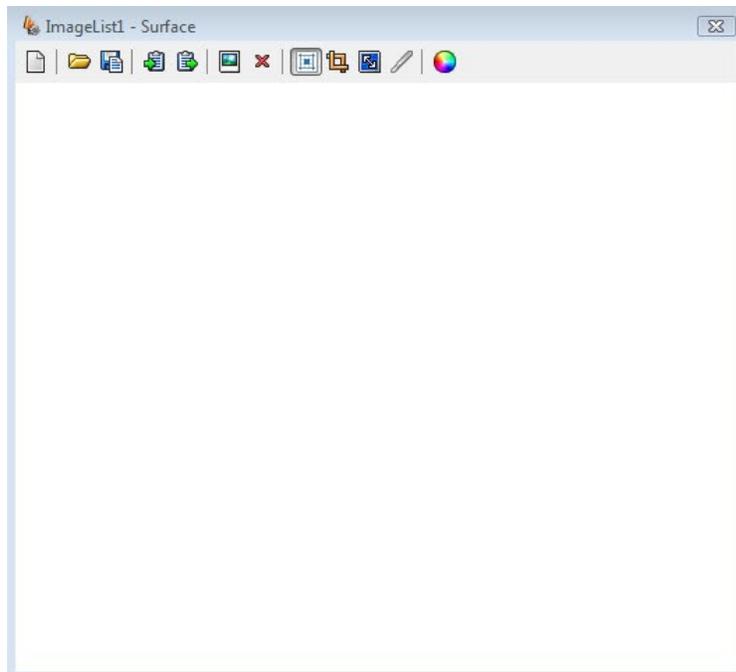


Figure 11: Surface property editor

With surface editor, users can very easily create large images by combining multiple bitmap

images. Very useable in creating sprite animations, surface editor includes various buttons for imparting complete control over the images. Buttons used in the surface editor are:

 New image list

 Open new image list

 Save current image list

 Copy item

 Paste item

 Add image

 Delete image

 Center image

 Crop image

 Stretch image

 Split image

 Change color mask button.

Font Property Editor – Used for imparting a specific font to an object, the font selector is an excellent tool that allows a user to visually select the different parameters of the font of an object. Users can gain easy access to the Font selector property by double clicking the font property.

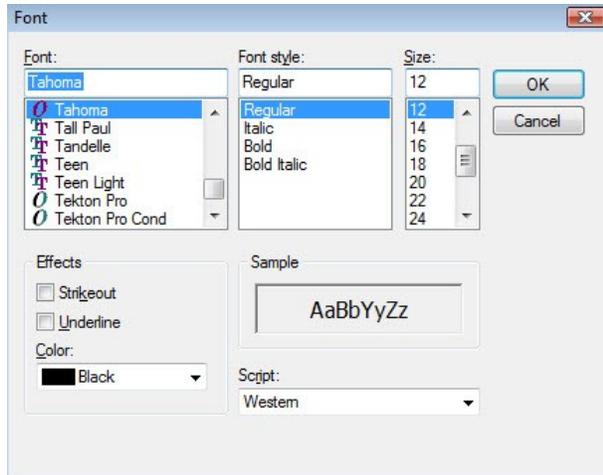
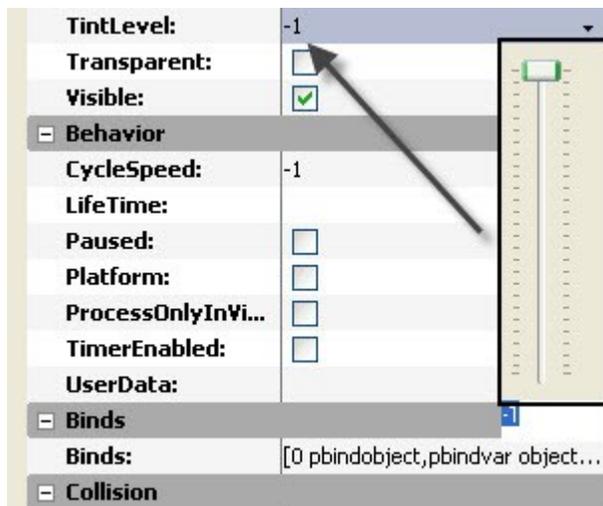


Figure 12: Font property editor

The Font selector or font editor window consists of various selections that can lead to many text transformations. While the **Font section** can be used to select the font, **Font style** is used to select the appearance of the font. The **Size** and **Effects** section determine the emphasis on a text and the **Color** section determines the color of a text. Using the various scripts in the **Script section**, users can select the outlook of their text. The sample section gives a preview of the text that is selected. After selecting the required font setup, click the **OK button** to accept the changes.

Range Selector – The range selector in PIDE2 allows a user to select the range instead of specifying a number. Used for specifying the range of a property, the range selector is an interesting tool that would allow a user to select the level of tint not by inputting the value but by selecting the appropriate level by scrolling the scroll button in the range selector.



Game Level Editor –

The game level editor is a simple to use yet very advanced utility of PIDE2. Allowing users to create games with no or minimum coding, the game designer in PIDE2 assists its users in creating games without any hassles just by dragging and dropping objects in the game form. Selecting a **Desktop Game** project in the **New Project Window** is the first step users will have to take for creating games in PIDE2. The default **Project Manager Window** of **Desktop Game** already consists of most of the objects that are required by a user but other objects will also need to be added. Some of the objects that are required in the Project Manager for creating a game are listed in the **Components Pane** under the **Swril** category, these are:

- **PGameForm** – The PGameForm class inherits its characteristics from the PForm class and is used to hold the contents of a game.

PGameForm Properties:

Property	Description
Name	This property specifies the name if the game form.
ClassName	This property lists the name of class this object belongs to.
Align	Used for aligning the object .
Anchors	Used to align an anchor in the desired location.
BorderIcons	Specifies the icons on the border i.e. close, minimize and maximize .
BorderStyle	Lists the border style followed by the PGameForm window.
Caption	Lists the Title of the PGameForm window.
Color	Sets the background color of PGameForm window.
Enabled	Specifies whether the object is enabled or not.
Font	Sets the font and the related properties of PGameForm.
ShowCursor	Decides whether to show cursor or not.
Visible	Specifies whether MyGameForm is visible or not.
FPSSpeed	Specifies the frame per second.
Speed	Specifies the number of game loops to process per second.
TabStop	Allows a user to switch windows while keeping stopping the

	execution of the game.
Binds	Specifies the object binding options. Double clicking this property opens the object binding property editor.
CycleMode	Allows a user to change the way a loop is processed in a game.
DrawMode	Allows a user to specify how a screen is updated at each game loop.
FullScreen	Whether game will display in full screen mode or not.
GameHeight	The height of the game window.
Gamewidth	The width of the game window.
InfoColor	Color of information
IsometricDisplay	Switch on the isometric display or not.
Orientation	Specifies whether the orientation of the screen.
OriginsX	This option specifies the start X coordinate of the game engine.
OriginsY	This option specifies the start Y coordinate of the game engine.
ShowInfo	Whether to show information or not.
UpdateMode	The mode to update the game display.
InputOrientation	This property specifies how the input methods will be oriented.
CurrentMap	Determines the current map that this game form belongs to.
MainMap	Specifies the main map of on which current game form will depend on.
MainFriction	This is a property that imitates the physics and determines the friction that needs to be applied to a PPhysicsSprite object.
MainGravity	This is a property that imitates the physics and determines the gravity that needs to be applied to a PPhysicsSprite object.

Height	This property determines the height position of the game form.
Left	This property determines the left position of the game form.
Position	Determines the position of the game form relative to the screen.
Top	This property determines the top position of the game form.
Width	This property determines the width of the game form.
AutoHideScrollBars	This property specifies whether to automatically hide scroll bars or not.
HScrollBar	Whether to display horizontal scroll bar or not.
ScrollIncrement	This property specifies the scroll increment by which the form is scrolled.
VScrollBar	Whether to display vertical scroll bar or not.

- **PGMap** – PGMap is a class that inherits the PObject class and represents the main screen of the game.

PGMap Properties:

Property	Description
BackMusic	This property determines the music to be played in the game background.
BackSurface	This property determines the image at the background of the game map.
Color	This property specifies the color of game map.
Binds	This property specifies the object binding properties.

- **PMusic** – The PMusic class is used to apply music to the game.

PMusic Properties:

Property	Description
Filename	This property specifies the filename of the music file.
PackageFile	Use this property specifies the package file that contains the music file.
Resource	This property specifies the resource that is used in the project.
Frequency	Use this property to specify the frequency of sound the music will play at.
Loop	Thy is used to specify whether the music should loop continuously or not.
Pan	The pan property can be used to increase or decrease the panning effect on the sound.
Volume	Use the volume property to set increase or decrease the default volume.

- **PSprite** – Psprite represents an object on the game map and is used to perform actions in the game.

PSprite Properties:

Property	Description
AltAlpha	This property is used to specify the alternate alpha settings.
AltIndex	This property is used to specify the alternate index.
AltRadius	This property is used to specify the alternate radius.
AnimLoopCount	Set this property for specifying the animation loop counts.
AnimSpeed	Set this property for specifying the animation speed.
AnimSpeedVel	This property can be used to set the animation speed based on the velocity of the sprite.
FirstIndex	Used to specify the first index of the animation.
HideWhenAnimDone	Used to specify whether to hide the animation when it is finished.

LastIndex	Used to determine the last index of the animation.
Alpha	This property can be used to provide the alpha settings.
Angle	Used to specify the angle of appearance of the sprite.
Color	Use this property to set the color of the sprite.
GreyScale	Specify whether sprite should be displayed in grayscale or not.
IsoTile	Use this option to set whether this sprite is tiled using isometric positioning or not.
MirrorX	Use this option to set if the sprite has is mirrored at X axis.
MirrorY	Use this option to set if the sprite has is mirrored at Y axis.
Negative	Use this option to set the negative of this sprite.
Parent	Used to specify the parent of the sprite object.
ParentClip	Specifies whether this sprite has a parent clip or not
Tint	Sets the tint of the sprite
TintLevel	Used to set the tint level of the sprite
Transparent	Used to specify if this sprite will transparent or not
Visible	Used to make this sprite visible or invisible
CycleSpeed	This property can be used to set the loop speed
LifeTime	This property is used to set the life time of a sprite
Paused	Use this property for setting whether the sprite will start paused or not.
Platform	This property determines if the sprite uses a platform or not.
ProcessOnlyInView	For processing the sprite only in view, use this option.
AccurateCheck	Makes sure the sprite does not collide with other objects.
BorderCheck	This property checks if the sprite collides with the screen borders.
BottomCollision	Check to see if the collision happens at the bottom of the

	sprite.
CallCollide	Use this property on two sprites to call the OnCollide property
CancelVelX	Property to cancel the velocity X of a sprite in case of a collision.
CancelVelY	Property to cancel the velocity Y of a sprite in case of a collision.
CheckCollide	Check if collision happened or not.
CollideBottom	Set property to trigger a collide event if collision occurs to the bottom of sprite.
CollideLeft	Set property to trigger a collide event if collision occurs to the left of sprite.
CollideRight	Set property to trigger a collide event if collision occurs to the right of sprite.
CollideTop	Set property to trigger a collide event if collision occurs to the top of sprite.
CollideWith	Use this property to specify the other sprites to collide with.
Id	Used to set the ID of a sprite.
LeftCollision	Used to check the occurrence of collision to the left of a sprite.
OvalShape	Used to set the shape of collision objects.
PixelCheck	Use this property to perform pixel check.
RightCollision	Used to check the occurrence of collision to the right of a sprite.
TopCollision	Used to check the occurrence of collision to the top of a sprite.
AccelerationX	This property can be used to set the acceleration of a sprite in x axis.
AccelerationY	This property can be used to set the acceleration of a sprite in y axis.

Velocity	Used to set the velocity of a sprite movement.
VelocityLimitHigh	Use this property to set the upper limit of the velocity
VelocityLimitLow	Use this property to set the lower limit of the velocity.
VelocityX	Set the velocity at the X axis for the sprite.
VelocityY	Set the velocity at the Y axis for the sprite.
IsoHeight	Set the isometric height.
Light	Use this property to set the intensity of light on the sprite.
LightRadius	Use this property to set the light radius on the sprite.
AutoOffsetX	This property allows a user to automatically set the Offset of X axis for the sprite
AutoOffsetY	This property allows a user to automatically set the Offset of Y axis for the sprite
OffsetX	This property allows a user to set the Offset of X axis for the sprite
OffsetY	This property allows a user to set the Offset of Y axis for the sprite
AutoScrollX	Use this property to set auto scroll options for the X axis
AutoScrollY	Use this property to set auto scroll options for the Y axis
FixedX	Check to set a fixed scroll X position
FixedY	Check to set a fixed Y position
Height	Set the height of the sprite
Left	Position the left side of the sprite
Top	Position the top side of the sprite
Width	Set the width of the sprite
ZOrder	Use this property to set the ZOrder of the sprite
AlphaSurface	Set the alpha settings for the surface of the sprite.
Index	Set the index for sprite.

Surface	This property specifies the surface object, this sprite will call its bitmap from.
TileX	Use this property to set the tiling for X axis.
TileY	Use this property to set the tiling for Y axis.

- **PPhysicsSprite** – The PPhysicsSprite class is the same as PSprite class but adds the physics properties to it.

PPhysicsSprite Properties:

Property	Description
Bounce	Check to set the bounce in the sprite.
Elasticity	Set the elasticity option for the sprite.
Friction	Set the amount of friction applied on the sprite.
Gravity	Set the gravity settings of the sprite.
Mass	Set the mass of a sprite object.
NoFriction	Check if the sprite will display friction or not.

- **PSound** – The PSound class inherits the properties of PSoundCollection class and is used to apply small sounds to the game.
- **PSoundEngine** – This class inherits the PObject class and specifies the sound settings in the game.

PSoundEngine Properties:

Property	Description
BitsPerSecond	This property is used to tweak the Bite per second setting of the sound.
Stereo	This property is used to determine if the sound played is stereo or mono.

- **PSurface** – The PSurface class inherits the PCollectionItem class and is used to set the surface property of a game map.

PSurface Properties:

Property	Description
ColorMask	This property specifies the color mask on the surface.
Filename	Used to specify the filename of the source.
FrameCount	This property determines the frame count of the surface object.

- **PWorldSprite** – The PWorldSprite class inherits the properties of the PSprite class and is used to provide faster drawing by sacrificing special effects that are made possible with PSprite and PPhysicsSprite objects.

Given below is an example that would show you how to create a simple game without much of a coding:

- Open PIDE and create a new **Desktop Game** project.

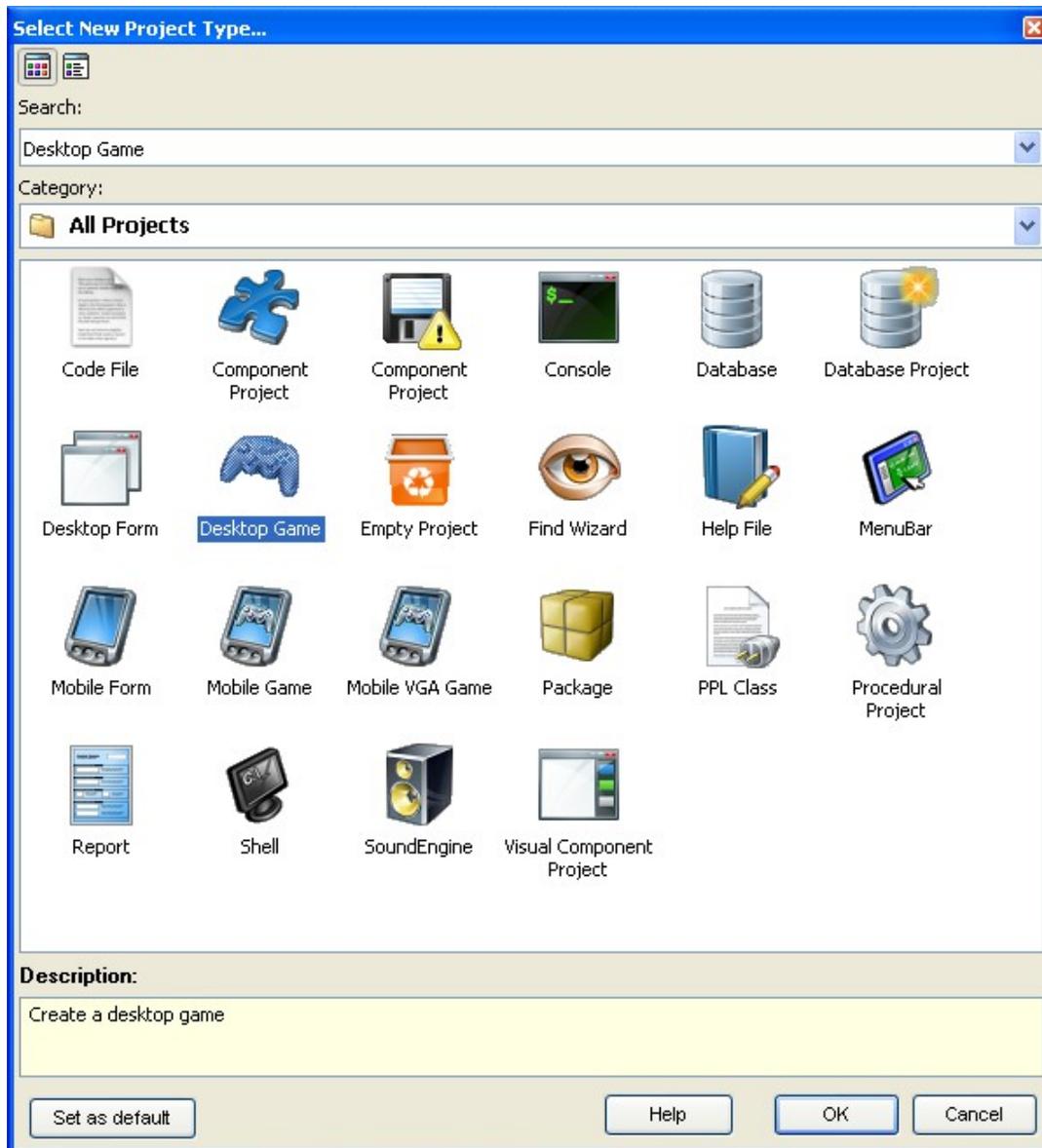


Figure 13: Create new desktop game

- Now, in the **Project manager** view of **Desktop Game** project, drag a **PSurface Object** to the **Surfaces** folder.

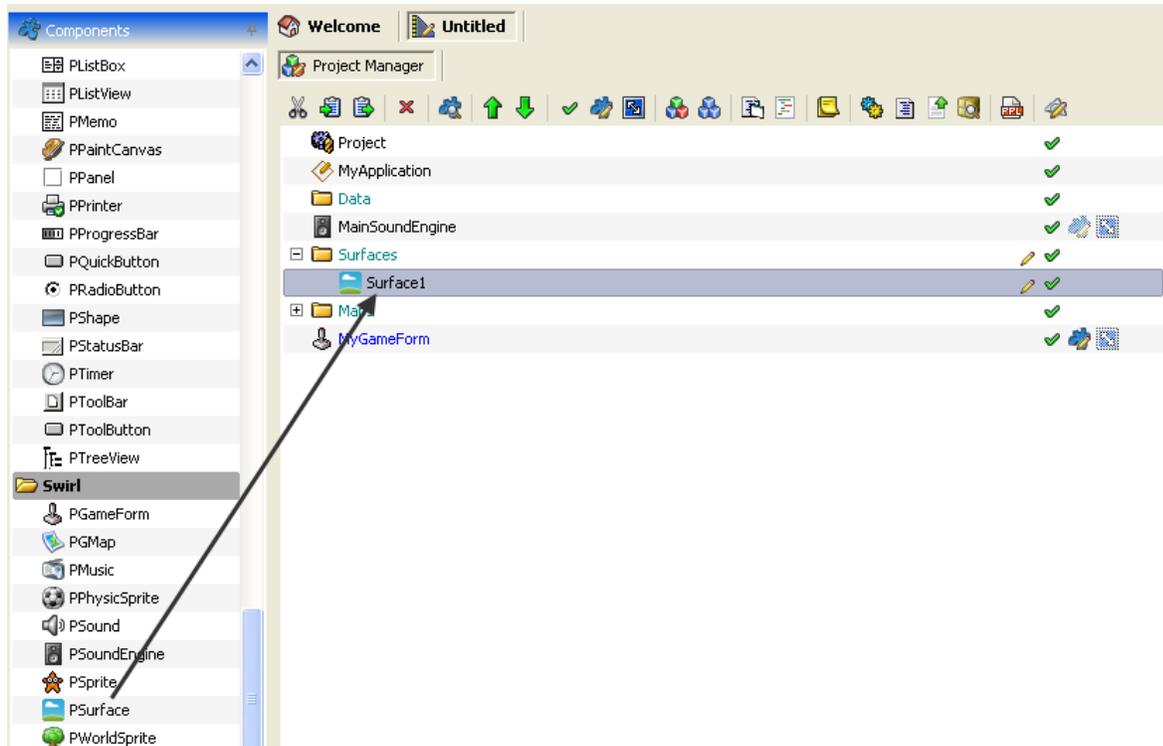


Figure 14: Drag PSurface to project manager

- Double click the **PSurface** editor to open **Surface Editor** and choose an image to be included in the bitmap. Close the **Surface editor** to proceed.

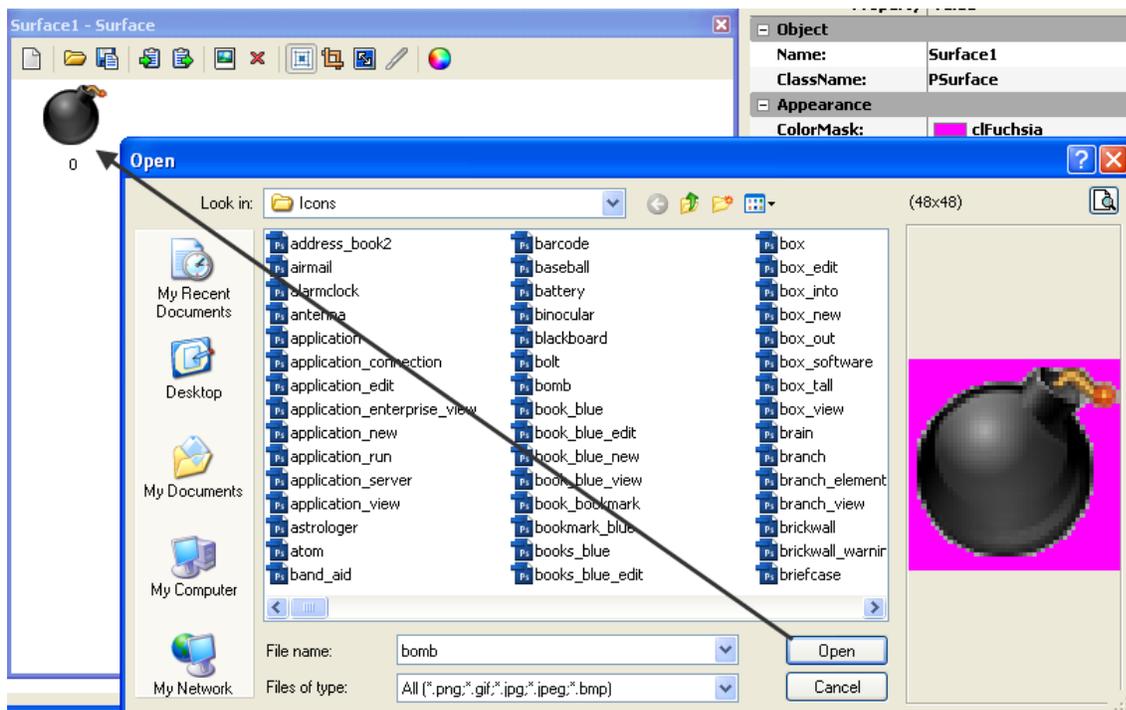


Figure 15: Open an image

- Double click the **PGameMap Object** and then double click on the **PSprite** class in the components pane under **Swirl** category to include a Sprite object in the **PGameMap**.

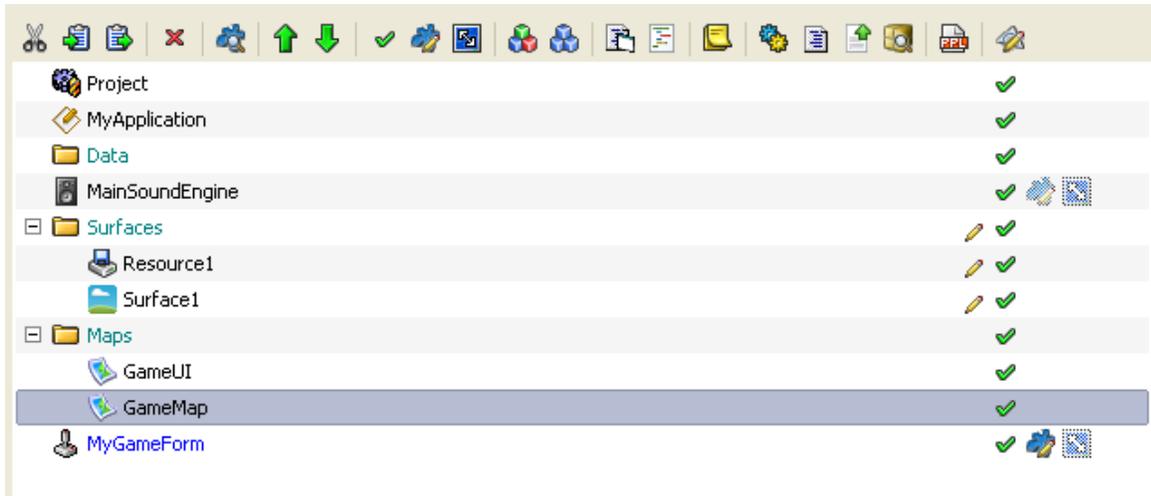


Figure 16: Double click GameMap

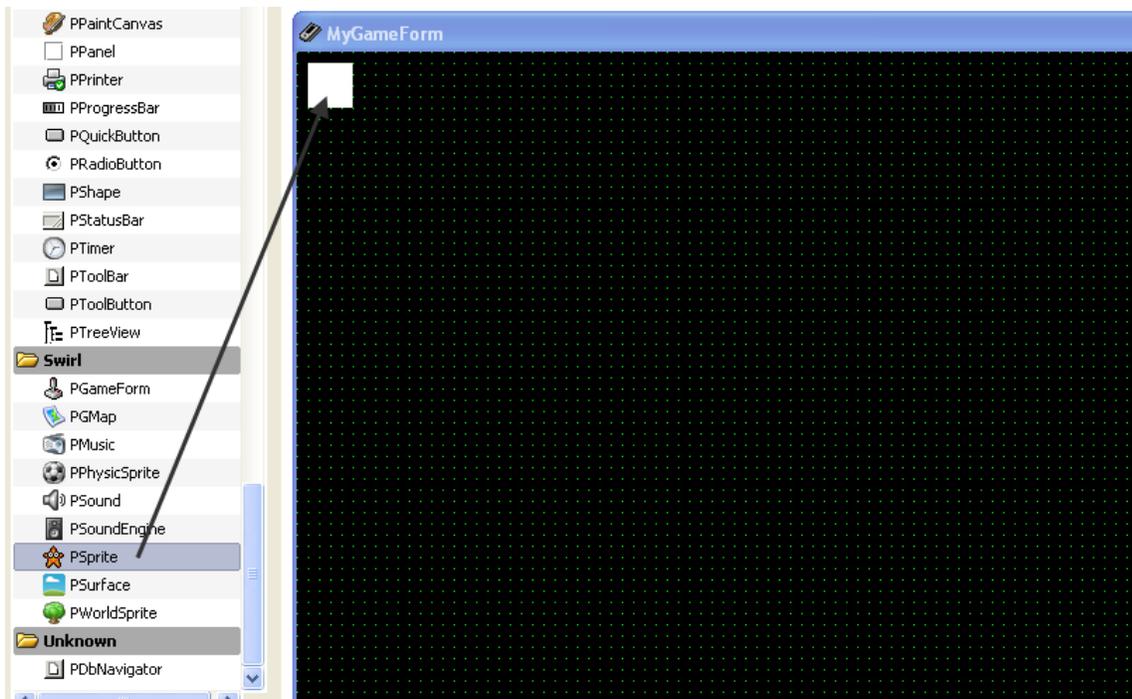


Figure 17: Drag PSprite to GameMap

- Go to the **Properties** pane and select the Surface you have included in your project in the **Surface Property**. This should make your **Sprite Object** have a background from the **Surface bitmap**. Now close the **GameMap Tab** to go back to the **Project Manager**.

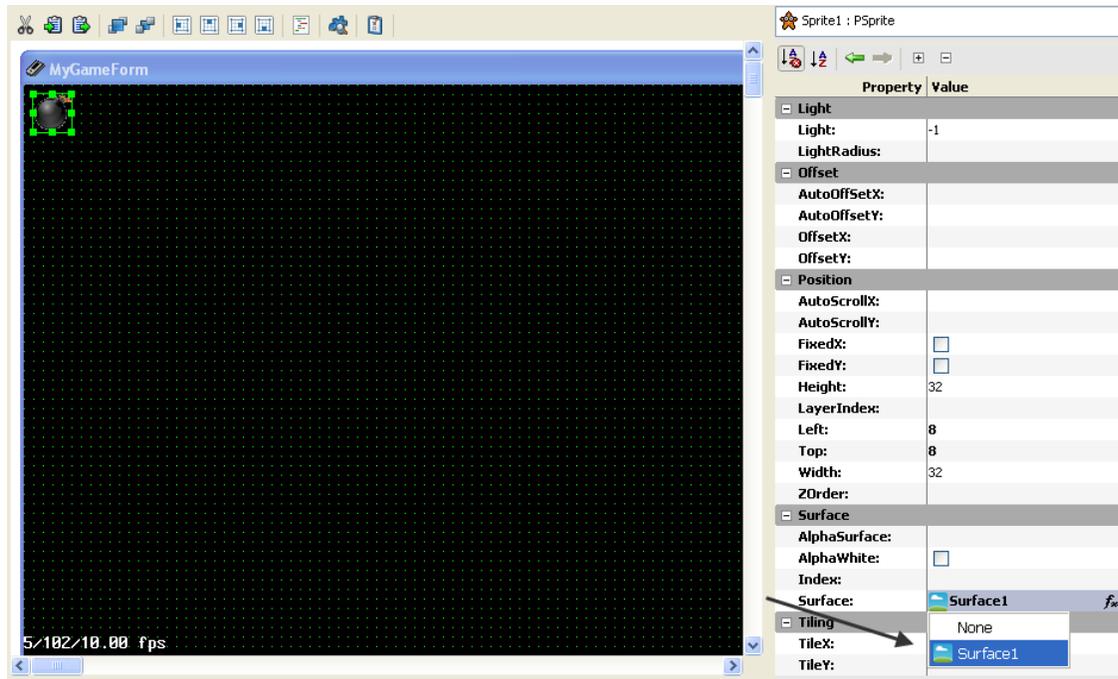


Figure 18: Change Surface property of PSprite

- Right click **MyGameForm** and select **OnMouseMove** event from the events context menu to create an event when a mouse moves.

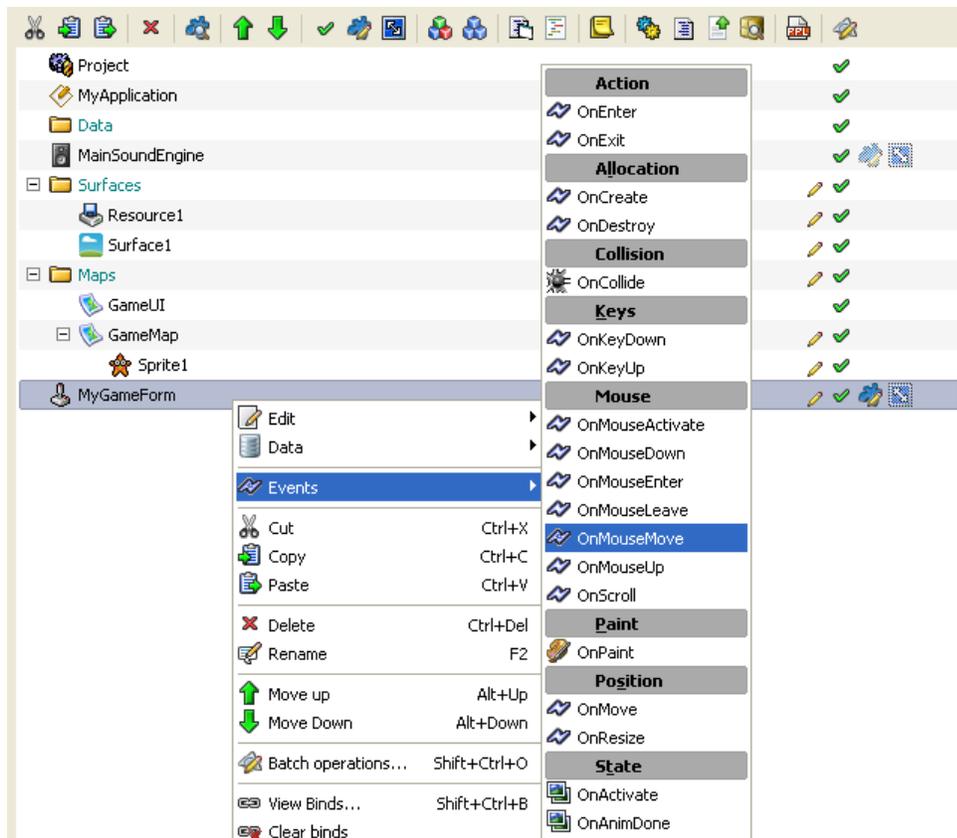


Figure 19: Create a move event on GameForm

- Drag the **PSprite Object** from your project to the **OnMouseMove** event created by you. This would enable an auto insert box to let you choose a method. Write **Move** and choose the **Move Method**.

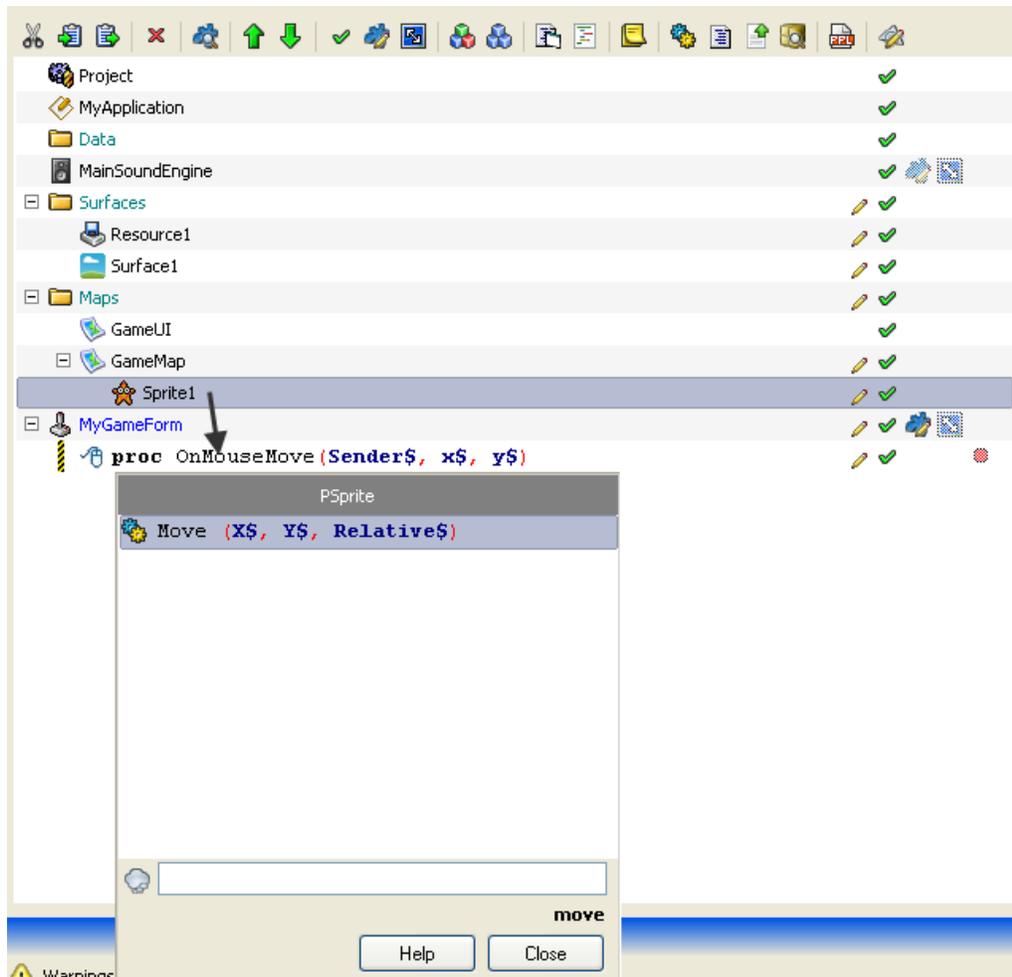


Figure 20: Drag PSprite to mouse move event

- Click the newly created **Move method** and change its **Parameters** to X\$ and Y\$ variables. Also set the **Relative property** to false.

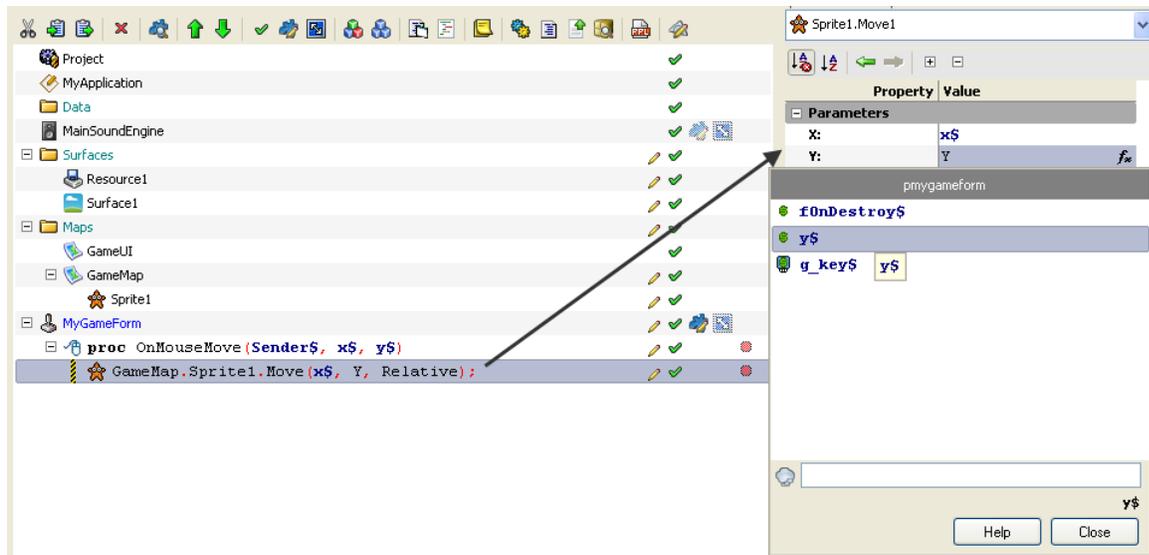


Figure 21: Set X and Y parameters of Move method

- Run the project to see the **Sprite object** following the mouse wherever it goes.

Code Navigator

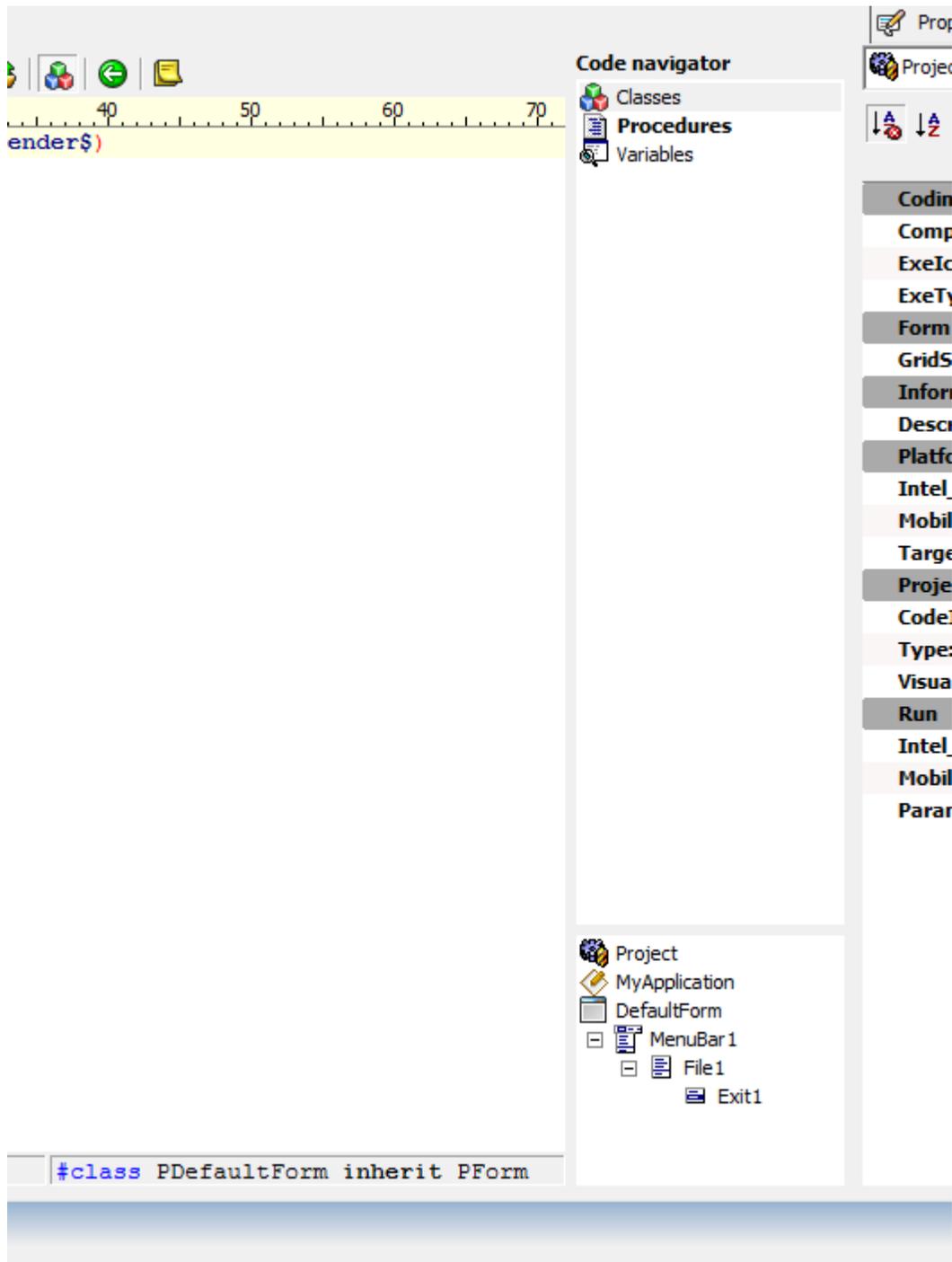


Figure 22: Code navigator

The code navigator is available in the Work Area by clicking on the Toggle Navigator project action button. The code navigator allows you to quickly interact with the three primary components of your code:

- Class,
- Procedures, and
- Variables.

Debugging Console

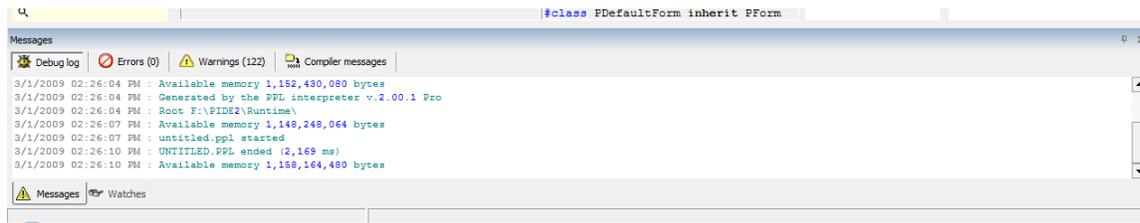


Figure 23: Debugging Console

The debugging console, located at the bottom of the interface, provides run-time analysis of code and project objects during execution. The debugging console of PIDE2 consists of five debugging tools to help a programmer debug his/her program while in execution. These tools are:

- Local Variables – Used to view the execution of variables declared local in the code.

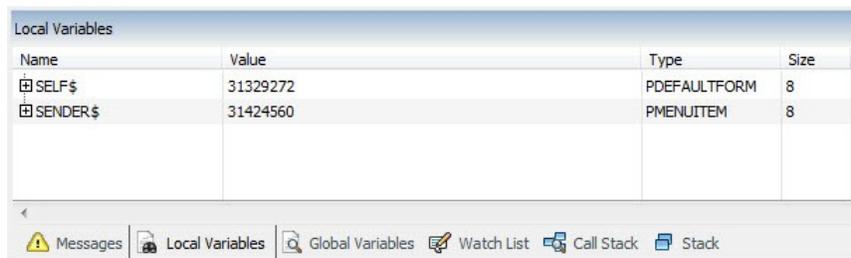


Figure 24: Debug Local variables

- Global Variables – Used to View the execution of variables declared global in the code.

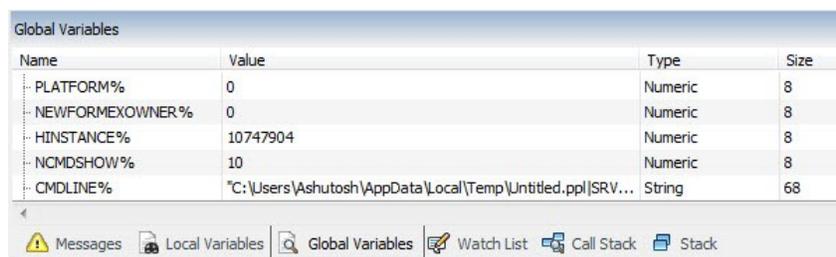


Figure 25: Debug Global Variables

- Watch List – Used to display the status of variables or expressions (during execution) that are selected by a user.

Name	Value	Type	Size
5632	0	Numeric	8
5308	0	Numeric	8

Messages Local Variables Global Variables Watch List Call Stack Stack

Figure 26: Debug Watch List

- Call Stack – Lists the functions or procedures in reverse order of their call.

Name	Parameters	Filename	Line
* 0x02a8e048 PDEFAULT...	31424560	1829852693	3
0x01b33890 PMENUITE...	19270076, 273, 2, 0	c:\pide2\pd\pdm...	349

Messages Local Variables Global Variables Watch List Call Stack Stack

Figure 27: Debug Call Stack

- Stack – Lists all the variables or expressions that are in the stack.

Name	Value	Type	Size
[0]	32761684	Numeric	8
FONCLICK\$	32761684	Numeric	8
[1]	31329272	Numeric	8
FOWNER\$	31329272	PDEFAULTFORM	8

Messages Local Variables Global Variables Watch List Call Stack Stack

Figure 28: Debug Stack

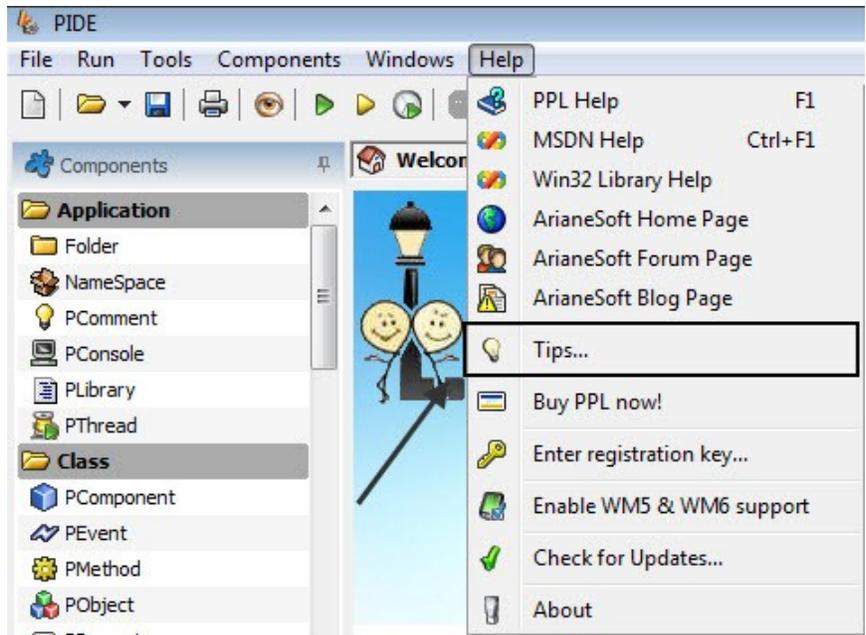
Tips on PIDE

Tips are used to provide a user with quick and efficient access to various functions of software. PIDE also uses a bunch of tips to allow its users to perform efficiently without any hassle.

By default, PIDE will show helpful tips at the startup of the application. These tips can be read by pressing the **Next Tip** button or closed by using the **Close button**. Users can also configure whether they want to see tips at the startup or not by using the **show tips at startup** drop down menu and selecting **always** or **never** option.



While in between the execution of the application, users can access tips by going to the Help menu and selection Tips.



Creating and Managing Projects

Projects created within PIDE are a combination of multiple files:

- Project file (.prj)—this file is an XML file that defines project parameters
- Code file (.ppl)—this is the .ppl code file that represents the project and all the programming objects within
- Help (.hlp)—this is a help file created to support the compiled application

Creating a New Project

Creating a new project can be carried out in a number of ways:

- Select the **New Option** from the **File Menu**, or
- Click on the **Create a New Project... Link** from the **Welcome Screen**.
- Both of these options will open the **Select New Project Type Window**.

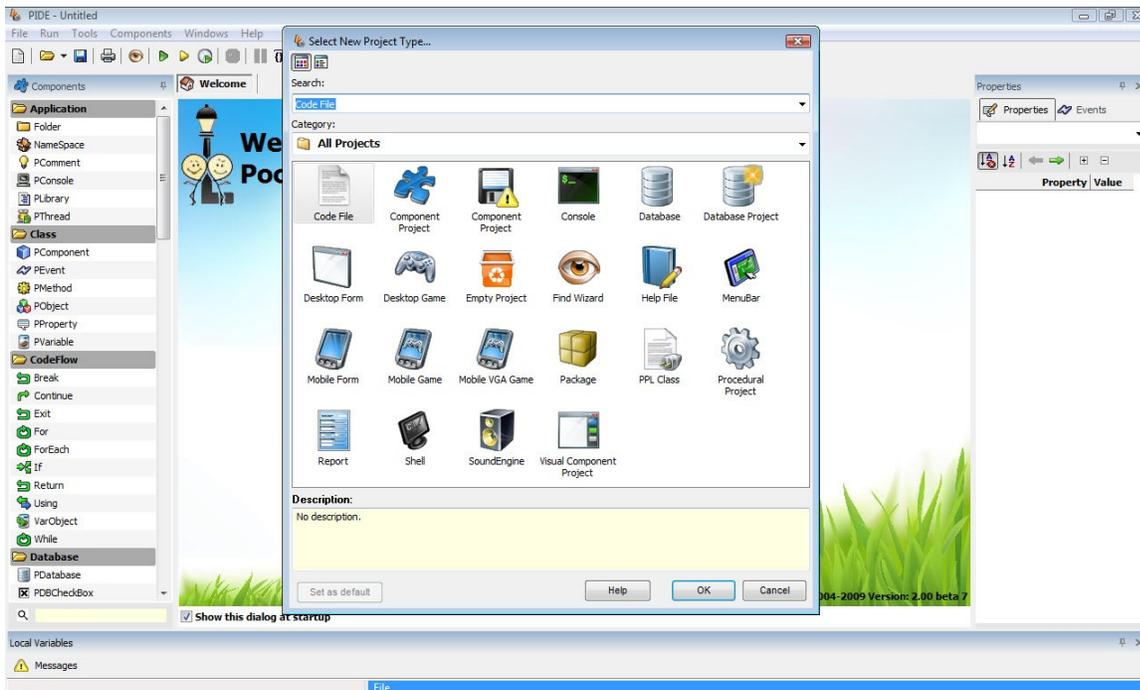


Figure 29: New Project Window

To create a new project through the Select New Project Type Window, simply select the type of project you want to create and click the **Ok Button**. Types include:

- Code File,

- Component Project,
- Console,
- Database,
- Database Project,
- Desktop Form,
- Desktop Game,
- Empty Project,
- Find Wizard,
- Help File,
- Menu Bar,
- Mobile Form,
- Mobile Game,
- Mobile VGA Game,
- Package,
- PPL Class,
- Procedural Project,
- Report,
- Shell,
- SoundEngine,
- Visual Component Project.

Opening a File into the Work Area

To open a file into the work area,

- Select the **Open Option** from the **File Menu**, or
- Click on the **Open File Button**,
- Both of these options will open the **File Dialog Window**.

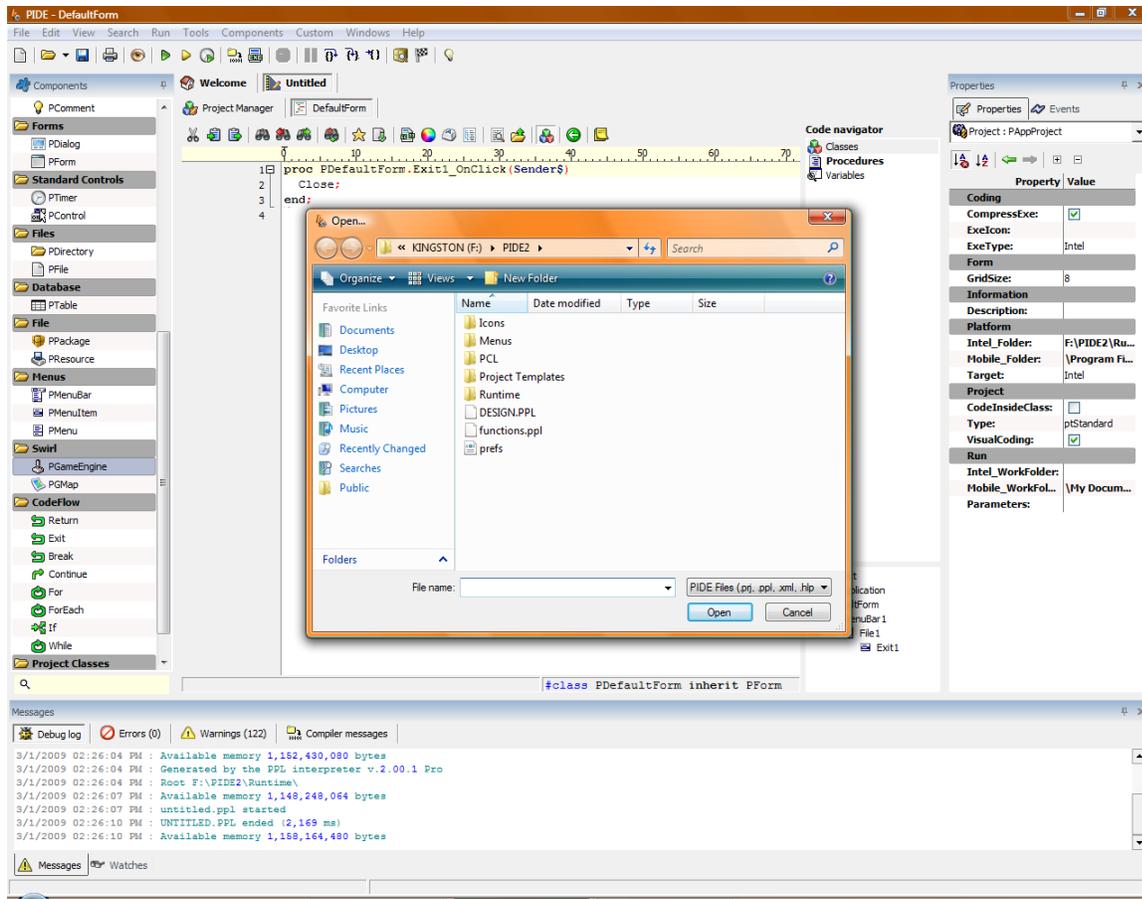


Figure 30: Opening a file

Use the File Dialog Window to find the project file you wish to open. Supported file types include:

- .ppl,
- .xml,
- .prj,
- .hlp,
- .bmp,
- .plg,
- .smd,
- .eed and
- .ctp

When you have found the file you wish to open, click the **Open Button** to open it into the work area.

Saving a Project

For saving the progress on a project, users have 4 options namely,

- Save
- Save as...
- Save Project
- Save Project As...

If you want to save the project as is, you can press the **Save** button from the **File Menu**. For saving the whole project you can use the **Save Project** or the **Save Project as...** option.



Figure 31: Saving a file

To save the current progress on a project,

- Select the **Save as...** Option from the **File Menu**,
- Write a **File name** for your file,
- Click the **Save Button**.

Note: you can select the Save As option from the File Menu to create a copy of the project. This will open the File Dialog Window and provide you the opportunity to create a new name for the project.

Close an Open Project

To close the current project,

- Right-click the name of the project above the Project Action Buttons and select the **Close** or **Close All Option** from the context-sensitive menu.



Figure 32: Closing a file

Note: if you have made changes, PIDE will prompt you to save the changes before closing the project.

- Click the **Save Button**.

Working with Code

As an integrated design environment, PIDE includes both design-time features (i.e., adding objects to your project and having the code automatically generated) as well as the opportunity to work directly with the code. Users can also work with the console environment to create programs.

About the Pocket Programming Language (PPL)

PPL is now a fully object-oriented programming environment but the old procedural programming is still supported. Most of the PIDE 2 file (projects, help, components definition, and preferences) are XML based. The PPL source code is a text file. The language syntax is a mix of C, Basic and Pascal.

Accessing the Code View

To access the Code view for a project (opening the .ppl file),

- Click on the **PPL Icon** in the Project Action Buttons, or
- If the code file is already open, click its name above the Action Buttons.
- Both these options will open the current project .ppl file into the work area.

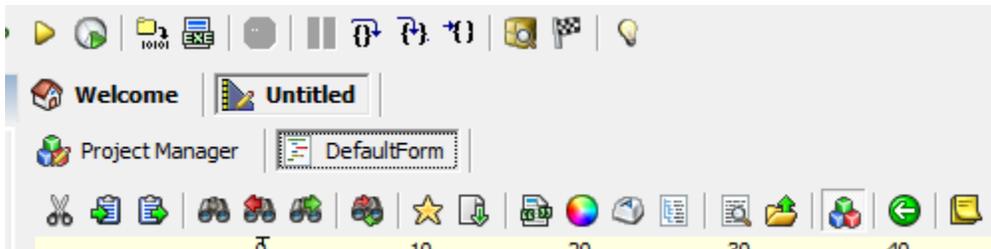


Figure 33: Code view

The Code View within PIDE is similar to other IDEs and provides a complete nested, organized view of the project code. Through the Code View, you can carry out a number of actions available from the Project Action Buttons on the top or the Components Navigator to the left. These actions include:

- Commenting code,
- Formatting code,
- Adding objects,
- Collapsing/expanding code,
- Adding color to code,

- Adding characters,
- Creating notes, and
- Viewing definitions.

Commenting code

There may be times when you need to quickly comment a block of code. This can be carried out simply through the Action Buttons.



Figure 34: Commenting code

To comment code,

- Select it from the Code View by highlighting it
- Click the **Comment Selected Code Button**. This will automatically comment all of the selected code.

Formatting Code

Keeping code properly formatted is critical for a neat, organized project. This will, in turn, make it easier to find code and functionality later.

```

1  proc PDefaultForm.Exit1_OnClick (Sender$)
2  Close;
3  end;
4

```

Figure 35: Before Formatting

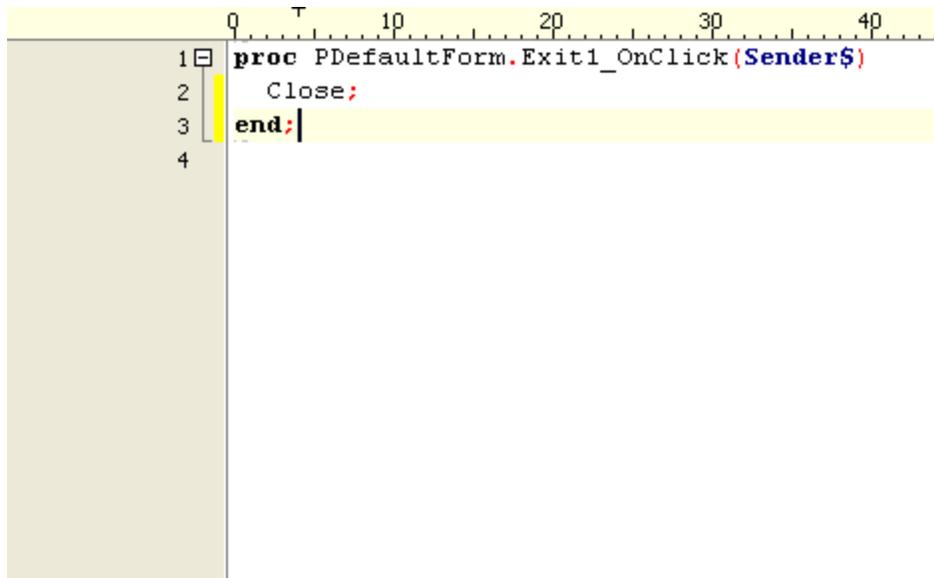
A screenshot of a code editor window. At the top, there is a ruler with markings at 0, 10, 20, 30, and 40. Below the ruler, the code is displayed on a light gray background. Line 1: `proc PDefaultForm.Exit1_OnClick(Sender$)` Line 2: `Close;` Line 3: `end;` Line 4: (empty line) The code is formatted with proper indentation and line numbers on the left. A yellow highlight is visible under the `end;` statement on line 3.

Figure 36: After formatting

To format all the code in the .ppl file,

- Click the **Format Code Button** from the Action Buttons.

Adding Objects

Application objects (i.e., returns, classes, etc.) can be easily added to the code by dragging them from the **Component Panel** to the left of the work area.

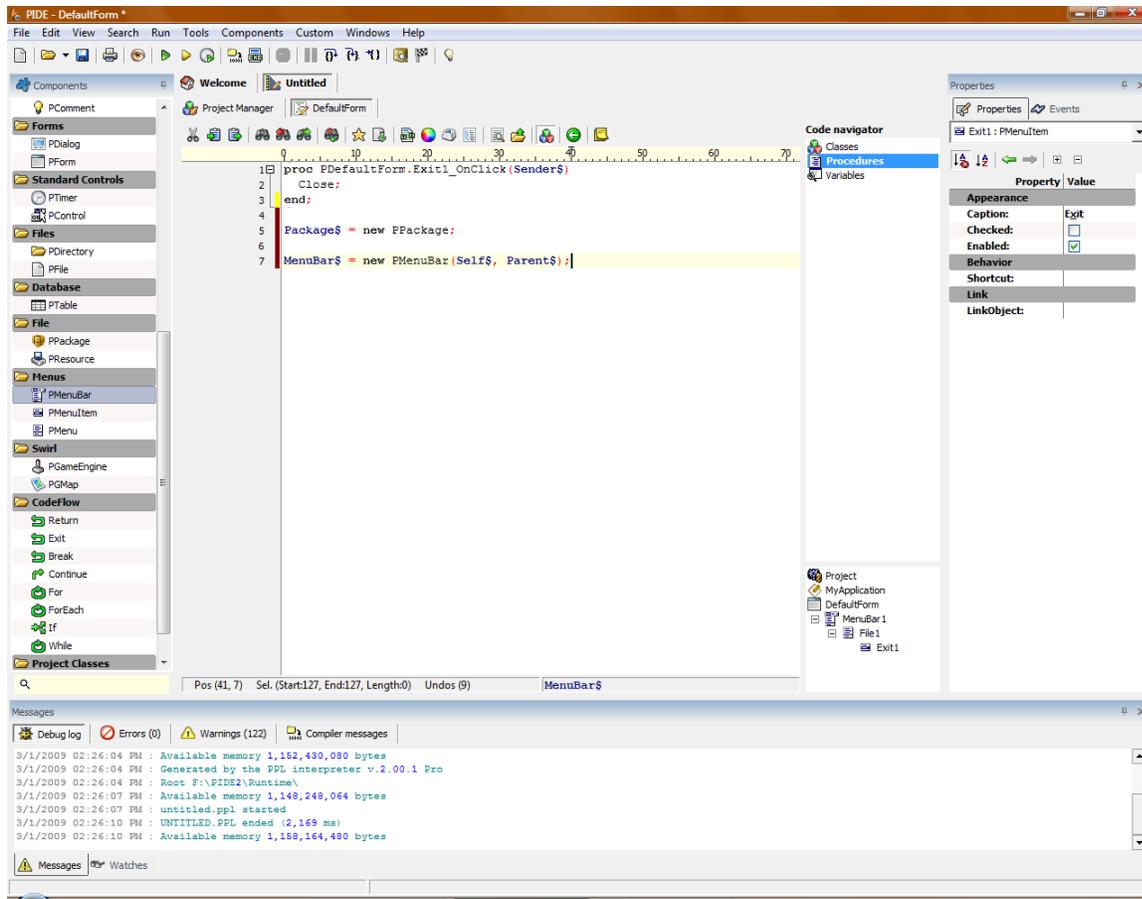


Figure 37: Adding objects

The Component Panel provides a number of objects that can be added to the code. These include:

- Application,
- Class,
- Code Flow,
- File,
- Standard Control,
- Forms,
- Menus,
- PPLs,
- Database,

- Swirl, and
- Project Classes

Note: See Appendix A for a definition of the available application objects.

Note: you can also install and uninstall components from the panel. For more information about the Component Panel, see the appropriate section in this manual.

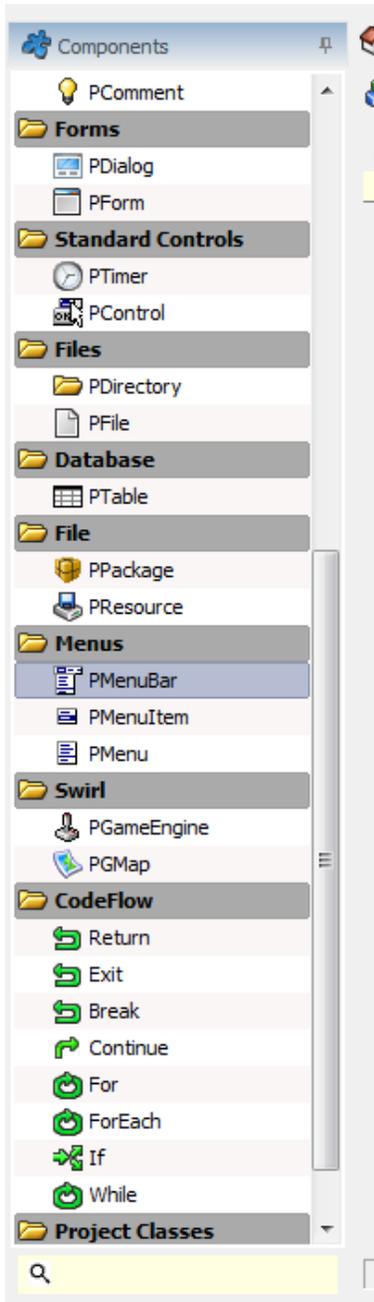


Figure 38: Components tab open

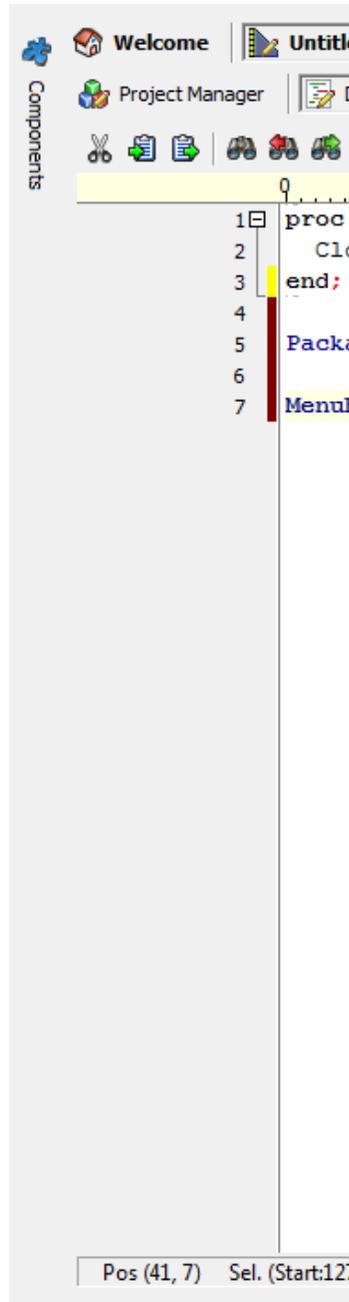


Figure 39: Components tab collapsed

To add an application object to the code,

- Click on the item you wish to add from the Component Panel and drag it to the appropriate location within your code. This will automatically add the appropriate code to reflect the application object.

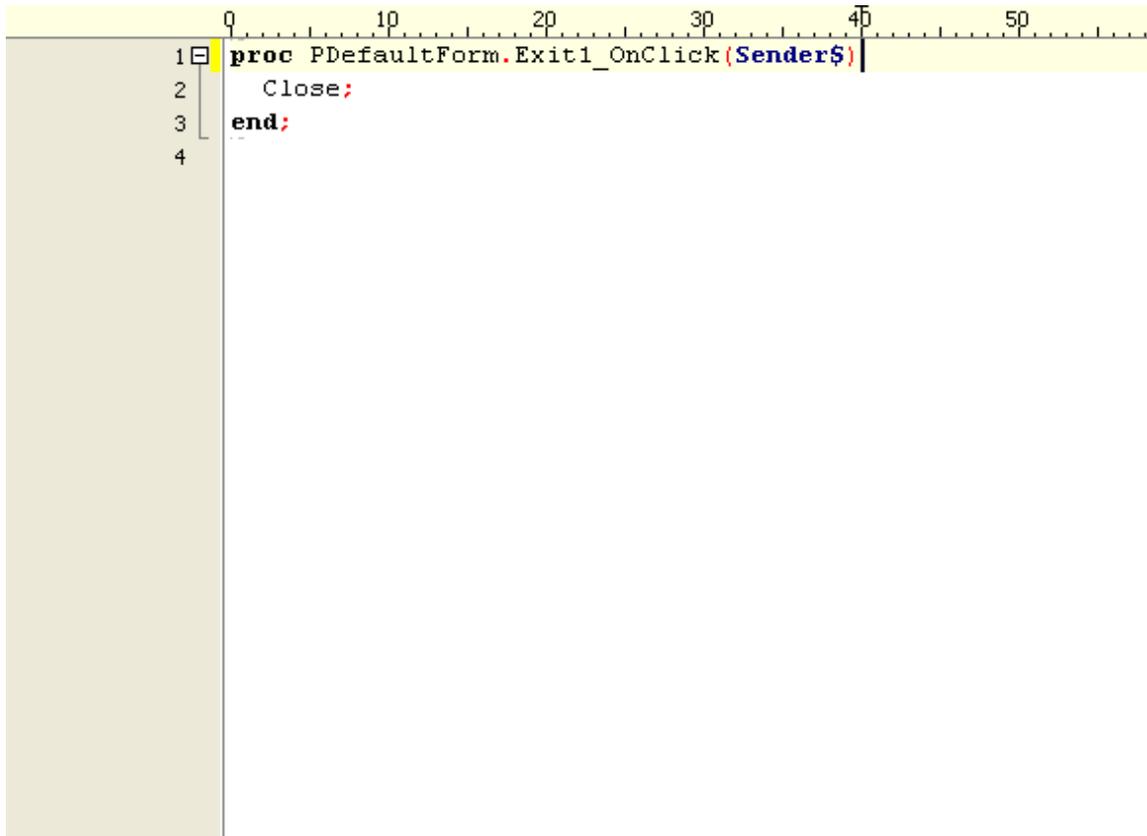
Collapsing/Expanding Nodes

To collapse or expand a node in your code view (making it easier to manage),

- Place the cursor in the code you wish to collapse or expand, and
- Click the **Collapse/Expand Button** in the Action Buttons.

Adding Color to Code

You may need, at times, to add a RGB color reference in your code.



The image shows a code editor window with a ruler at the top. The code is as follows:

```
1 proc PDefaultForm.Exit1_OnClick(Sender$)
2     Close;
3 end;
4
```

The code is color-coded: 'proc' is in black, 'PDefaultForm.Exit1_OnClick' is in blue, '(Sender\$)' is in red, and 'Close;' and 'end;' are in black. The background of the code area is a light beige color.

Figure 40: Adding color

To add a RGB color to your code,

- Insert the cursor at the point in the code where you wish to add the RGB color and click the **Insert RGB Color Button**. This will open the **Color Selector Window**.

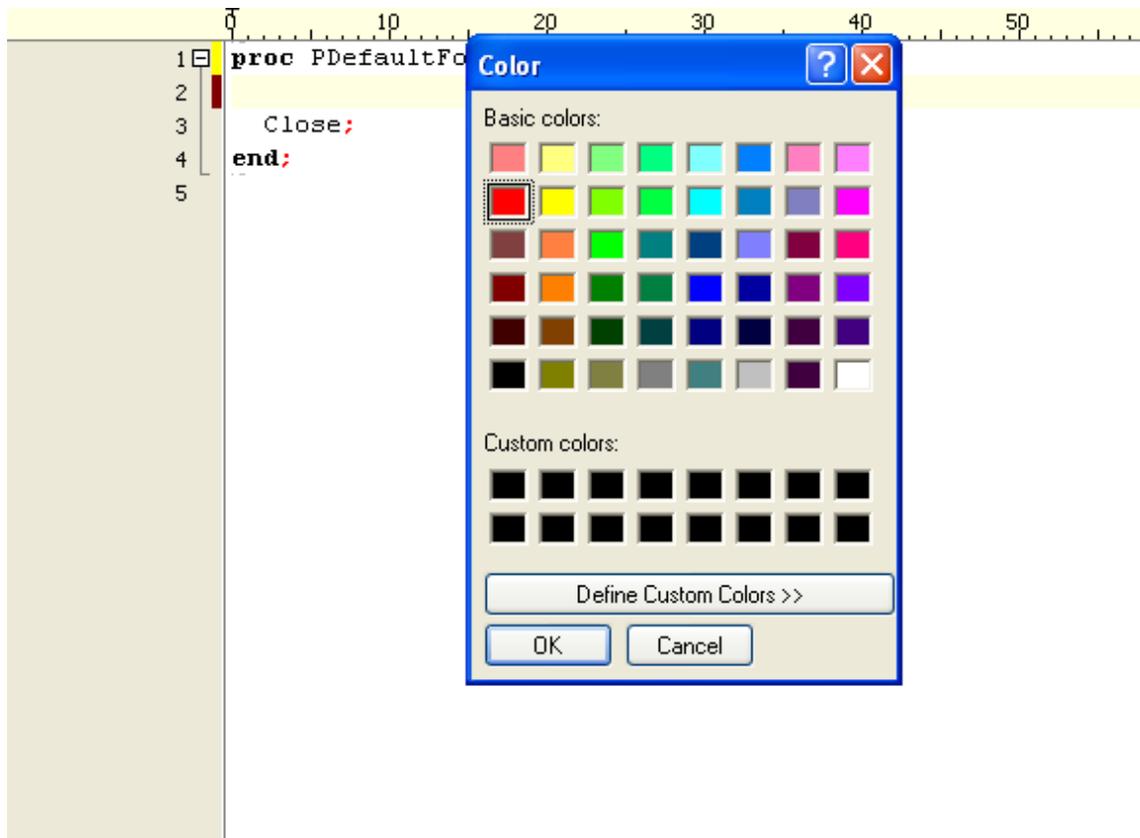
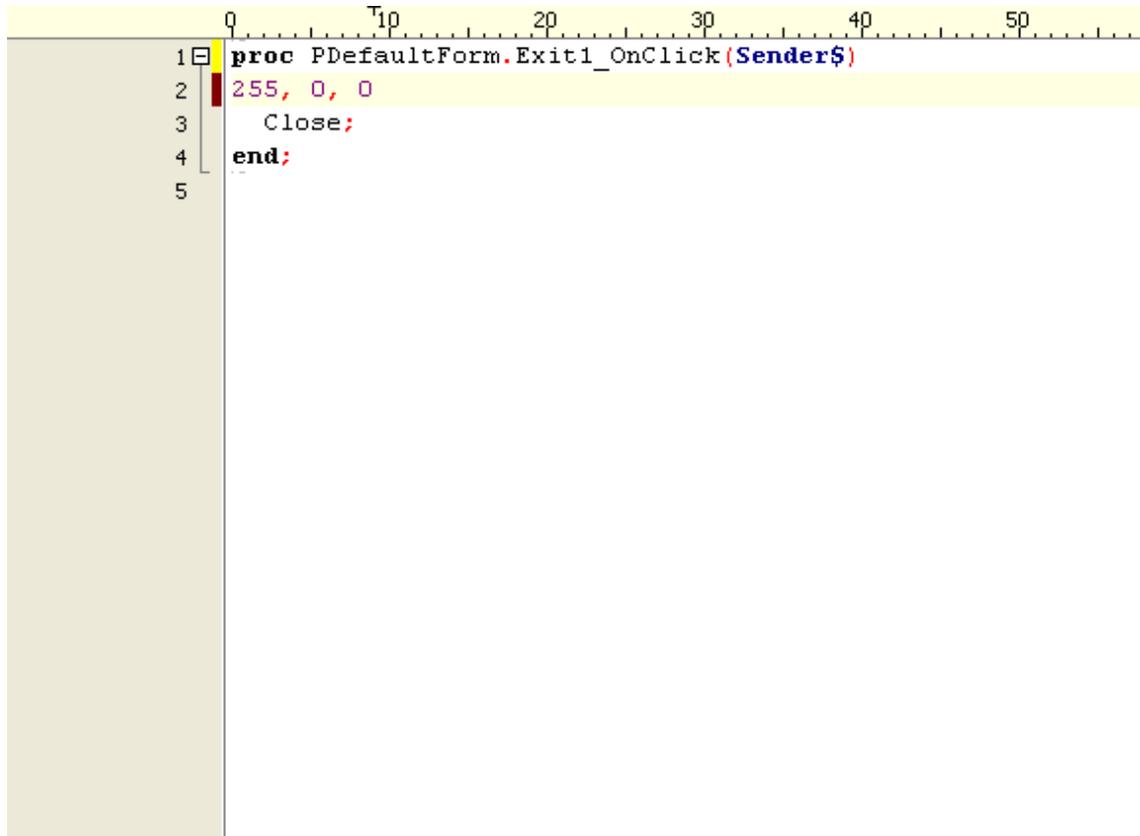


Figure 41: Color Selector Window

- Click on a color to insert and click the **Ok Button**.



```
0 10 20 30 40 50
1 proc PDefaultForm.Exit1_OnClick(Sender$)
2   255, 0, 0
3   Close;
4 end;
5
```

Figure 42: Inserted Hexidecimal Color

- The RGB color selected in the Color Selection Window will appear at the point in the code where you clicked the color button as three numerical values:
 - Red,
 - Green, and
 - Blue.

Adding Characters

You can add special characters into your code and point of the cursor. To add special characters,

- Place the mouse cursor in your code at the point where you want to add the characters
- Select the **Characters Option** from the **Edit Menu**. This will display the special characters selection pop-up at your mouse cursor.

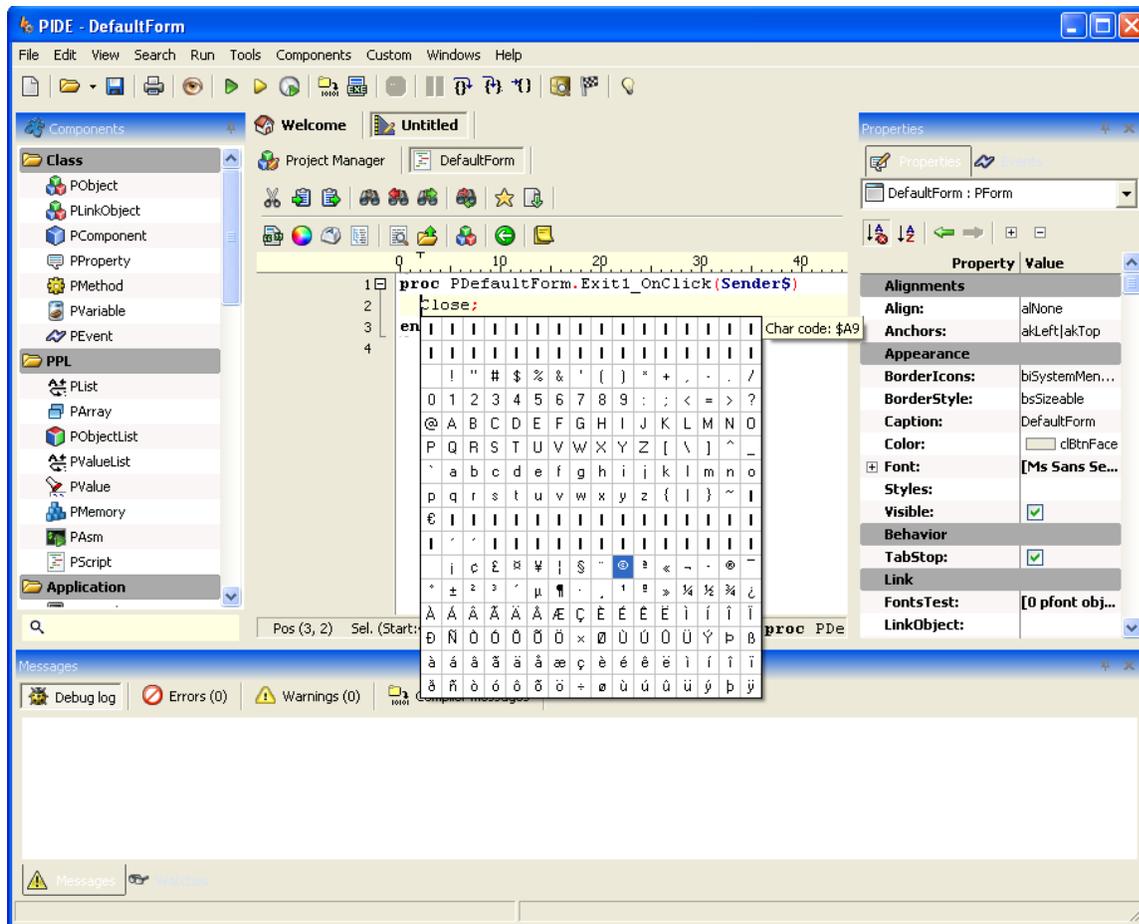


Figure 43: Insert Special Characters

- To insert, simply click on the desired character.

Creating Notes

Notes are a great way to add code level-comments.

To add a note,

- Put the cursor at the point in the code where you wish to add the note
- Click the **Note Button** from the Action Buttons. This will open the **Edit... Window**.

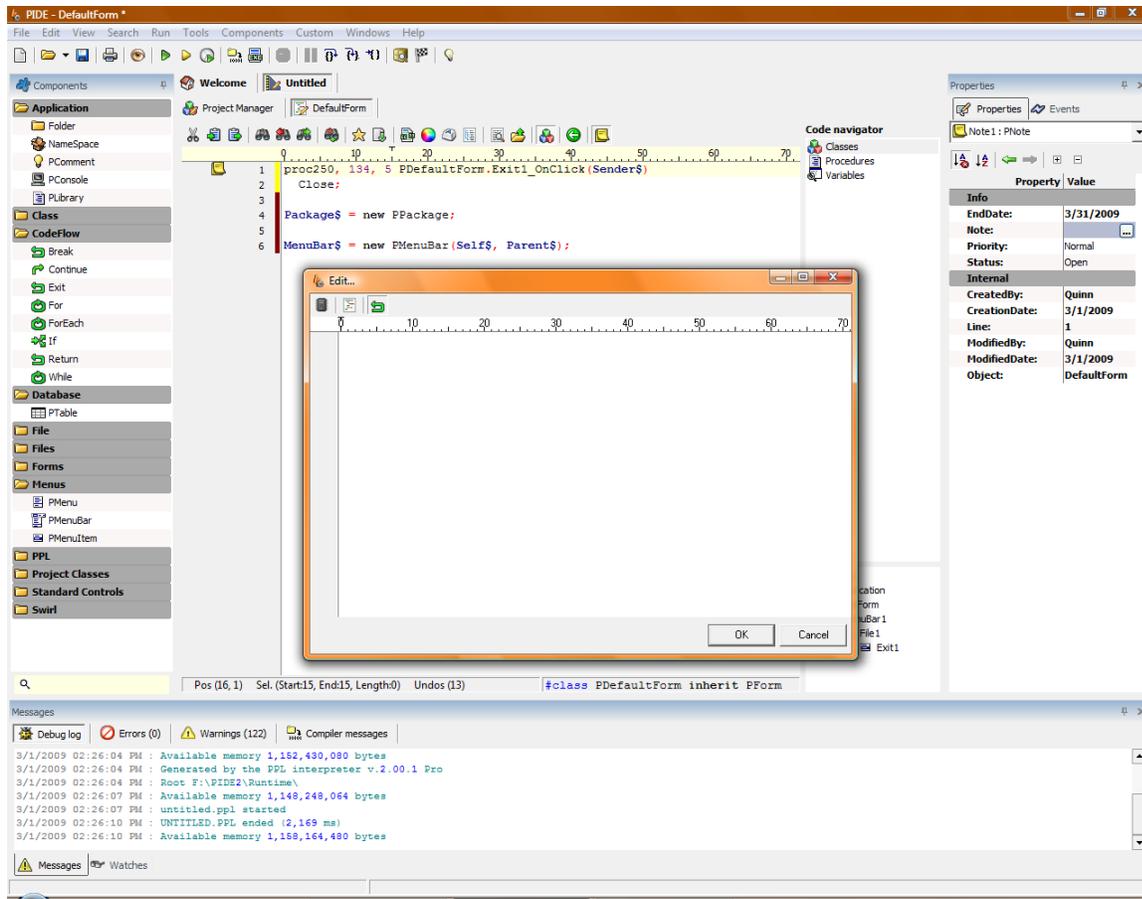


Figure 44: Adding notes

The Note Editor allows you to create text-based notes for your code.

Note: each note you create will be visible from the Visual Designer interface. The note is also assigned to a specific line in the code.

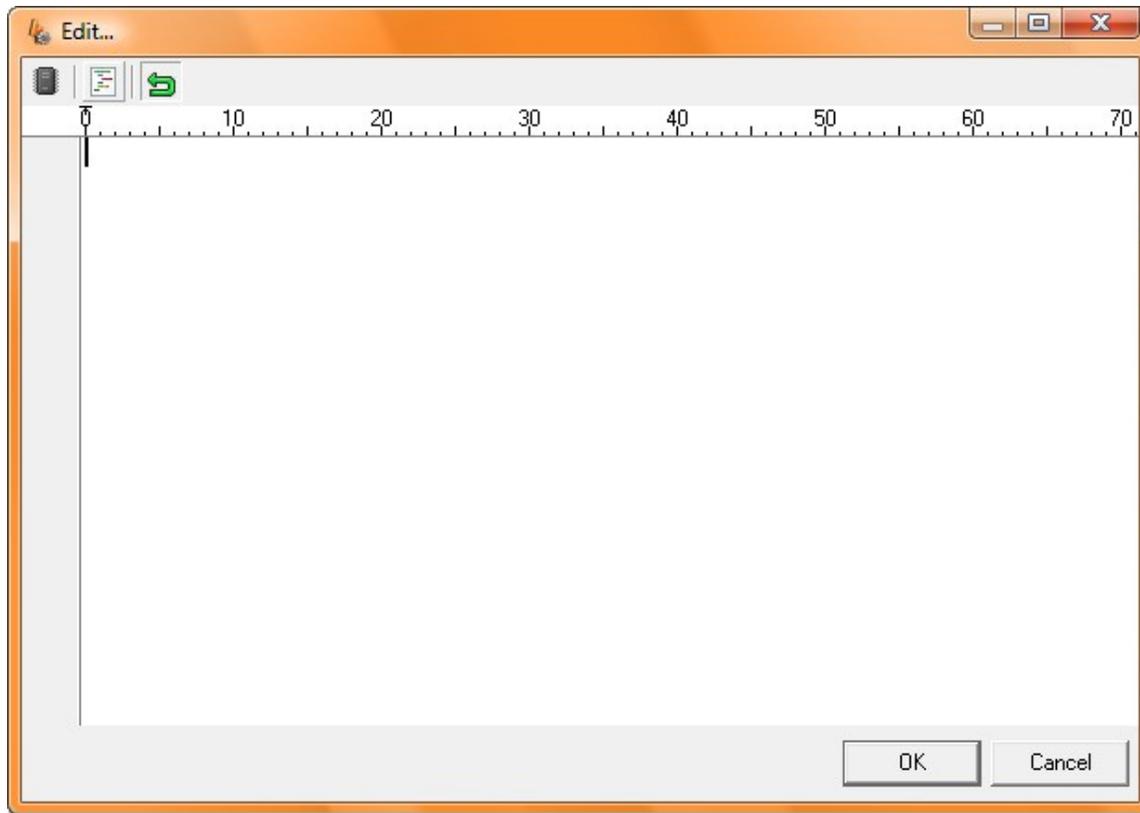


Figure 45: Textual note

To write a text note,

- Click the **Text Button**. This will allow you to compose a textual note. *Note: this is the default note format so if you wish to compose a textual note when first clicking the Note Button, you do not need to click the Text button.*

Edit a Note

To edit a note,

- Double click on the note you wish to edit, or
- Select the note by clicking on it and then click on the **... Button** of the **Note Field** in the **Properties Panel**.
- Both of these options will open the note for editing.

Procedure List

While project manager makes it very easy to locate an object, code editor is not that useful in finding something. In the code view, users can use the Procedure list to find and locate functions as well as procedures form with a code. Especially useful for people who work with thousands of lines of code, Procedure list can be used by pressing the **Ctrl + G** key in

the code view.

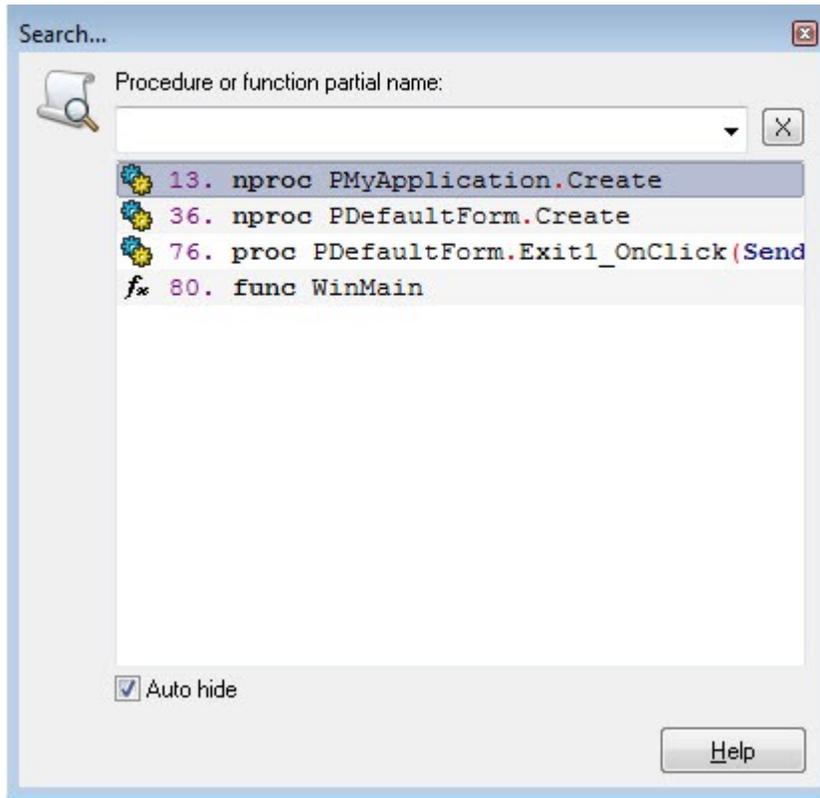


Figure 46: Procedure list

Components

There may be times when you add components that refer to external .ppl files (i.e., the game engine uses swirl.ppl). In this event, the associated .ppl file will be included at the top of the project .ppl file using an include statement.

For more information about the Component Library, see the appropriate chapter in this manual.

Navigating Code

Through the Code View, you can also quickly access categories of elements through the Code Navigator:

- Classes,
- Procedures, and
- Variables.

To open the code navigator,

- Click on the **Toggle Navigator Button** from the Action Buttons. This will display the Code Navigator.

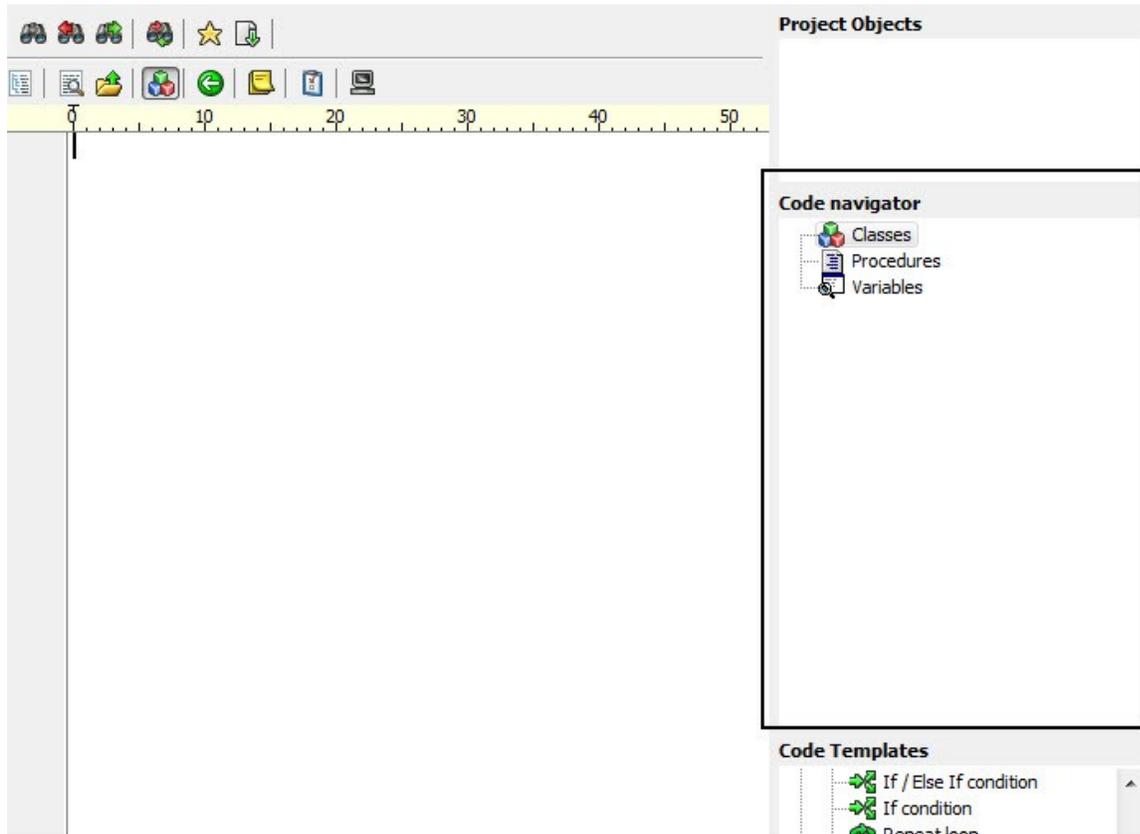


Figure 47: Code navigator

To see your project elements within a specific category of code, double click on the category name. This will expand the category to display the elements of your project that fall within that category. You can then double-click on the element to jump to that specific point within the source code of your project.

The Design Environment

PIDE includes a comprehensive and powerful visual designer, allowing you to create robust applications (including games) by dragging & dropping components as well as specifying component behavior through properties and events.

The visual designer is opened by default when you open a project and provides you with a variety of tools by which to easily create your application:

- Drag and Drop Components,
- Application Object Properties and Events,
- Layout,
- Auto Create, and
- Object editing

Drag and Drop Components

Perhaps the most powerful feature of the visual designer is dragging and dropping components from the Component Library into the application flow. Whether fully-encompassed objects (such as a form or menu) or application logic (such as a return or a while loop), you can easily and quickly add functionality to your application with little to no coding.

The Component Panel includes the following objects you can add:

- Application,
- Class,
- Code Flow,
- File,
- Standard Control,
- Forms,
- Menus,
- PPLs,
- Database,
- Swirl, and

- Project Classes

Note: See Appendix A for a definition of the available application objects.

Note: you can also install and uninstall components from the panel. For more information about the Component Panel, see the appropriate section in this manual.

To add an application object to the visual designer,

- Click on the item you wish to add from the Component Panel and drag it to the appropriate location within your application flow. This will automatically add the respective item in the visual editor.
- All you need to do is to set the various property and events values, you will be able to perform many functions with the help of events, methods and classes.

Note: you cannot drag an application to any spot. You can only place objects appropriately within your flow.

Once an object has been added to the application flow, you can carry out additional actions by clicking on the appropriate Action Button:

-  Auto Create,
-  Sub-Class,
-  Expand Class,
-  Edit, and
-  Edit code.

Auto Create

PIDE can auto-create code for elements that are added to the flow. There may be times when you don't want the code for a component generated automatically. To turn on or off this feature,

- Select the application object for which you wish to change the auto create setting.
- Click the **Auto Create Button** from the Action Buttons. This will either turn the auto code generation for the selected object off or on and will be indicated by either a solid or transparent green check.

Sub-Class

This creates a sub class that is added to the palette so that you can re-use it by dragging it into the visual editor

Expand Class

Expand on a base class by adding code. All instances of the class used throughout your project will be automatically updated with the additions. Furthermore, adding the expanded class will include your additions.

Edit

Some objects have visual components that can be edited in the visual layout tool. To edit an object,

- Select the object (i.e., a form)
- Click the **Edit Button** from the Action Buttons. This will open the object into the visual layout tool.

Note: for more information about the visual layout tool, see the Layout Section of this manual.

Edit Code

To edit the code for an application object,

- Select the object (i.e., a form)
- Click the **Edit Code Button** from the Action Buttons. This will open the object into the code view.

Note: for more information about editing code, see the Code View section of this manual.

Application Object Properties and Events

Each object that you add to your application from the Component Library has properties that can be specified through the Properties Pane. In addition, some objects have events (i.e., onclick) that can be specified through the Events Panel.

Properties Panel

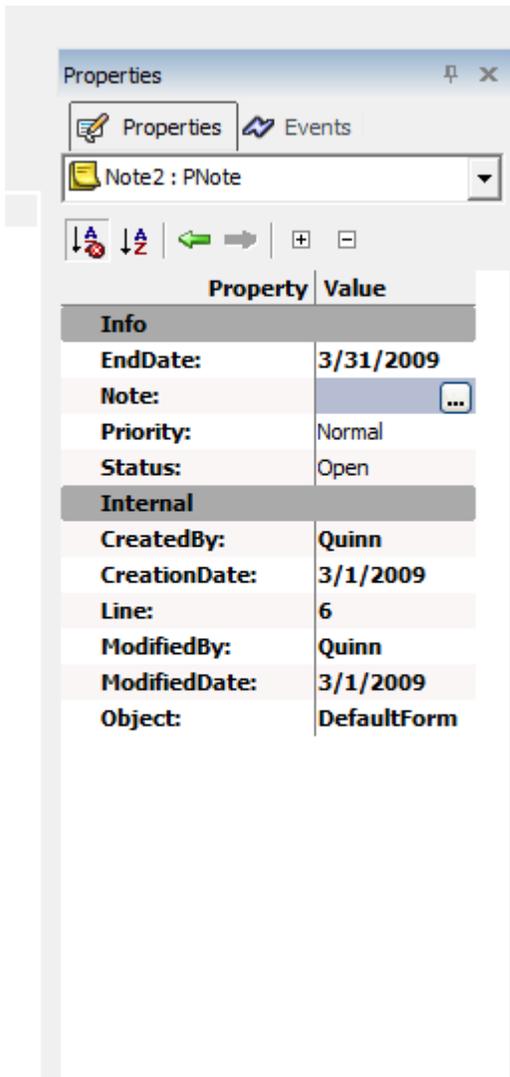


Figure 48: Properties panel

The Properties Panel allows you to specify properties of an application or project object.

To specify an object's properties,

- Select the object in the application flow.
- This will automatically populate properties pane with the object's current properties.

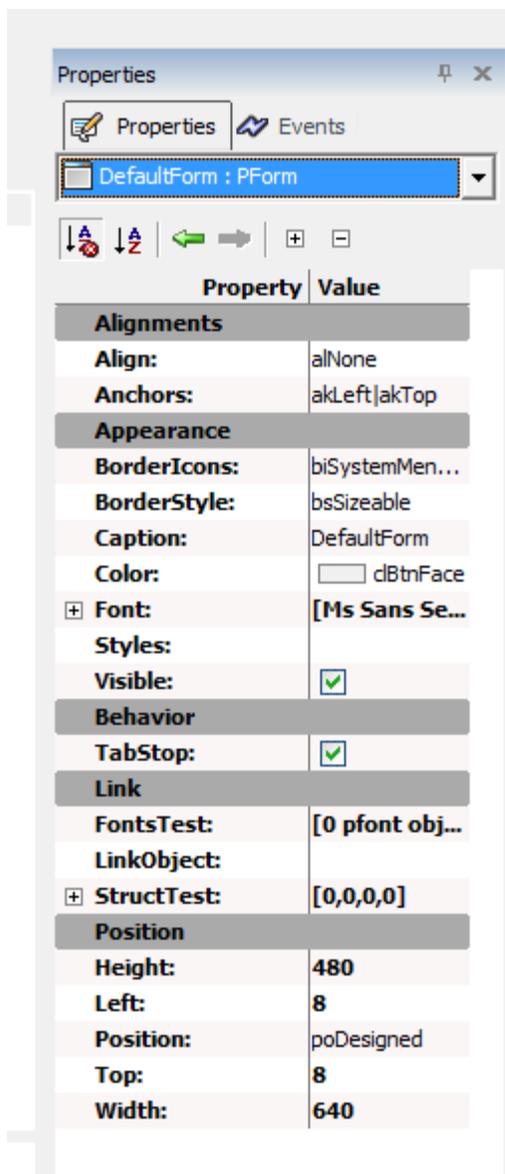


Figure 49: Selecting Properties

Each object can have a variety of properties categorized into one of several categories depending upon the type of object:

- Alignments—applicable to objects with layout properties such as forms
- Appearance—applicable to objects with layout properties such as forms
- Behavior
- Link
- Position—applicable to objects with layout properties such as forms

- Input—applicable to objects that have graphical interfaces (i.e., the game engine)
- Parameters, and
- Status.

To set the properties for a specific item in a category,

- Click on the Value field of the property you wish to set and specify a value (or select it)

Note: you can use the drop-down menu at the top of the Properties Pane to jump to other application objects associated with the object that you initially selected.

Events Panel

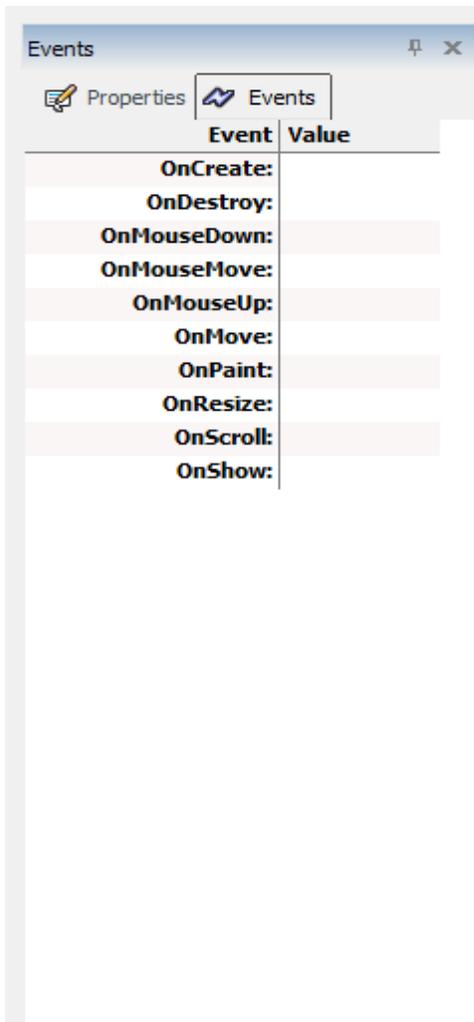


Figure 50: Events panel

In addition to properties, some objects (like menus and buttons) have events that you can specify.

To specify an object's events,

- Select the object in the application flow.
- This will automatically populate properties pane with the object's current properties.
- Click the Events Tab.

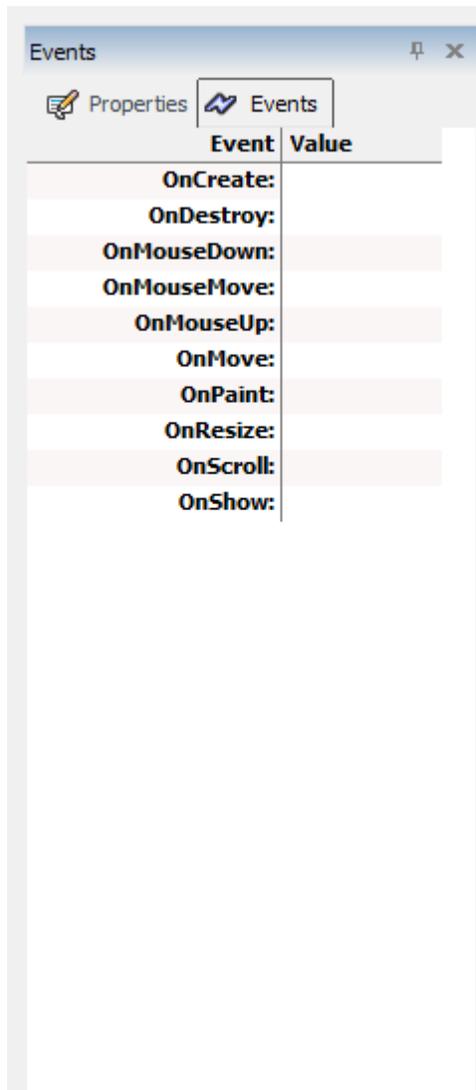


Figure 51: selecting events

PIDE includes a number of default events that you can specify including:

- OnAnimDone,
- OnCollide
- OnClick,

- OnCreate,
- OnDestroy,
- OnMouseDown,
- OnMouseMove,
- OnMove,
- OnPaint,
- OnResize,
- OnScroll,
- OnShow, and
- OnTimer

To set the value for a specific event,

- Click on the Value field of the event you wish to set and select the application object that will be called upon that event's instantiation, or
- Double-click on the event you wish to add. If an event already exists of that type, it will jump to that portion of your code where the event is located; otherwise, it will create a new event.

Note: if you are in the visual editing mode and add an event, it will create the event in the visual framework.

Layout

PIDE also includes a visual layout tool to design the interface-components of application objects such as forms.

To open the visual layout tool,

- Double click on an application object in the visual design environment (i.e., a form). This will open the visual layout tool.

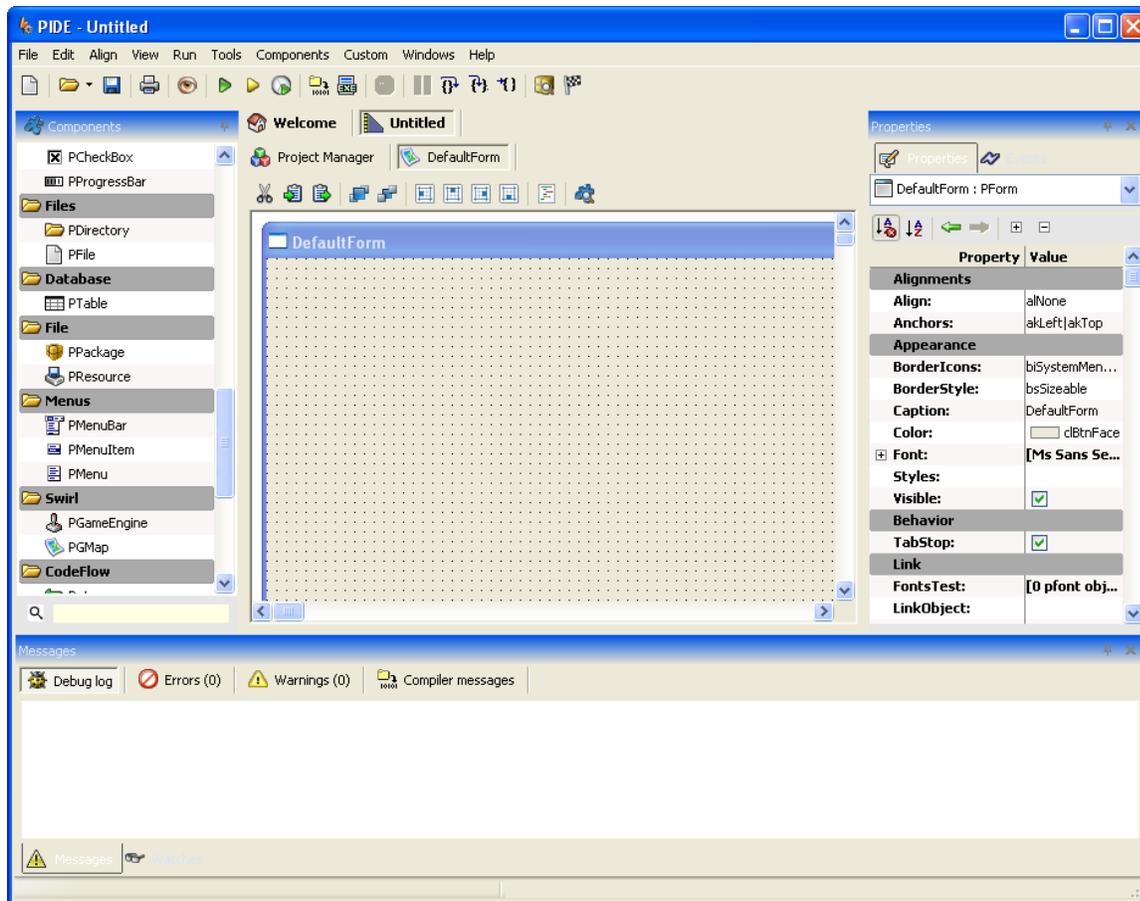


Figure 52: Visual Editor

Through the visual layout tool, you can add a variety of controls (with associated events) to an application object such as a form. When you are in the visual layout tool, an additional component library category becomes available, Standard Controls, that include:

- PControl,
- PButton,
- PTimer,
- PEdit,
- PMemo,
- PGroupBox,
- PLabel,
- PListBox,
- PComboBox, and

- PGrid

Through the visual layout tool, you can:

- Add Elements,
- Position Elements,
- Edit Control Code, and
- Edit Events and Properties.

Add Elements

To add an element to the visual layout tool,

- Double click on the control you wish to add to the work area.

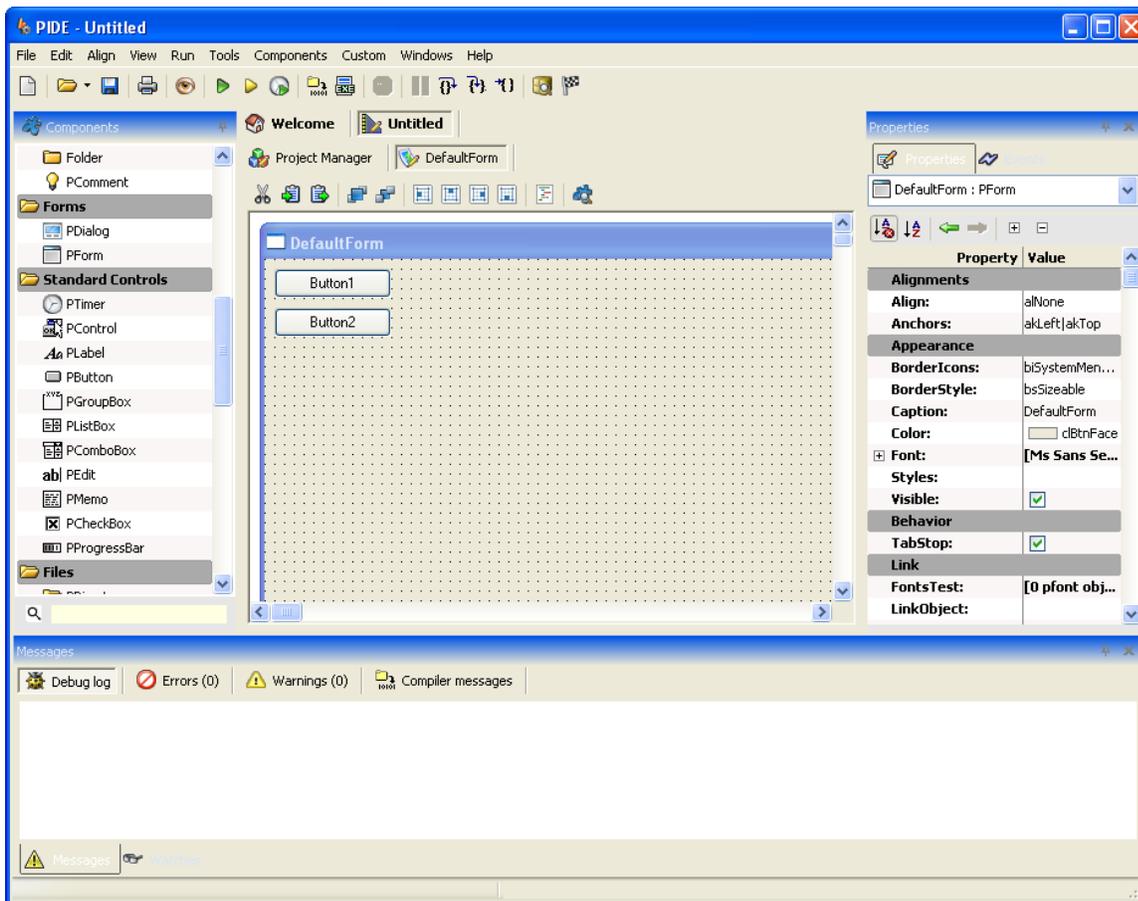


Figure 53: Elements Placed in the Visual Editor

Once the control has been placed, you can drag it around the layout editor to position it where you wish.

Position Elements

Once the control has been placed, you can carry out additional actions to place and arrange the control by first select the control and then clicking the appropriate Action Button:

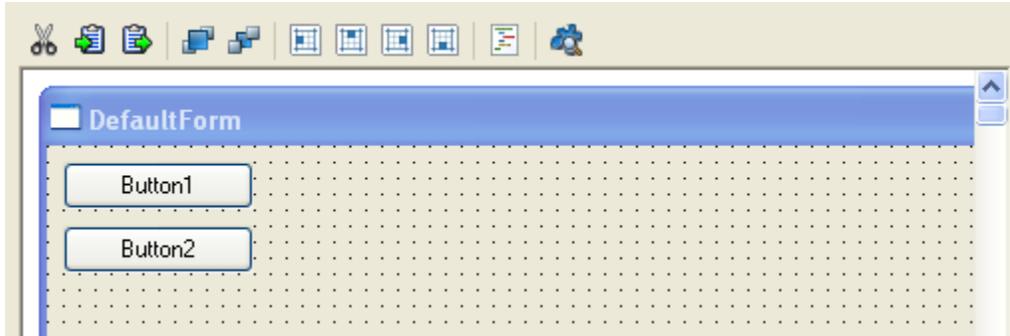


Figure 54: Visual Editor Placement Tools

- Bring control to front—bring the control to the topmost layer
- Send control to back—send the control the bottom most layer
- Align—you can align one or more controls at the same time by selecting them and clicking the appropriate Action Button:
 - Left,
 - Right,
 - Top, and
 - Bottom.
- Placement—you can click and drag the control to any part of the visible area

Edit Control Code

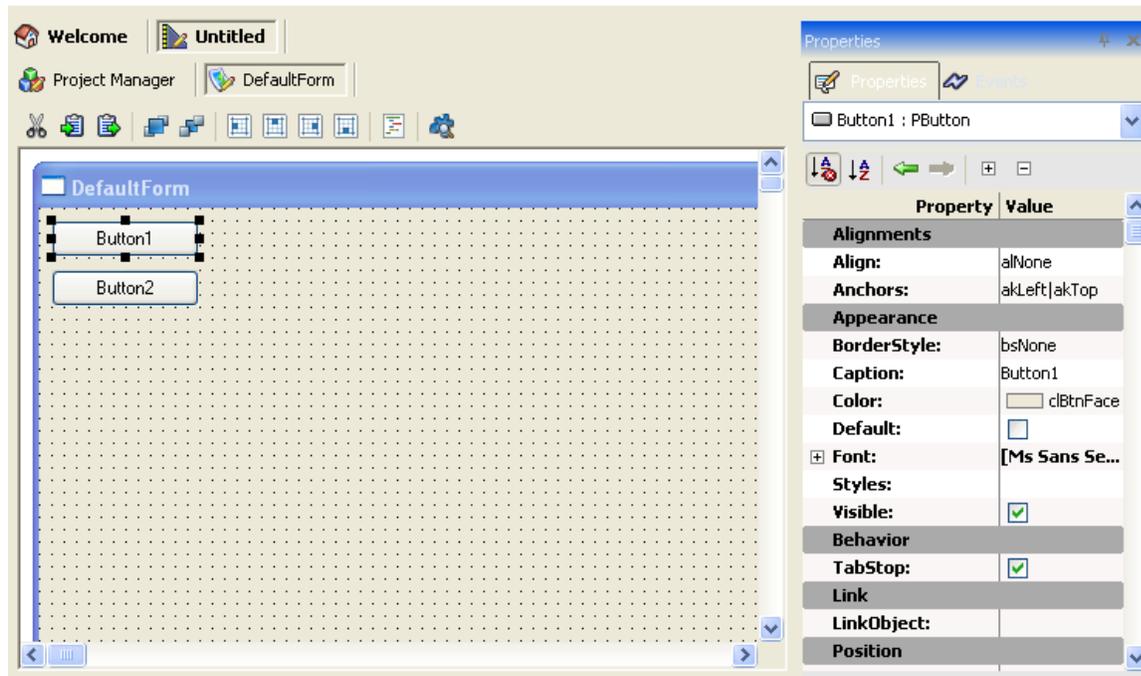


Figure 55: Selected Control in Visual Editor with Properties Open

You can edit the control of a selected code by clicking on the **Edit Code Button** from the Action Buttons or by double-clicking on the control.

Non Visual Elements

In many instances, application objects that have visual components (which can be edited using the visual layout tool) will also have non-visual components. You can see these components by clicking the **Non Visual Components Button** from the Action Buttons. This will open the Non Visual Components panel.

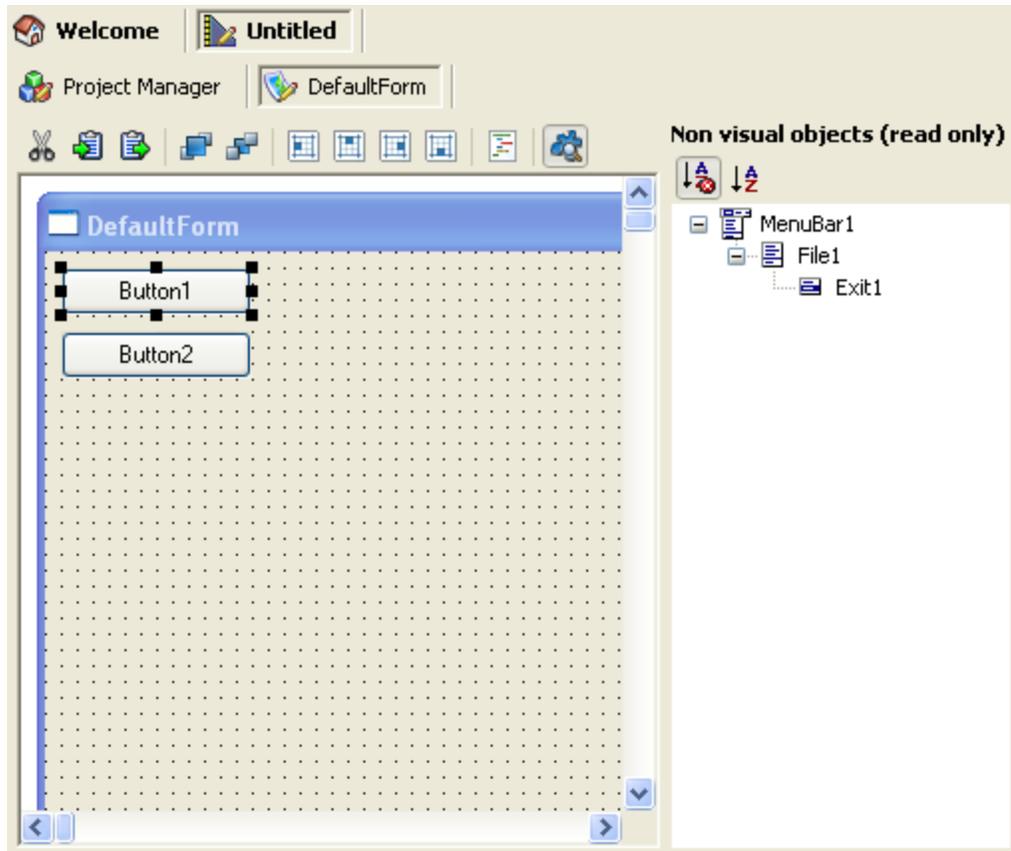


Figure 56: Non-Visual Elements in Visual Editor

Through the Non Visual Components panel, you can:

- Move a component up or down—select on the component you wish to move and click the **Up Arrow** or **Down Arrow**.

Events and Properties

You can edit the events and properties of a control by selecting it and modifying the values in the Properties panel.

Creating An Event

There are many ways by which you can create an event in PIDE. Given below are the steps by which you can create an event in PIDE:

- Method 1:
 1. Create a project and use the **Project Manager** view

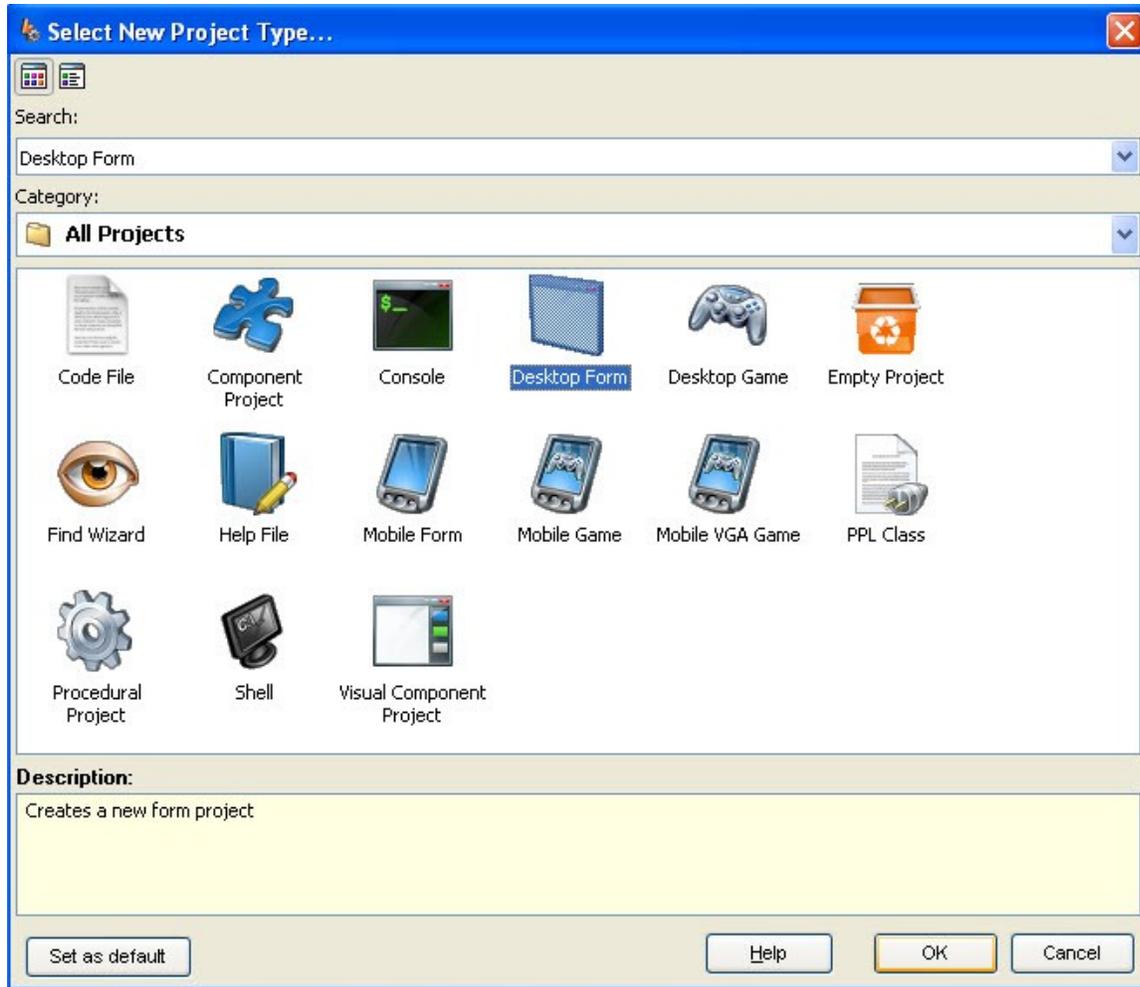


Figure 57: Creating New Project

2. Click on an **Object** and view the events panel

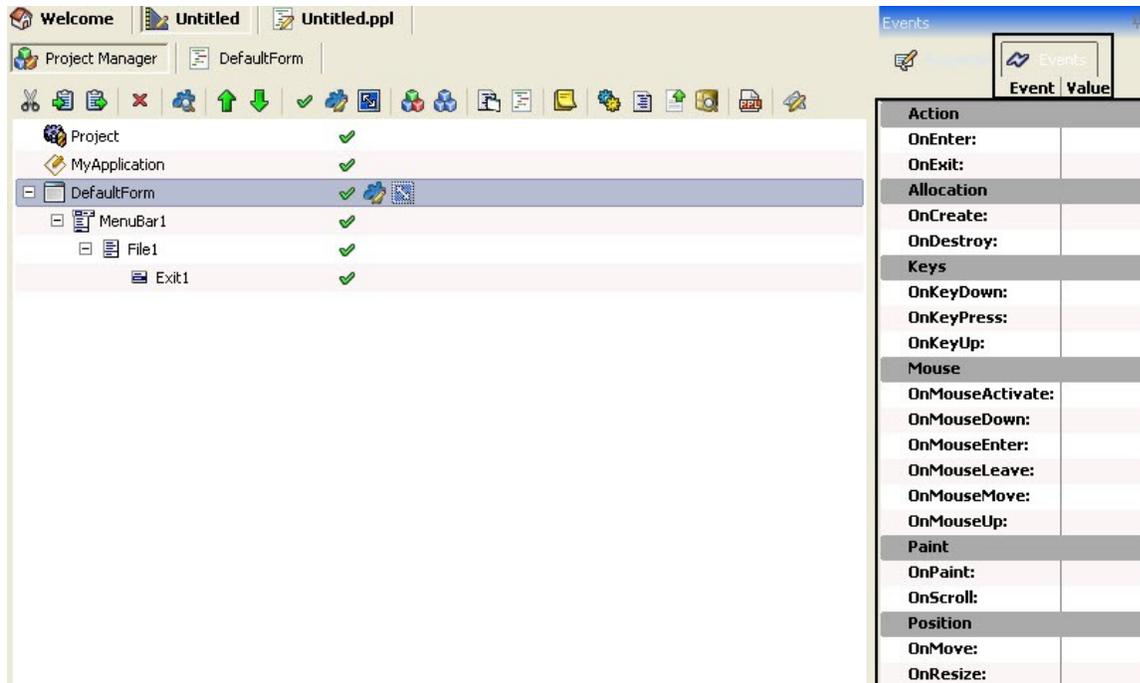


Figure 58: Events Panel

3. Double click the object event name in **Events Panel** to create an event

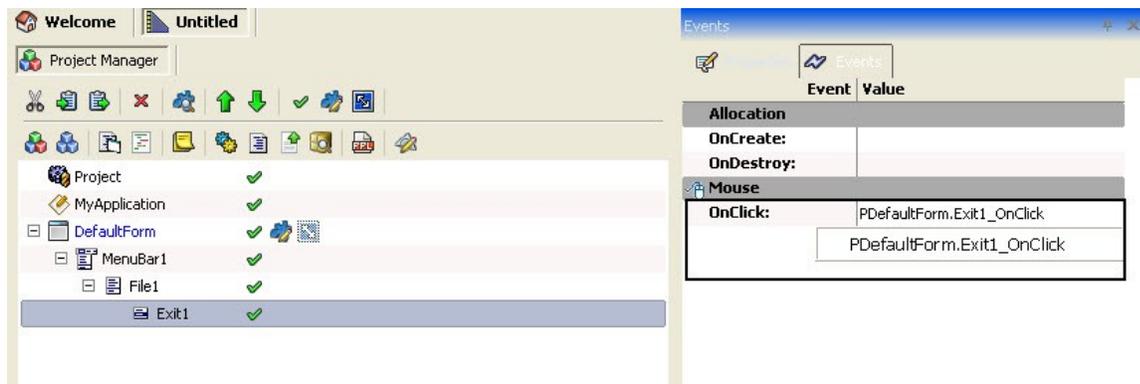


Figure 59: Creating an event

- Method 2:
 1. In the project manager, double click on an object to create a default event



Figure 60: Creating Default Event

- Method 3:
 1. Right click on the object and select **Events** from the context menu.
 2. Within Events, select the appropriate events to attach to an object.

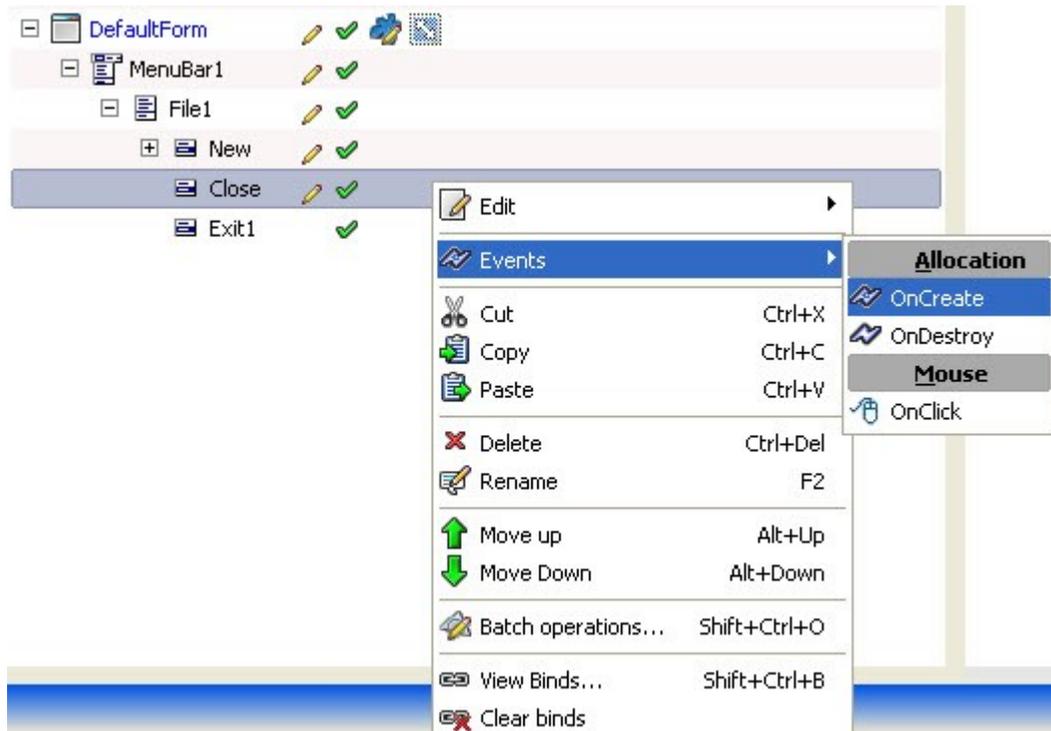


Figure 61: Selecting Events

Adding Objects within Events

After creating an event with any of the methods specified above, you can add objects within events with ease. Follow the steps given below to add objects and methods in the event.

- After creating an event, drag an object to the event in the **Project Manager**.

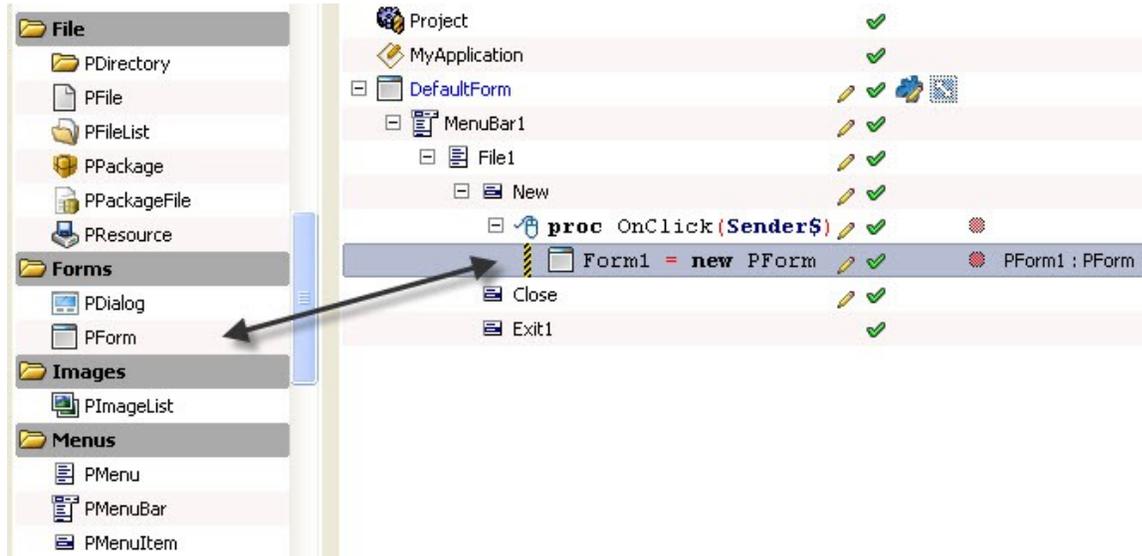


Figure 62: Drag to the event

- After adding an object to the event, you will be able choose methods that will apply to an object from the menu that appears.

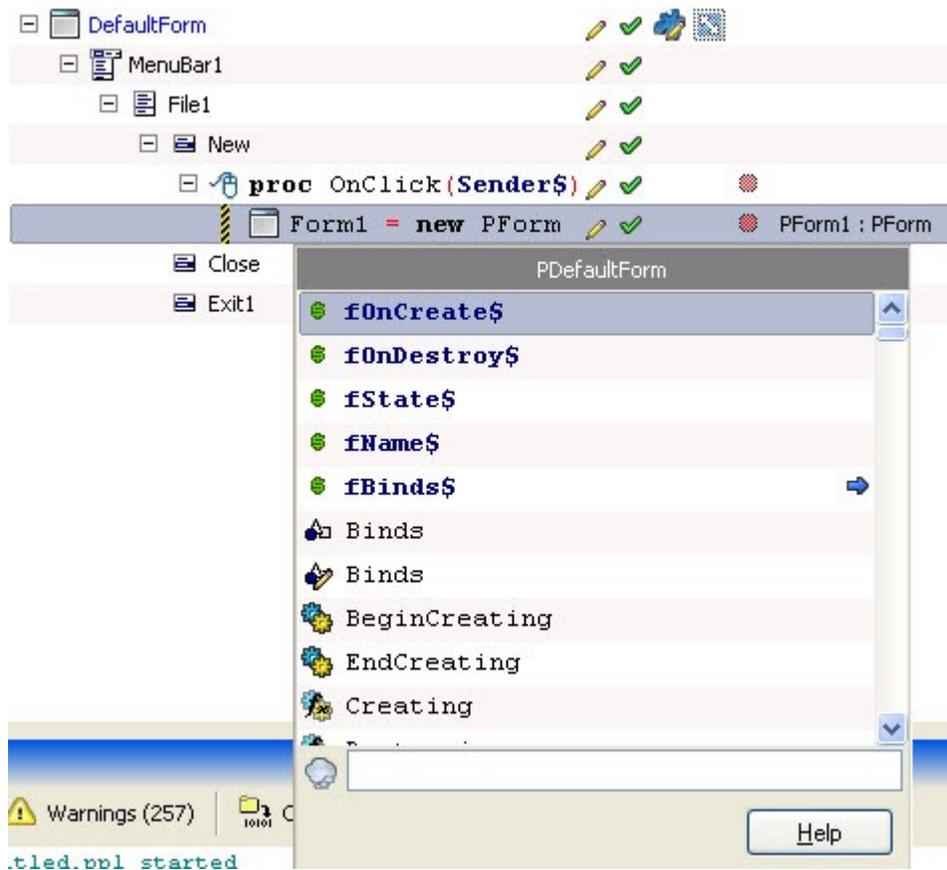


Figure 63: Choose a method

- If you have chosen a property, you will be able to set its value from **expr Property** field.

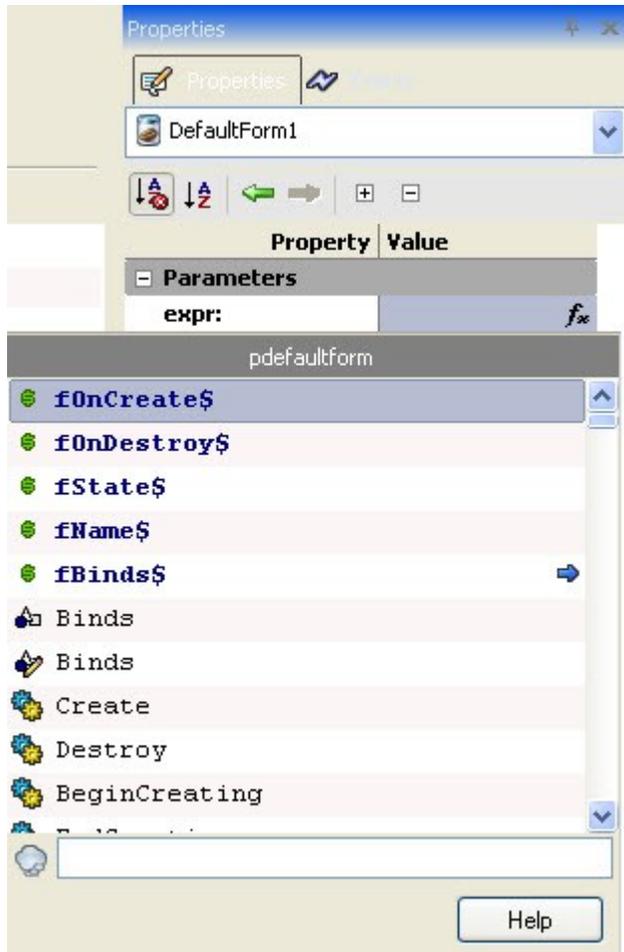


Figure 64: Using exper property

Note: Instead of searching for a new objet in the components panel, users can also use the Alt+Drag feature of PIDE to create an object belonging to the same class of the object from where it was dragged.

Testing Projects

As a robust development environment, PIDE includes a host of features to enable run-time debugging and testing. The PIDE testing features are categorized as follows:

- Design-time testing,
- Run-time testing,
- Profiling,
- Watches, and
- Test/Debug Console.

Design-Time Testing

The PIDE design-time testing features include processes and actions that can be taken as you develop your application. These include:

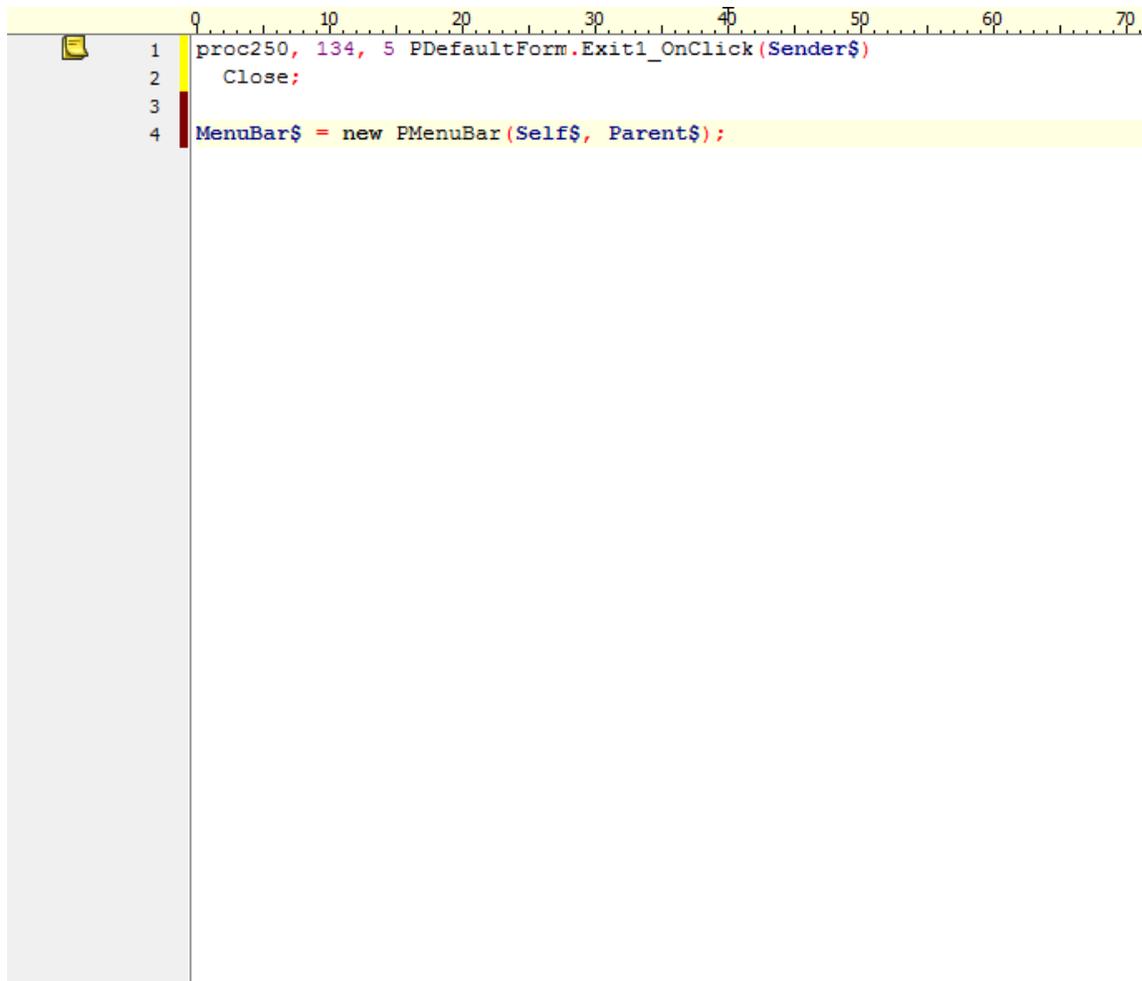
- Breakpoints, and
- Code stepping

Breakpoints

When viewing the source code for your project, you can easily set breakpoints where your code will stop upon executing. This is especially useful if you are testing to see outputs of actions (i.e., clicking a button) or data capture.

To set a breakpoint,

- First, open the source code of the project component by right-clicking and selecting the **View Source Code** option from the context-sensitive menu.



The image shows a code editor window with a ruler at the top. The ruler is marked from 0 to 70 in increments of 10. The code is as follows:

```
1  proc250, 134, 5 PDefaultForm.Exit1_OnClick(Sender$)
2      Close;
3
4  MenuBar$ = new PMenuBar(Self$, Parent$);
```

A red dot is positioned to the left of line number 4, indicating a breakpoint. The line containing the code for line 4 is highlighted in yellow.

Figure 65: Viewing the source code

When your source code is displayed, you can set a breakpoint by

- clicking to the left of the line for which you wish to add a break, or
- selecting the **Toggle Breakpoint Option** from the **Run Menu**.
- Both methods will add a red dot (to the left of the line number you selected) indicating a breakpoint is now present on that line.

You can also manage all of your breakpoints (throughout the project) by...

- selecting the **Breakpoints... Option** from the **Run** menu or
- pressing CTRL+B
- Both of these options will open the **Breakpoints window**.

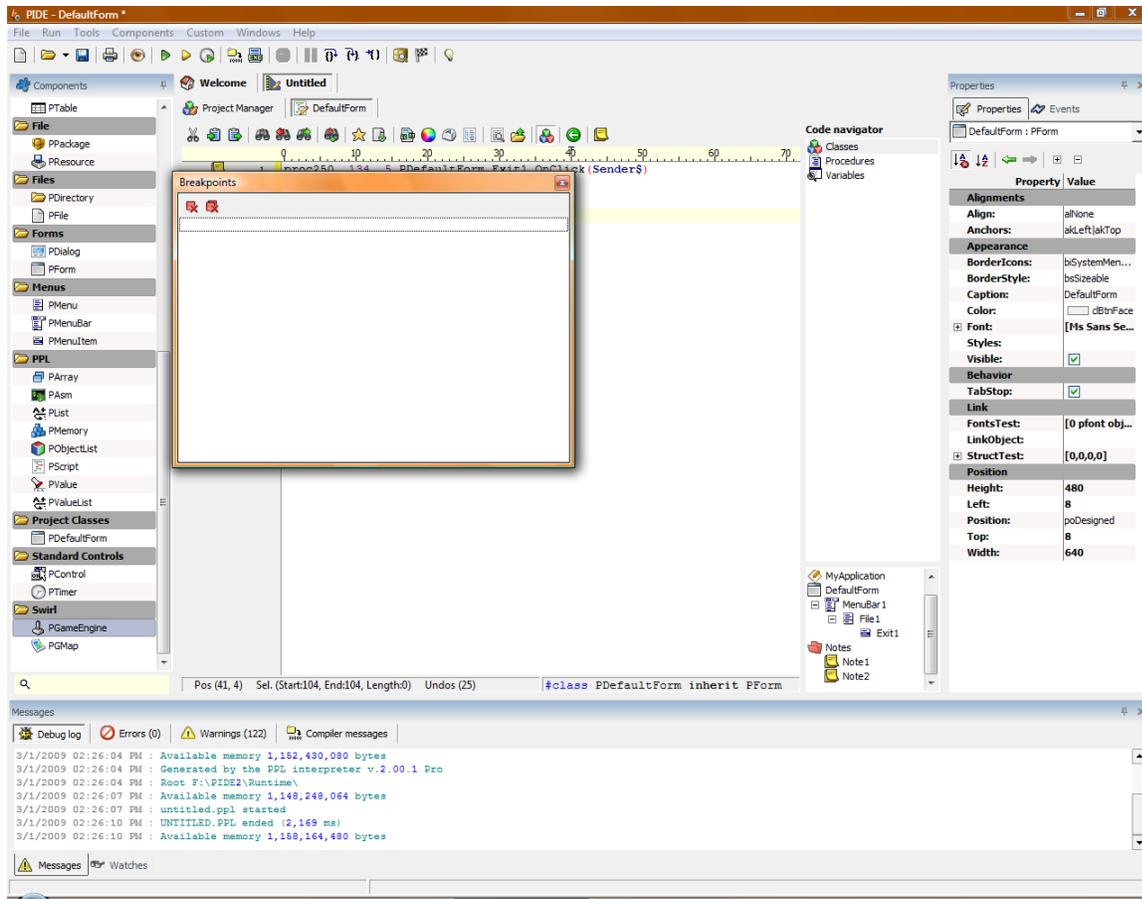


Figure 66: breakpoint window

The Breakpoints window makes it easy to manage your breakpoints across the entire project. Through the Breakpoints window, you can

- Delete individual breakpoints by selecting them and clicking the **Delete Breakpoint Button** or
- Delete all breakpoints by clicking the **Delete All Breakpoints Button**.

Note: when running your code with breakpoints, output will appear in the Debug Console at the bottom of the PIDE interface. For more information about this console, see the appropriate section in this chapter.

Code Stepping

Code stepping is a feature that allows you to move line-by-line through your code in debugging mode, capturing valuable information about each line in the Debug Console.

PIDE provides three types of code stepping:

- Step to next line—step the running application to the next line (skipping any sub lines). *Note: if the next line calls a specific function, that function will be called in its*

entirety. This is good if you already know the function works.

- Step to next line within procedure or function—step the running application to the next line within a procedure or function (allowing you to test sub-lines). *Note: this is good to test individual functions and their output.*
- Run to current line—this runs the entire application up to the currently selected line. *Note: this is great to test if a specific line is receiving information from previous lines/functions.*

*Note: when you utilize code-stepping, you automatically put PIDE into a debugging state, represented by the **Stop Button** in the main toolbar. To exit the debugging environment, click the **Stop Button**.*

Note: all debugging information during code stepping will appear in the Debug Console. For more information about the Debug Console, see the appropriate section in this chapter.

Run-time Testing

Run-time testing allows you to see application-level output during run-time (as if the application were actually running on the user's device).

To enable run-time testing,

- Click the **Run Button** or
- Select the **Run Option** from the **Run Menu**.
- Both of these options will put your application into a run-time simulation, capturing all run-time problems in the Debug Console.

*Note: you can also prevent run-time checks from happening (i.e., if you have a known problem that does not impact functionality and want to work with other parts of the application's run-time experience) by clicking the **Run Without Run-Time Check Button** or selecting the **Dedicated Run Option** from the **Run Menu**.*

Profiling

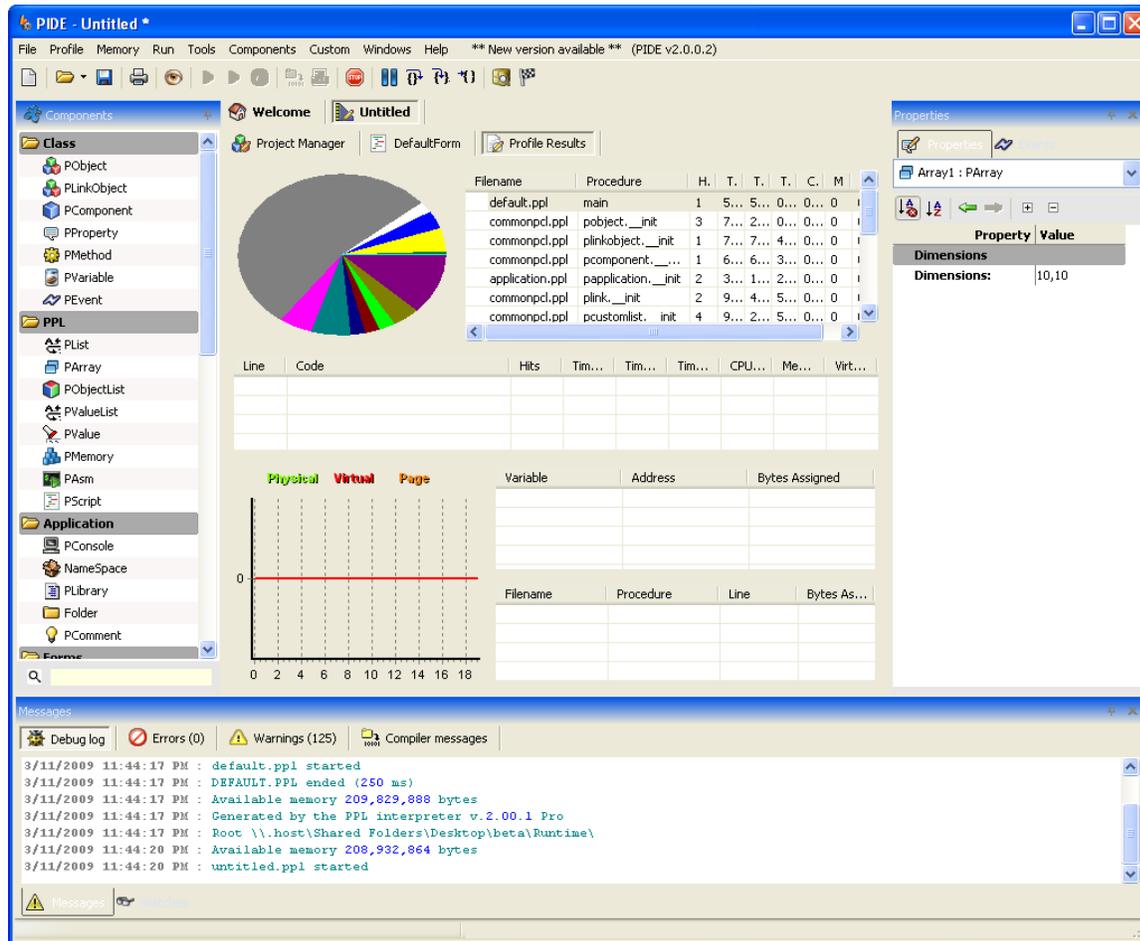


Figure 67: Profile Report

Profiling allows you to get detailed information about the performance of your application. Vectors that PIDE tracks during debugging include:

- Application memory consumption,
- Time spent processing each line of code,
- Time spent processing each procedure, and
- CPU utilization for each line of code and procedure.

To run a profile, select the **Profile Option** from the **Run Menu**.

Watches

Watches allows you to see variable assignment at run-time. This is a great way to double-check that your variables are populated with expected value and data as your application executes.

Debug Console

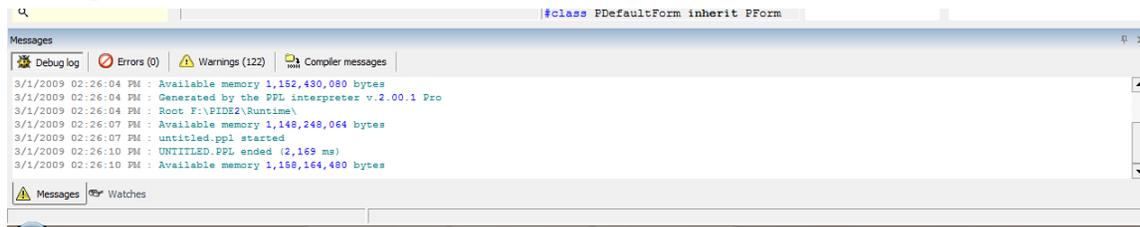


Figure 68:Debug tab

The Debug Console is a powerful feature within PIDE that captures design-time testing and run-time testing debugging information. The console is divided into four tabs:

- Debug,
- Errors,
- Warnings, and
- Compiler Messages

Debug Tab

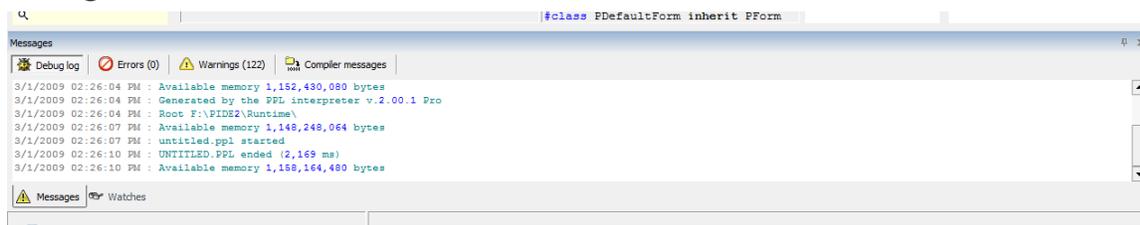


Figure 69: Debug tab

The debug tab provides output information as the application runs. To access the Debug portion of the Debug Console, click on the **Debug Tab**.

Errors Tab

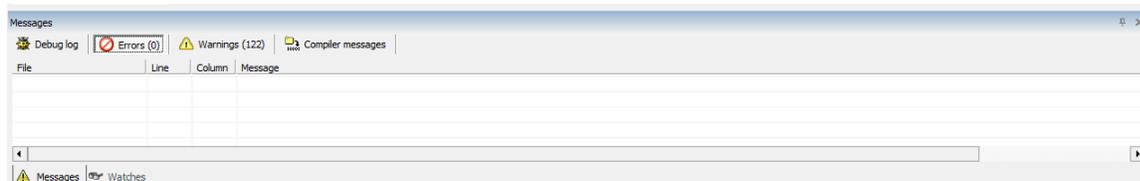


Figure 70:Errors tab

The errors tab provides specific errors in the application that prevent it from running. To access the Errors portion of the Debug Console, click on the **Errors Tab**.

Note: To jump to the code that caused the error, simply double-click on the error message. This will open the source code window of the appropriate component.

Warnings Tab

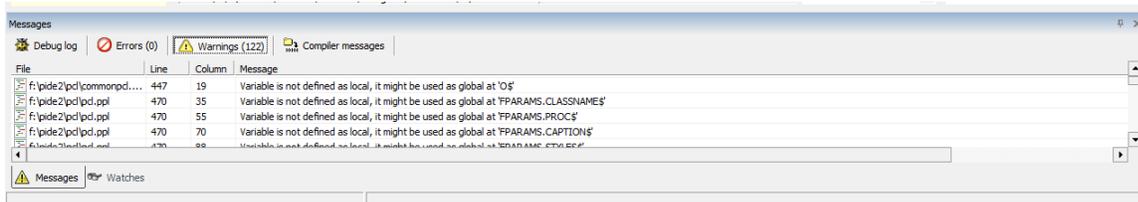


Figure 71: Warnings tab

The warnings tab provides information about application behavior during execution that won't cause the application to fail but may cause long-term issues (i.e., memory leakage, data formatting, etc.) To access the Warnings portion of the Debug Console, click on the **Warnings Tab**.

Note: To jump to the code that caused the warning, simply double-click on the warning message. This will open the source code window of the appropriate component.

Compiler Messages Tab

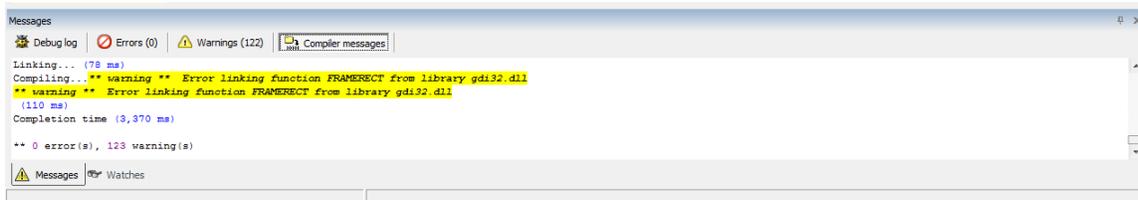


Figure 72: Compiler tab

The compiler messages tab provides information about messages generated during the compilation of the application. To access the Compiler Messages portion of the Debug Console, click on the **Compiler Messages Tab**.

Note: This also includes success messages

Component Library

In an effort to enable developers to more quickly create and launch their applications, PIDE includes a robust component library that allows for rapid software development.

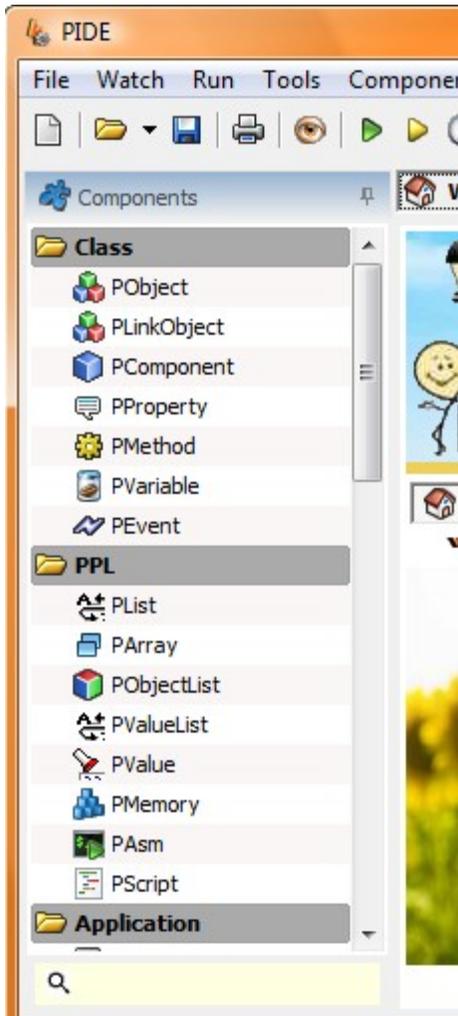


Figure 73: Components tab

The Component Library is the left-hand panel of the interface and is divided into the following categories:

- Class,
- PPL,
- Application,
- Forms,

- Files,
- Database,
- File,
- Standard Controls,
- Menus,
- Swirl,
- Codeflow,
- Project Classes, and
- Unknown.

In addition to drag-and-drop capabilities, you can also access property information about each component. To access a component's properties,

- Hover your mouse over the component in question. This will display the context-sensitive menu.

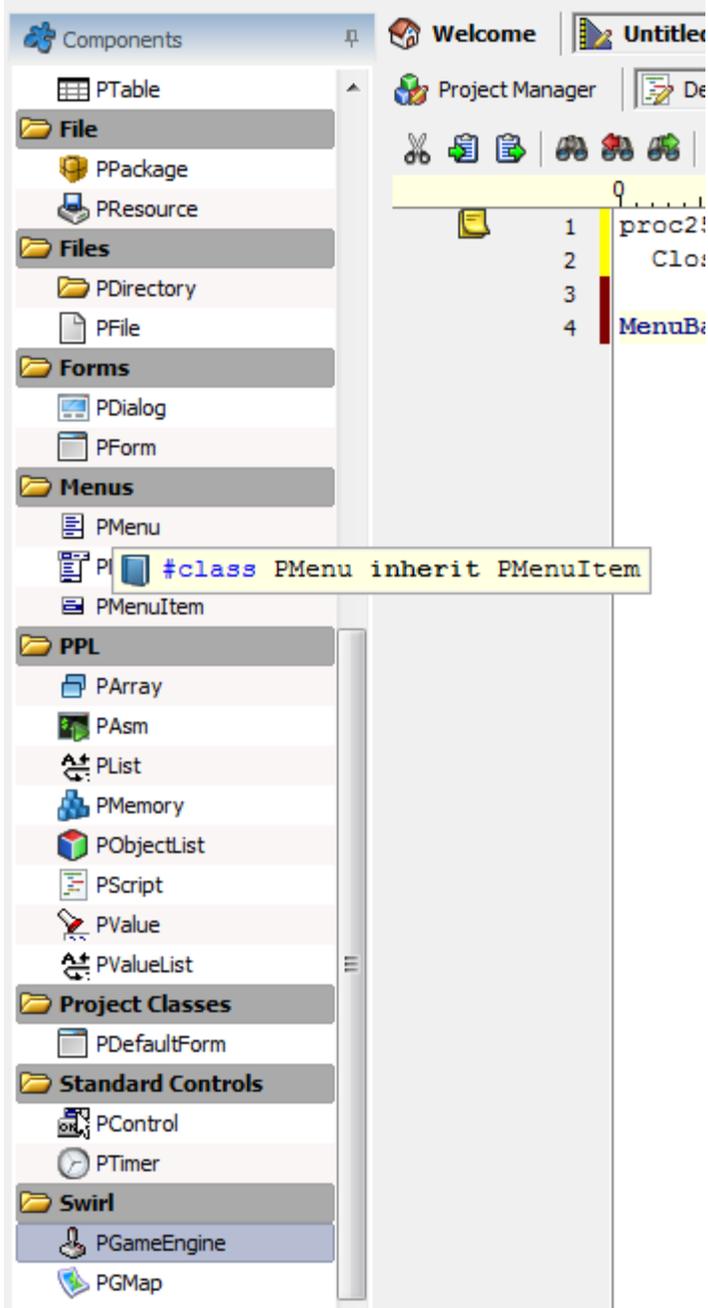


Figure 74: Components tab hover feature

- Click on the **Book Icon**. This will open the **Hint Window**.

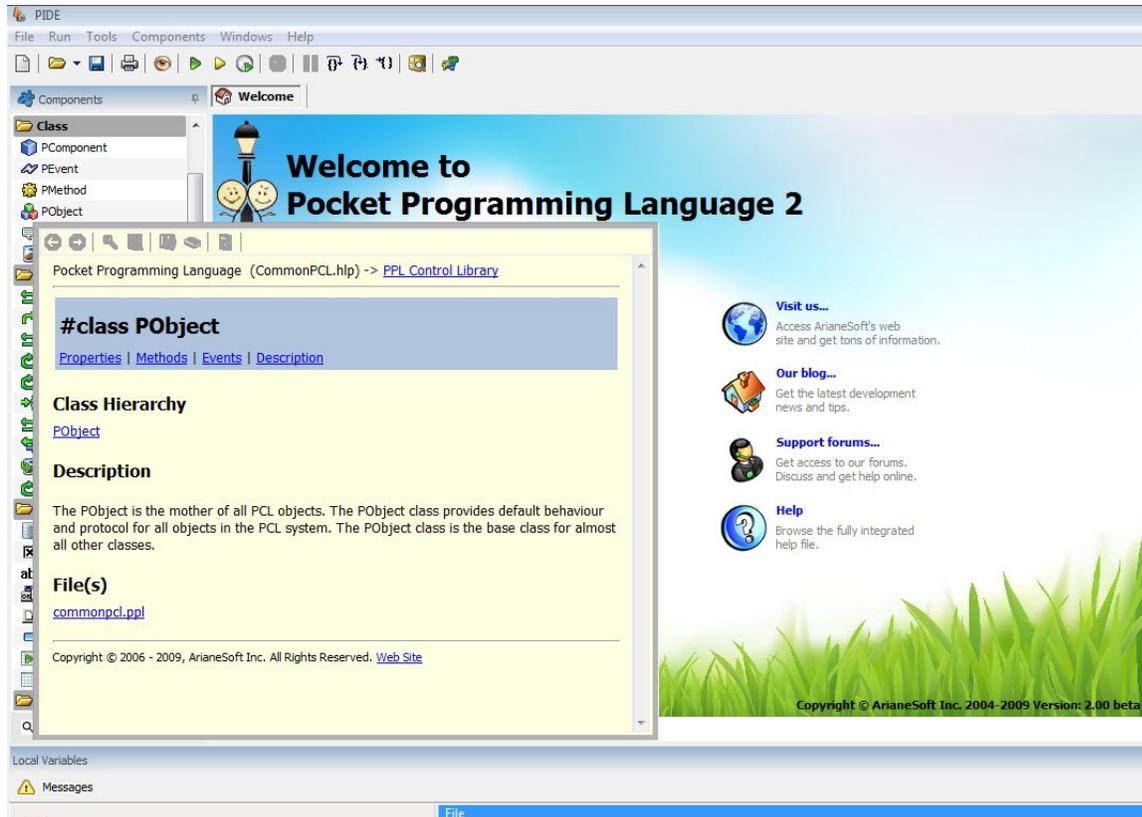


Figure 75: Hint Window for Component Properties

The Hint Window displays the following information about the component:

- The class to which the component belongs
- Any inheritances
- The position of the component within the class hierarchy
- .ppl Code files associated with the component, and
- Related objects

The information displayed in the Hint Window is often a link that leads to additional information (such as related objects). When links within the Hint Window are click, the additional information will be displayed.

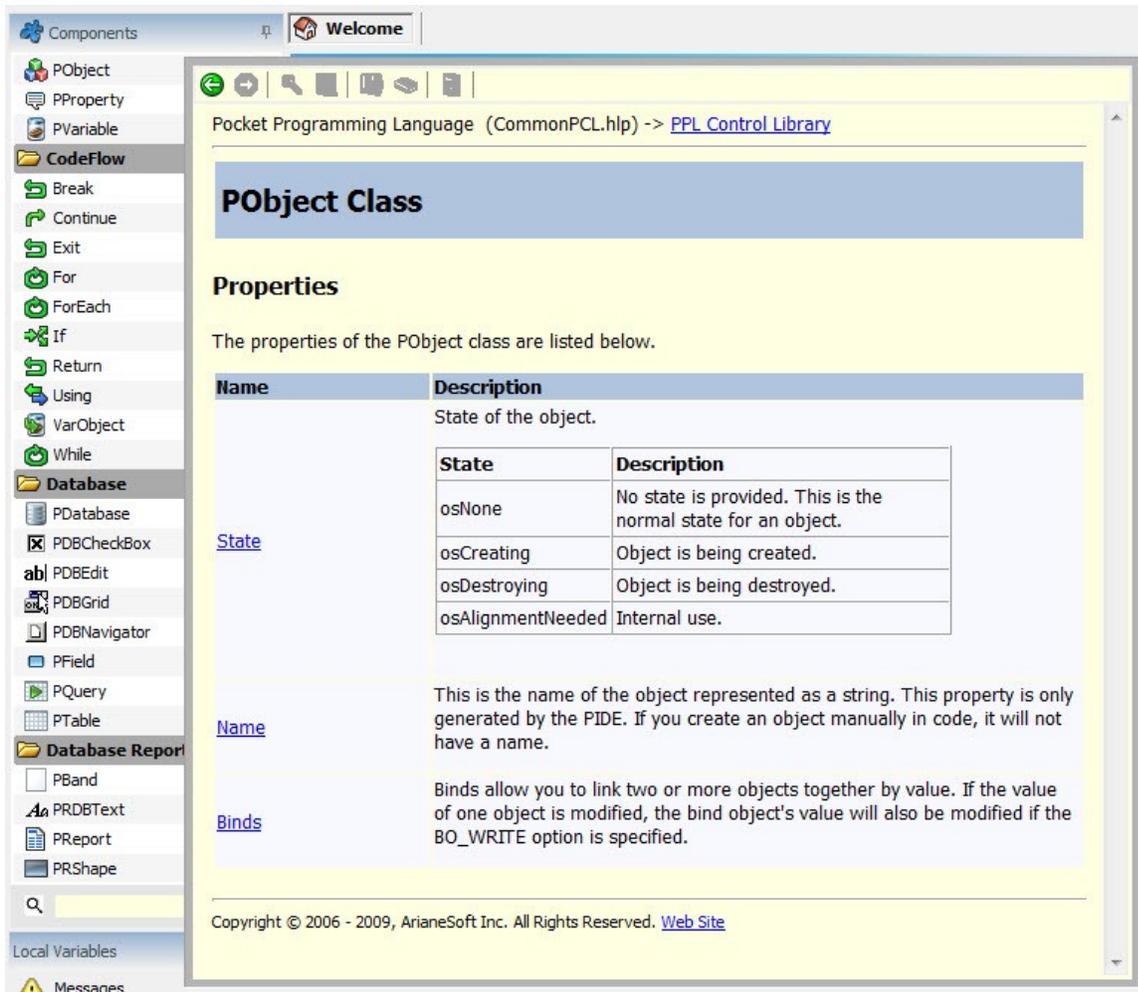


Figure 76: Hint Window with Navigation

The Hint Window includes navigational elements to help you move through the additional pages as you click on links. The illustration above shows the available back arrow as a link as been clicked from the previous screen in the Hint Window.

- To return to the previous information, click on the **Back Button** in the navigation bar
- To move forward to a page that you have visited, click on the **Next Button** in the navigation bar

Finally, you can open the actual code of a component by clicking on the .ppl link within the Hint Window. This will open the code within the code editor, allowing you to make changes to a component. *Note: this is especially useful if you have added your own components to the library.*

What is a Component?

A component in the PIDE software is composed of two elements:

- One or more .ppl files (source code), and
- XML definitions file.

These two elements are combined into a **Component Package**.

Class

PIDE comes installed, by default, with a number of components already available in the library. You can get more information on these components by hovering your mouse over the component and clicking on the book to open the component help pop-up window.

Adding Custom Components

Through the Component Library, you can also your own components (or additional components released by other developers or ArianeSoft). To install a new component,

- Select the **Install New Component(s) Option** from the **Components Menu**. This will open the **Choose a Component Window**.

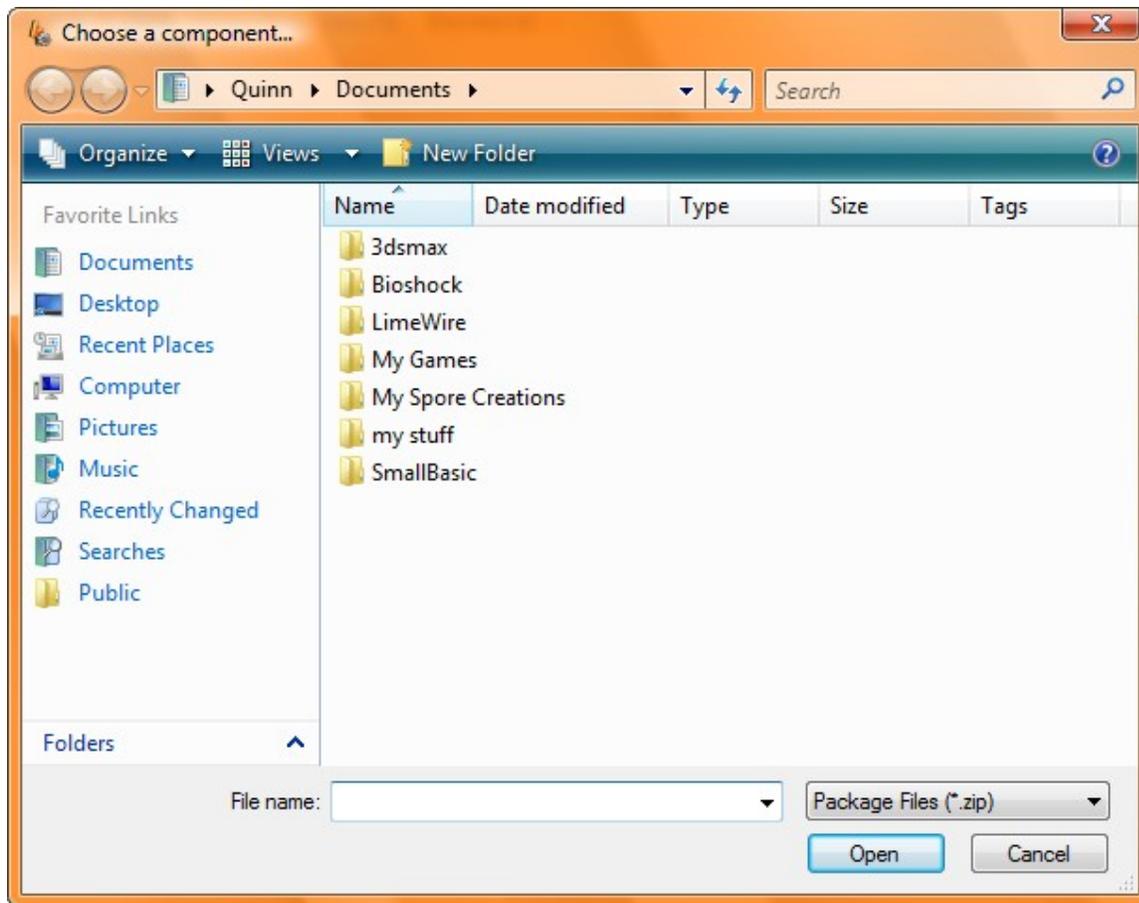


Figure 77: installing a new component

The Choose a Component Window allows you to select a component package (.zip) file from your hard drive. After you have located the file, click the **Open Button** to load the component into your library.

Managing the Component Library

There may be times when you want to remove components from your library. This is accomplished by uninstalling the component. To uninstall a component from your library,

- Select the **Uninstall Component Option** from the **Components Menu**. This will open the **Selection Component Definition File to Uninstall... Window**.

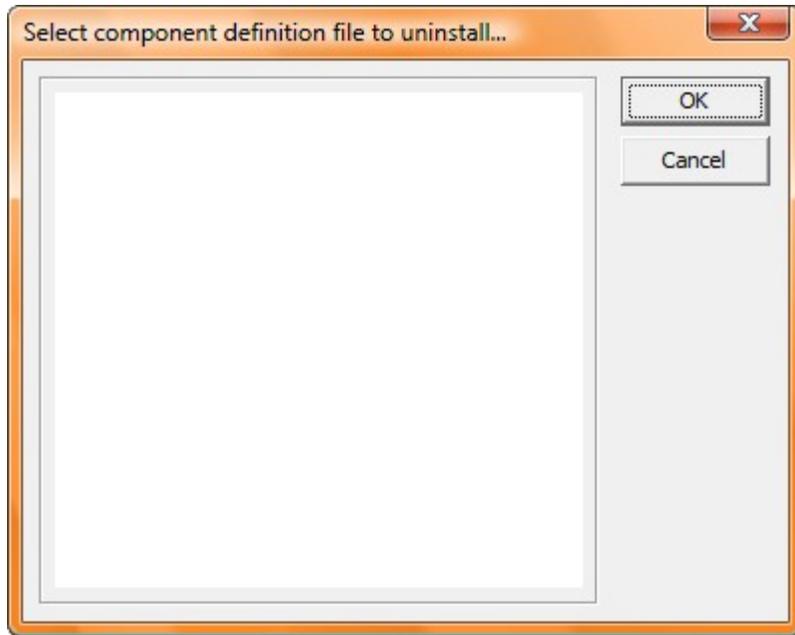


Figure 78: Uninstalling components

- To uninstall a component, select the definition file of the component to remove and click the component. *Note: to select more than one component to remove, press the CTRL key while you are clicking components.*

Once you remove or add components, it's a good idea to refresh the library. This is accomplished by selecting the **Reload Components and Help File Option** from the **Components Menu**.

Creating Your Own Components

You can also create your own components and provide them to other PIDE developers for inclusion in their library.

To create a component, you must package the following:

- .ppl files (source codes),
- Help files, and
- XML definition files

XML Definition Files

PIDE includes a built-in function that will automatically generate a XML file from a .ppl file. To generate your XML definition file,

- Select the **Create XML From PPL File... Option** from the **Components Window**. This will open the **Open a File Window**.

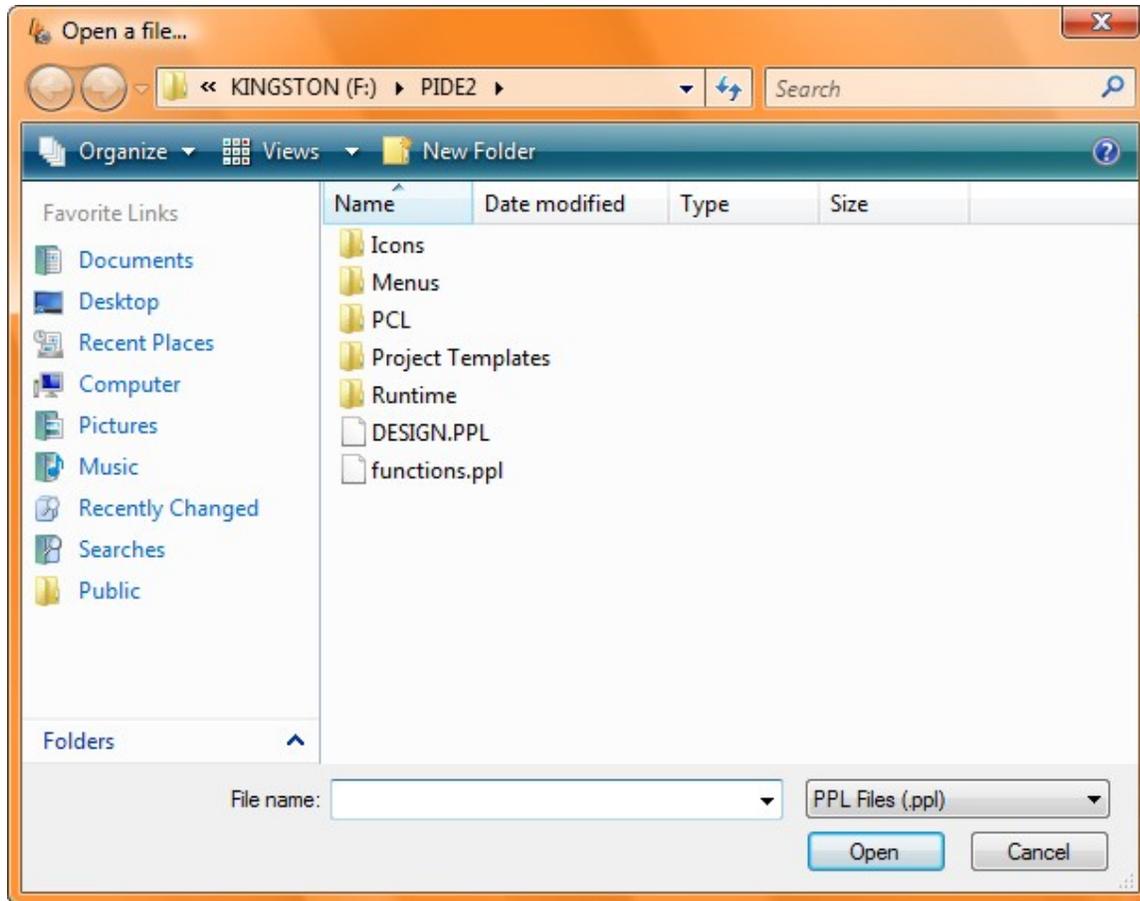


Figure 79: creating a xml from a .ppl file

The Open a File Window allows you to search your hard drive for the .ppl file. Once you have identified it, click the **Open Button**. This will open the **Save As Window** through which you can specify a name and location for the .xml file.

Generate the Component Package

Once you have the .xml file definition 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**.

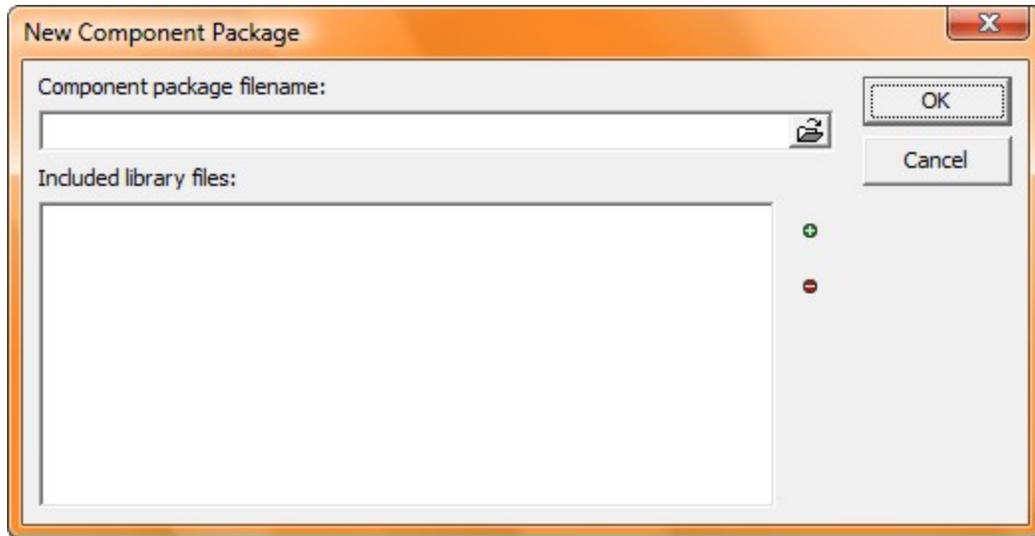


Figure 80: creating a new component package

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.

Creating classes from current project in components panel

Creating components is very important not only because it allows a programmer to create his/her own components, it is also useful because it allows a programmer to reuse that component as many times as he/she wants. If you are working on a project and want to create components from it so that you can use those same components in other projects, you can use the '**Install components from current project**' button available in the **Components** menu.

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 panel.

- Create a new **Component Project** in PIDE2 and delete the existing components class as we will be creating that on our own. Drag a **PToolBar class** to the project and rename it as DbNavigator to make it look like a different class.

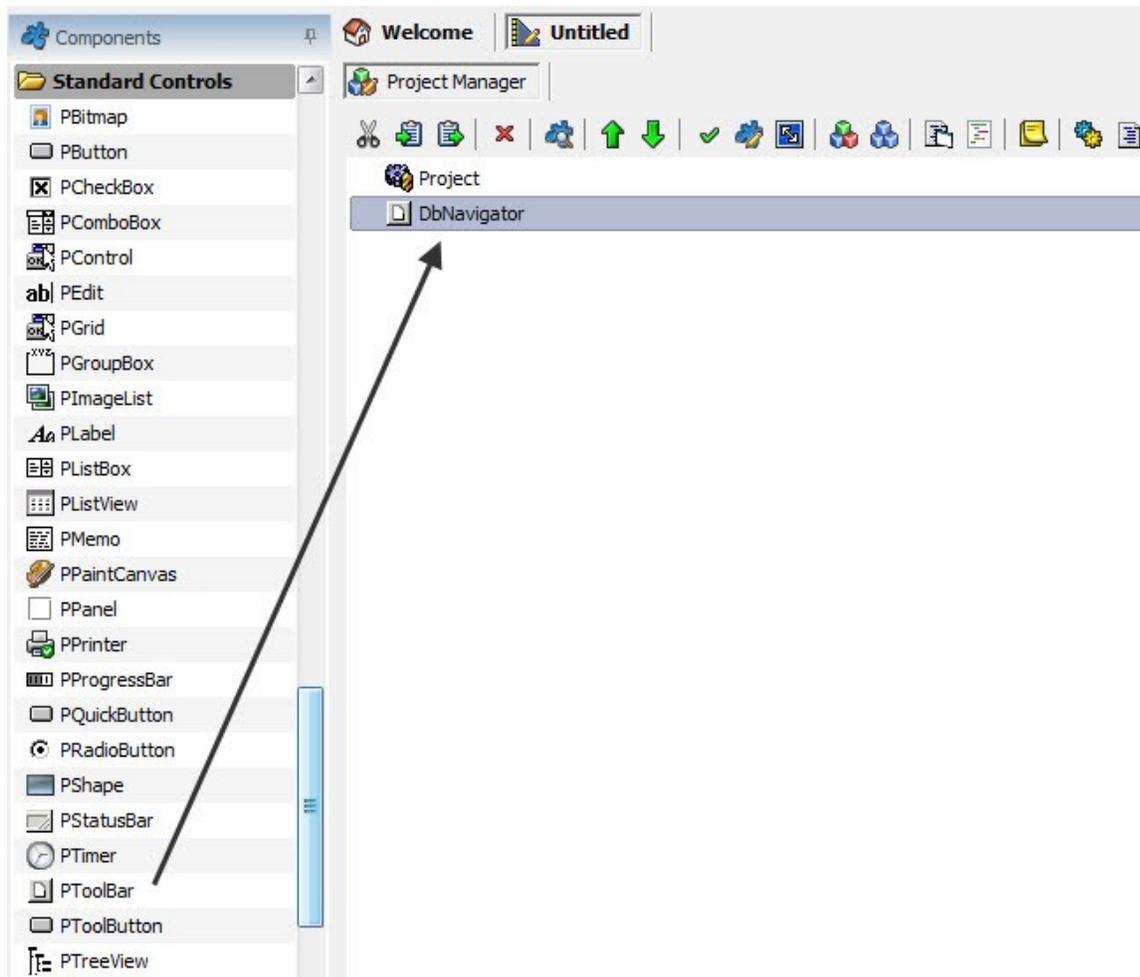


Figure 81: Drag a PToolBar

- 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**.

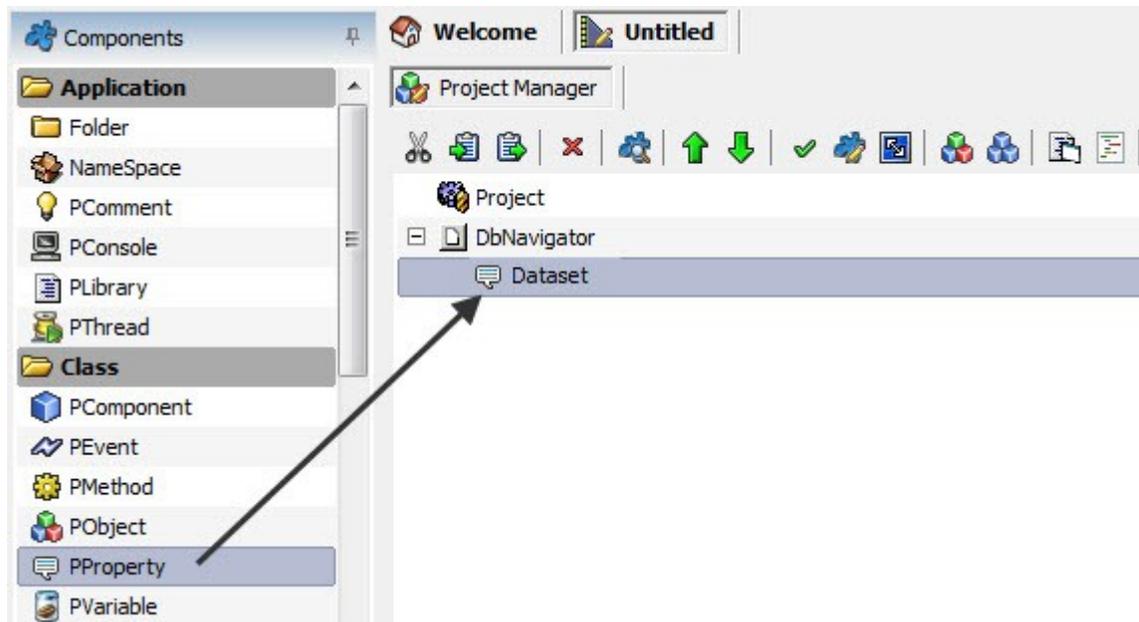


Figure 82: Drag PProperty to DbNavigator

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

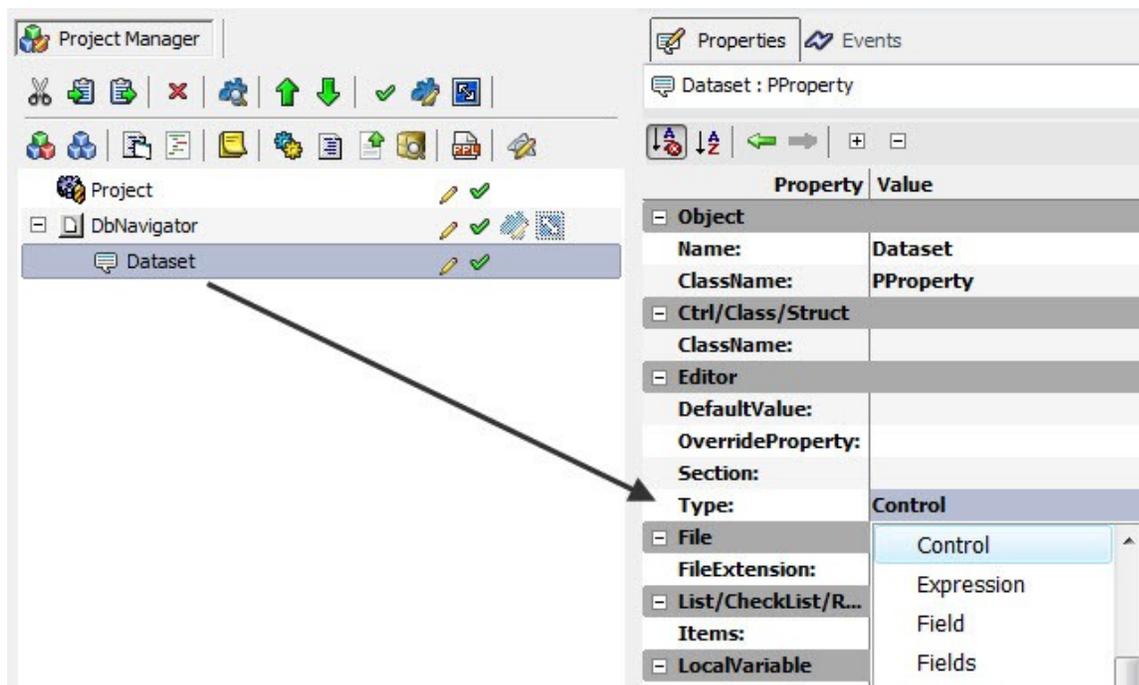


Figure 83: Set the Type property

- After changing the Type property, select PDataSet in the **ClassName Property**.

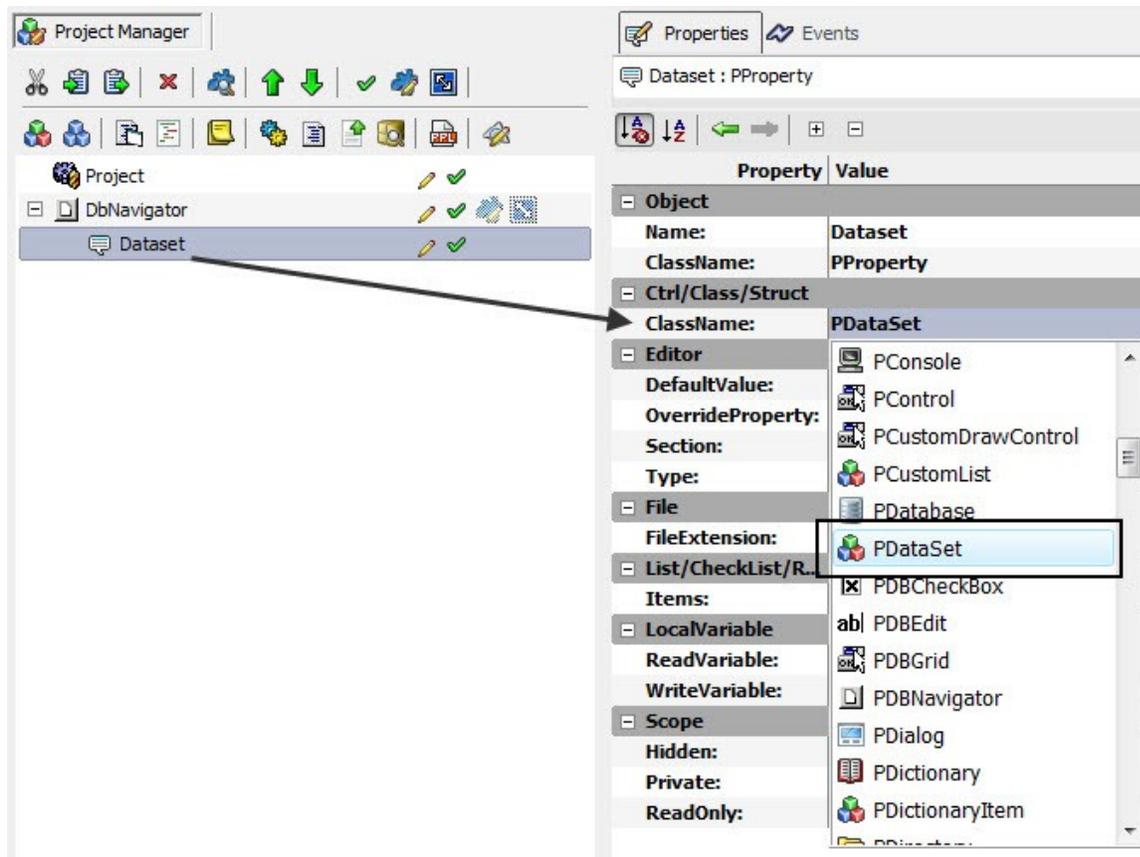


Figure 84: Set the ClassName property

- 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**.

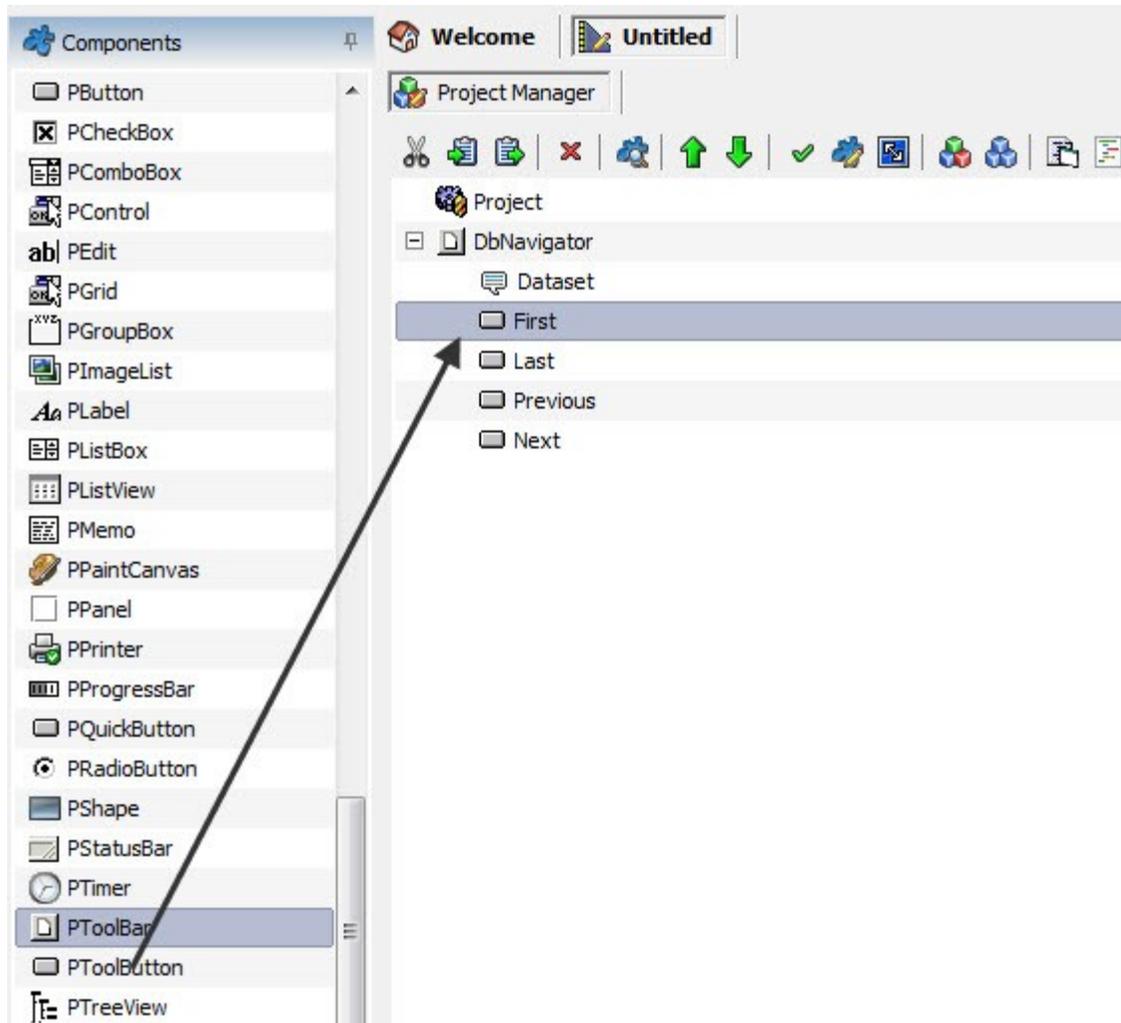


Figure 85: Drag four PToolBar Objects

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

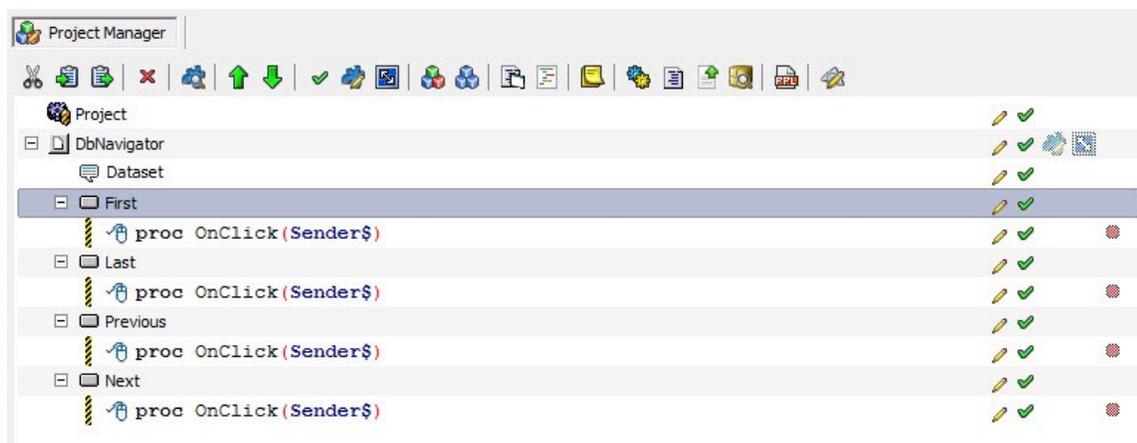


Figure 86: Double click the buttons to create events

- 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.

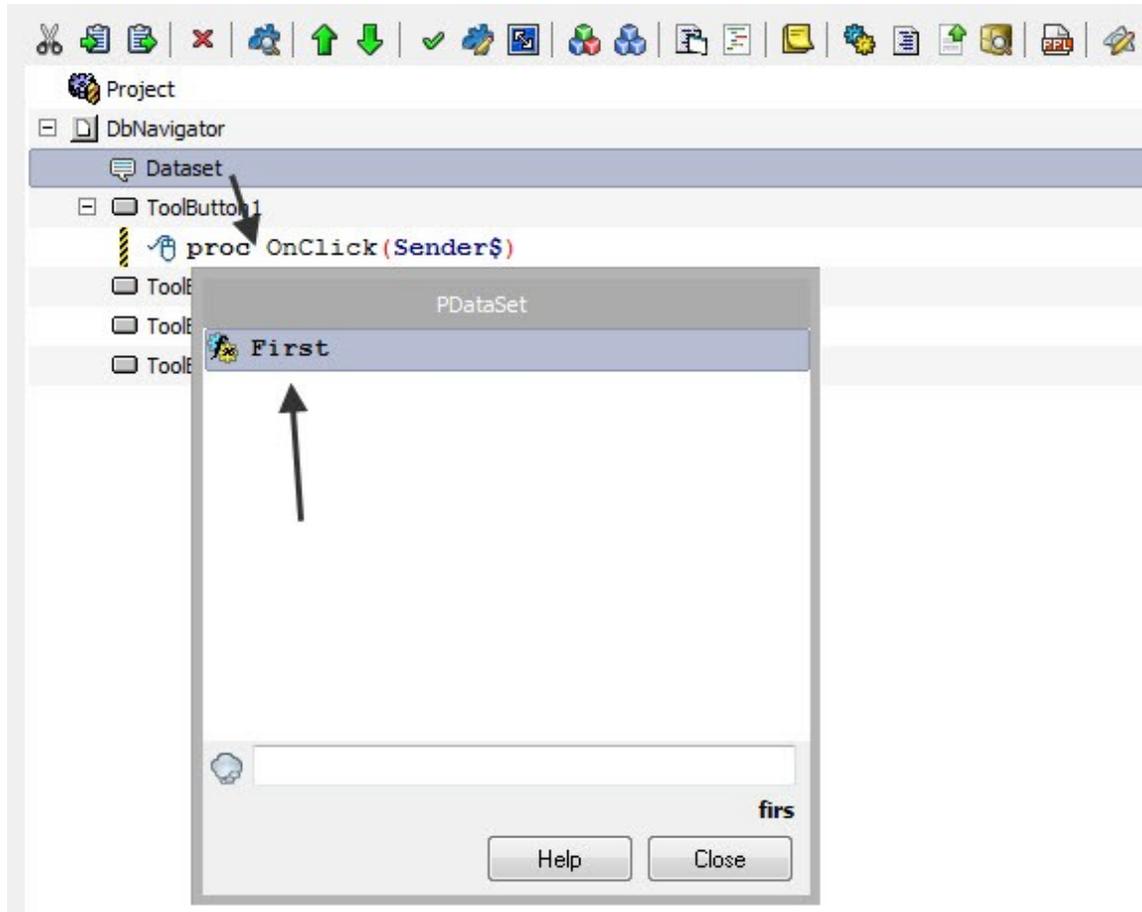


Figure 87: Select a message

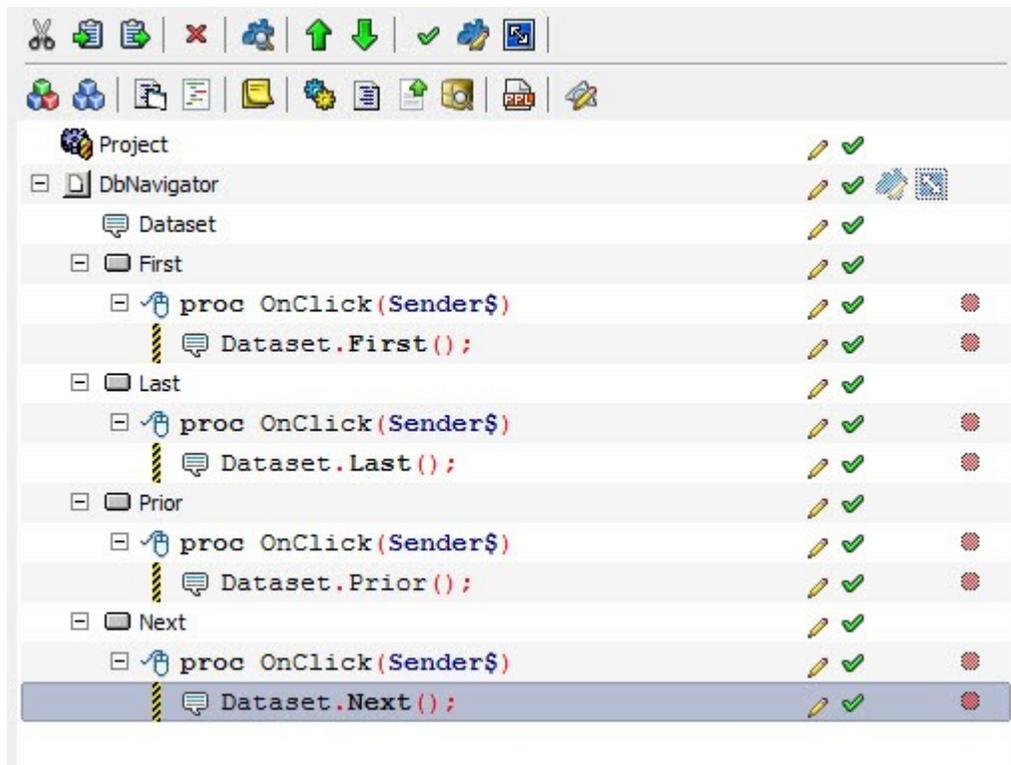


Figure 88: Select messages for all buttons

- 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. 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.

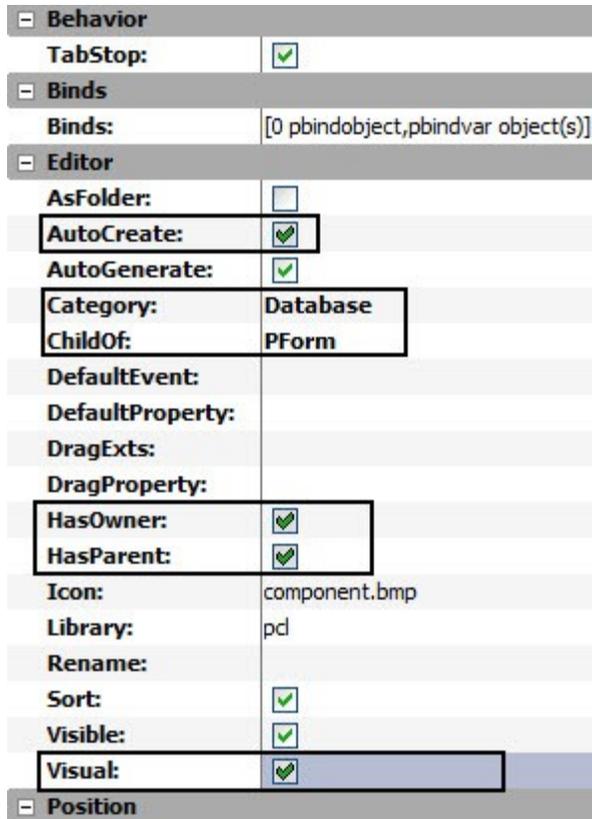


Figure 89: The property panel

- 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.

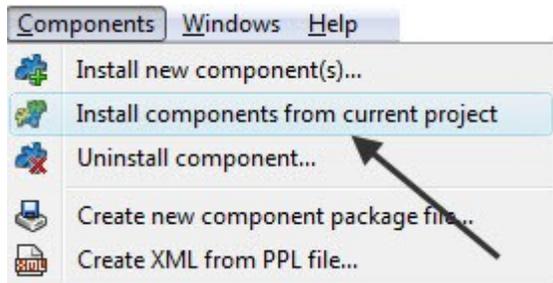


Figure 90: Components Panel

- 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.

Creating A SQLite Database In PIDE

Creating as well as manipulating SQLite database in PIDE is an easy task. Generating quick databases with PIDE is just a task of dragging and dropping the correct components from the component panel to the project manager. Given below are the guidelines that would help you in creating a SQLite database in PIDE.

- Create a **New Project**

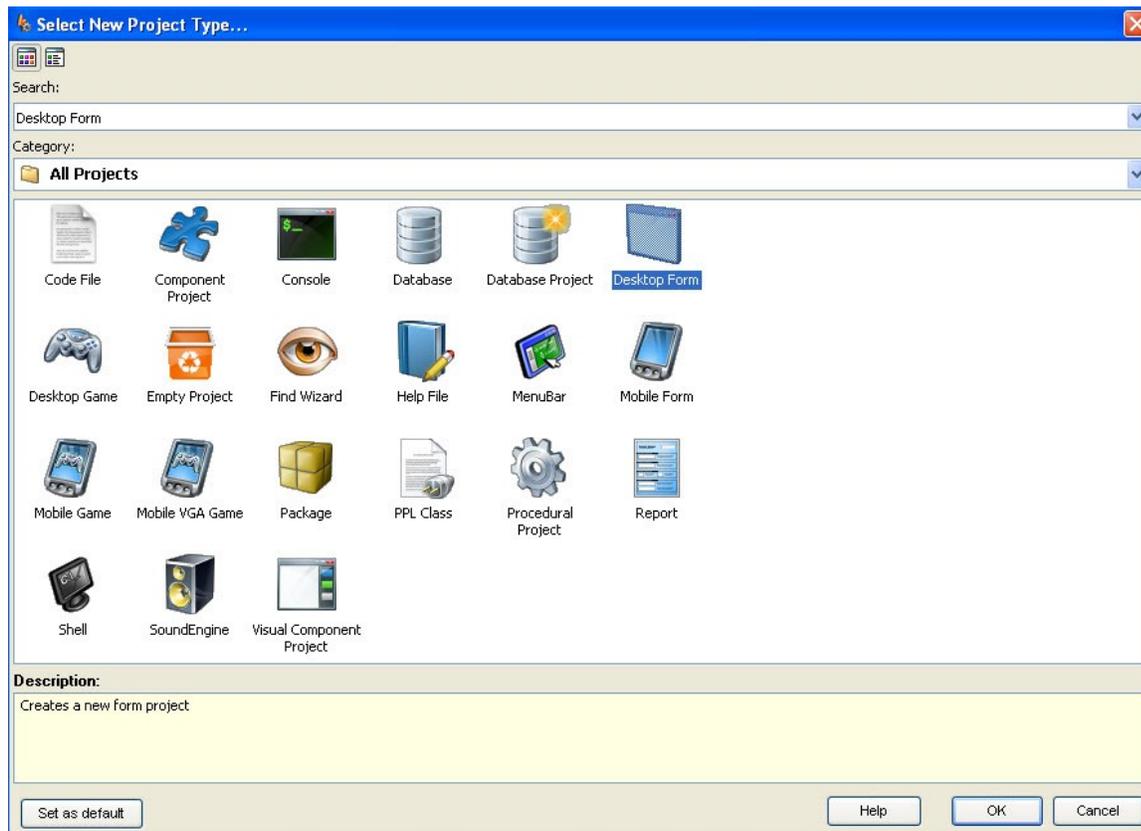


Figure 91: Creating new project

- Select a project of your choice and open it in the **Project Manager** view
- Drag **PDatabase Object** from the **Components Panel** to the project manager

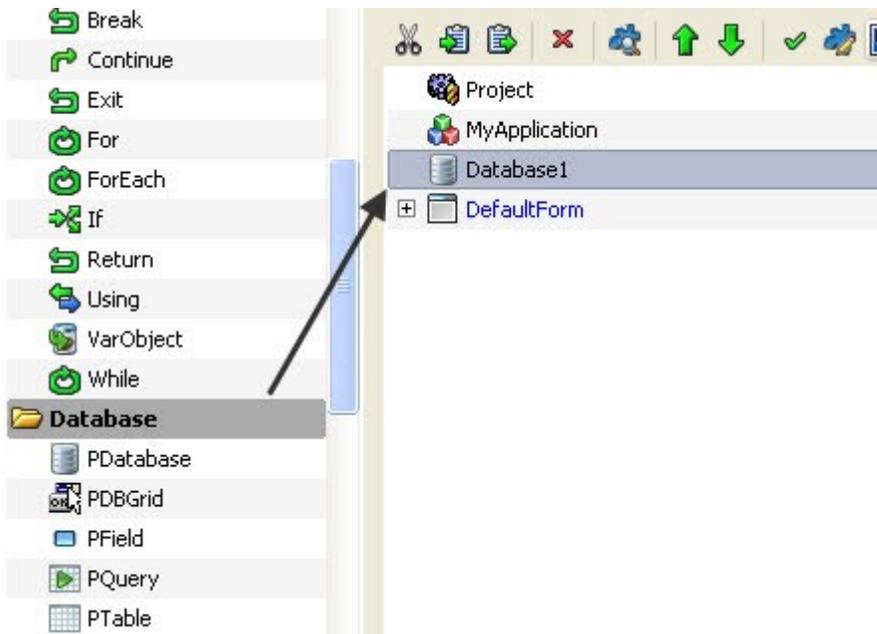


Figure 92: Drag a database

- Drag a **PTable Object** to the newly created **PDatabase Object**

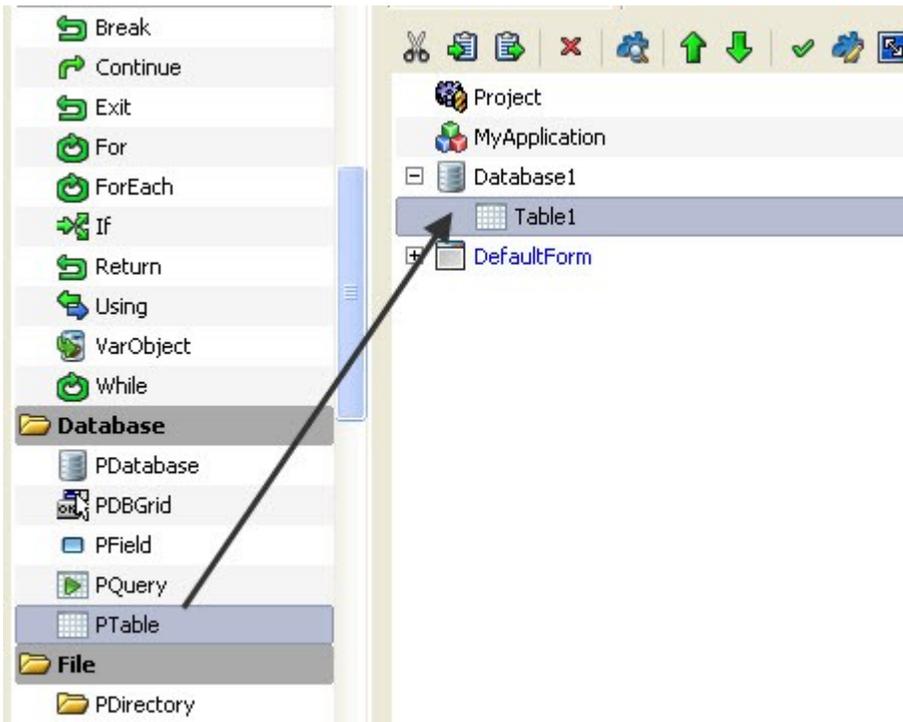


Figure 93: Drag a PTable

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

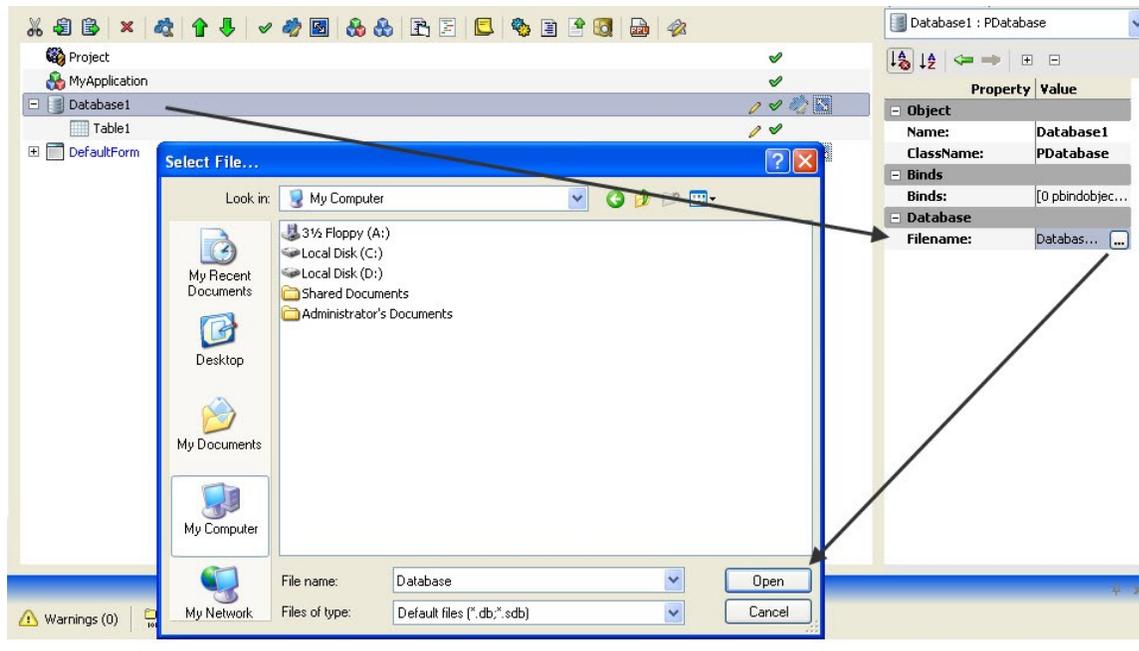


Figure 94: Specify a filename

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

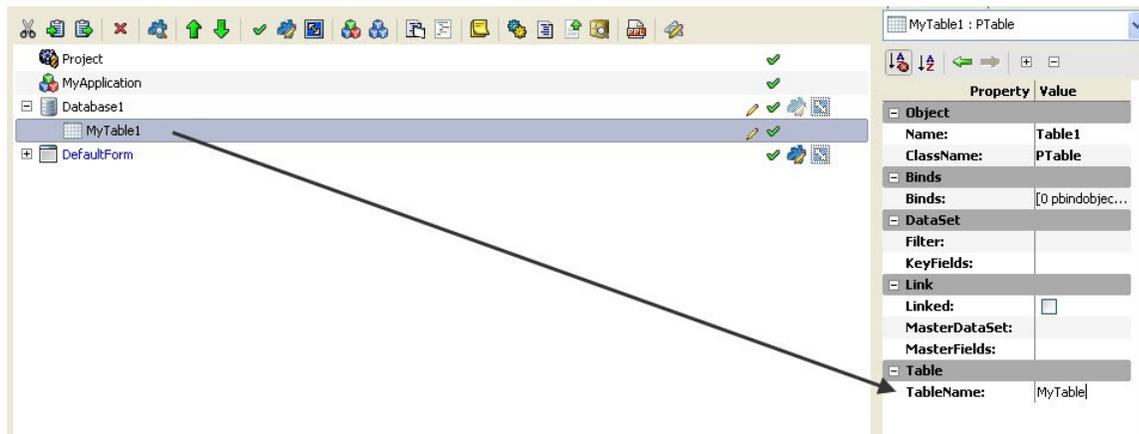


Figure 95: Give TableName

- Drag some **PField** Objects to the table object to add fields

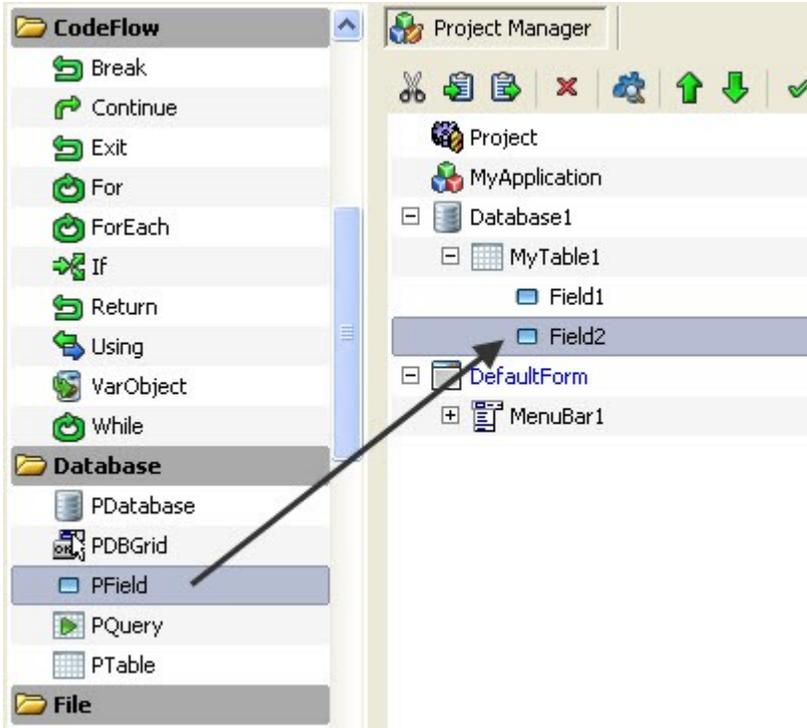


Figure 96: Add PField object

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

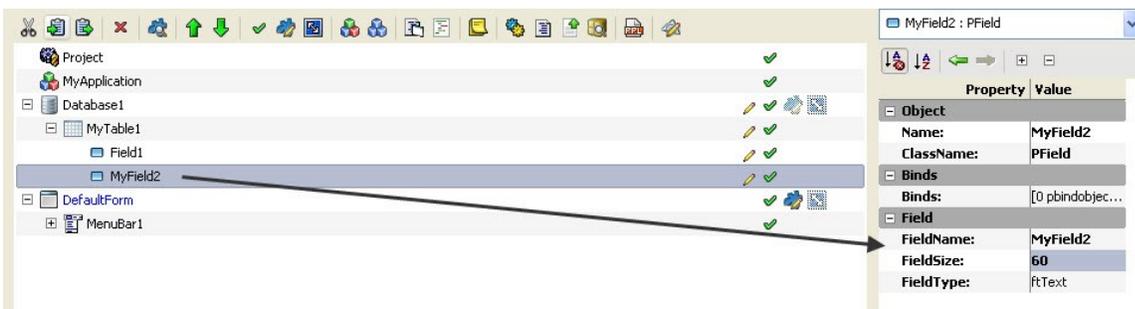


Figure 97: Specify Property

- In the data menu, Click on **Create Database** to create your database file in the location specified earlier.

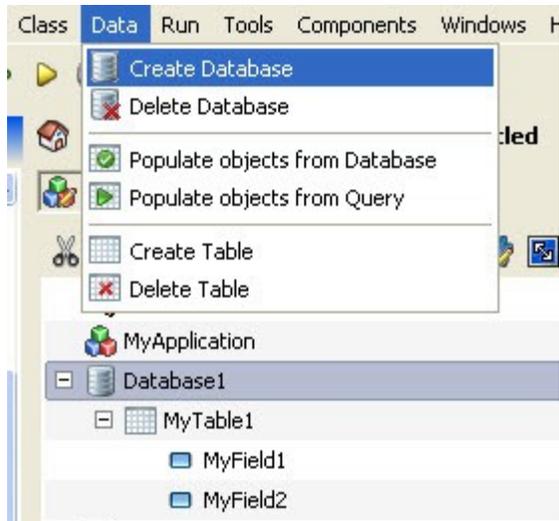


Figure 98: Creating Database

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

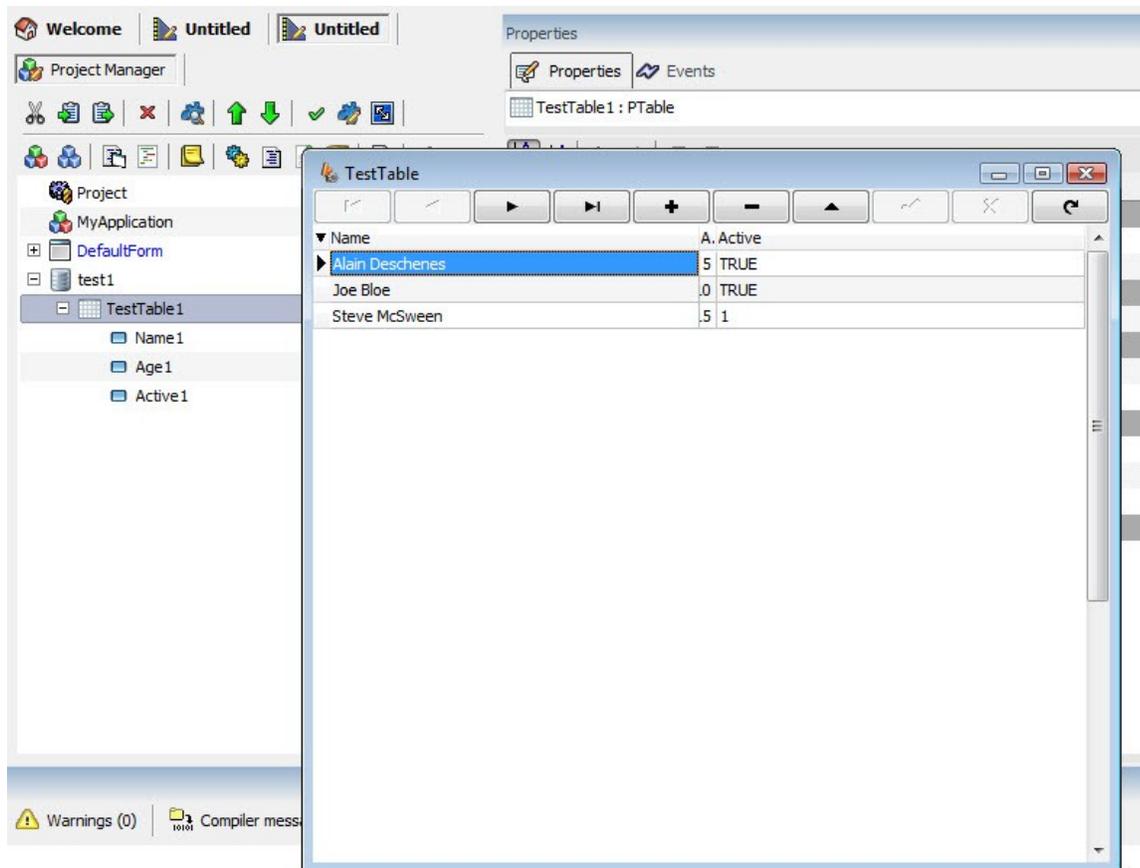


Figure 99: Viewing Table

- The database along with table has been created. Now you can use other objects like **PDBGrid** to arrange the fields the way you want.

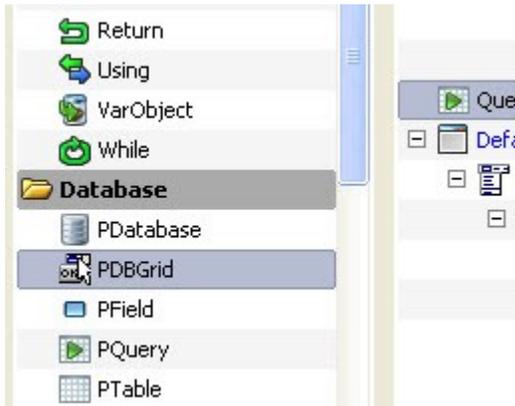


Figure 100: insert PDBGrid

- 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 PDBGrid to PForm' to create a table with a grid view in your form.

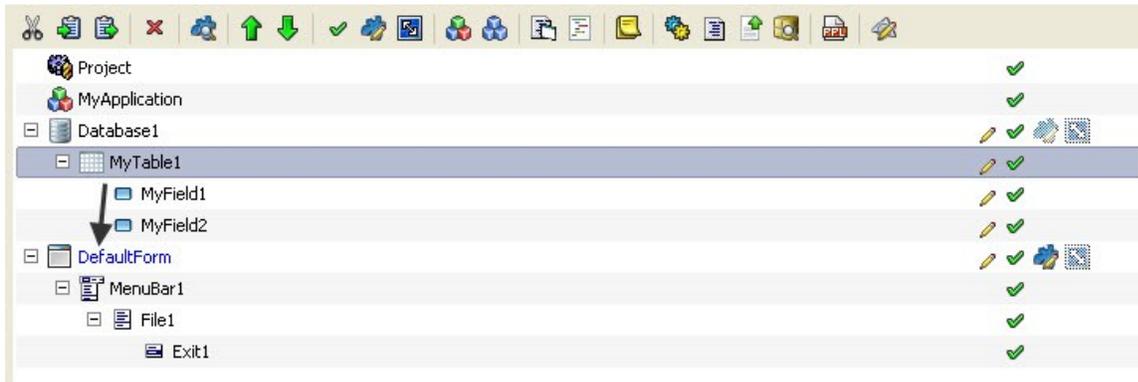


Figure 101: Drag table to form

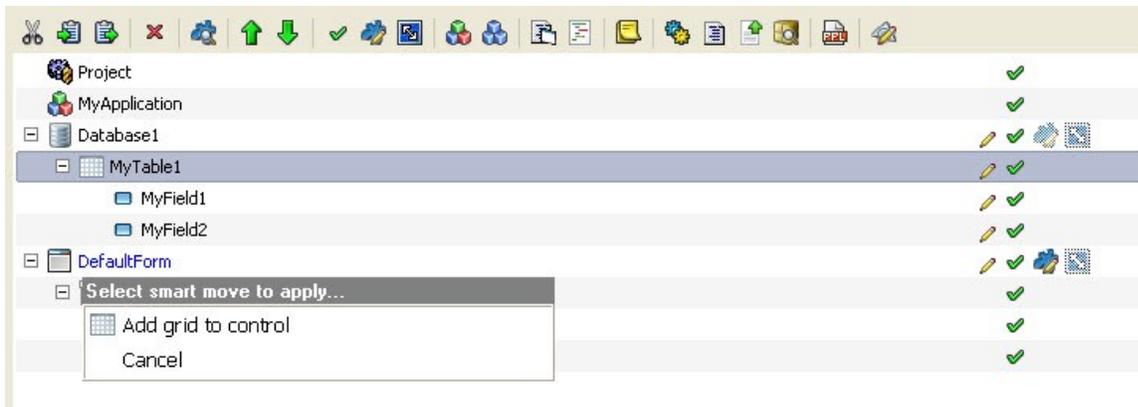


Figure 102: Smart move

- If you want to build your SQL query visually, you can also drag a **PQuery Object** from the **Components Panel** into the **Project Manager**.

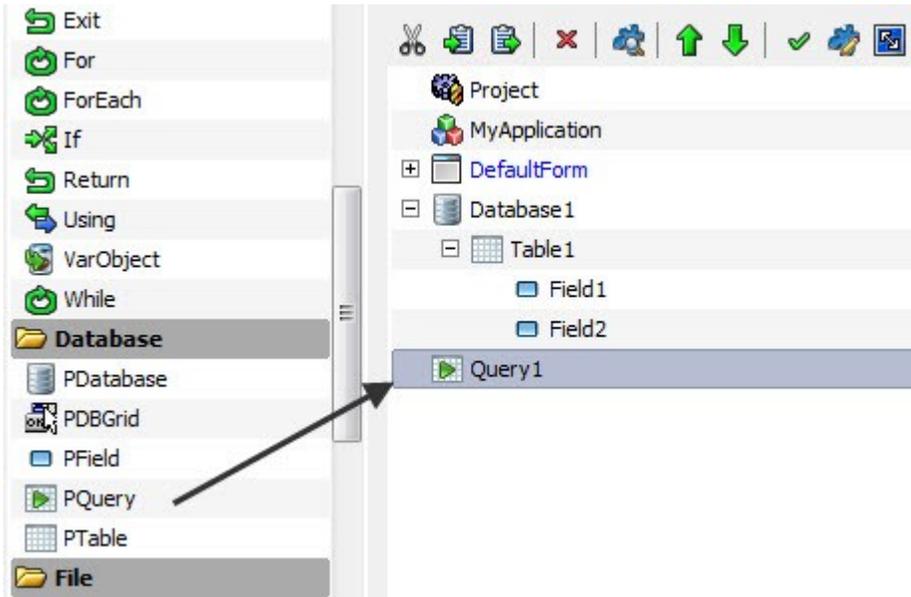


Figure 103: Drag PQuery

- After dragging and dropping the **PQuery**, go to the **Property Panel** and change the **database property** to a database that is already created.

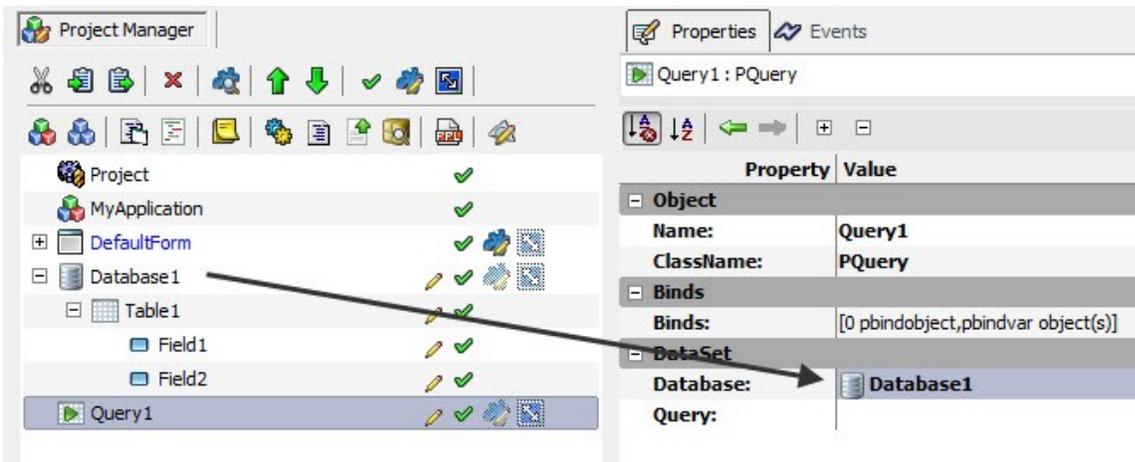


Figure 104: Change Database property

- Now double click the **PQuery Object** to create a **SQL query visually**.

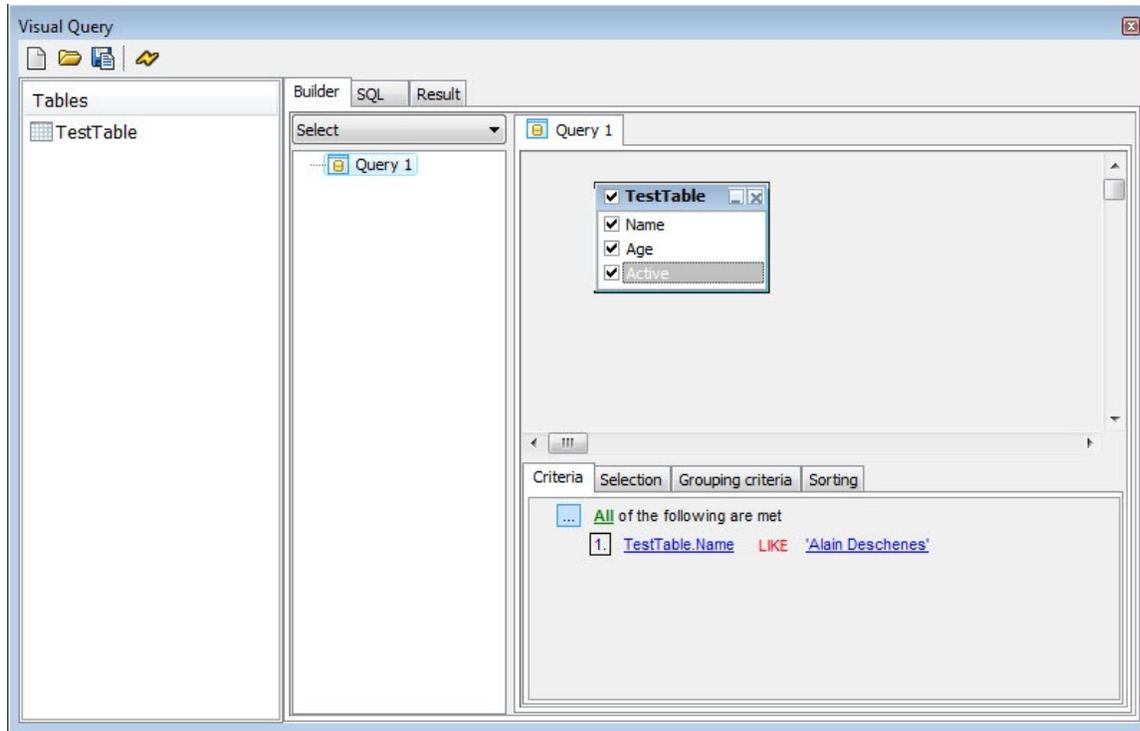


Figure 105: Visual query editor

PQuery is a handy utility that allows users to run their own SQL query in PIDE environment, we will learn more about PQuery in the next section.

Visual Query Editor

Visual query editor is a handy tool that not only allows a user to effectively create his/her own SQL query, it also imparts higher level of control and flexibility to the PIDE2 programming environment. After learning how easy it is to create databases and tables in PIDE2, we will learn about the visual query editor that will allow us to create SQL manipulations visually as well as through manual methods.

For editing a query in visual query editor, one must have access to a project with a database that can be associated with the **PQuery Object**. If you do not know how to create a database, follow the topic given earlier in the manual. Follow the guidelines given below to edit a query in visual query editor:

- Open PIDE2 and create a new project with a database, tables and respective fields.

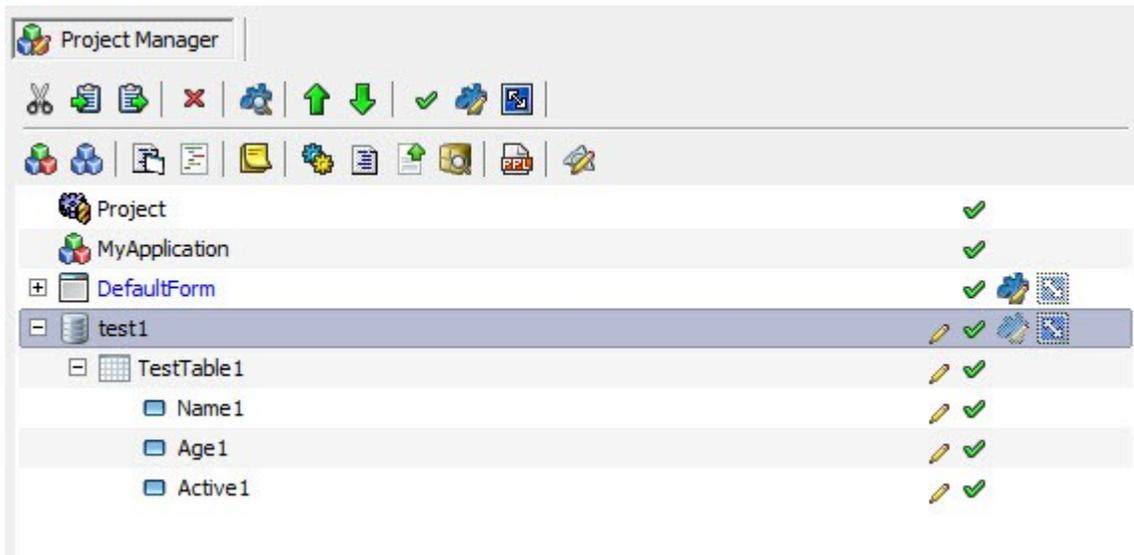


Figure 106: New project with database

- Drag a **PQuery Object** from the **Components Panel** to the **Project Manager**.

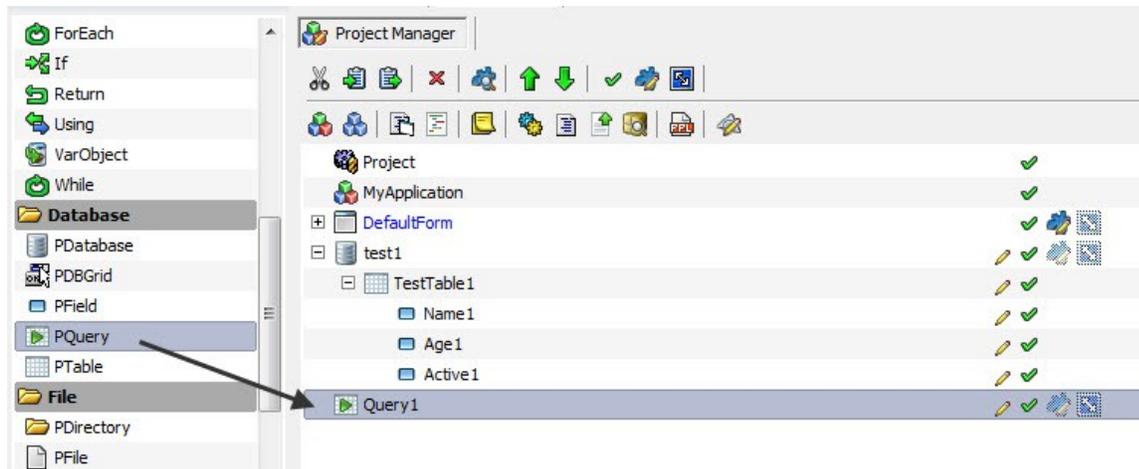


Figure 107: Drag PQuery

- Select your database name in the **Database Property** of PQuery.

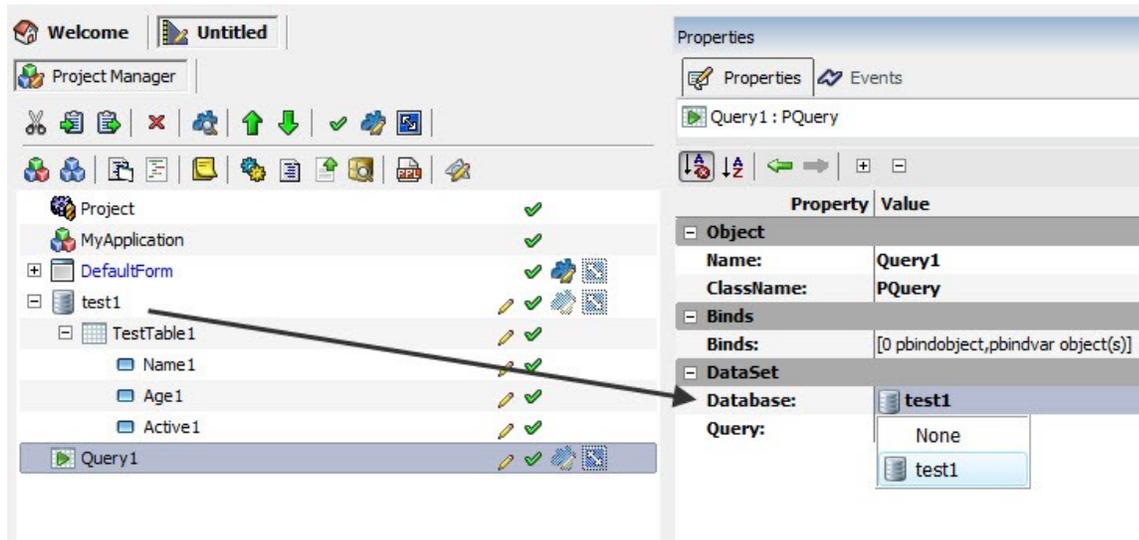


Figure 108: Select appropriate database name

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

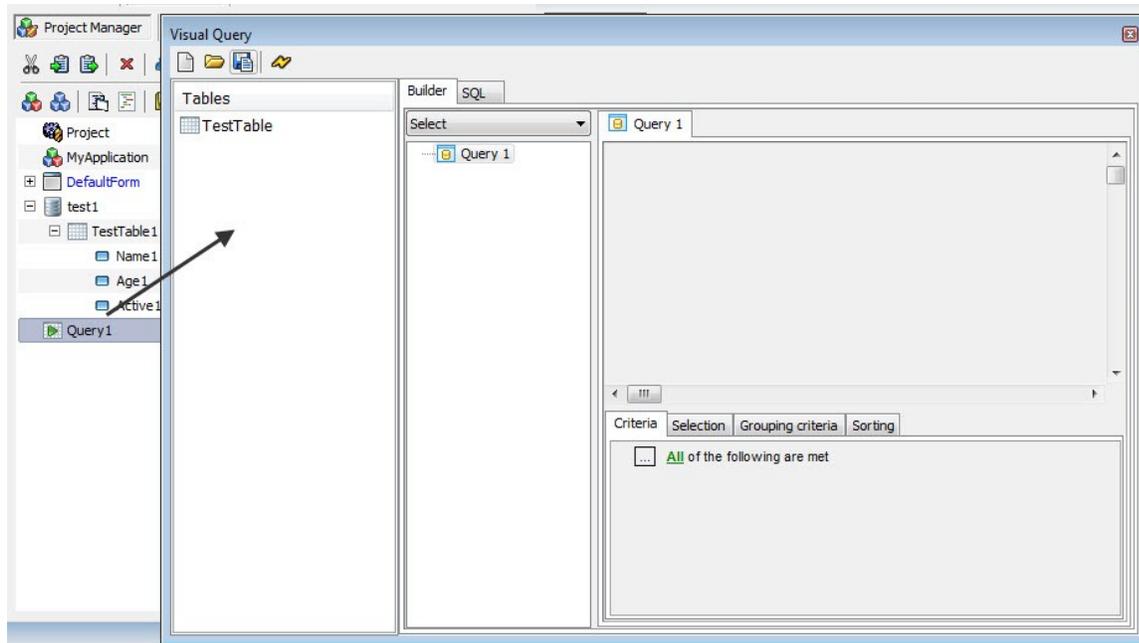


Figure 109: Double click PQuery

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

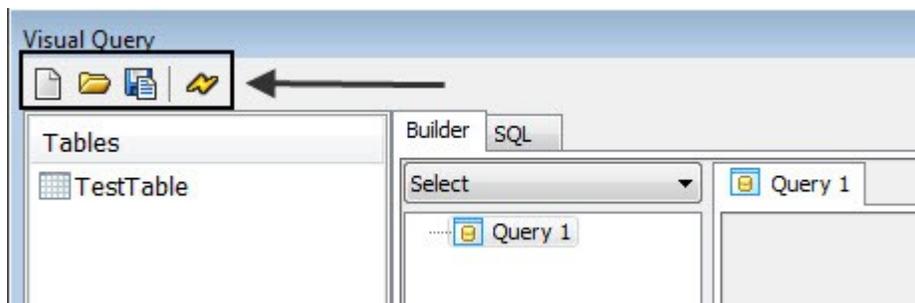


Figure 110: The four buttons of visual query editor

- 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.

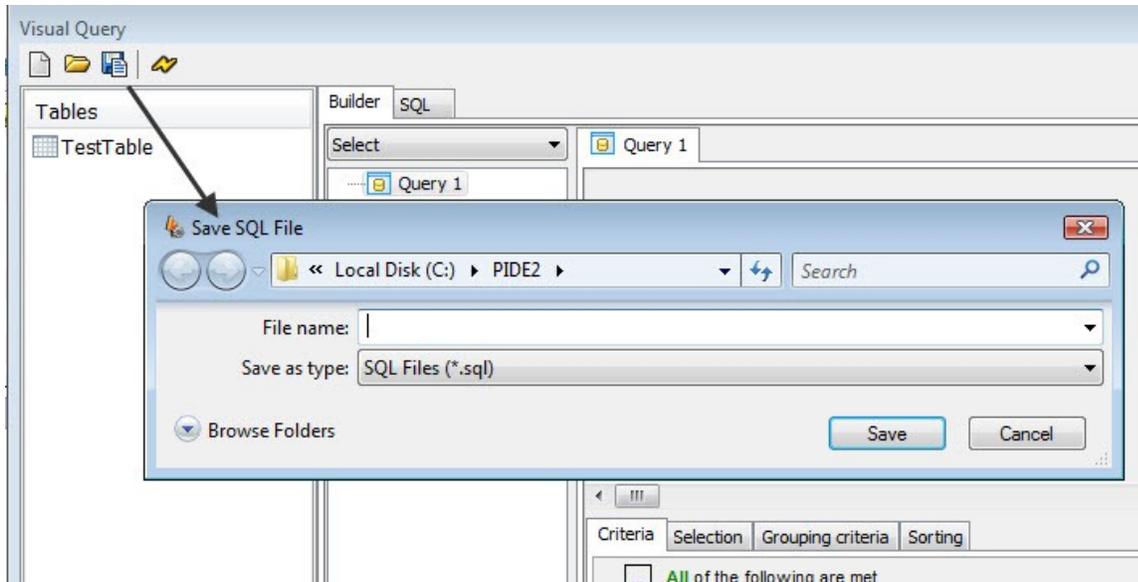


Figure 111: Save file button

- 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.

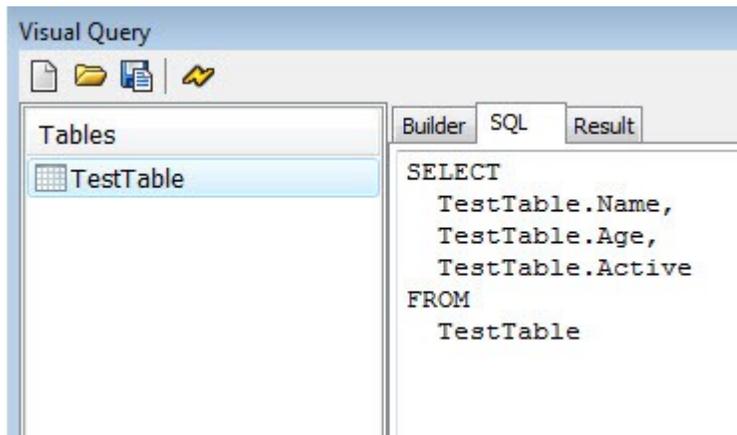


Figure 112: The SQL Tab

- 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**.

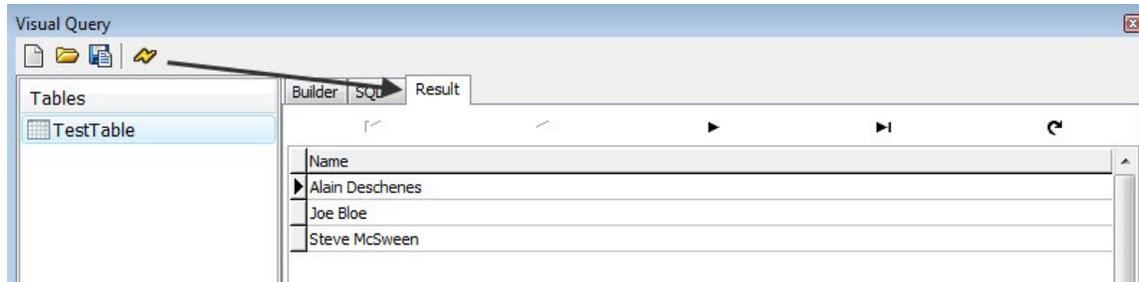


Figure 113: Results Tab

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

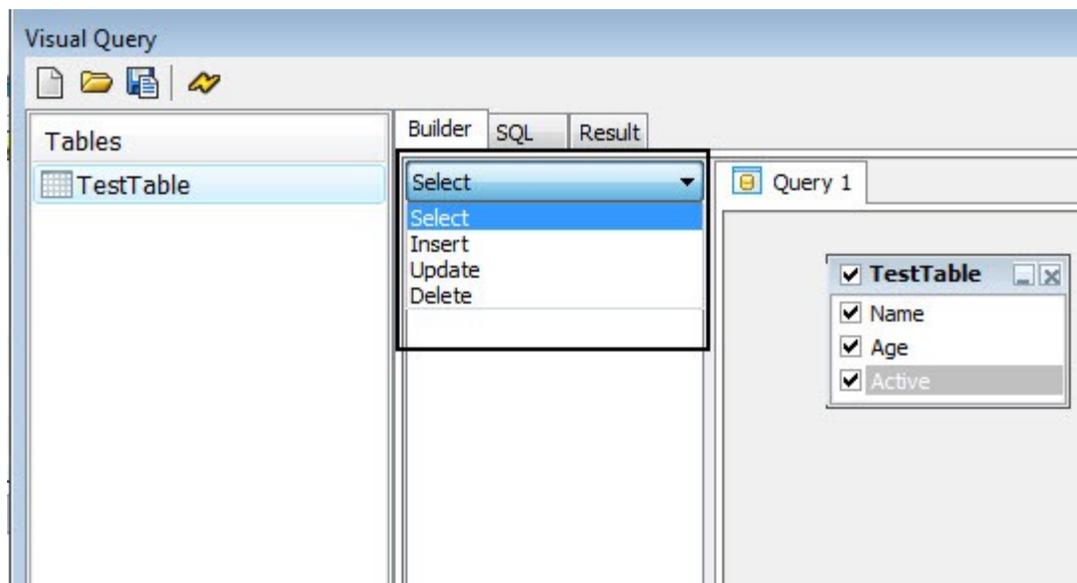


Figure 114: Selecting Query Type

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

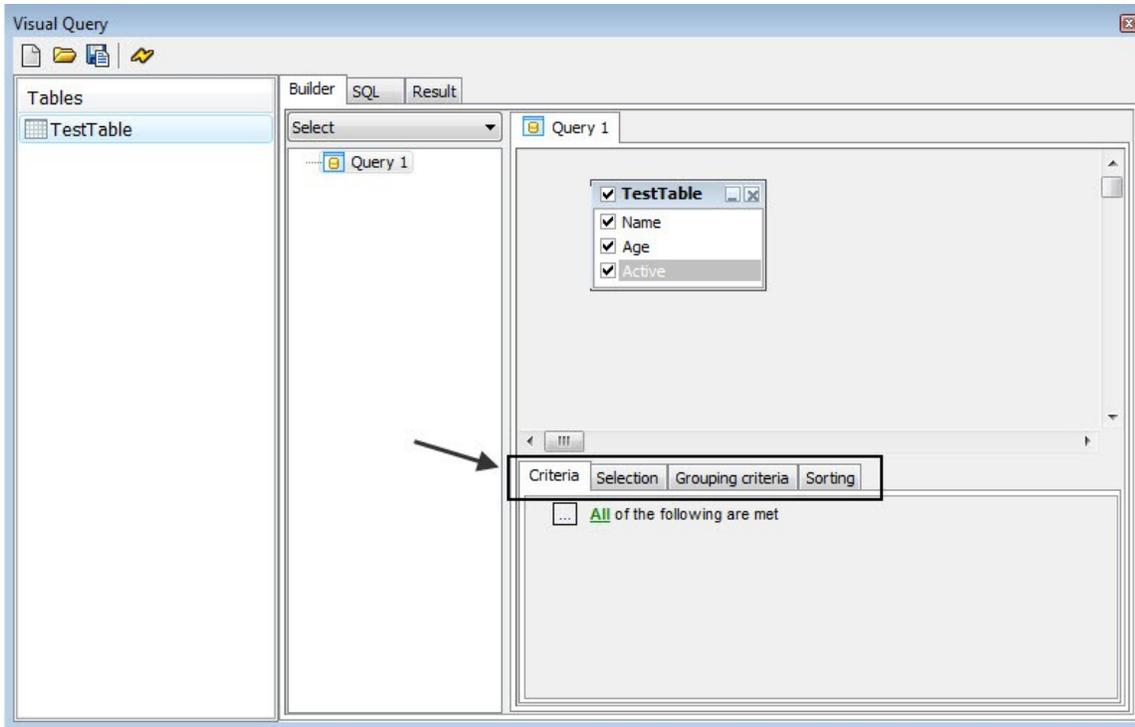


Figure 115: Creating advanced query

Select SQL Query and Visual Query Editor

Once with visual query builder, creating and performing SQL manipulations is never a big deal! Just by a few clicks, users can create and manipulate data in tables by using SQL queries. Let us see how to create a SELECT query with the help of visual query builder:

- Open a PIDE project in the project manager view with a database, tables and the associated fields.
- Drag a **PQuery Object** to the project manager and change its **Database Property** to point to the database of your project.
- Double click the **PQuery Object** to open visual query builder and double click on the table you want to work with. This will open a dialog box with the fields present in the table.

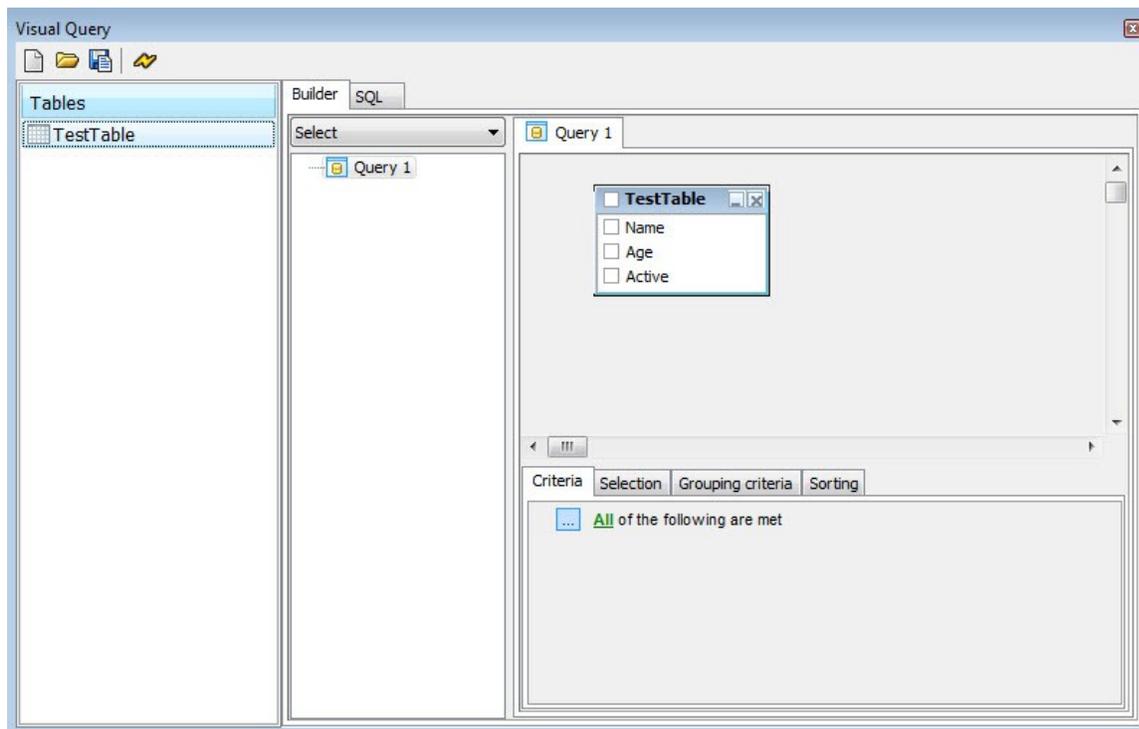


Figure 116: Visual query editor

- In the **Builder Tab**, select the **SELECT** query option. (SELECT query is the default option).
- Select all the columns you want to view in the **SELECT query**.

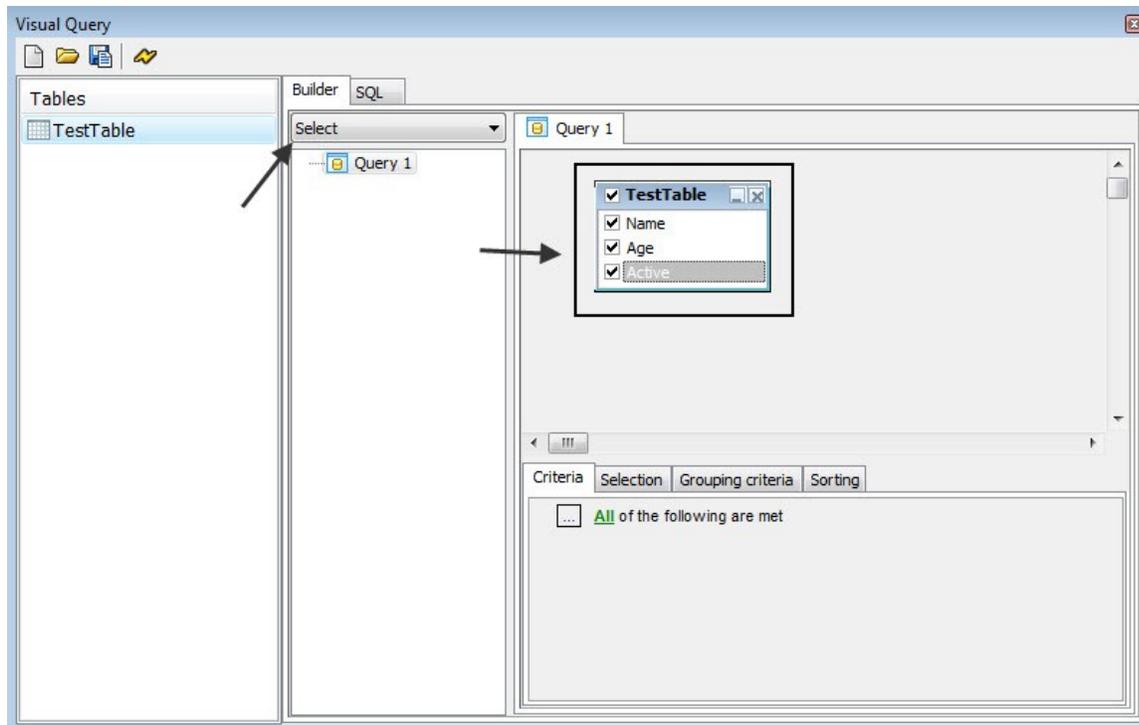


Figure 117: Creating select query

- To display the **SELECT query** without any criteria, simply press the **Run button** to execute the query and it will display all the results according to the columns that are selected.

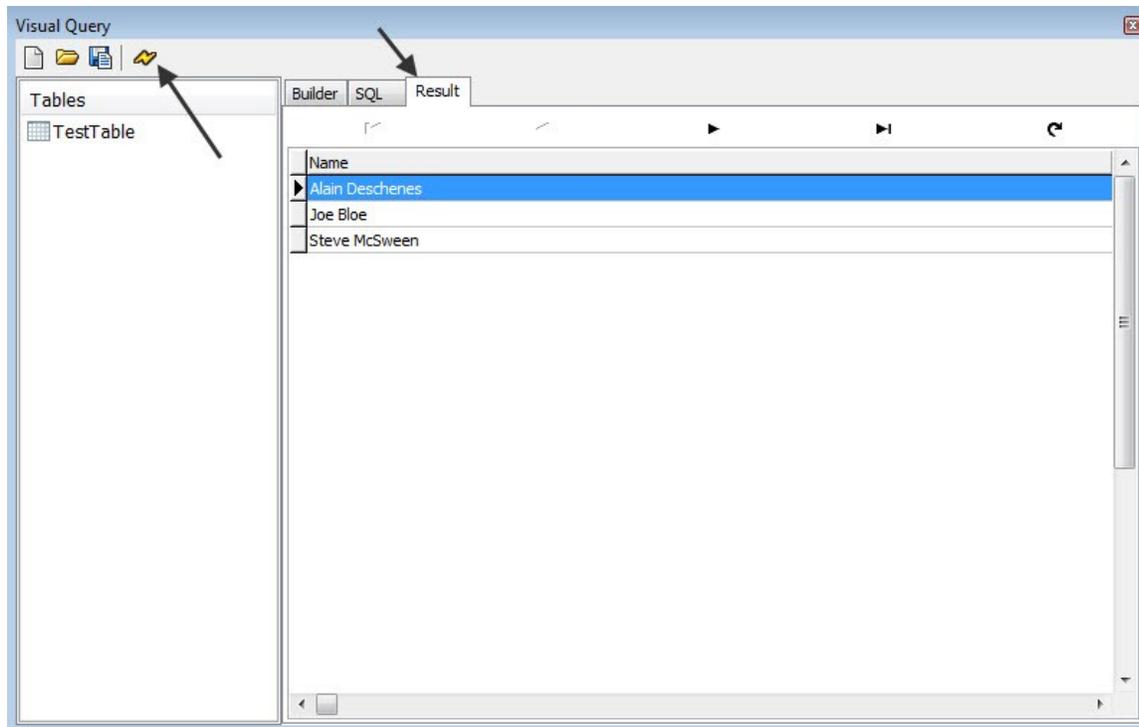


Figure 118: Run SELECT query

- For adding a condition, press the box in the criteria tab and press **Add Condition** to create a **WHERE** condition. Doing this will create two dashes with one equal sign in between them.

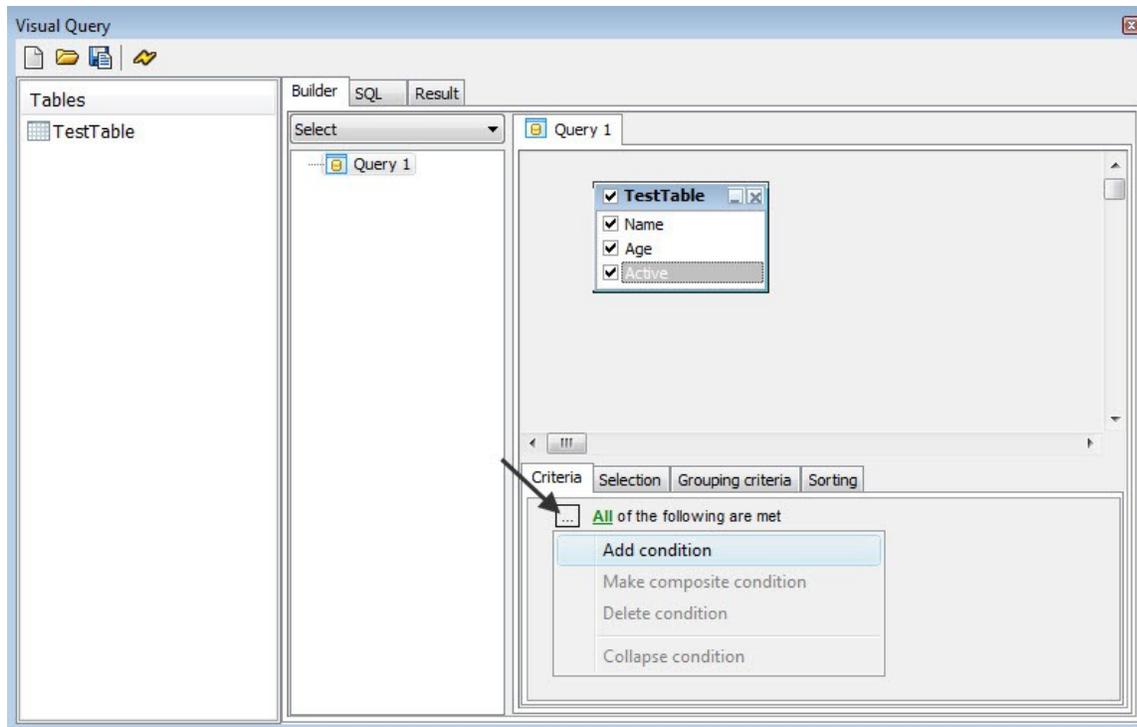


Figure 119: Add Conditions

- Click the first dash and click the table name to specify the left hand side of your **WHERE clause**. In the second dash, double click to fill a field from a table or input your own text.

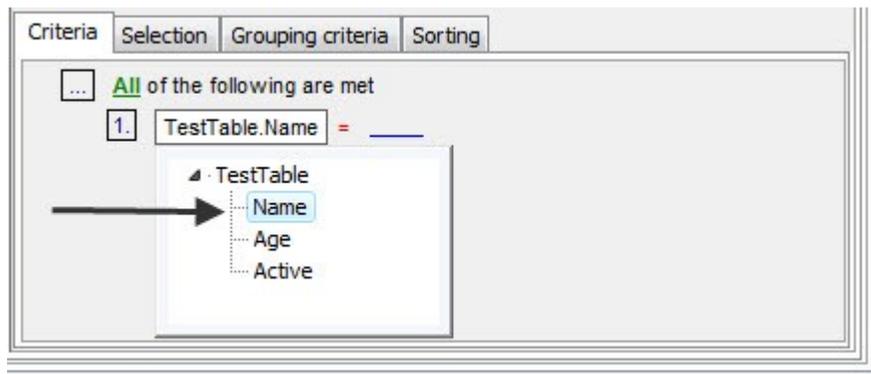


Figure 120: Creating first part of a condition

- For comparison operator, click the equal sign to select the **Comparison Operator** of your choice.

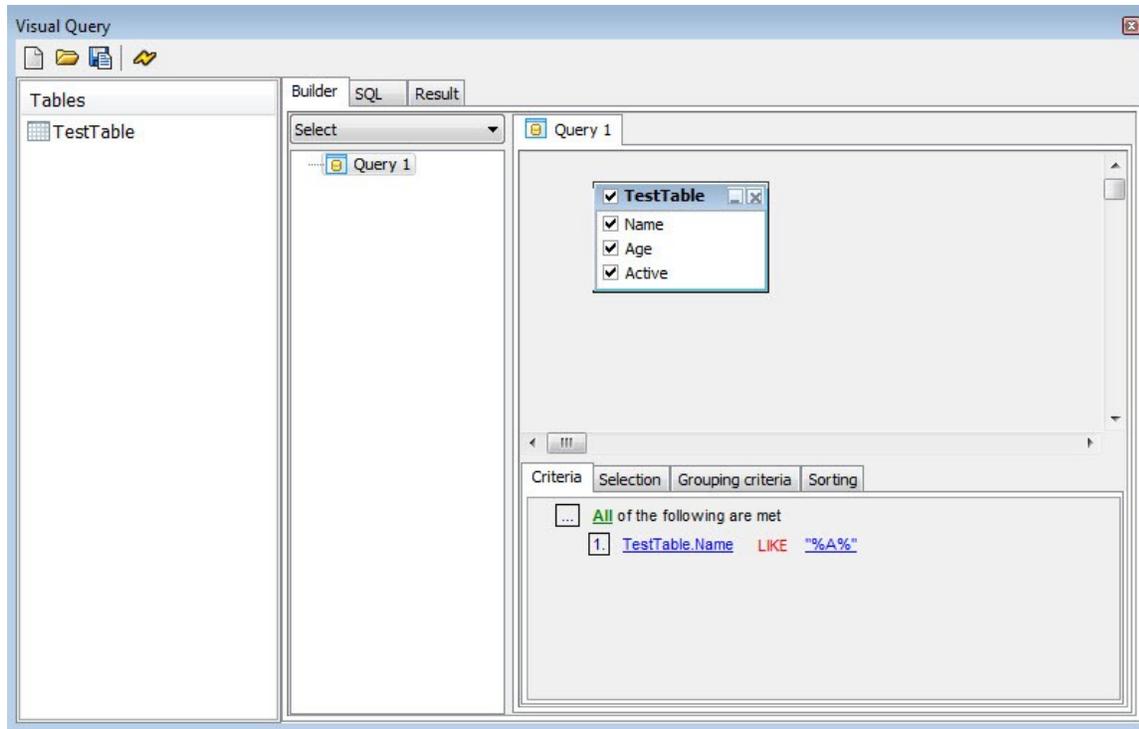


Figure 121: Choose a comparison operator

- Once your condition is created, click the **Run Button** to execute the **SELECT** query with **WHERE** clause.

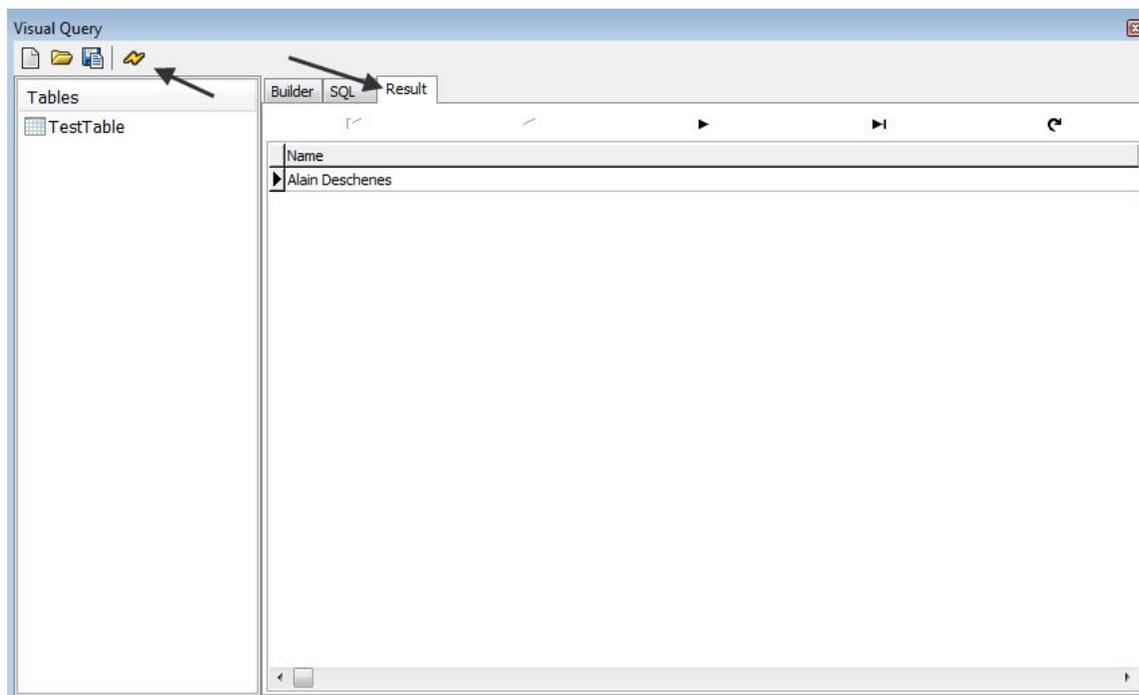


Figure 122: Run the query

- Advanced users can use the **Selection Tab** to specify how their fields will be displayed as well as to specify the **grouping and aggregate options**.

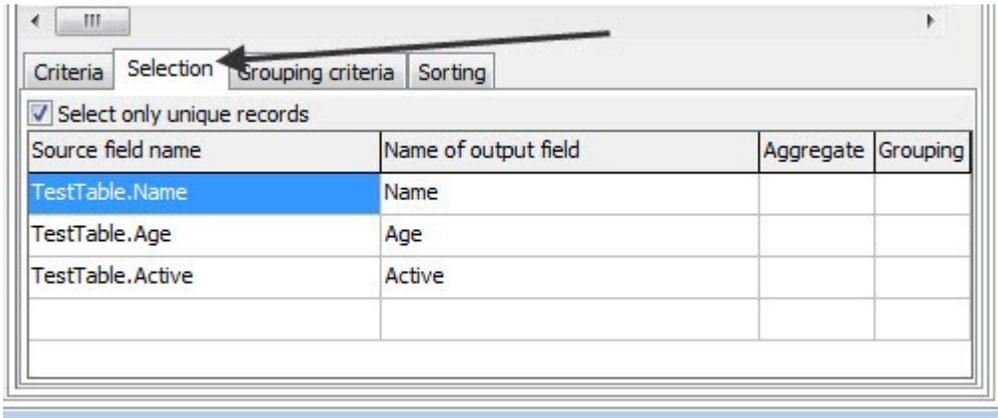


Figure 123: Selection Tab

- Grouping condition** can also be specified just like it was specified in the **Criteria Tab**.

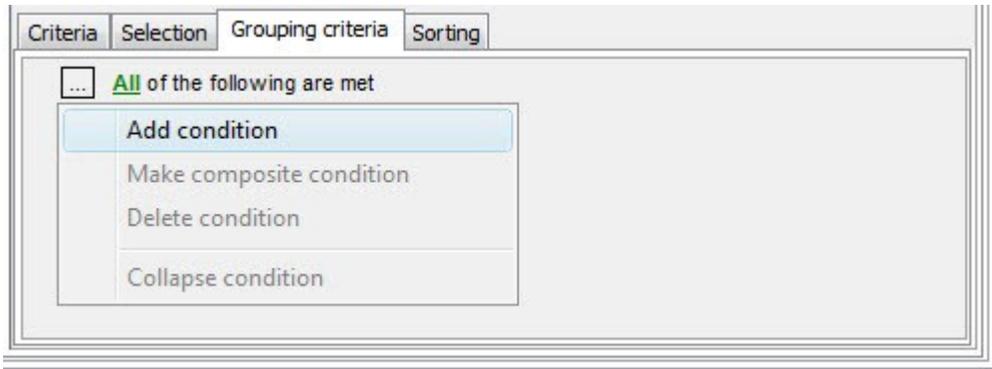


Figure 124: Grouping Criteria Tab

- The **Sorting Tab** can be used to specify how the fields listed in your table will look like.

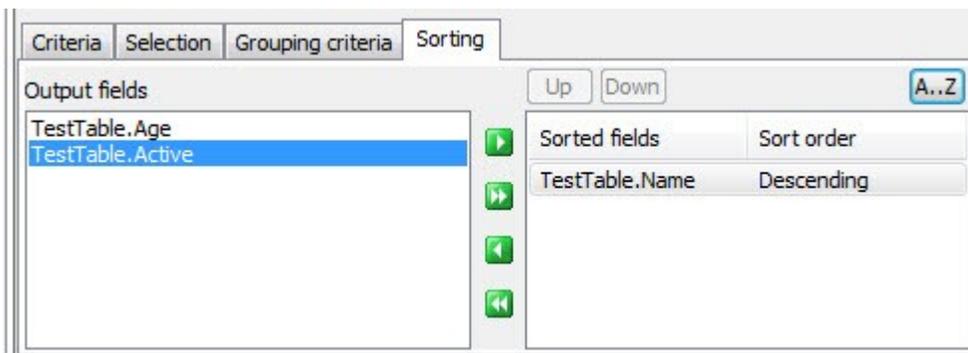


Figure 125: Sorting Tab

- After creating a query with the help of **visual query editor**, users can also switch to the **SQL Tab** to look at the **SQL query** created by them. In case the query is deemed to be inappropriate, it can also be changed very easily from the **SQL Tab** only.

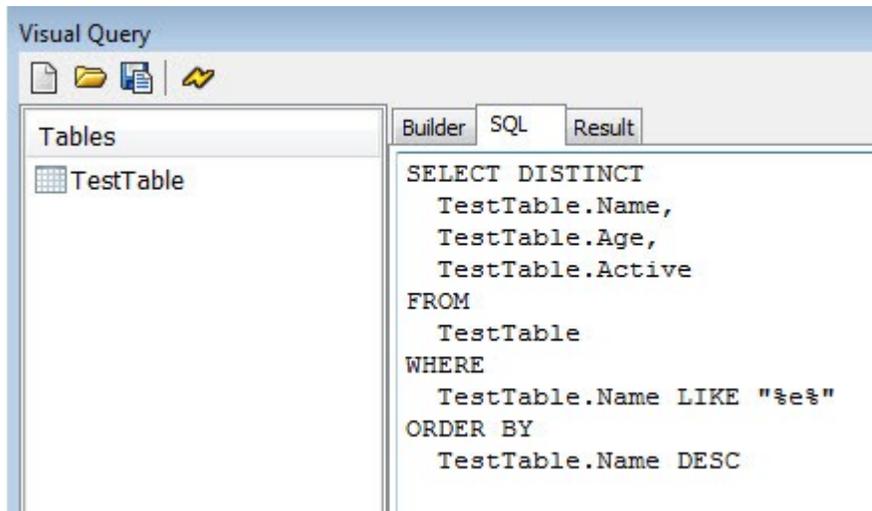


Figure 126: The SQL Tab

Insert Query with Visual Query Editor

INSERT is one of the most important operations performed by a SQL query. By using the INSERT query, users can insert data into the fields and use that data to extract meaningful information. Let us see how to insert data in a table with the help of visual query builder in PIDE:

- Open a **PIDE Project** in the project manager view with a database, tables and the associated fields.
- Drag a **PQuery Object** to the project manager and change its **Database Property** to point to the database of your project.
- Double click the **PQuery object** to open visual query builder and double click on the table you want to work with. This will open a dialog box with the fields present in the table.
- Select **INSERT** in the builder tab and select the fields you want to insert the data in.

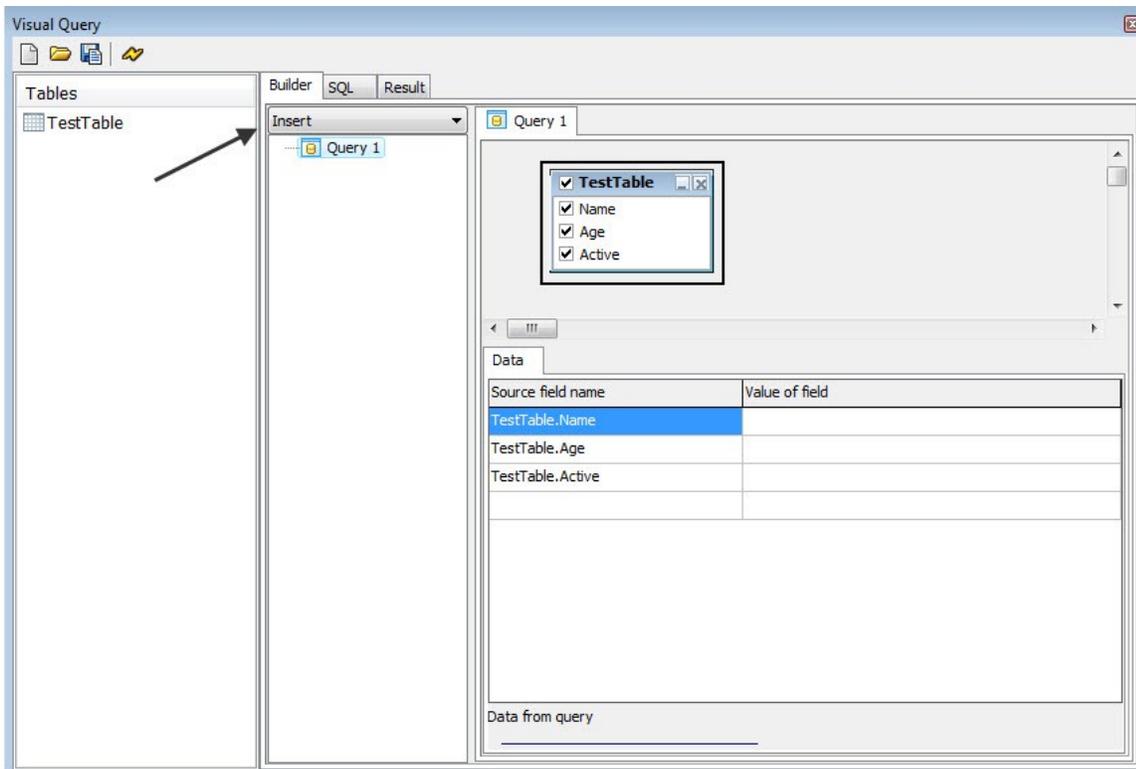


Figure 127: Selecting Insert Query

- In the data tab, write the value of the fields you want to input and press the **Run Button** to execute the query.

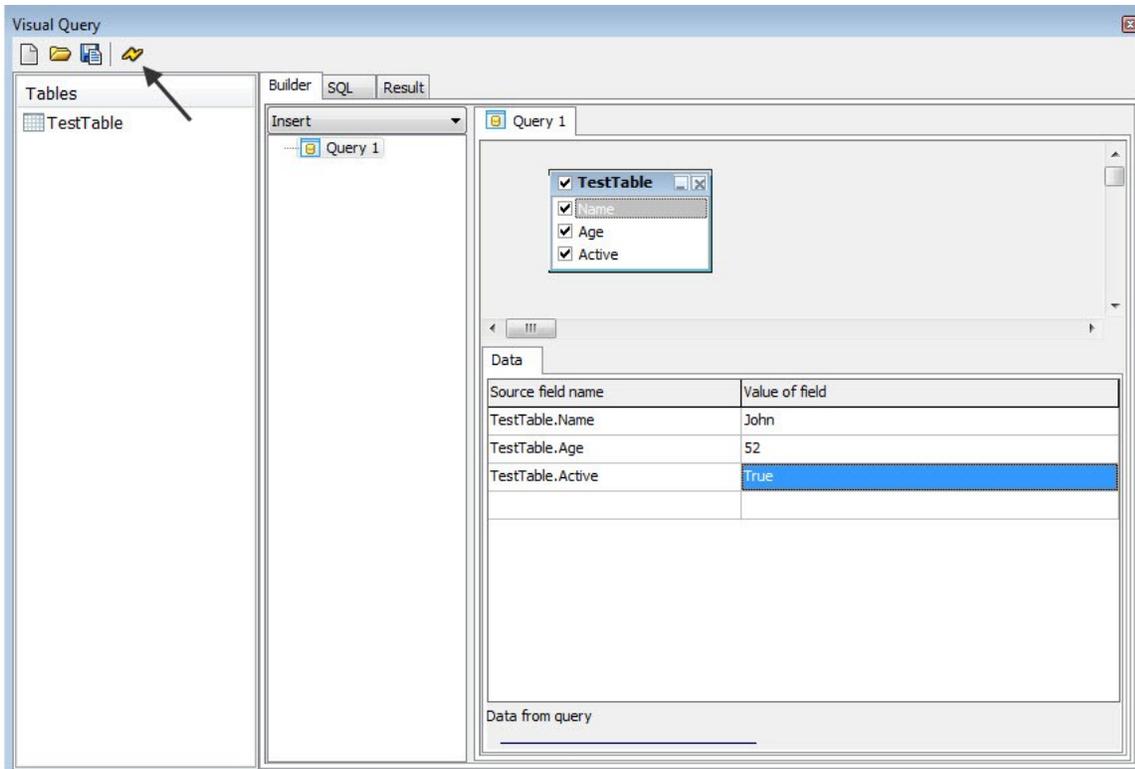


Figure 128: Run Query

- After the query has been successfully executed, go to the **SELECT** option and run a select query to check if the row has been added in the table or not.

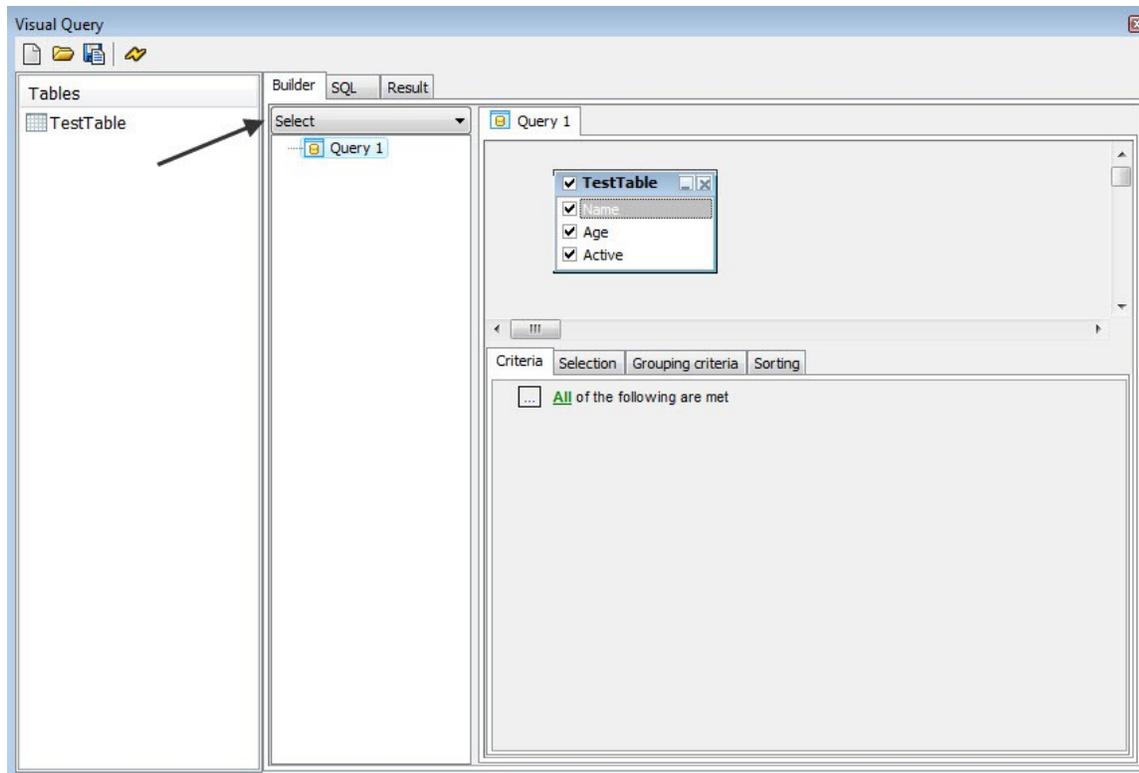


Figure 129: Select Query

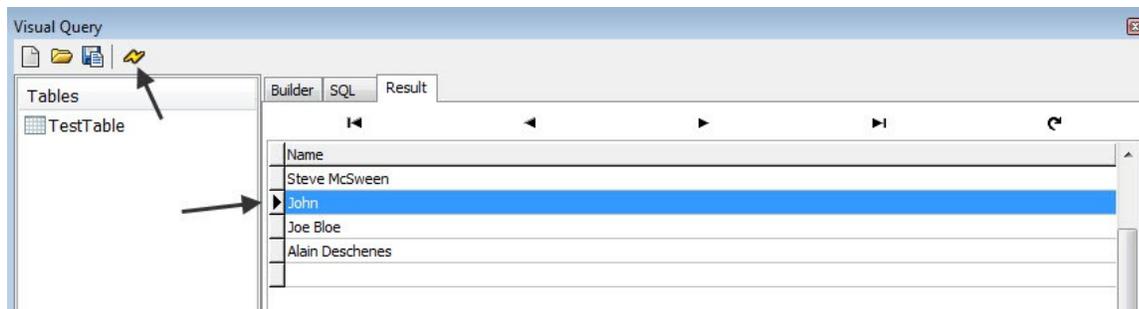


Figure 130: Executing The Select query

Update Query with Visual Query Editor

Updating data in tables is a task that is frequented by database administrators and allows them to rectify any mistakes made to the data. Here we will see how to use the UPDATE query in PIDE2 and work with it effortlessly from visual query builder:

- Open a PIDE project with database and **PQuery Object** already configured for use and double click the **PQuery Object** to open a visual query editor.
- Here, double click the table you want to work with and select **UPDATE** in the drop down menu of the **Builder Tab**.

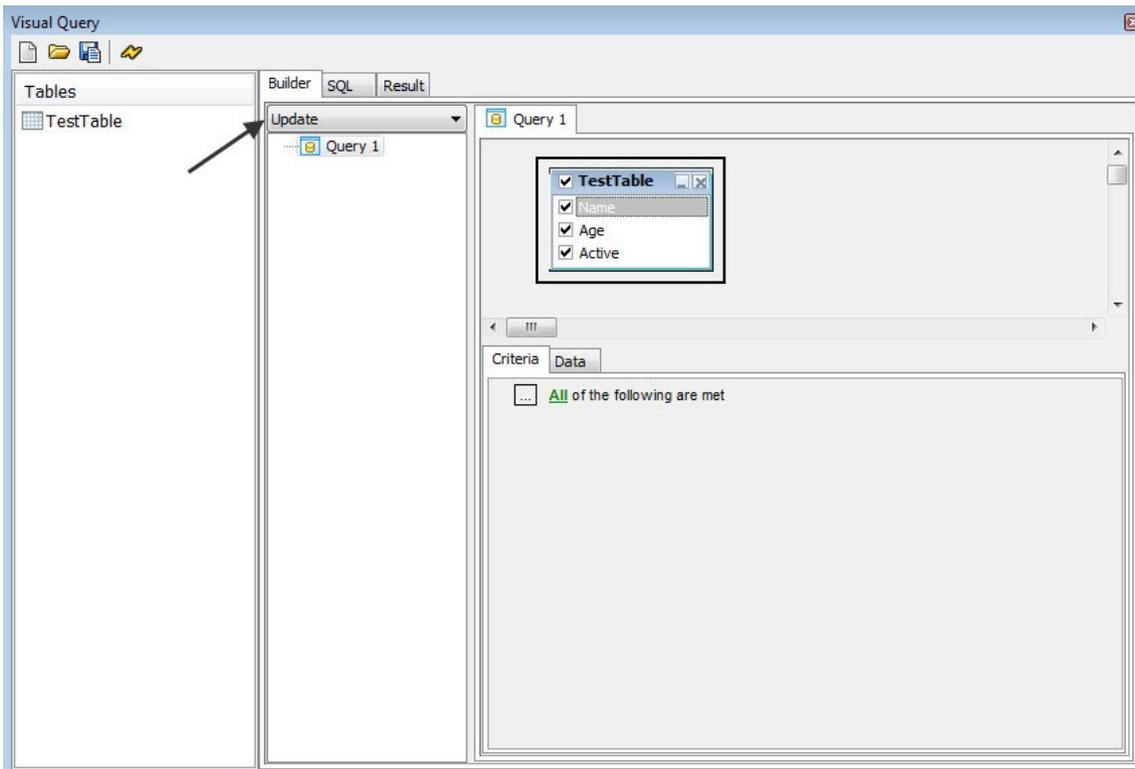


Figure 131: Choose Update Query

- In the **Criteria Tab**, click on the box and select **Add Condition**. This will create two dashes with an equal to sign in between.

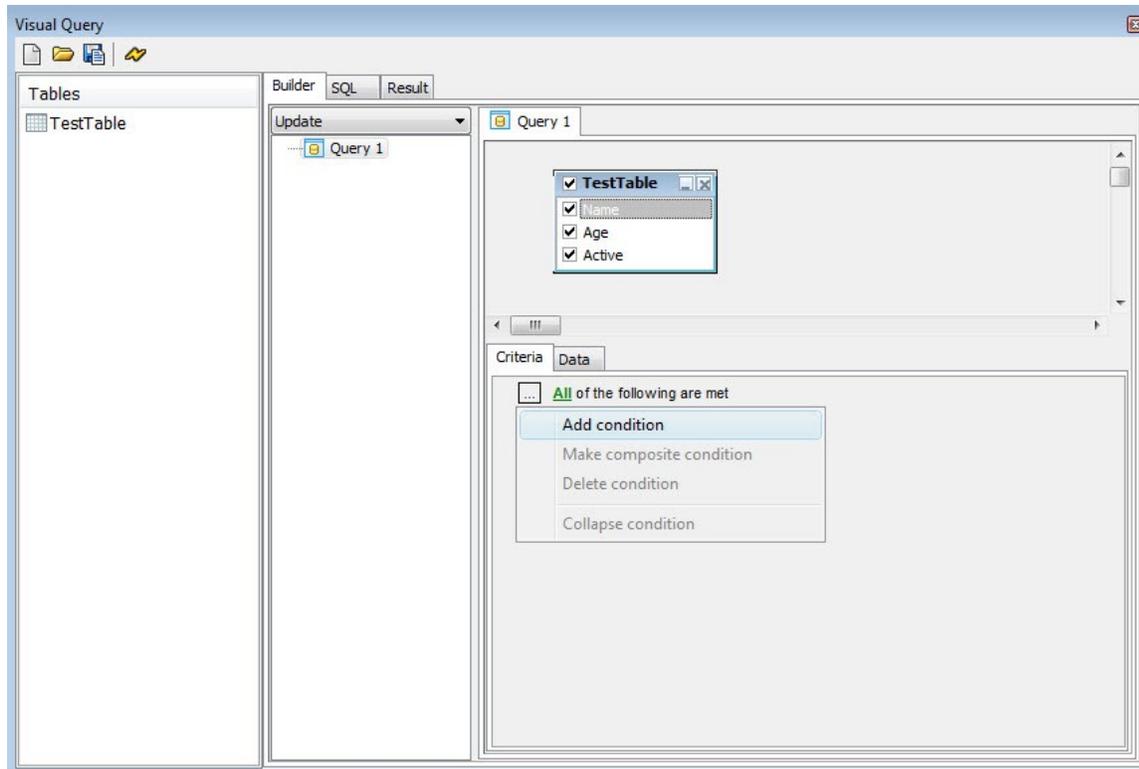


Figure 132: Add Condition

- Click on the first dash and select the field based on which your **UPDATE Query** will be executed. Click on equal sign and the second dash to write or select the proper field to create a condition which, if true, would lead to the selection of that row for an update.



Figure 133: Complete condition

- After creating a condition, go to the data tab and enter the data you want to update in the field that satisfies the stated condition. Press the **Run Button** to execute the query.

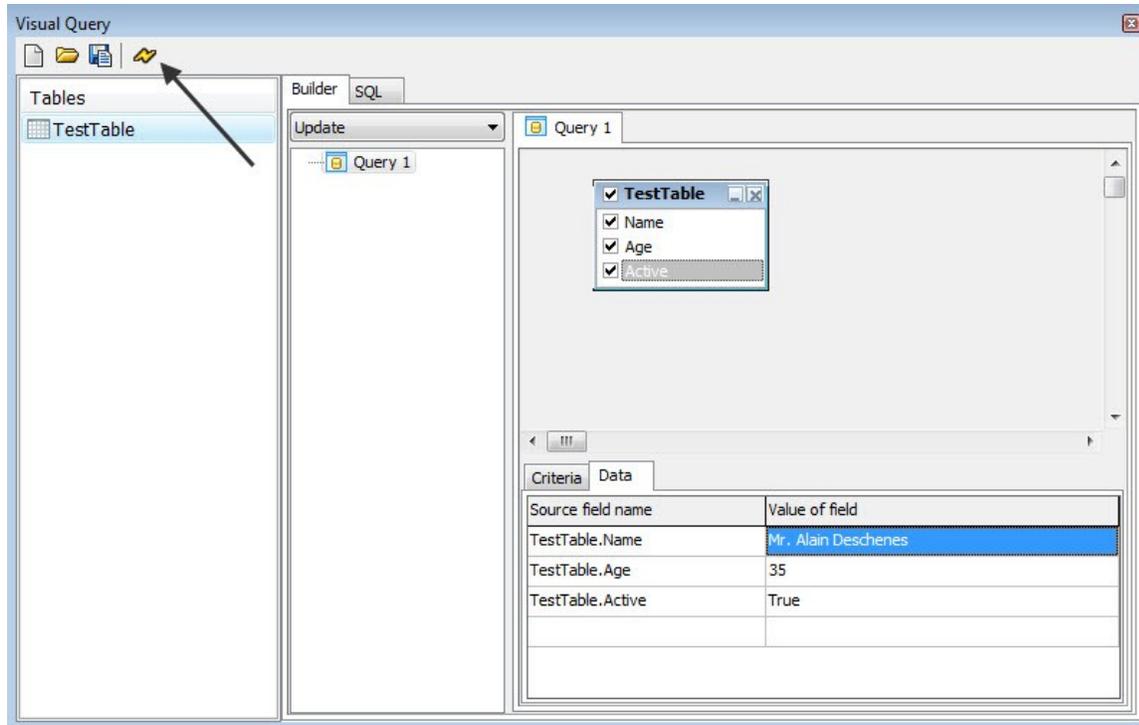


Figure 134: Input data

- Choose **SELECT** from the drop down menu in the **Builder Tab** and run a select query to check if row has been updated or not.

Deleting Operation with the help of Visual Query Editor

After Selecting, Inserting and Updating a row in a table, Deleting is the next most important operation that needs to be implemented for the proper upkeep of a database. Given below are the steps to delete a row from the table with the help of visual query editor in PIDE2:

- Open a PIDE project with database and **PQuery Object** already configured for use and double click the **PQuery Object** to open a visual query editor. Here, Select **DELETE** option in the **Builder Tab**. Double click the table you want to work with and check all the fields.

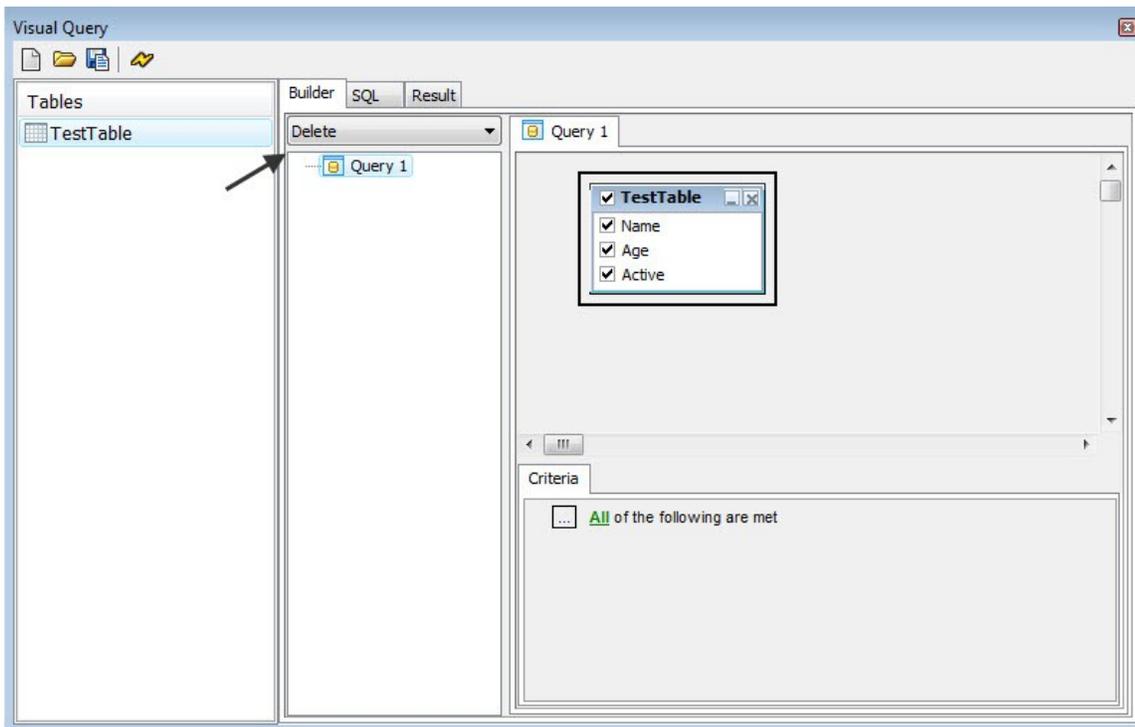


Figure 135: Choose Delete query

- In the criteria tab, click on the box and select **Add Condition** option to create a condition on which the row will be deleted. This will create two dashes and a comparison operator in between them.

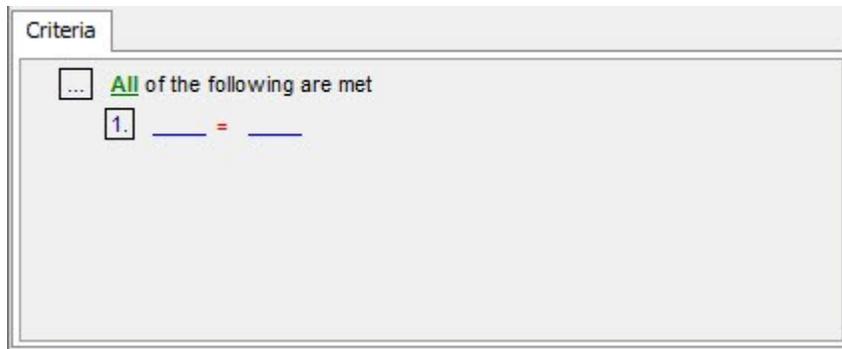


Figure 136: Specify criteria

- In the first condition that is created, click the first dash and **select the field name**.
- Choose the right comparison operator by clicking the equal sign and click the second dash to select or write a condition that needs to be checked.

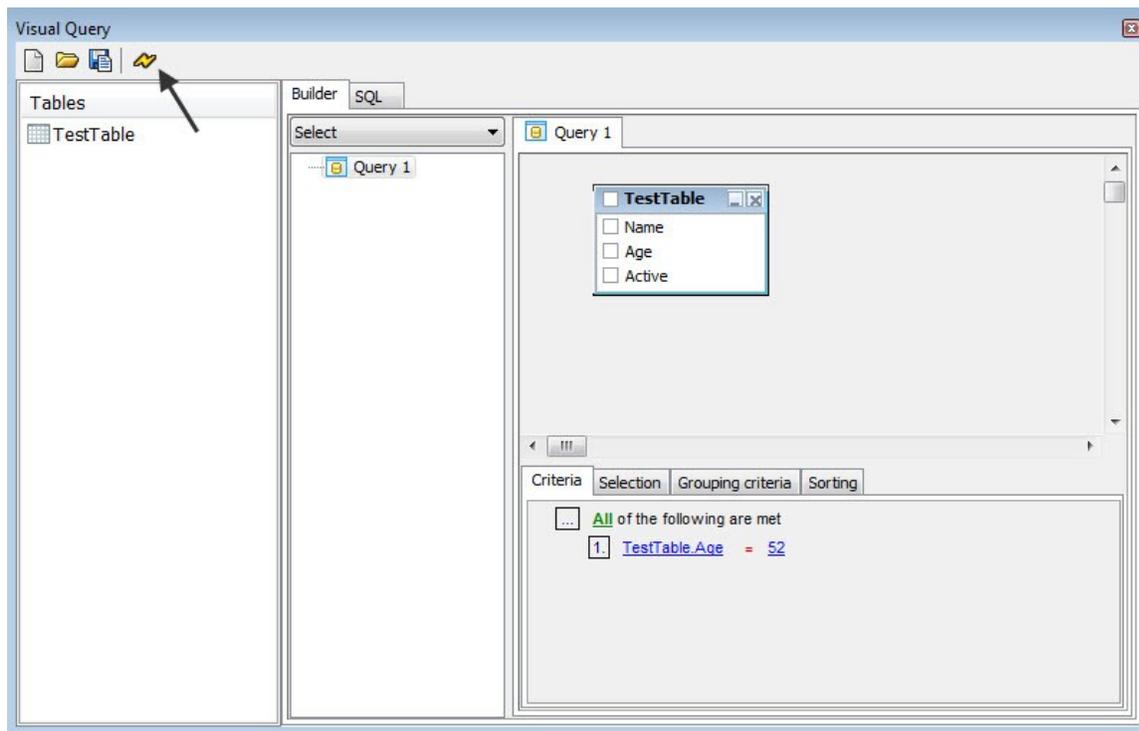


Figure 137: Run query

- Click the run button and check the table for changes by running the select query in visual query editor.

Help File Editor

Help file editor in PIDE2 allows a user to create help files (.hlp). Providing the facility to create your own help files, PIDE is an excellent platform for creating comprehensive help files with all the features present in a usual help file. Users can easily open previously made help files and tweak them according to their choice.

While you can create a help file by choosing Help file in the **New project window**, opening an already created .hlp file will open it in the help file editor so that you can edit all its contents.

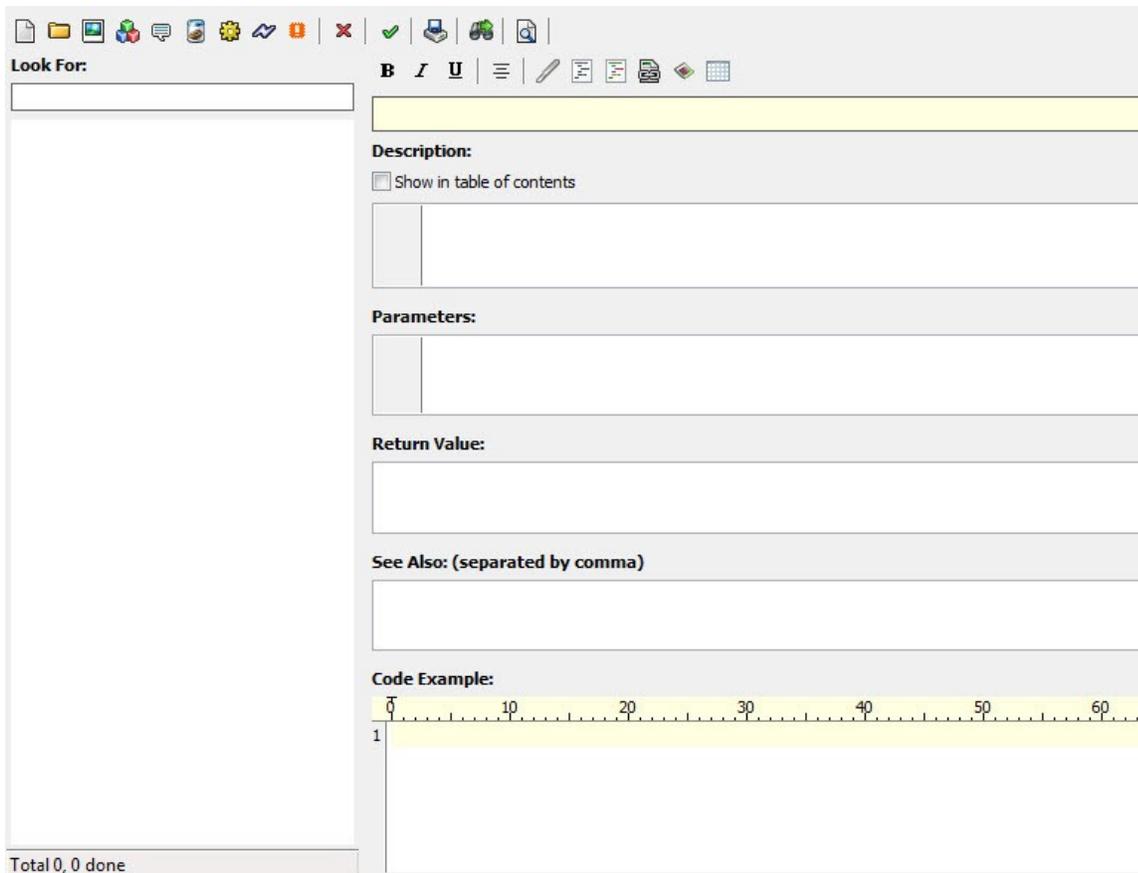


Figure 138: Help file editor

The help file editor has many components one can use to create a complete help file. The top layer of the help editor consists of the common components that are used for building a help file.



Components can be created in a help file just by clicking these buttons. Given below is a brief of these buttons and their usage:



The **New Topic Button** can be used to create a new topic in the help file.



New Folder Button is useful in creating a new folder that can contain all the other components of a help file.



For inserting an image in the help file, use the **Image Button**.



The **Class Button** is used to specify a class in the help file.



For specifying a property, the **Property Button** can be used.



This button signifies a **variable**.



Method is represented by this button; use it to create help on a method.



The **Event Button** is used to specify help on an event.



This button is used to specify a **Define Element** in the help file.



This button is used to **Delete** a topic in a help file.



This button is used to specify that a topic is **Complete**.



For **Importing Topics** from another help topic, use this button.



Use this button for **Searching** the occurrence of next text in the help file.



This button can be used to **Preview** the help file.

Appearance in Help file

Apart from the ones given above, there are also many layout design buttons available in PIDE2 that would allow a user to design the layout of the help file. Just like a text editor, these buttons too allow effective tweaking of the appearance of a help file.



Given below is an explanation of the above buttons:

B Use to give a **Bold** appearance of a tag.

I Italics button is used to give an **Italics** appearance to the text.

U Use to **Underline** a text.

☰ This button is used to **Center Align** a text.

✎ This button is used to specify a **line with some Formatting**.

📄 This button is used to specify a **Preformatted Text**.

📄 This button is used to specify a text that represents **Code**.

📄 For **Linking To Another Topic**, use this button.

🖼️ This button **Links An Image** with the help file.

📊 The grid button can be used for representing data in a **Grid Form**.

All the buttons given above will create a tag representing the specific appearance button you have clicked. For example, clicking the **B** or the bold button would create a **text tag** which would give a bold appearance to any text that is included within the starting as well as ending tags of the bold tag. Likewise, all these buttons will create tags and anything included within these tags will have the appearance marked by the tags.

The Help Editor Interface

The Help editor interface consists of two panes. While the left hand side pane is used to find and include topics in the help file, the right side pane is used to include information in those topics.

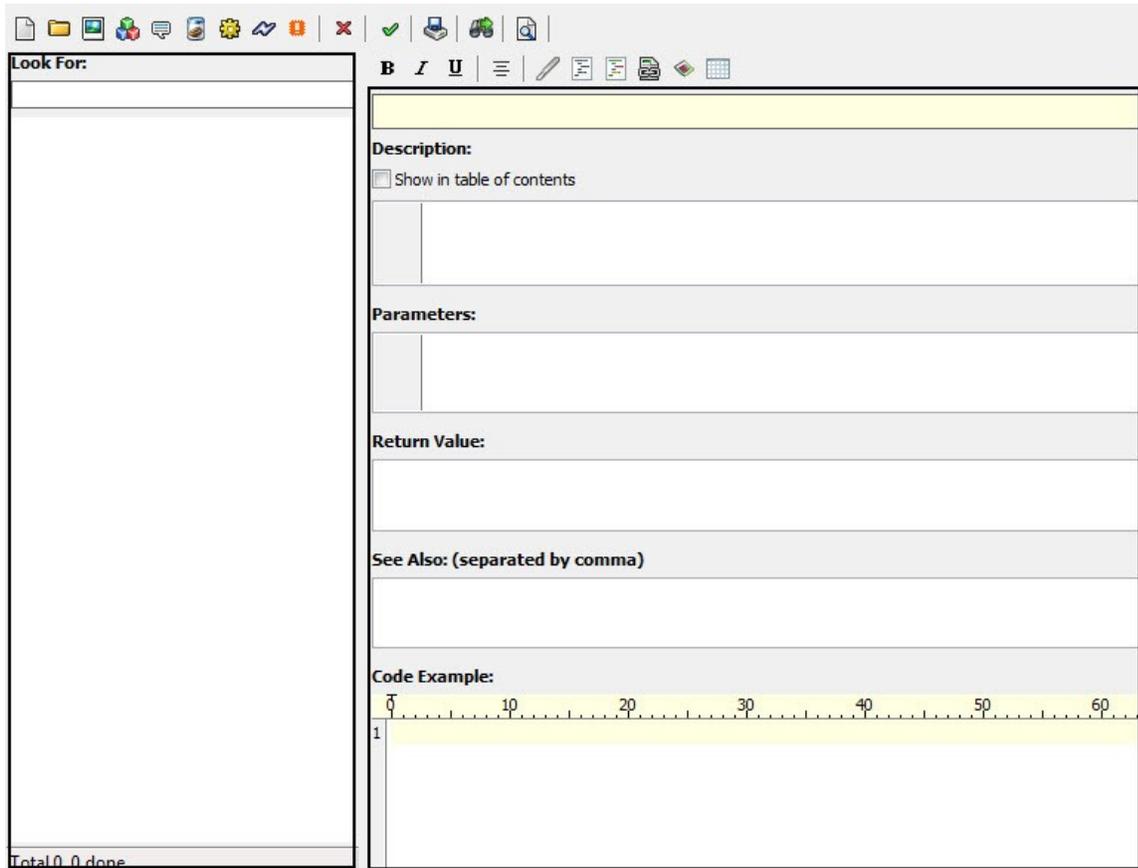


Figure 139: Help file editor interface

From the right side pane, a user can easily write:

- The heading of the topic
- Description of the topic
- Parameters to be shown in the help file
- Return value of the component (if any)
- See also contents
- Code examples for better understanding

Once all the above information about a topic is deemed complete, help file can be previewed

with the help of the preview help button.

Editing .hlp files in PIDE2

For editing help files created in PIDE2:

- Open PIDE2 and open an existing **.hlp file** that was created in PIDE from the open option in the file menu.
- As the file opens in the help file editor, you will be able to change everything from its **Description, Parameters, Return Values, See Also Text** etc.
- After editing a topic, users should mark a topic as **Done** by pressing the done button or by pressing **Ctrl+D key**.

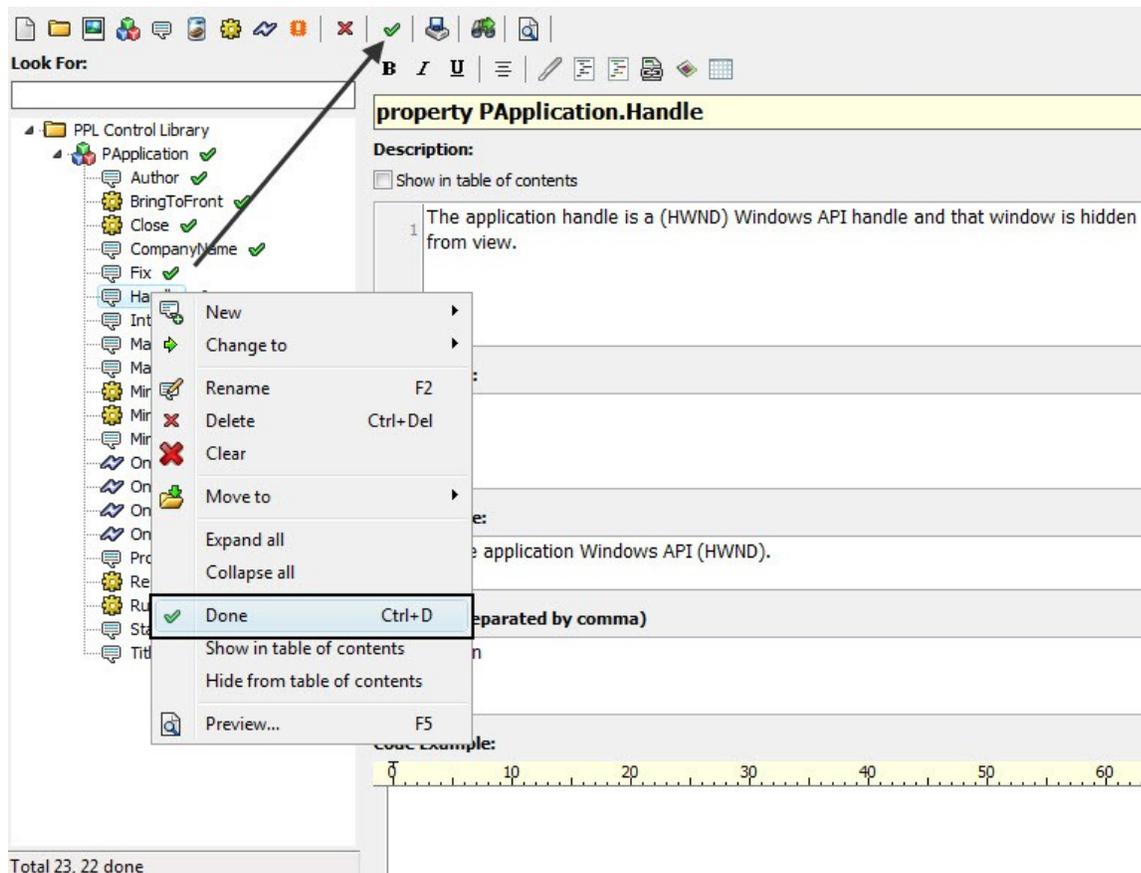


Figure 140: Completing a topic

- After all the deemed topics are complete, users can save the **.hlp file** and preview it by pressing the **Preview Help Button**.

Creating topics from files

With the help editor in PIDE2, you will also be able to create Help File Topics from .ppl or .xml files automatically. Just by scanning these files, PIDE automatically lists all the objects and classes along with methods, events and other code objects in the topic list. From here, users can edit the various topics individually and add the things that are required in a help file.

For adding topics to a help file:

- Open an existing or create a new help file in the help file editor.

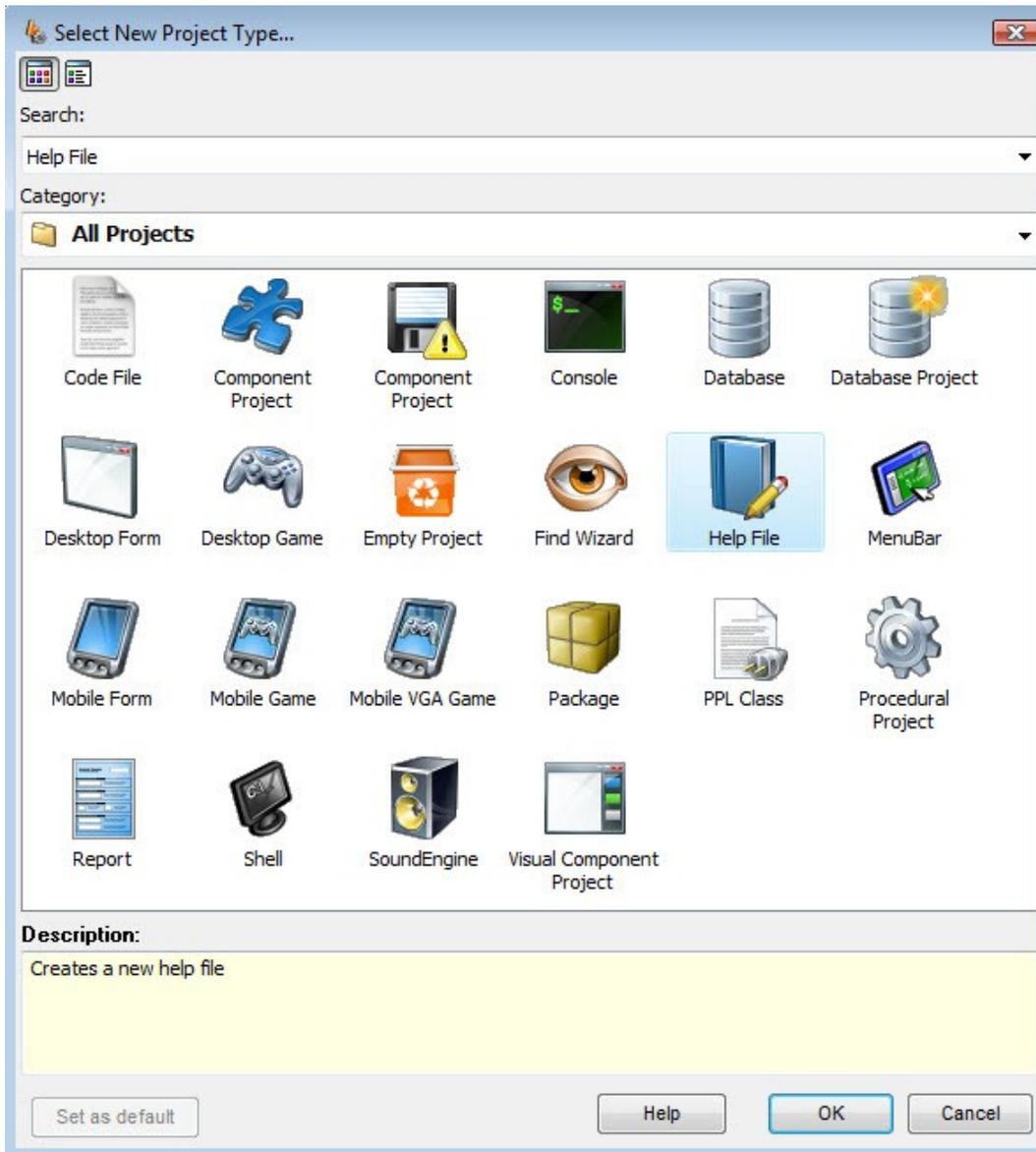


Figure 141: Create Help file project

- Click the **Import/Export** menu and select **Create topics from file...** option.

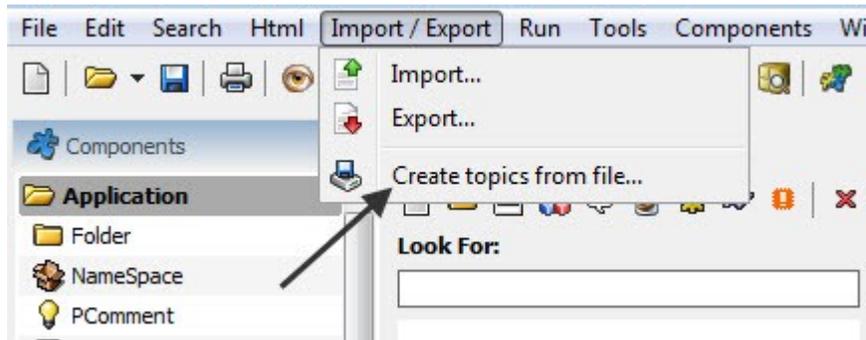


Figure 142: Select Create topics from file...

- Select an .xml or .ppl file from the **import from a file...** window. This will automatically scan the file and include the topics contained in the file.

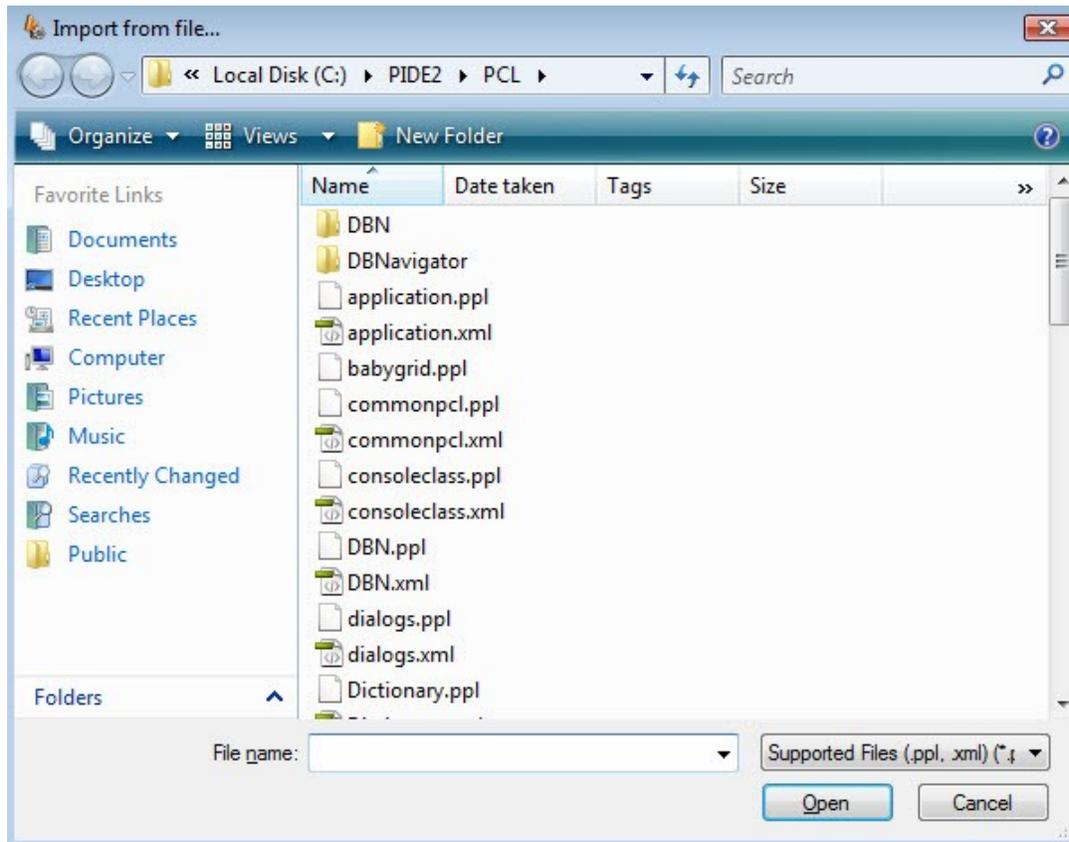


Figure 143: Choose a file

- Now, users can select a topic and edit its properties according to their preference for creating a help file.

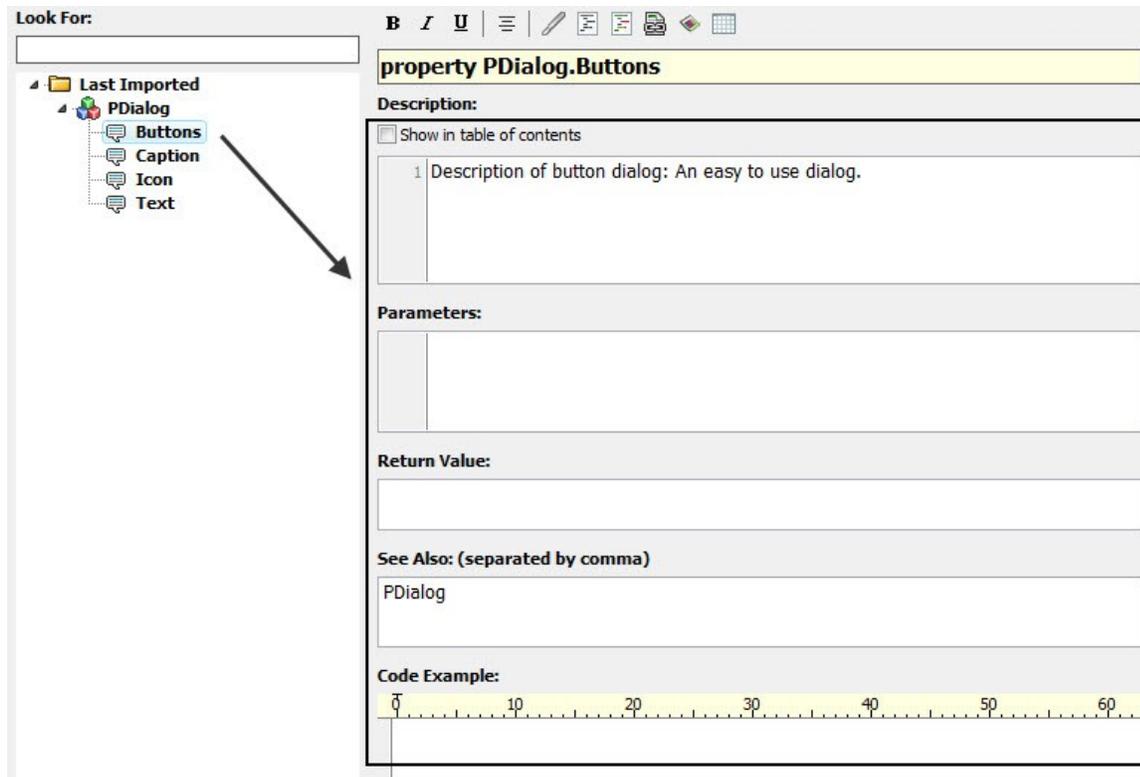


Figure 144: Fill in the requirements

Code Templates In Code Editor

Code templates in PIDE provide the functionality to write code in an easier and hassle free way. Just by creating your own code templates or by manipulating the PPL.ctp code template created as the default code template file, users can create a coding environment that is tweaked according to their coding preferences.

One of the best examples of using a code example is the use of **If condition**. In PIDE, whenever a user enters ‘**ifi**’ and follows it with a **Tab key**, the text is automatically converted to an ‘**if – else**’ statement that just needs the user to fill in the required expression and statements.

Using the Default Code Template

Using the default PIDE code template is a straightforward process. Follow the example given below to use the default code template:

- Create a new code file project in PIDE by pressing **Ctrl+N key** or by selecting “**Create a new project**” from the welcome screen itself.

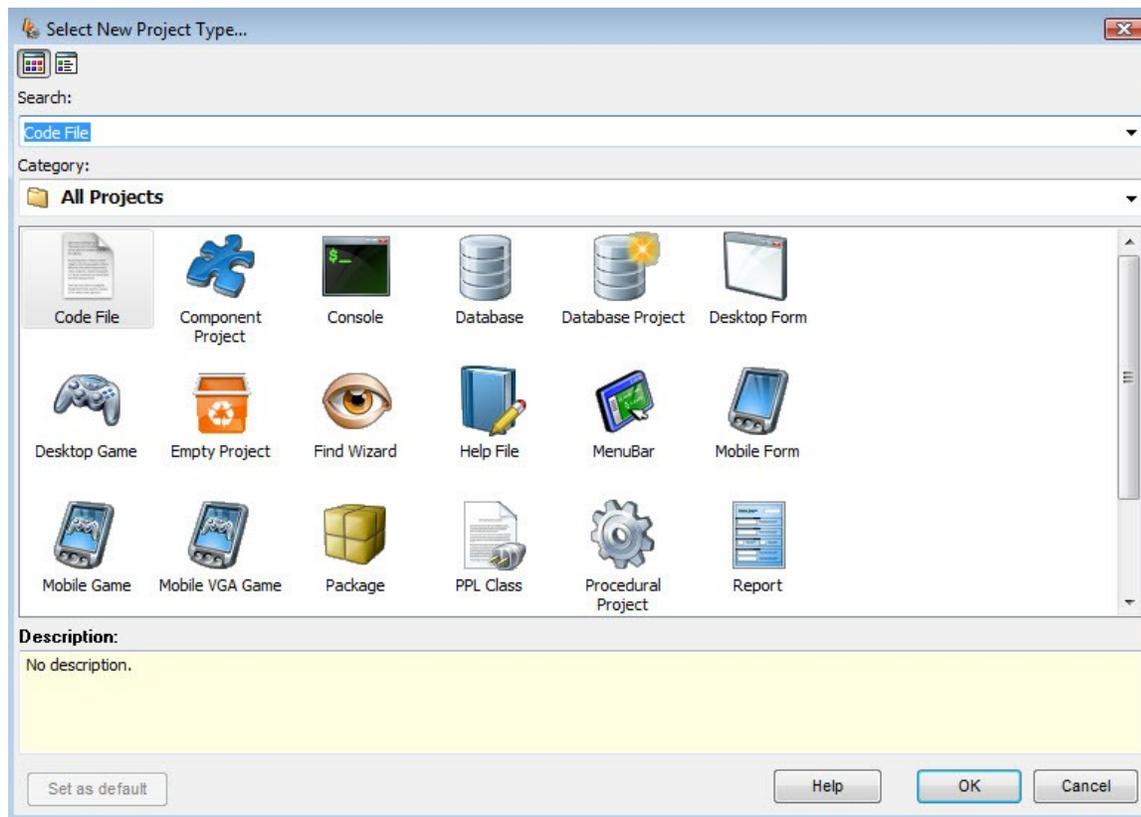
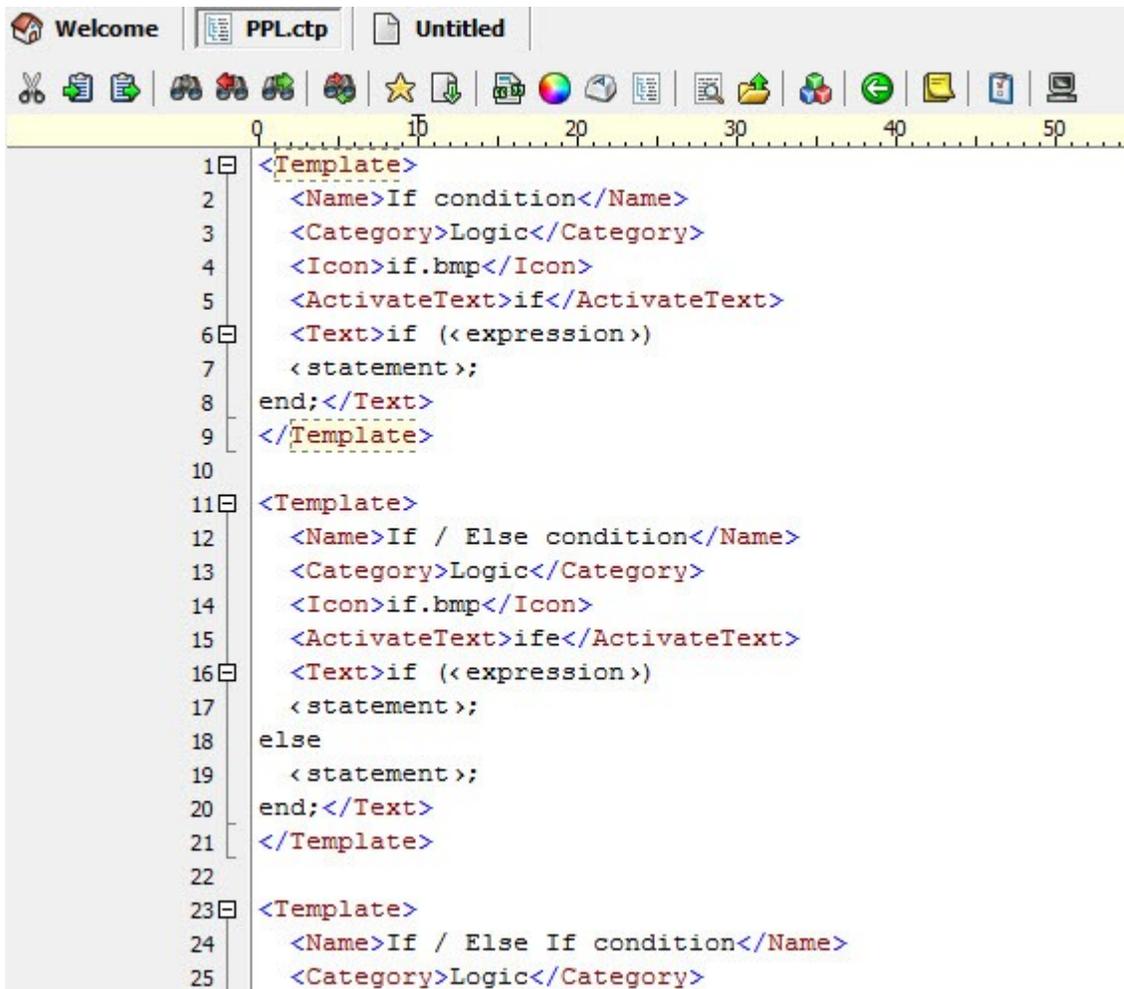


Figure 145: Creating new project

- Go to the file menu and click **Open** to open PPL.ctp file from **/PCL directory** of your

PIDE installation.

- Have a closer look at the **PPL.ctp** file and study the various templates. Notice carefully the `<ActiveText>` tag, this tag will allow you to call a specific code the way it is specified in the `<text>` tag.



The screenshot shows the PIDE editor interface with a toolbar and a code editor. The code editor displays three code templates for conditional logic. The first template is for a simple 'if' statement, the second is for an 'if / else' statement, and the third is for an 'if / else if' statement. Each template is enclosed in `<Template>` tags and contains specific tags for name, category, icon, and active text, followed by the code text to be inserted.

```
1 <Template>
2   <Name>If condition</Name>
3   <Category>Logic</Category>
4   <Icon>if.bmp</Icon>
5   <ActivateText>if</ActivateText>
6   <Text>if (<expression>
7     <statement>;
8   end;</Text>
9 </Template>
10
11 <Template>
12   <Name>If / Else condition</Name>
13   <Category>Logic</Category>
14   <Icon>if.bmp</Icon>
15   <ActivateText>if</ActivateText>
16   <Text>if (<expression>
17     <statement>;
18   else
19     <statement>;
20   end;</Text>
21 </Template>
22
23 <Template>
24   <Name>If / Else If condition</Name>
25   <Category>Logic</Category>
```

Figure 146: Code Template

- Now, in the code file create a class by writing `#class` and press the **Tab key**. This will automatically write the syntax for a class and would require you to enter the necessary values only.

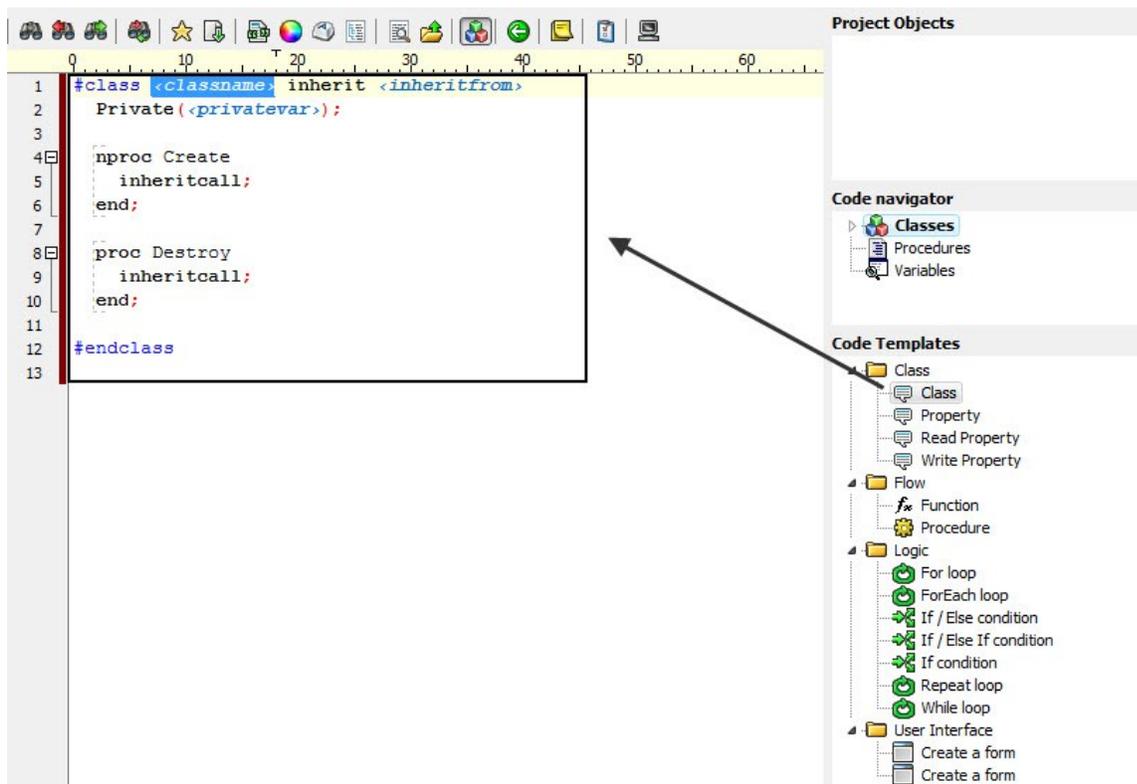
```

1  #class <classname> inherit <inheritfrom>
2      Private (<privatevar>);
3
4  nproc Create
5      inheritcall;
6  end;
7
8  proc Destroy
9      inheritcall;
10 end;
11
12 #endclass
13

```

Figure 147: Using Code template

- Instead writing the **activate text** and hitting the **tab key**, users can also use the code templates directly by using the navigator panel. Under the **Code Templates** pane, double click any code template to include that template automatically on the code.



Modifying a .ctp or code template file

For individual needs, users can modify any.ctp or code template file with their own code. Follow the steps given below to write your own template code in a code template file:

- In PIDE, go to file menu and **select Open** to open the **.ctp file** from a location in a hard disk.
- Scroll down to the last template of **.ctp file** and write `<Template>` to create a new code template tag.

```
146 <Icon>method.bmp</Icon>
147 <ActivateText>proc</ActivateText>
148 <Text>proc <name> (<parameters>)
149 <statement>;
150 end;
151 </Text>
152 </Template>
153
154 <Template>
155 <Name>Function</Name>
156 <Category>Flow</Category>
157 <Icon>function.bmp</Icon>
158 <ActivateText>func</ActivateText>
159 <Text>func <name> (<parameters>)
160 <statement>;
161 return (<returnvalue>);
162 end;
163 </Text>
164 </Template>
165
166 <Template>
167 |
168
169
170
```

Figure 148: Code Template

- Write the subsequent tag as **<Name>** tag and write the name of the element you want to create the template of. End this tag with an **</Name>** tag.
- Next, add the **<Category>** tag to specify the category of the element. This tag too will end with a **</Category>** tag.
- As the name suggests, the **<Icon>** tag specifies an icon for the element whose template is being created. Like every tag, this tag too will end with an ending tag i.e. **</Icon>**.
- **<ActivateText>** tag is the most important tag in the template and it will specify the

text with which this template will be called. End this tag with a **</ActivateText>** tag.

- Lastly, the **<Text>** tag will contain the text that will be written in place of the text specified in the **<ActivateText>** tag. This tag will end with **</Text>**.
- End the template with **</Template>** tag.
- Save this template by pressing '**Ctrl+S**' and write the element written in your **<ActivateText>** tag with a Tab key to transform it automatically in the text that was specified in the **<Text>** tag.

Creating and using a Code Template file

Follow the guidelines given below to create your own code template files:

- Create a new code project in PIDE2 and save it as a **.ctp file** under the **/PCL** folder.

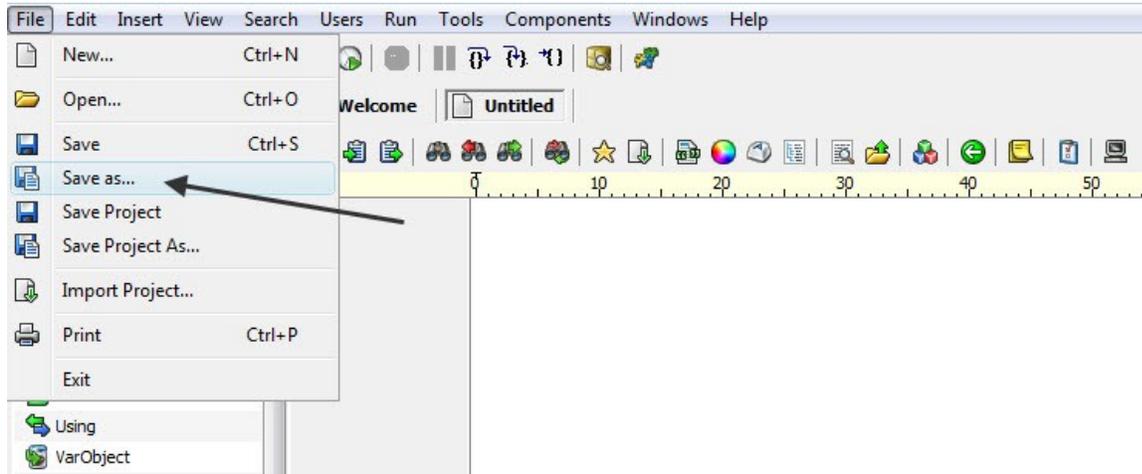


Figure 149: Save project as

- In the file write as many code templates you want to write while following the proper syntax.
- Given below is the syntax for creating a code template:

```
<Template>
<Name>Name of code</Name>
<Category>Category of code</Category>
    <Icon>Icon representing the code</Icon>
<ActivateText>Text that needs to be changed</ActivateText>
<Text>Code that will be used</Text>
</Template>
```

- After creating your code template, save the file again and restart PIDE2. Alternatively, you can also go to Tools in the standard menu and select the Reload definition files option to reload the file.

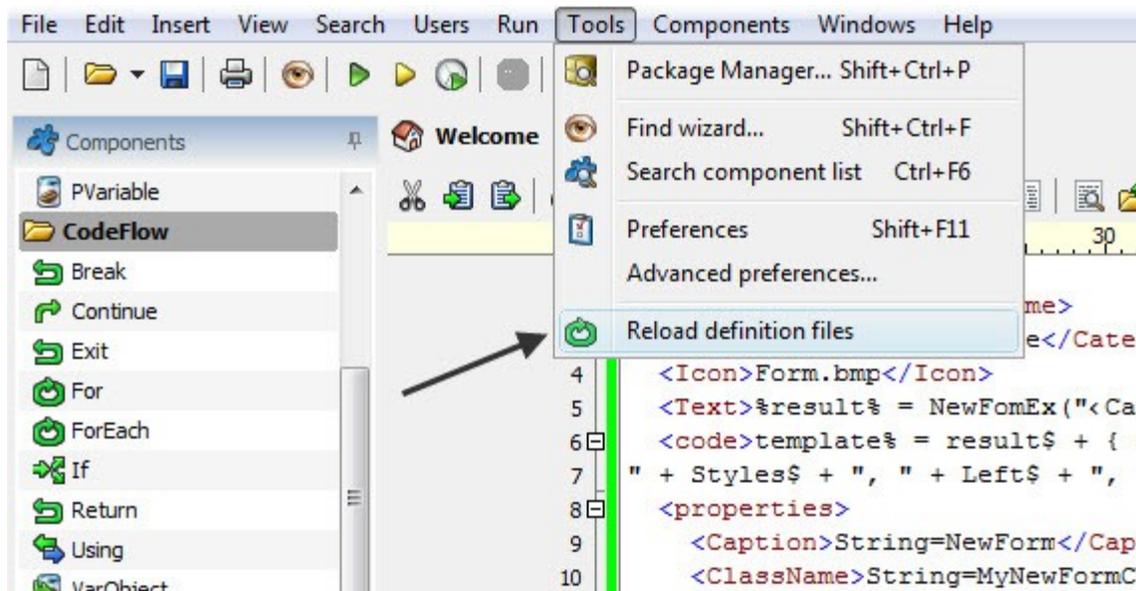


Figure 150: Reloading Definition Files

- Once in the memory, every **.ctp file** present in the **/PCL** folder is used by PIDE2 automatically.

The PIDE Shell

Shell or the Dos prompt is a useful utility that is easier to use and vary capable of providing advanced programming control over the execution of an application. Even if you do not use shell prompt a lot, you will love the way PIDE works just like the dos prompt and allows you to do pretty much everything that one is able to do in a dos prompt. Here we will have a quick look at opening the PPL shell and working with it:

- Open PIDE application and double click the **Shell icon** to open a PPL shell

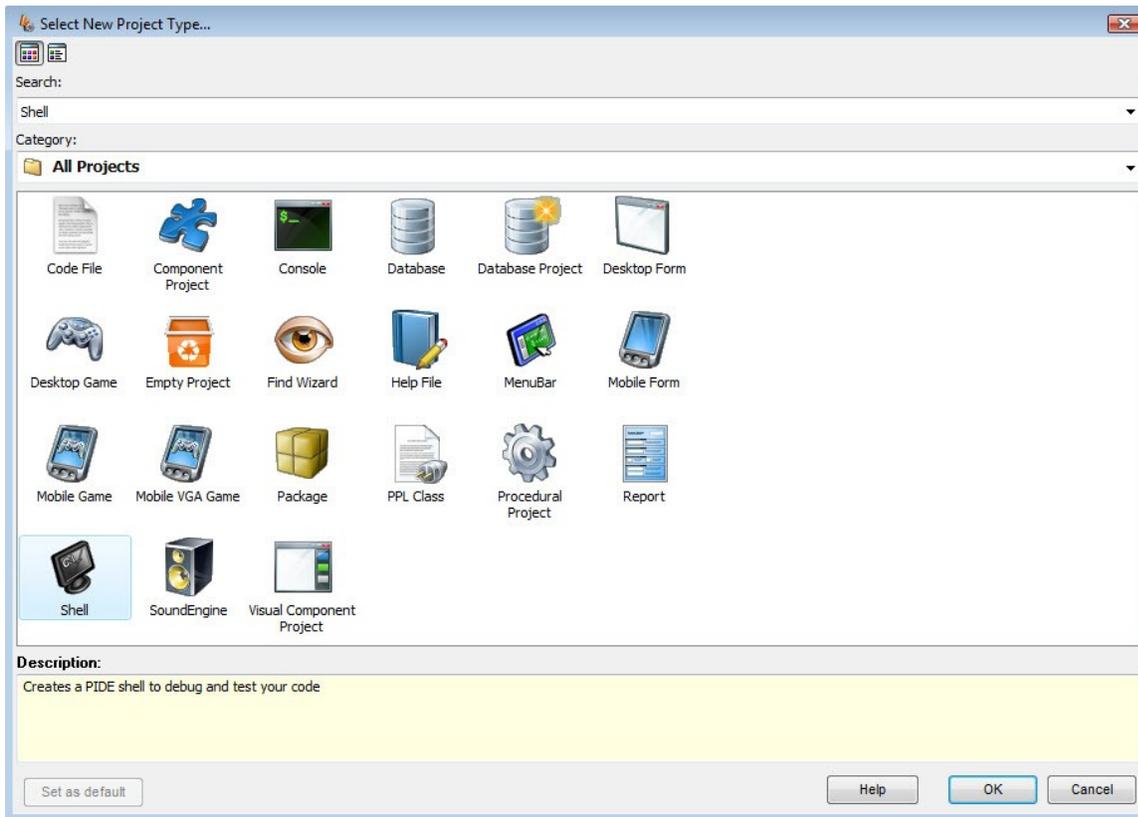


Figure 151: Creating new project

- In the PPL shell, you can write any shell command and work with it like you do in dos prompt. Here, the **shell prompt** will be represented by the location of your PIDE directory.

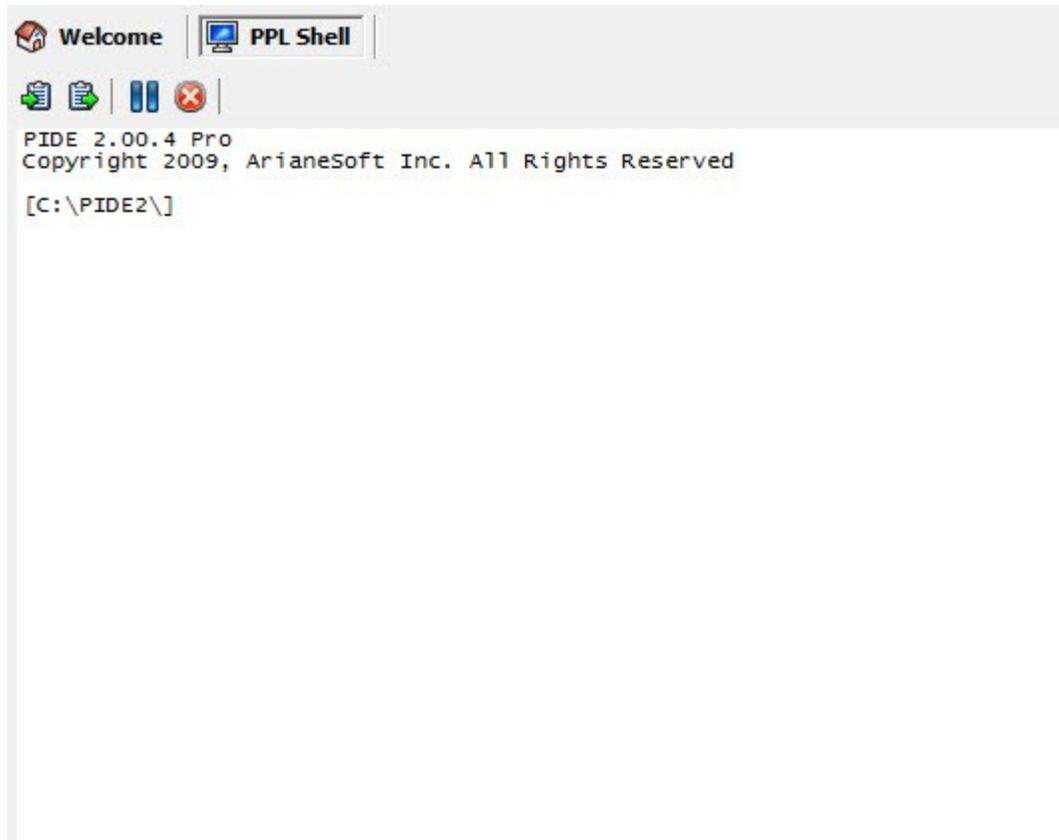


Figure 152: PIDE shell

- Users can also enter other directories by writing **CD** followed by the **directory name** and pressing the **enter key**.

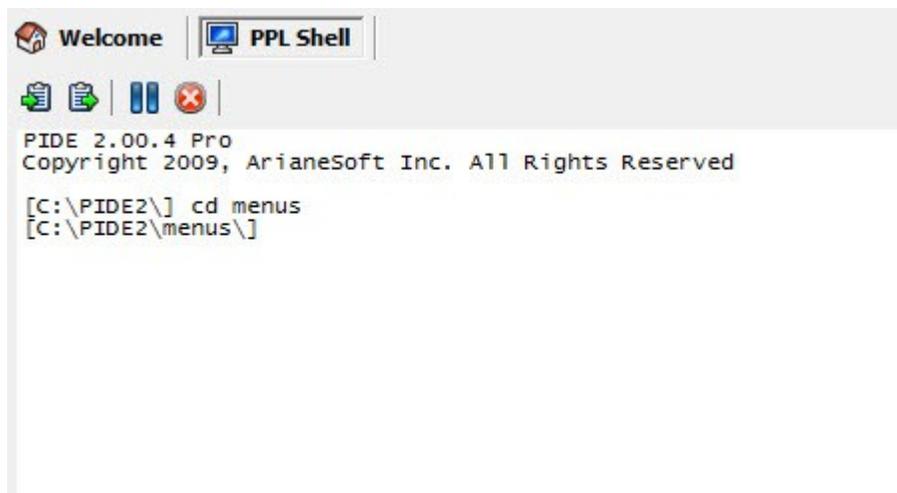
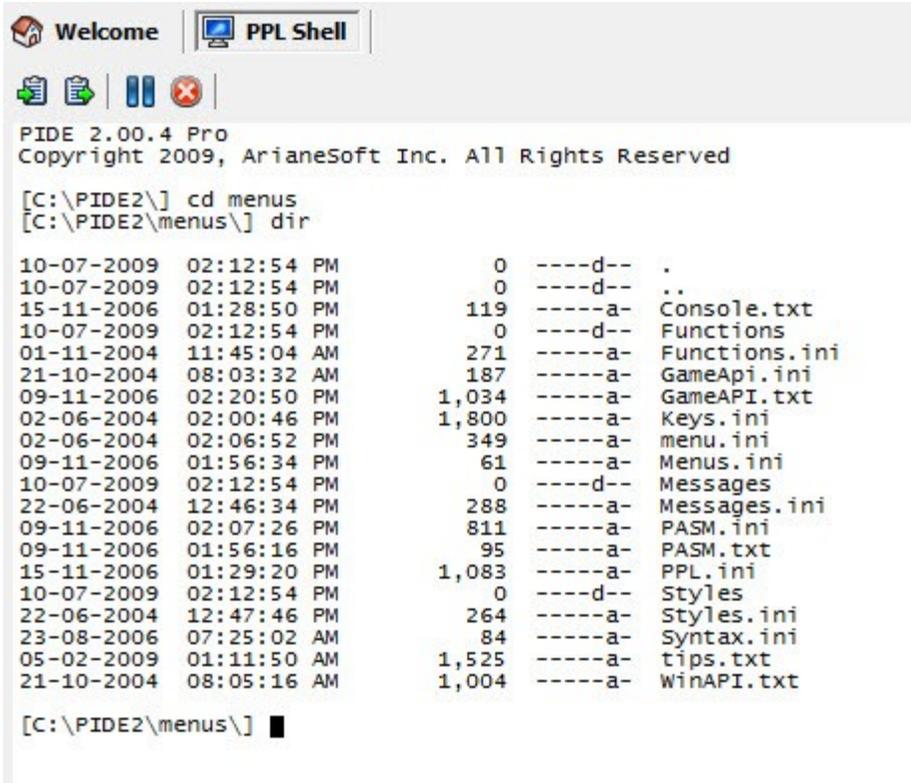


Figure 153: Working in PIDE shell

- Just like dos prompt, writing the **dir command** and pressing the **enter key** would result in a list of all the directories present within the PIDE directory.



```

Welcome | PPL Shell |
PIDE 2.00.4 Pro
Copyright 2009, ArianeSoft Inc. All Rights Reserved

[C:\PIDE2\] cd menus
[C:\PIDE2\menus\] dir

10-07-2009 02:12:54 PM          0  ----d--  .
10-07-2009 02:12:54 PM          0  ----d--  ..
15-11-2006 01:28:50 PM        119  ----a-   Console.txt
10-07-2009 02:12:54 PM          0  ----d--  Functions
01-11-2004 11:45:04 AM        271  ----a-   Functions.ini
21-10-2004 08:03:32 AM        187  ----a-   GameApi.ini
09-11-2006 02:20:50 PM       1,034  ----a-   GameAPI.txt
02-06-2004 02:00:46 PM       1,800  ----a-   Keys.ini
02-06-2004 02:06:52 PM        349  ----a-   menu.ini
09-11-2006 01:56:34 PM         61  ----a-   Menus.ini
10-07-2009 02:12:54 PM          0  ----d--  Messages
22-06-2004 12:46:34 PM        288  ----a-   Messages.ini
09-11-2006 02:07:26 PM        811  ----a-   PASM.ini
09-11-2006 01:56:16 PM         95  ----a-   PASM.txt
15-11-2006 01:29:20 PM       1,083  ----a-   PPL.ini
10-07-2009 02:12:54 PM          0  ----d--  Styles
22-06-2004 12:47:46 PM        264  ----a-   Styles.ini
23-08-2006 07:25:02 AM         84  ----a-   Syntax.ini
05-02-2009 01:11:50 AM       1,525  ----a-   tips.txt
21-10-2004 08:05:16 AM       1,004  ----a-   WinAPI.txt

[C:\PIDE2\menus\] █
```

Figure 154: DIR command in PIDE shell

- Likewise, pressing enter with **ls command** will list all the contents present in a directory.

```

Welcome | PPL Shell
[PIDE 2.00.4 Pro
Copyright 2009, ArianeSoft Inc. All Rights Reserved

[C:\PIDE2\] cd menus
[C:\PIDE2\menus\] dir

10-07-2009 02:12:54 PM          0 ----d-- .
10-07-2009 02:12:54 PM          0 ----d-- ..
15-11-2006 01:28:50 PM        119 -----a- Console.txt
10-07-2009 02:12:54 PM          0 ----d-- Functions
01-11-2004 11:45:04 AM        271 -----a- Functions.ini
21-10-2004 08:03:32 AM        187 -----a- GameApi.ini
09-11-2006 02:20:50 PM       1,034 -----a- GameAPI.txt
02-06-2004 02:00:46 PM       1,800 -----a- Keys.ini
02-06-2004 02:06:52 PM        349 -----a- menu.ini
09-11-2006 01:56:34 PM         61 -----a- Menus.ini
10-07-2009 02:12:54 PM          0 ----d-- Messages
22-06-2004 12:46:34 PM        288 -----a- Messages.ini
09-11-2006 02:07:26 PM        811 -----a- PASM.ini
09-11-2006 01:56:16 PM         95 -----a- PASM.txt
15-11-2006 01:29:20 PM       1,083 -----a- PPL.ini
10-07-2009 02:12:54 PM          0 ----d-- Styles
22-06-2004 12:47:46 PM        264 -----a- Styles.ini
23-08-2006 07:25:02 AM         84 -----a- Syntax.ini
05-02-2009 01:11:50 AM       1,525 -----a- tips.txt
21-10-2004 08:05:16 AM       1,004 -----a- WinAPI.txt

[C:\PIDE2\menus\] ls

.
Functions
Keys.ini
Messages.ini
Styles
..
Functions.ini
menu.ini
PASM.ini
Styles.ini
Console.txt
GameApi.ini
Menus.ini
PASM.txt
Syntax.ini
GameAPI.txt
Messages
PPL.ini
tips.txt

[C:\PIDE2\menus\]

```

Figure 155: ls command in PIDE shell

- There are many options for opening files in PIDE. For opening files users can either use the type command or the open command.
- If you want to open files in their own window, use the **open command** along with the path of the file

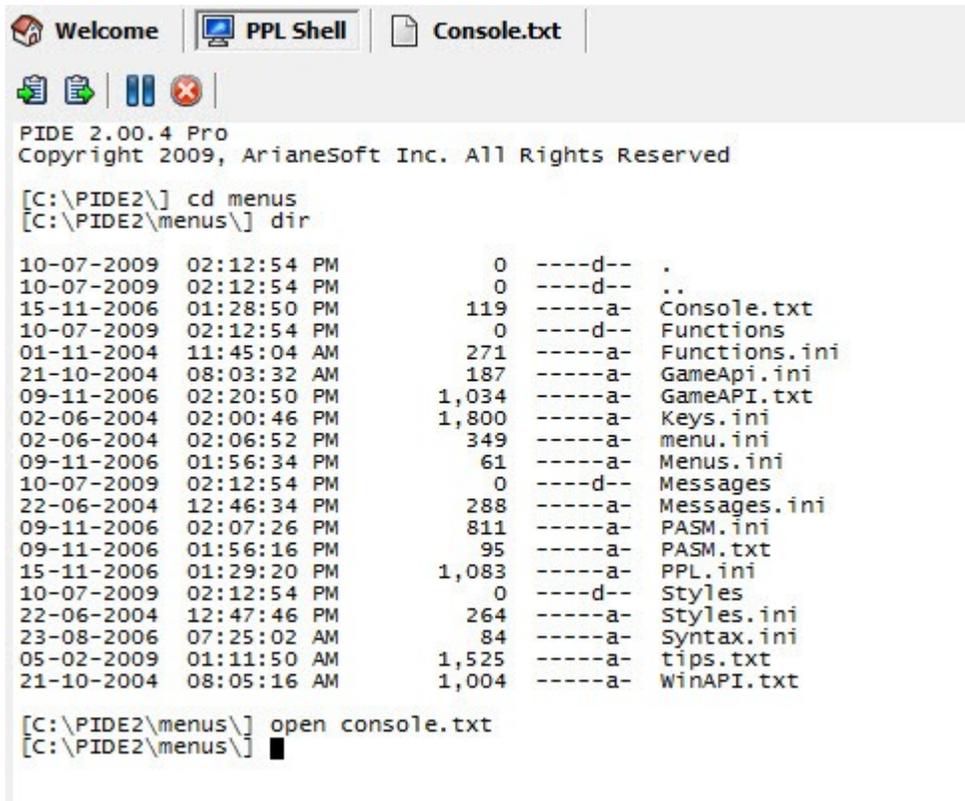


Figure 156: Opening files in PIDE Shell

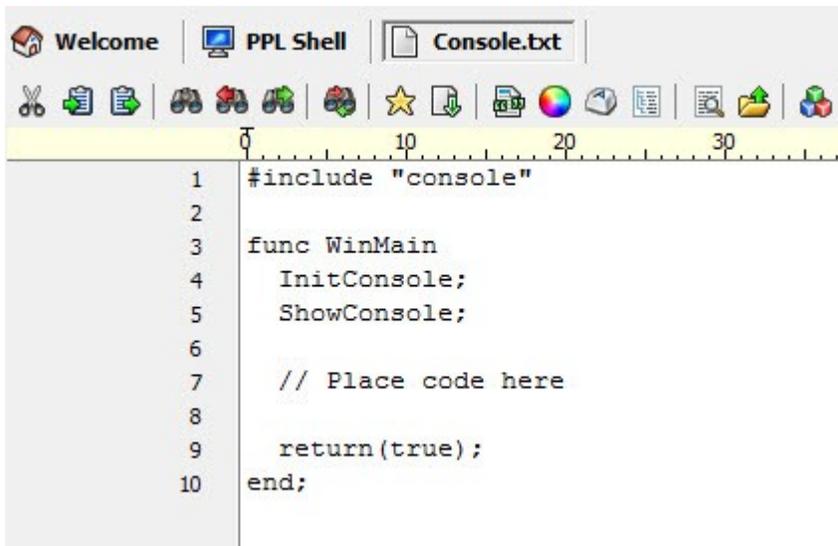


Figure 157: File open in different window

- If you want to open files within the shell, use the **Type command** followed by the path of the file

```

Welcome | PPL Shell
|
[C:\PIDE2\] cd menu
[C:\PIDE2\menu\] dir
10-07-2009 02:12:54 PM          0  ----d--  .
10-07-2009 02:12:54 PM          0  ----d--  ..
15-11-2006 01:28:50 PM        119  -----a- Console.txt
10-07-2009 02:12:54 PM          0  ----d--  Functions
01-11-2004 11:45:04 AM        271  -----a- Functions.ini
21-10-2004 08:03:32 AM        187  -----a- GameApi.ini
09-11-2006 02:20:50 PM       1,034  -----a- GameAPI.txt
02-06-2004 02:00:46 PM       1,800  -----a- Keys.ini
02-06-2004 02:06:52 PM        349  -----a- menu.ini
09-11-2006 01:56:34 PM         61  -----a- Menus.ini
10-07-2009 02:12:54 PM          0  ----d--  Messages
22-06-2004 12:46:34 PM        288  -----a- Messages.ini
09-11-2006 02:07:26 PM        811  -----a- PASM.ini
09-11-2006 01:56:16 PM         95  -----a- PASM.txt
15-11-2006 01:29:20 PM       1,083  -----a- PPL.ini
10-07-2009 02:12:54 PM          0  ----d--  Styles
22-06-2004 12:47:46 PM        264  -----a- Styles.ini
23-08-2006 07:25:02 AM         84  -----a- Syntax.ini
05-02-2009 01:11:50 AM       1,525  -----a- tips.txt
21-10-2004 08:05:16 AM       1,004  -----a- WinAPI.txt

[C:\PIDE2\menu\] type console.txt
Filename: C:\PIDE2\menu\Console.txt
#include "console"

func WinMain
  InitConsole;
  ShowConsole;

  // Place code here

  return(true);
end;

[C:\PIDE2\menu\]

```

Figure 158: File oprn in PIDE shell

- Just like the **dos command prompt**, PIDE shell prompt too complies with the **CLS command** and allows you to clear everything that is on the screen.

Batch Operations In PIDE

Batch operations in PIDE is a handy utility that either allows a user to insert new objects with properties assigned to them in a batch manner or allows users to set the properties in of a group of objects in a single go.

- For opening the batch operations, open a new project and use the batch operations button to open batch operations window.
- Users can select the Operation drop down menu to select the operation they want to perform. For doing so, just click the **drop down menu** and select the **Insert new objects** for inserting new objects with properties assigned in a batch manner. For setting the properties for a set of objects, choose the other option i.e. **Set Properties**.

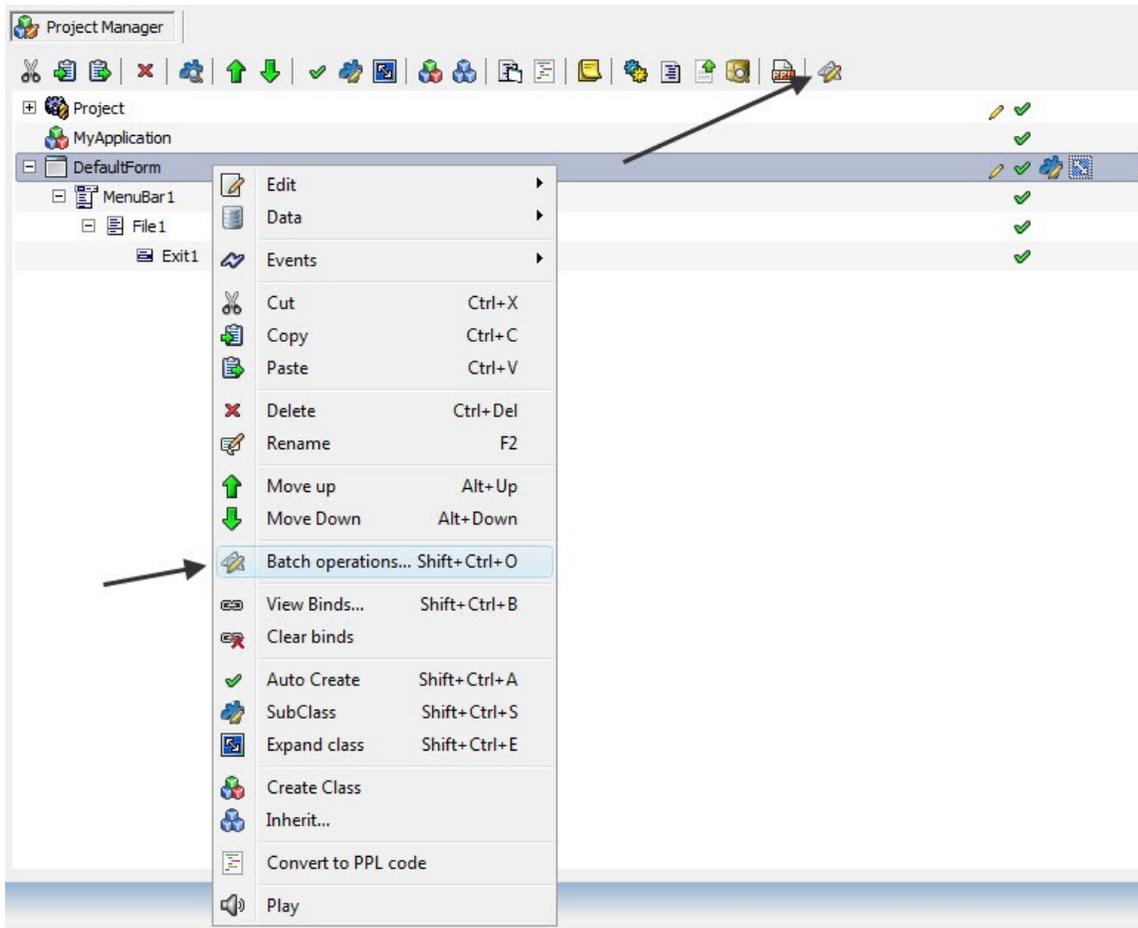


Figure 159: Access batch operations

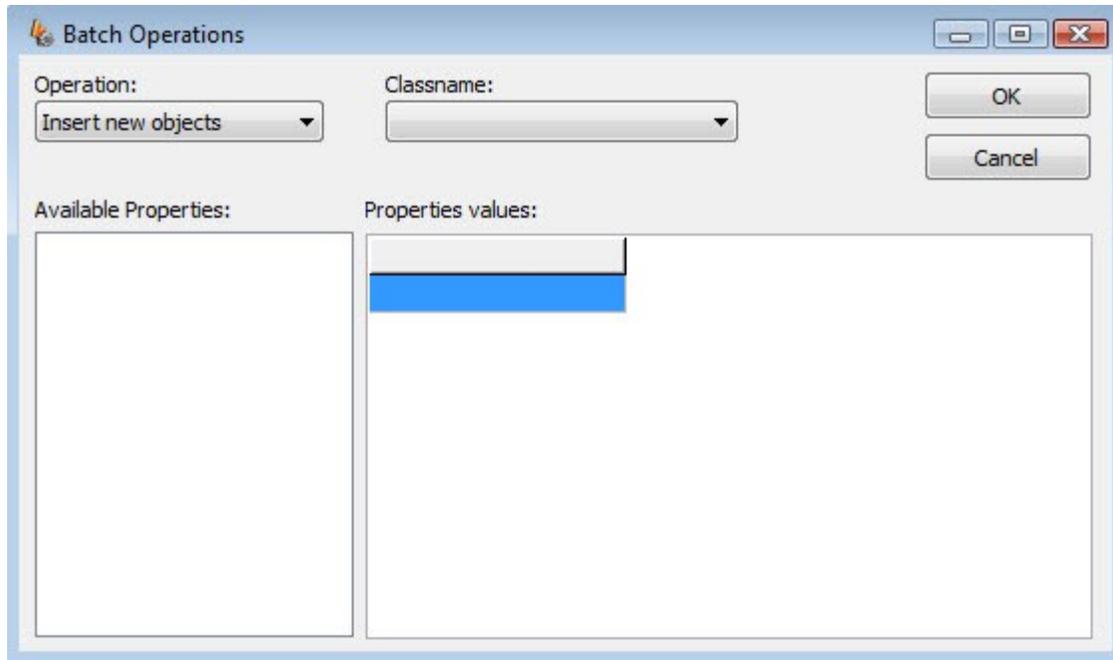


Figure 160: Batch Operations window in PIDE

- After setting the operation select a **classname** that you would want to apply a batch operation to.
- After selecting a **classname**, select the property from the list of available properties by double clicking the property name (alternatively you can also perform drag). Once the property is selected, give a value for and **press ok** to apply.

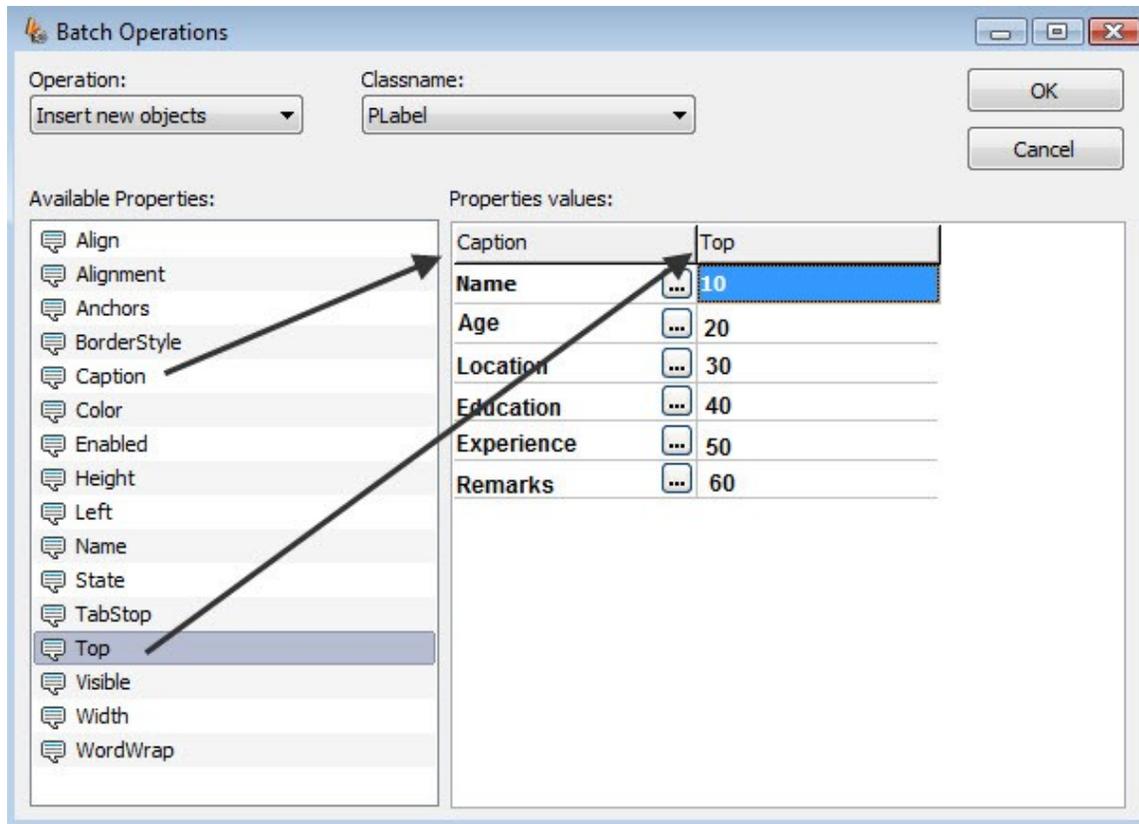


Figure 161: Setting batch operations

Inserting Files With PIDE

Inserting files in PIDE2 is as easy as a drag and drop operation. With its advanced programming techniques, PIDE2 makes it very for a user to include files from the file system directly to the PIDE project. After inserting the file in the project manager, all one needs to do is to specify the type of file it is. For example, while a database file should be specified as a **PDatabase object**, a disk image file can be specified as **PResource object**. Let us look at how to insert files in a project with PIDE2:

- Open a PDIE project in the **project manager view**.
- From **windows explorer**, drag a file in the **project manager**.

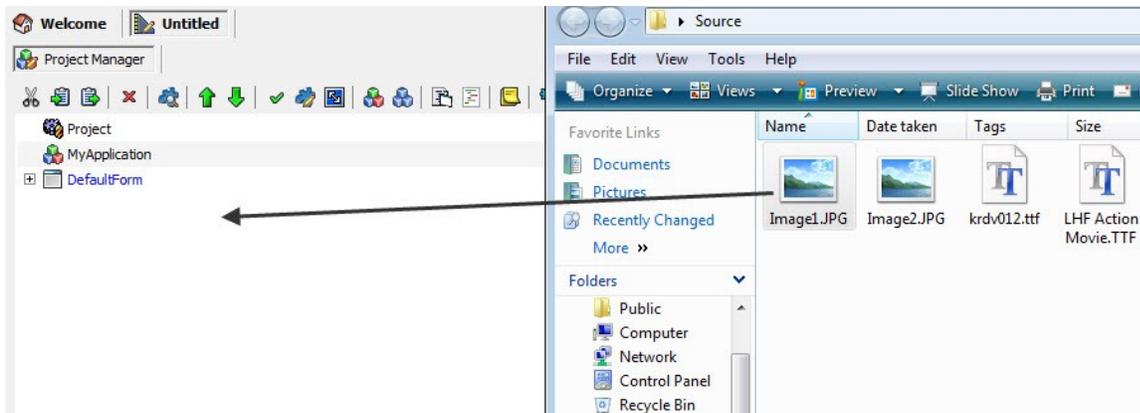


Figure 162: Dragging files

- The drag operation will enable the **SmartMove** feature which will ask you to specify which type of object is being inserted. Here select the appropriate object according to the file and its use you are inserting it for.

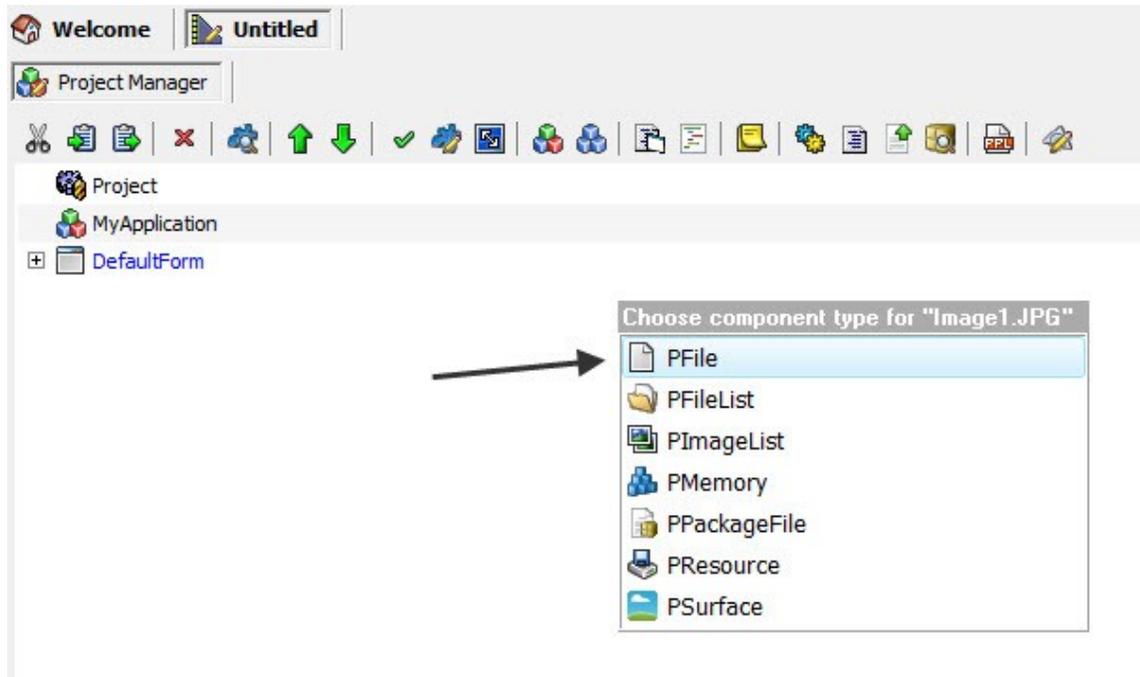


Figure 163: Choosing file type

- Save the project and use the inserted file for other purposes.

Package Files With PIDE

Package files contain multiple files that are encrypted to provide extended security and are made easily accessible in PPL2. Here we will learn how easy it is to package files with PIDE.

- Before we go on to package files with projects in PIDE, we will have to open a project. Open PIDE2 and choose any project like a desktop form.

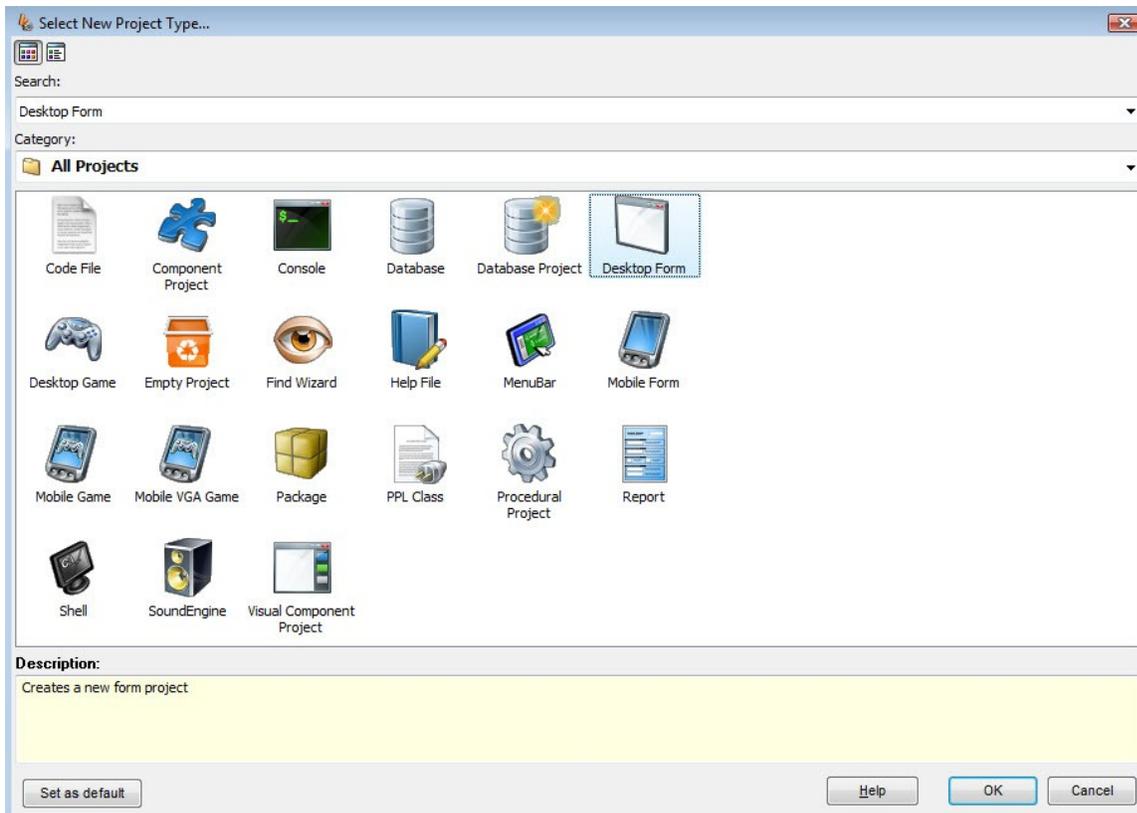


Figure 164: New project

- Once a project is opened, drag a **PPackage Object** from the file group of **components panel** to the **Project Manager**.

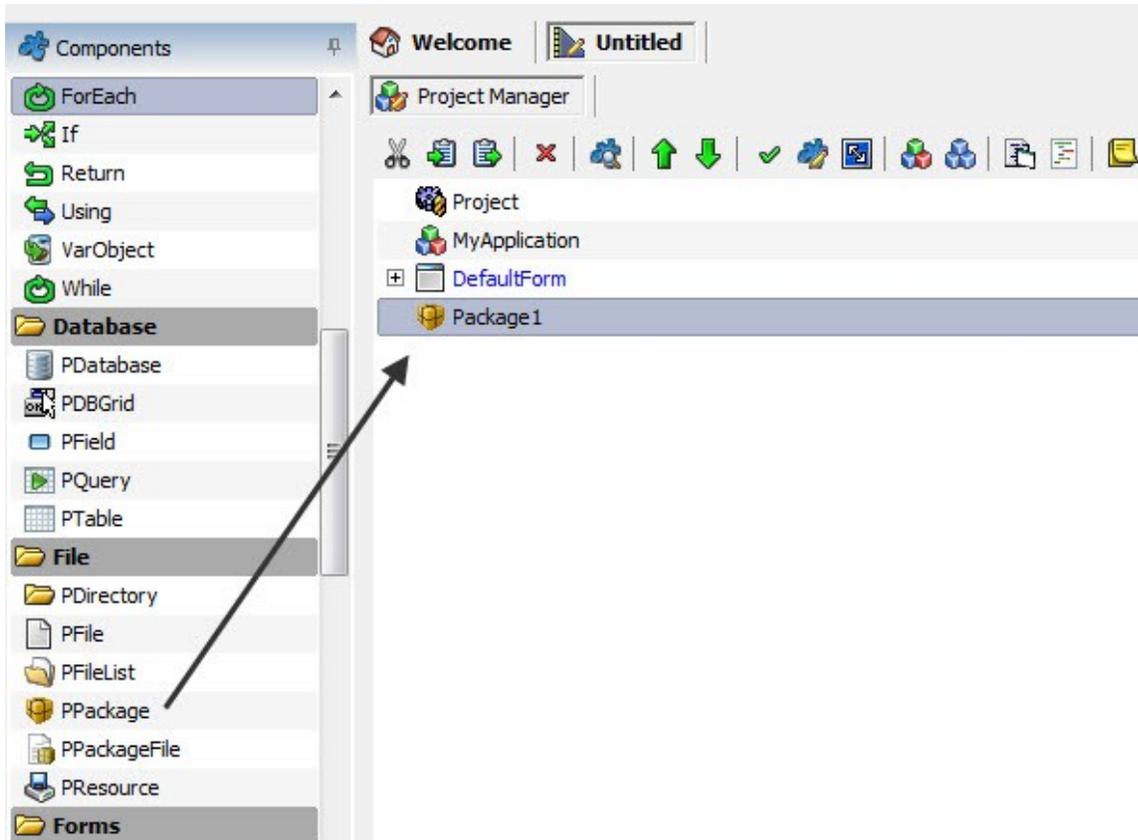


Figure 165: Drage PPackage Object

- After you have dragged a **PPackage** object to the project manager, set its **PackageFilename** property to point to a location on the hard disk.

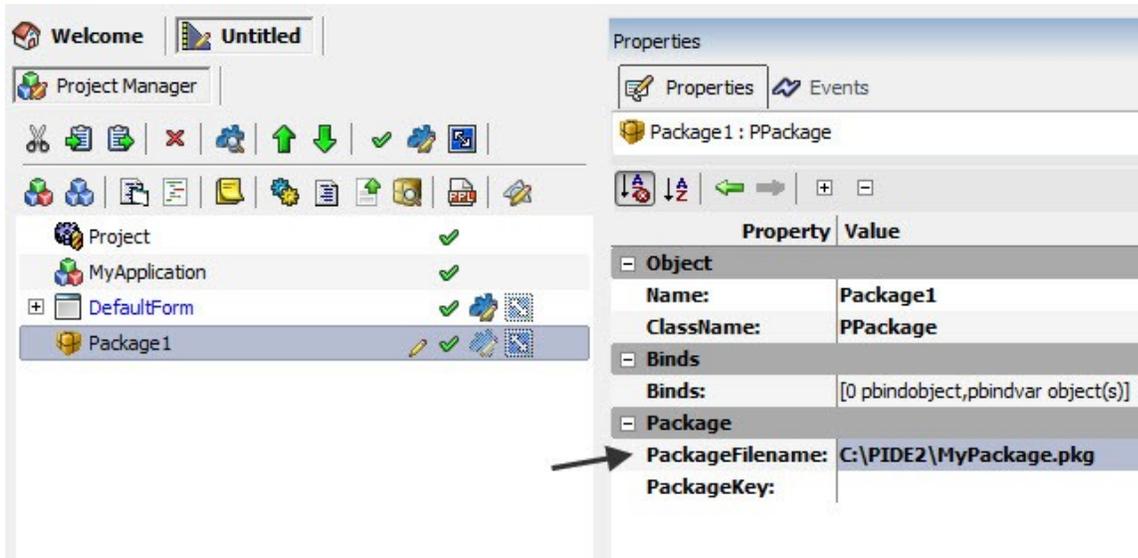


Figure 166: Setting PackageFilename property

- Next, set the **PackageKey** property and specify an **encryption key**.

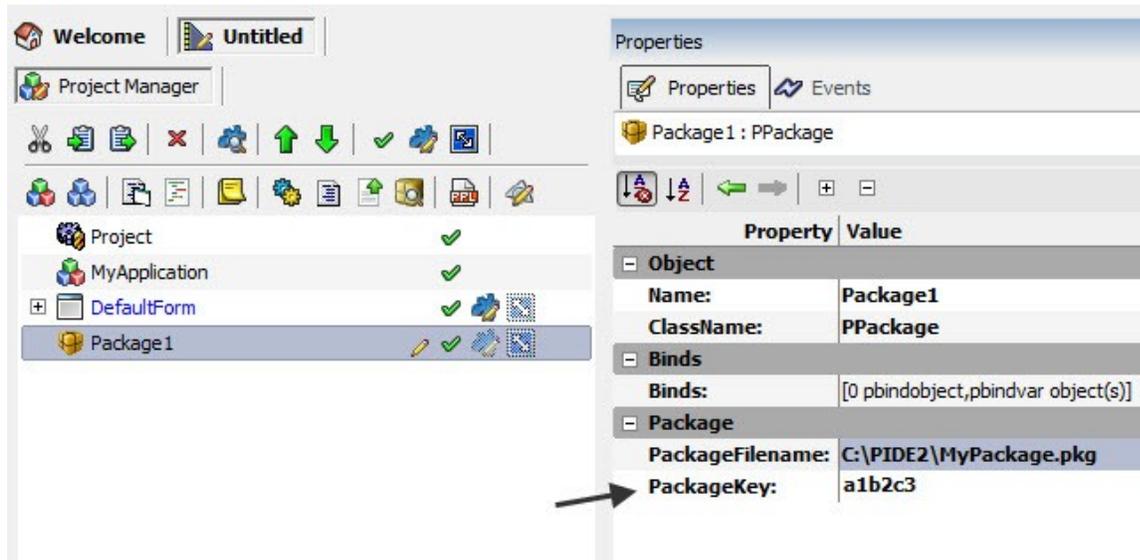


Figure 167: Specifying encryption key

- After performing the above steps, use **windows explorer** and drag files from there to the package object.

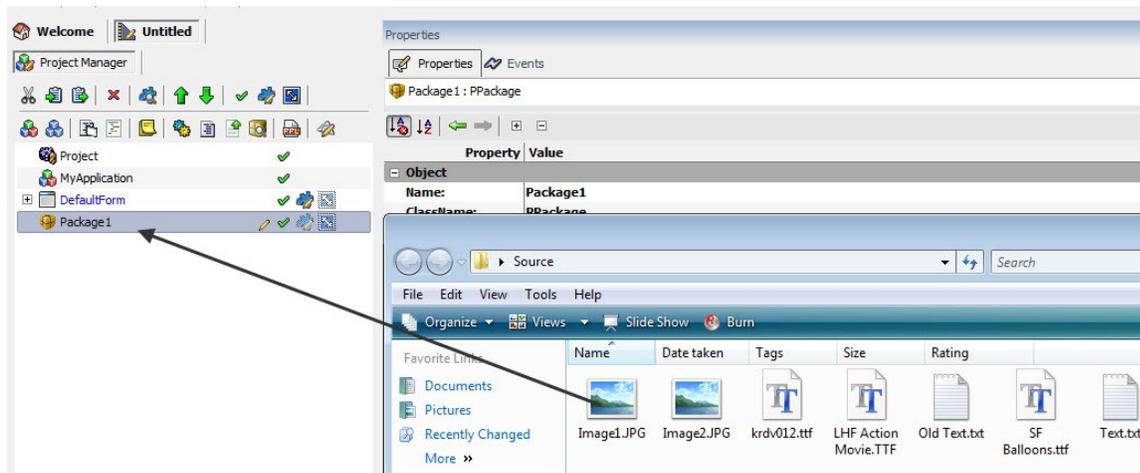


Figure 168: Dragging files to package

- After you have dragged a file, the **SmartMoves** feature will ask you to specify the nature of this file. Here you will have to select the **PPackageFile** option. This will add the file to your package. Just like this file, you can drag as many files as you want from the windows explorer to your package.

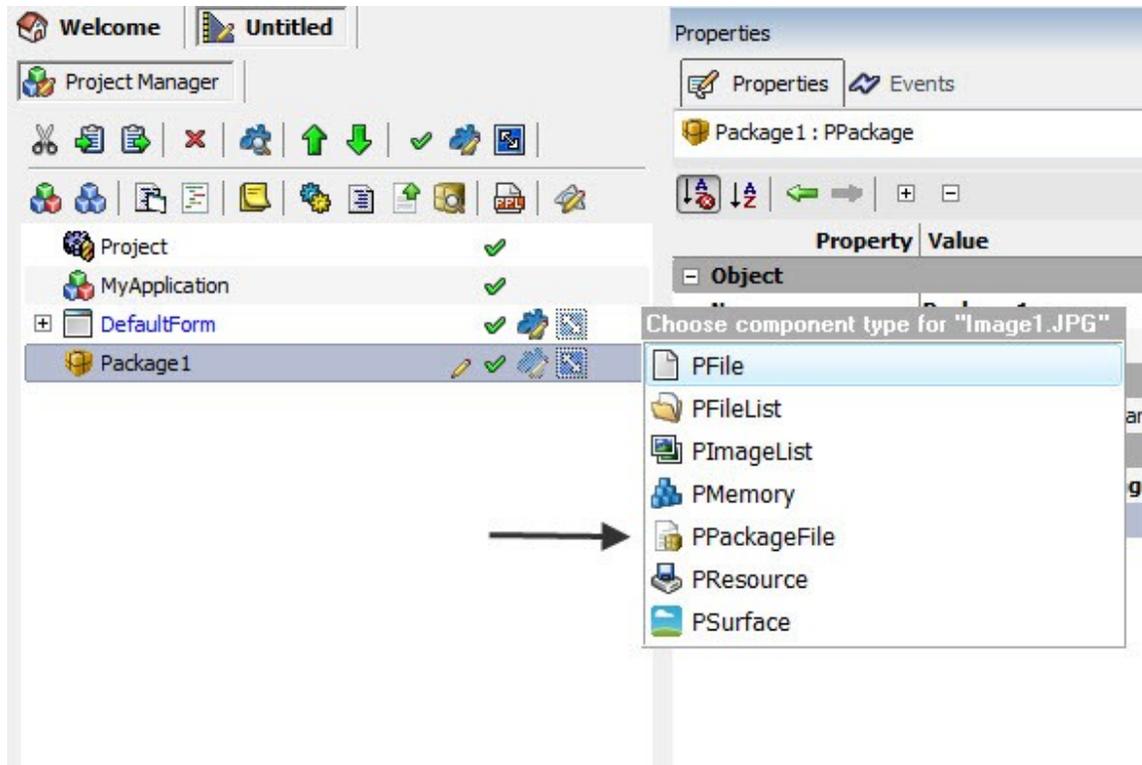


Figure 169: Select PPackage type object

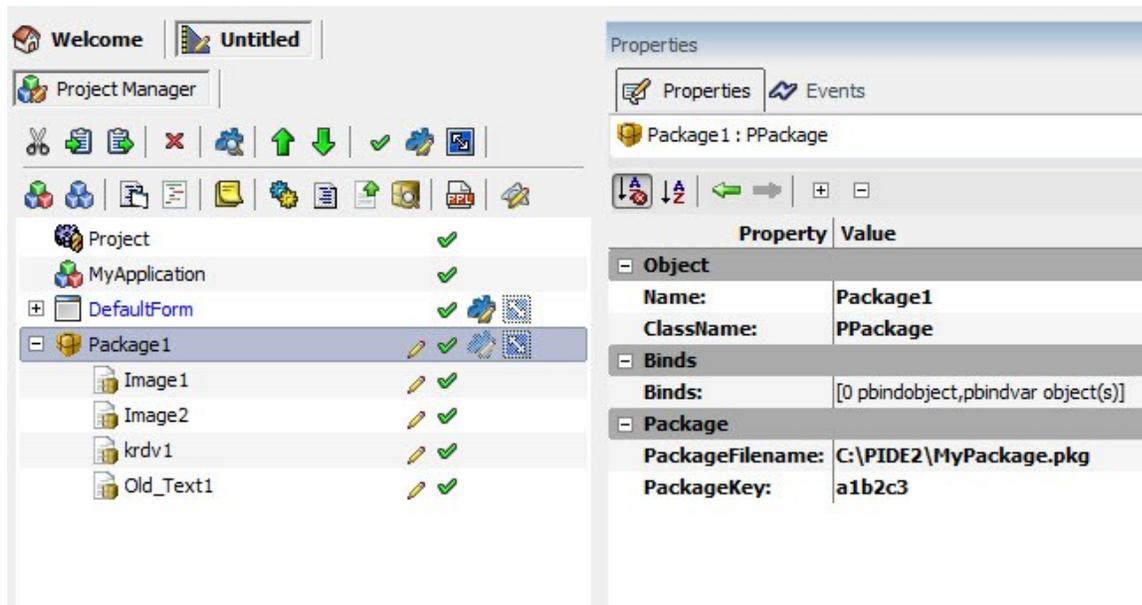


Figure 170: Package object with files that were dragged

- After including all your files in the **PPackage Object**, Go to run and press the **Compile Option** or press **F7** from the keyboard. This will create the package file at the location specified in the **PackageFilename Property**.

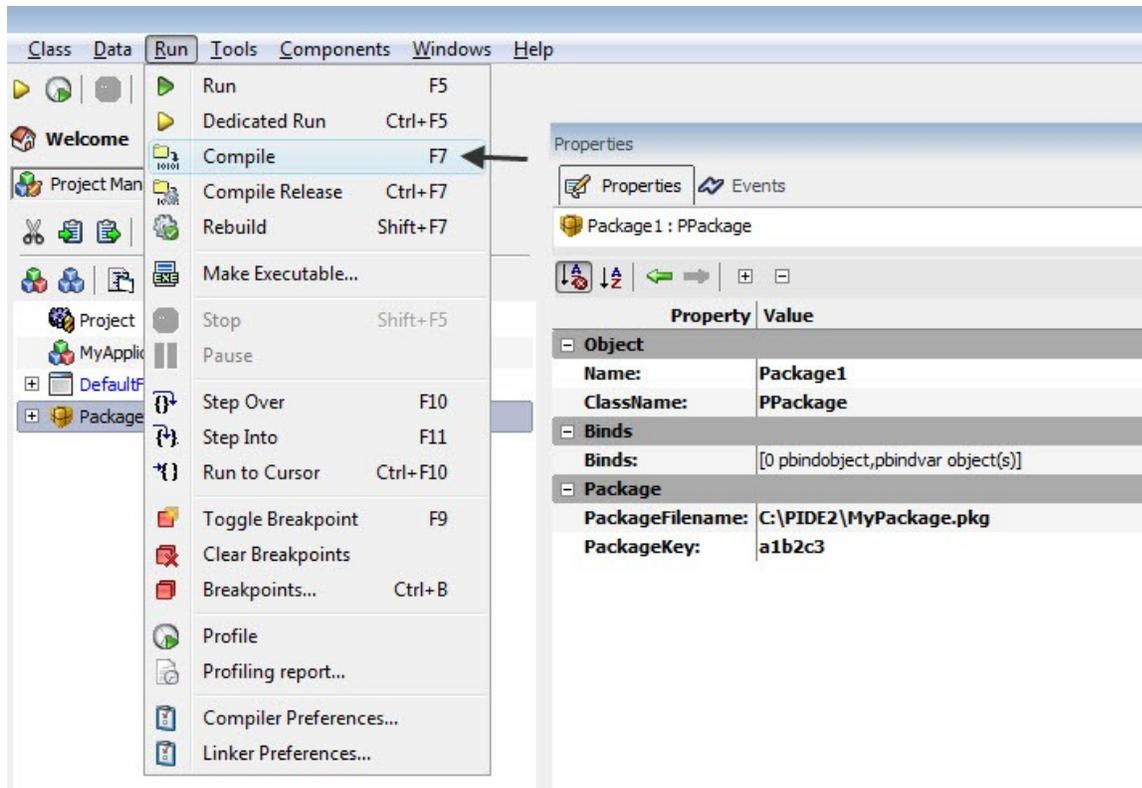


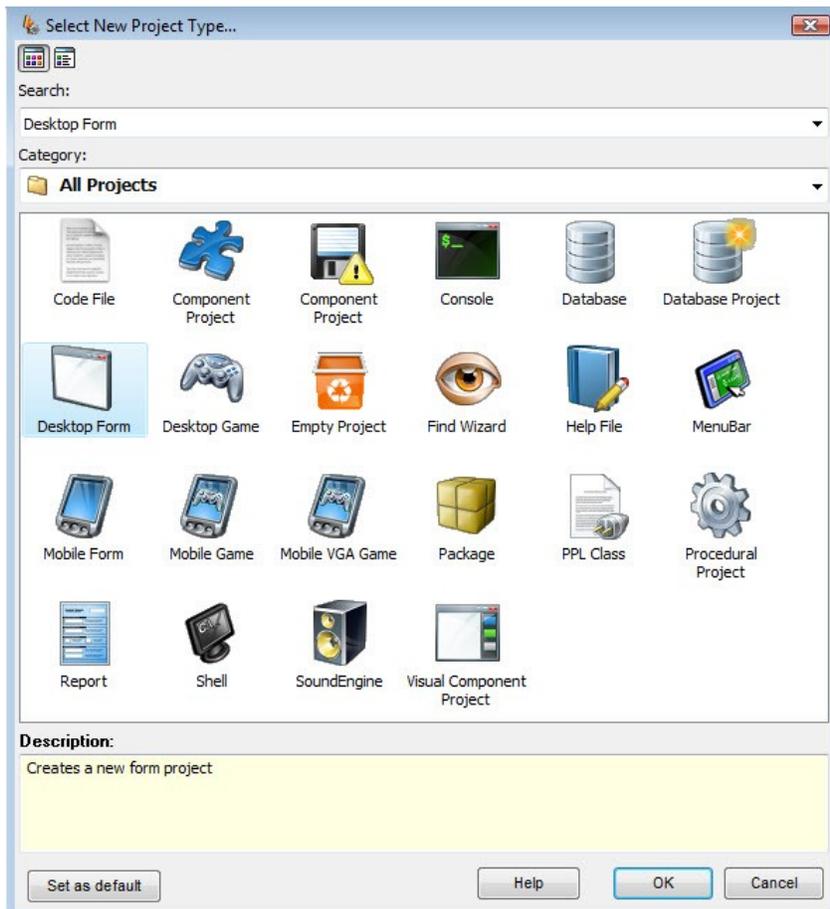
Figure 171: Compiling the package

Code Collection Editor

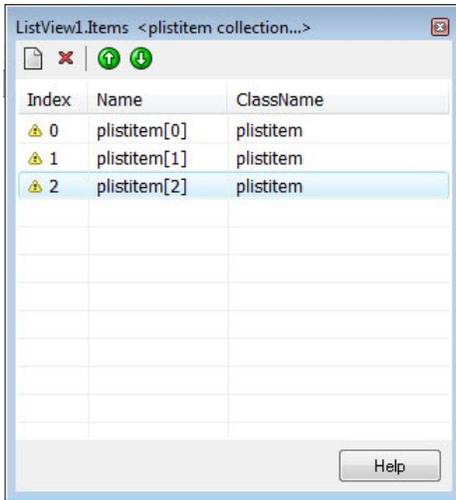
The code collection editor in PIDE works by storing various objects so that they can be used by the property of another object.

A **PListView Object** uses code collection editor to its fullest. By using a list of objects in its **Items Property**, **PListView** allows a user to input a huge many items simultaneously. Follow the given guidelines to create items through code editor:

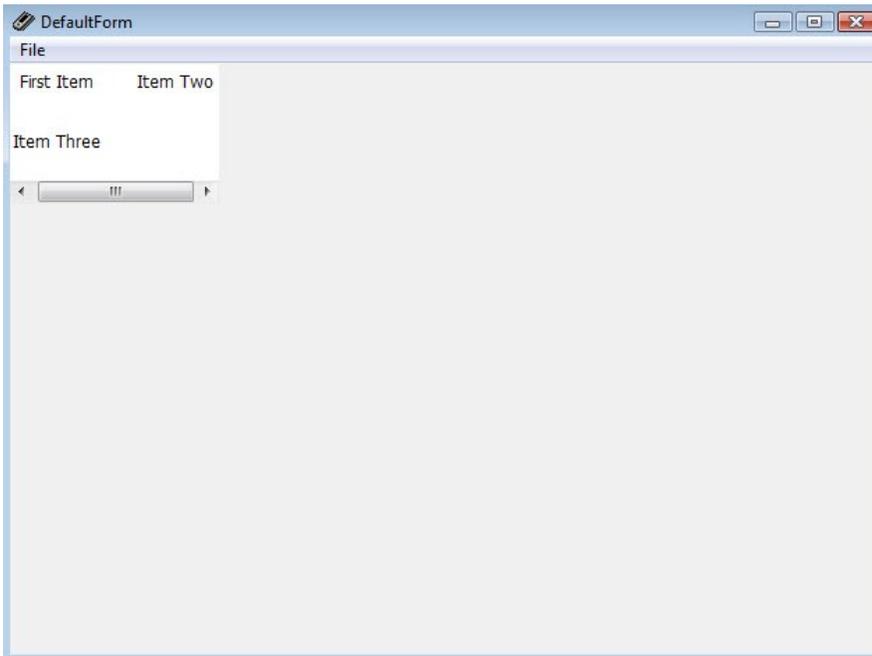
- Create a new **Desktop Form** project



- Drag a **PListView Object** from the standard controls pane in the components panel to the project manager.



- After the code collection editor is closed, run the project to view the items that were created in the **code collection editor**.



Object Binding With PIDE

Object binding is a powerful feature that allows a PIDE user to bind or link the values of two different objects. Apart from linking property values of two objects, the ability to link the property value of an object with a variable provides a programmer with extended flexibility and enhanced control over program execution.

There are many instances when a programmer needs to create objects that depend on the value of the property of other objects. A popular example would be while changing the theme of the application. When a user changes the color theme of an application, other things like text color, text size, menu bar size, selection color etc will also need to be changed. By binding the values of various properties of the color theme object with other objects of the interface, a programmer can easily create a perfect themed environment.

- 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**.

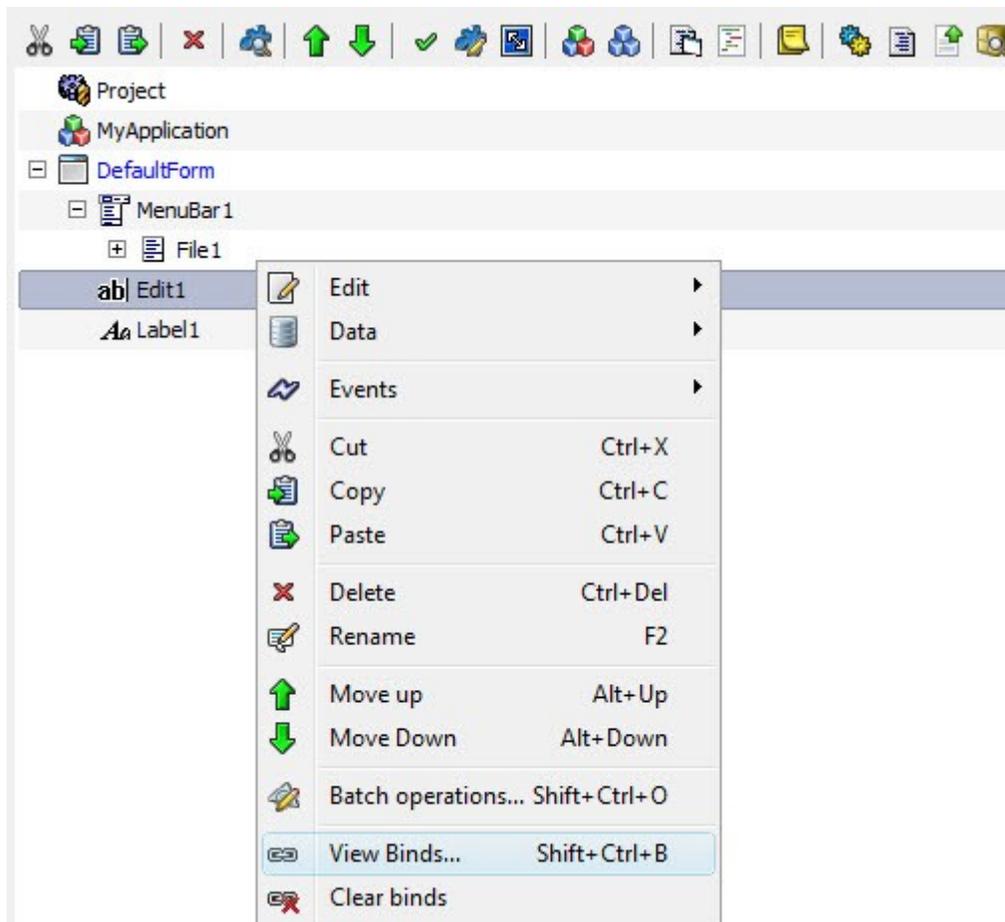


Figure 172: View Binds option

- 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.

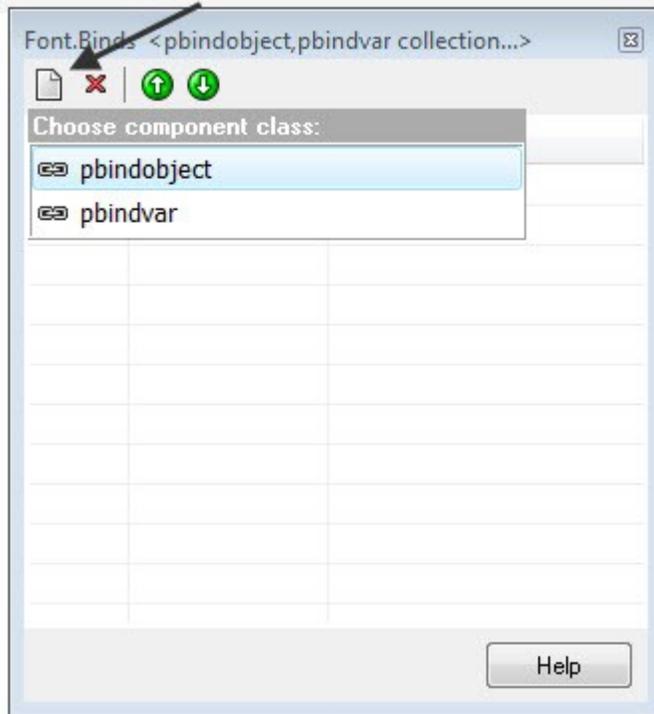


Figure 173: choosing components class

- 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.

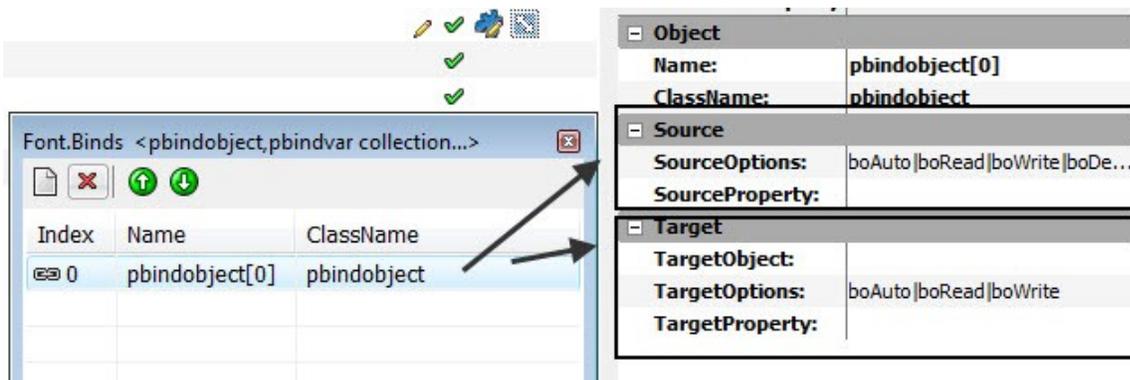


Figure 174: Source and Target sections

- 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.

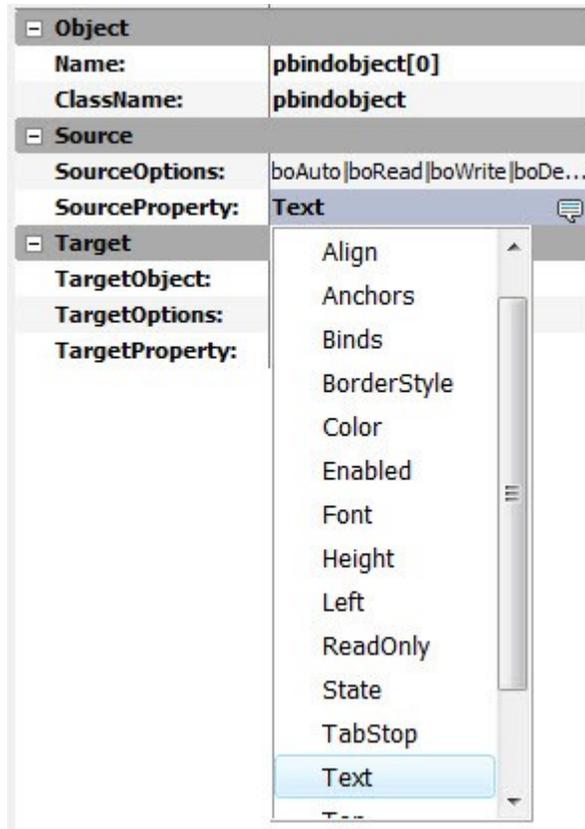


Figure 175: SourceProperty

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

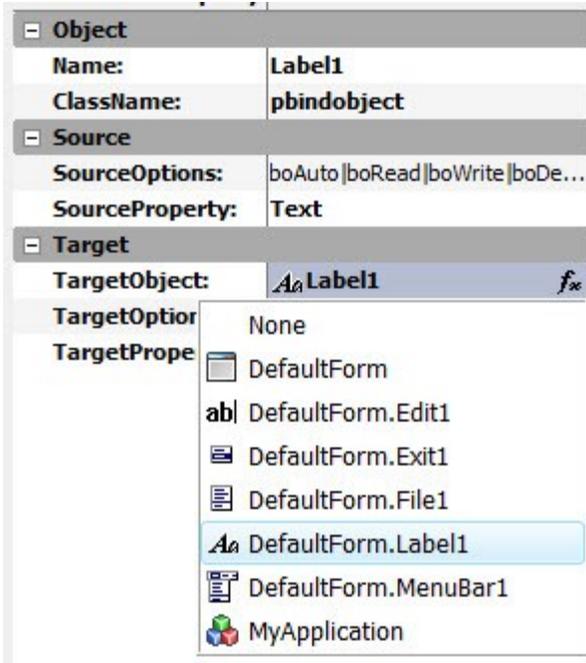


Figure 176: Selectiong a TargetObject

- 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**.

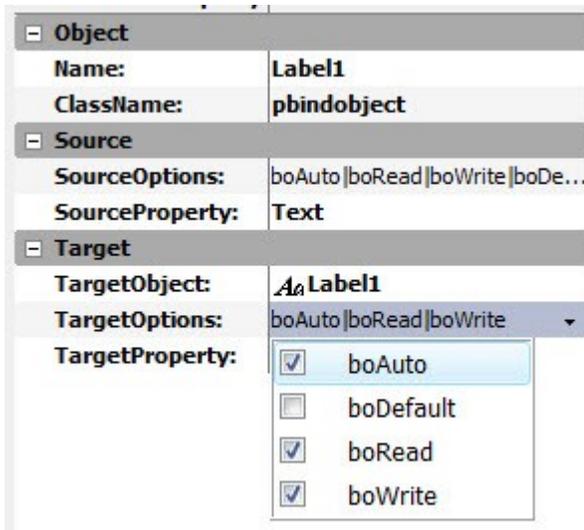


Figure 177: Target options

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

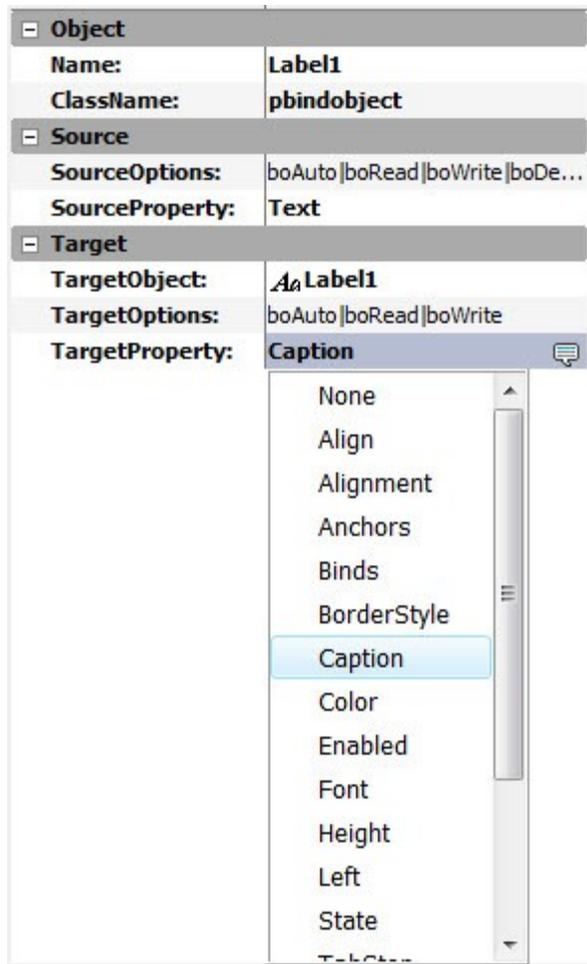


Figure 178: Target property

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

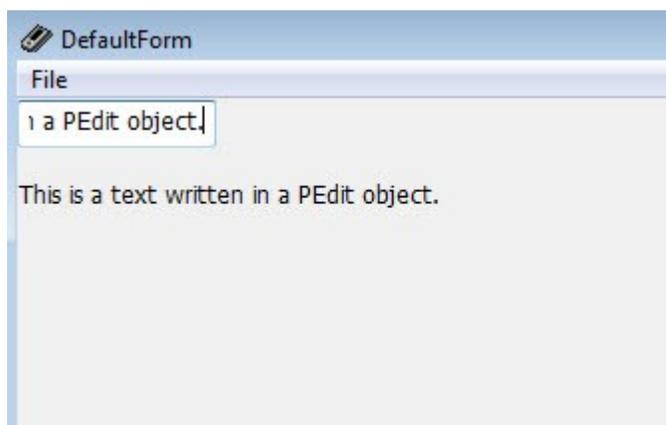


Figure 179: PEdit text binded with PLabel object

Packaging for Deployment

The last step in building an application within PIDE is to deploy it. You can do this through the integrated Compiler.

To compile your application,

- Open the project you wish to compile,
- Click the **Create Executable Button** in the main menu bar. This will open the **Create Executable File Window**.

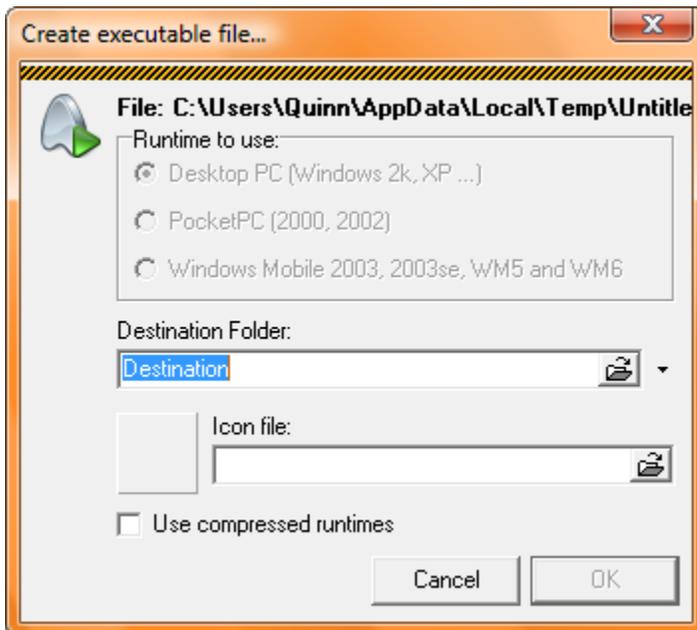


Figure 180: Creating an .exe file

Through the Create Executable File Window, you can set the following parameters for the compilation of your application:

- **Destination Folder**—click the selection button to specify the location on your hard drive where you want to save the compiled application
- **Icon File**—click the selection button to find the icon file, from your hard drive, that you wish to use for the compiled application
- **Use compressed runtimes**—every PPL application utilizes a set of run-times that enable the compiled application to run. You can check this box to utilize the compressed version of those runtimes. *Note: this will require less space for your compiled application on the target device but also reduce overall performance.*

Additional Support

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

Appendix A: Available Application Objects

The following application objects are available in the base version:

- PObject
- PAsm
- PMemory
- PScript
- PCustomList
- PList
- PArray
- PObjectList
- PValue
- PValueList
- PApplication
- PComponent
- PConsole
- PThread
- PDialog
- PMatrix
- PArray
- PBitmap
- PGrid
- PImageList
- PListView
- PTreeView
- PPaintCanvas

- PPanel
- PQuickButton
- PShape
- PToolBar
- PToolButton
- PControl
- PFont
- PCanvas
- PForm
- PMenuBar
- PMenu
- PMenuItem
- PGroupBox
- PTimer
- PLabel
- PButton
- PCheckBox
- PRadioButton
- PListBox
- PComboBox
- PEdit
- PMemo
- PProgressBar

Appendix B: Figures

Figure 1: Main Interface.....	11
Figure 2: Menu Bar.....	12
Figure 3: Action Buttons.....	12
Figure 4: Components.....	13
Figure 5: Work area.....	14
Figure 6: Project manager.....	15
Figure 7: Code View.....	17
Figure 8: Properties/ Events tab.....	19
Figure 9: Matrix Property Editor.....	21
Figure 10: Color Property Editor.....	22
Figure 11: Surface property editor.....	22
Figure 12: Font property editor.....	24
Figure 13: Create new desktop game.....	34
Figure 14: Drag PSurface to project manager.....	35
Figure 15: Open an image.....	35
Figure 16: Double click GameMap.....	36
Figure 17: Drag PSprite to GameMap.....	36
Figure 18: Change Surface property of PSprite.....	37
Figure 19: Create a move event on GameForm.....	38
Figure 20: Drag PSprite to mouse move event.....	39
Figure 21: Set X and Y parameters of Move method.....	40
Figure 22: Code navigator.....	41
Figure 23: Debugging Console.....	42

Figure 24: Debug Local variables.....	42
Figure 25: Debug Global Variables.....	42
Figure 26: Debug Watch List.....	43
Figure 27: Debug Call Stack.....	43
Figure 28: Debug Stack.....	43
Figure 29: New Project Window.....	46
Figure 30: Opening a file.....	48
Figure 31: Saving a file.....	49
Figure 32: Closing a file.....	50
Figure 33: Code view.....	51
Figure 34: Commenting code.....	52
Figure 35: Before Formatting.....	52
Figure 36: After formatting.....	53
Figure 37: Adding objects.....	54
Figure 38: Components tab open.....	55
Figure 39: Components tab collapsed	55
Figure 40: Adding color.....	56
Figure 41: Color Selector Window.....	57
Figure 42: Inserted Hexidecimal Color.....	58
Figure 43: Insert Special Characters.....	59
Figure 44: Adding notes.....	60
Figure 45: Textual note.....	61
Figure 46: Procedure list.....	62
Figure 47: Code navigator.....	63
Figure 48: Properties panel.....	67
Figure 49: Selecting Properties.....	68

Figure 50: Events panel.....	69
Figure 51: selecting events.....	70
Figure 52: Visual Editor.....	72
Figure 53: Elements Placed in the Visual Editor.....	73
Figure 54: Visual Editor Placement Tools.....	74
Figure 55: Selected Control in Visual Editor with Properties Open.....	75
Figure 56: Non-Visual Elements in Visual Editor.....	76
Figure 57: Creating New Project.....	77
Figure 58: Events Panel.....	78
Figure 59: Creating an event.....	78
Figure 60: Creating Default Event.....	79
Figure 61: Selecting Events.....	79
Figure 62: Drag to the event.....	80
Figure 63: Choose a method.....	81
Figure 64: Using exper property.....	82
Figure 65: Viewing the source code.....	84
Figure 66: break point window.....	85
Figure 67: Profile Report.....	87
Figure 68: Debug tab.....	88
Figure 69: Debug tab.....	88
Figure 70: Errors tab.....	88
Figure 71: Warnings tab.....	89
Figure 72: Compiler tab.....	89
Figure 73: Components tab.....	90
Figure 74: Components tab hover feature.....	92
Figure 75: Hint Window for Component Properties.....	93

Figure 76: Hint Window with Navigation.....	94
Figure 77: installing a new component.....	96
Figure 78: Uninstalling components.....	97
Figure 79: creating a xml from a .ppl file.....	98
Figure 80: creating a new component package.....	99
Figure 81: Drag a PToolBar.....	100
Figure 82: Drag PProperty to DbNavigator.....	101
Figure 83: Set the Type property.....	101
Figure 84: Set the ClassName property.....	102
Figure 85: Drag four PToolButton Objects.....	103
Figure 86: Double click the buttons to create events.....	103
Figure 87: Select a message.....	104
Figure 88: Select messages for all buttons.....	105
Figure 89: The property panel.....	106
Figure 90: Components Panel.....	106
Figure 91: Creating new project.....	107
Figure 92: Drag a database.....	108
Figure 93: Drag a PTable.....	108
Figure 94: Specify a filename.....	109
Figure 95: Give TableName.....	109
Figure 96: Add PField object.....	110
Figure 97: Specify Property.....	110
Figure 98: Creating Database.....	111
Figure 99: Viewing Table.....	111
Figure 100: insert PDBGrid.....	112
Figure 101: Drag table to form.....	112

Figure 102: Smart move.....	112
Figure 103: Drag PQuery.....	113
Figure 104: Change Database property.....	113
Figure 105: Visual query editor.....	114
Figure 106: New project with database.....	115
Figure 107: Drag PQuery.....	116
Figure 108: Select appropriate database name.....	116
Figure 109: Double click PQuery.....	117
Figure 110: The four buttons of visual query editor.....	117
Figure 111: Save file button.....	118
Figure 112: The SQL Tab.....	118
Figure 113: Results Tab.....	119
Figure 114: Selecting Query Type.....	119
Figure 115: Creating advanced query.....	120
Figure 116: Visual query editor.....	121
Figure 117: Creating select query.....	122
Figure 118: Run SELECT query.....	123
Figure 119: Add Conditions.....	124
Figure 120: Creating first part of a condition.....	124
Figure 121: Choose a comparison operator.....	125
Figure 122: Run the query.....	125
Figure 123: Selection Tab.....	126
Figure 124: Grouping Criteria Tab.....	126
Figure 125: Sorting Tab.....	126
Figure 126: The SQL Tab.....	127
Figure 127: Selecting Insert Query.....	128

Figure 128: Run Query.....	129
Figure 129: Select Query.....	130
Figure 130: Executing The Select query.....	130
Figure 131: Choose Update Query.....	131
Figure 132: Add Condition.....	132
Figure 133: Complete condition.....	132
Figure 134: Input data.....	133
Figure 135: Choose Delete query.....	134
Figure 136: Specify criteria.....	135
Figure 137: Run query.....	135
Figure 138: Help file editor.....	136
Figure 139: Help file editor interface.....	139
Figure 140: Completing a topic.....	140
Figure 141: Create Help file project.....	141
Figure 142: Select Create topics from file.....	142
Figure 143: Choose a file.....	142
Figure 144: Fill in the requirements.....	143
Figure 145: Creating new project.....	144
Figure 146: Code Template.....	145
Figure 147: Using Code template.....	146
Figure 148: Code Template.....	147
Figure 149: Save project as.....	149
Figure 150: Reloading Definition Files.....	150
Figure 151: Creating new project.....	151
Figure 152: PIDE shell.....	152
Figure 153: Working in PIDE shell.....	152

Figure 154: DIR command in PIDE shell.....	153
Figure 155: ls command in PIDE shell.....	154
Figure 156: Opening files in PIDE Shell.....	155
Figure 157: File open in different window.....	155
Figure 158: File oprn in PIDE shell.....	156
Figure 159: Access batch operations.....	157
Figure 160: Batch Operations window in PIDE.....	158
Figure 161: Setting batch operations.....	159
Figure 162: Draging files.....	160
Figure 163: Choosing file type.....	161
Figure 164: New project.....	162
Figure 165: Drage PPackage Object.....	163
Figure 166: Setting PackageFilename property.....	163
Figure 167: Specifying encryption key.....	164
Figure 168: Draging files to package.....	164
Figure 169: Select PPackage type object.....	165
Figure 170: Package object with files that were dragged.....	165
Figure 171: Compiling the package.....	166
Figure 172: View Binds option.....	170
Figure 173: choosing components class.....	171
Figure 174: Source and Target sections.....	171
Figure 175: SourceProperty.....	172
Figure 176: Selectiong a TargetObject.....	173
Figure 177: Target options.....	173
Figure 178: Target property.....	174
Figure 179: PEdit text binded with PLabel object.....	174

Figure 180: Creating an .exe file.....175