Pocket Programming Language Newsletter
October 2006

Welcome to our very first PPL newsletter. We will be releasing a new one every month with lots of news, new releases information, in-depth articles, coding tips and tricks to make your life easier. The PPL newsletter is open to contributions, if you have an article, tips, advertising or anything else related to PPL, we will be very happy to include your submissions.

Table of content

Editorial

PPL has been released with great success on September 14th 2006. We are very proud and this is only the beginning, PPL will evolve on a regular basis. After more than 2 ½ years of active development, PPL as matured into a robust product over the years. It has been tested by hundreds of people all over the world. It is an immense project with only a small team, PPL has Object-Oriented syntax, a cross-platform assembler (write your assembler code and run it on all supported devices), ActiveX support, linked-list variables, compression and encryption functions, matrices support in the language syntax, garbage collection (no need to declare variables and variables have no types), a visual debugger (with variable values watching), a full-featured IDE on the desktop and on the PocketPC, a high-level Gaming API with 2d physics and particles engine, a visual Game Editor, a profiling tool (to optimize your code), a memory analyzer tool (to detect memory leaks), an optimized interpreter and so much more.

Our first release 1.0 installer had some problems. We've quickly turned around and released 1.01 the next day. Version 1.02 came out about one week later with many more fixes and added features, it was finally what we intended on releasing with 1.0 but time was not on our side at that time. We've even started to offer a free 15-days trial of the Pro version to let the users experiment the full power of it. 1.03 came out just a couple of days after 1.02 was released fixing a few issues with the install process. We are working on 1.04 which should be coming out soon after you receive this newsletter.

We are giving our 100% effort to serve the PPL community with frequent updates, we listen to our customers and users to improve the product, we try to create new tutorials every week or so and one thing is for sure, PPL is an ever evolving tool that will just get better with time.

What's new

New contest $300 in prizes.

We will be holding a new coding contest starting this month (October) with $300 USD in prizes. We will be looking for the best graphical demo running on a PocketPC. The first prize will be $150, the second $100 and the third prize will be $50. Don't miss this chance to win big money and impress the community with your talent! The contest deadlines are set for November 30th. Watch our forums and news section on www.arianesoft.ca for more information.

New version of PPL v1.1 to come.

We are already planning the release of version 1.1. We have a nice list of new features and enhancements in store. We will keep you informed this month on what is on this list and when it will come out.

Here is brief unofficial list of what to expect:

- .OGG music file support is on the list.
- PIDE, form objects list in code editor for easy access to object's event codes.
- Fix for RS232 with a better demo.
- PIDE, find declaration, support for .frm files
- PIDE, find in files, support for .frm files.
- Particles sprite object in Game Level Editor.
- Improved form object's properties editor.
- Adoce library.
- PocketOutlook library.
- Game Level Editor user interface sprites. (Like, button, editbox, listbox etc...)

The list will definitely grow bigger in the next few weeks.

Expect version 1.1 this November and 1.2 by the end of the year. Check our forums for updates later this month on www.arianesoft.ca/forum.php

Coding Tips section:

How to create a memory allocation and return the pointer from a function?

```
func mytest (string$)
  new(n$, sizeof(string$)+10);
  static(n$);
  n$  = string$ % " new";
  return(&n$);
end;

proc main
  ptr$ = mytest("Old ");
  ShowMessage(@ptr$);
  free(ptr$);
end;
```

In function MyTest we create a new variable based on the size of the variable string$ adding 10 to the new length. We use the **static(n$)** to make sure the pointer of ptr$ will not be destroyed when the function MyTest exit. We then concatenate string$ to the string " New" and put the result into n$. We then have to return the pointer of n$ by doing **return(&n$)**;.

When we call MyTest from the Main procedure, the return value is a pointer value assigned to variable ptr$. We then need to output the memory content of that pointer. We use **@**ptr$. The **@** will convert the pointer to the memory content. We display it as a string. Then we need to free the memory allocated in MyTest of ptr$ because PPL cannot do it anymore since ptr$ contains a numerical value (a pointer address value).

How to use lists as parameters and return list values?

```
proc test(l$)
  add(l$, 4, 5, 6);
end;

proc main
  list(l$);
  add(l$, 1, 2, 3);
  test(l$);
  ForEach (l$)
    ShowMessage(l$);
  end;
  test(&l$);
  ForEach (l$)
    ShowMessage(l$);
  end;
end;
```

Lists are handled as normal variables by the PPL interpreter. It means that the current list position index value is used by the interpreter at all time. If you assign a value to a list, the current list position value is set, the same for returning a list variable from a func. Only the current list position value will be returned. The best way to return list variables to a proc or func is to pass the parameter variable as a pointer with the **&** operator. In the example above, we first pass l$ as a non-pointer value, therefore the values 4, 5 and 6 are not added to the list. The second time we call the test procedure, we pass the list variable as pointer, therefore the 4, 5 and 6 values are properly added to the list on return.

An SQL Primer by Eric Pankoke.

SQLite is a nice, lightweight database available for both the PC and PocketPC, and can be accessed through PPL using the Sql.ppl header. If you are interested in adding database support to your applications, I strongly suggest giving SQLite a try, since the database format is identical between the Windows and Windows CE platforms. Here's a quick look at how to use SQLite.

Of course, you'll first want to open the database. You must keep in mind that SQLite's open function will create a database if one with the specified name does not exist, so if your application should not run unless using a database supplied with the install, make sure you check for the existence of the file before calling the open function.

```
if not (FileExists(AppPath$ + "mydata.db"))
  ShowMessage("Data does not exist.  Please reinstall application");
  //Exit function or possibly application
end;

db$ = SqlOpen(AppPath$ + "mydata.db");
if(db$ == 0)
  ShowMessage("error opening database");
  //Exit function or possibly application
end;
```

Now you have a variable called db$ that holds a reference to your database. To retrieve data from a table, simply do the following:

```
rs$ = SqlExec(db$, "SELECT * FROM tblName", &data$, &rows$, &cols$);
if(rs$ <> SQLITE_OK)
  ShowMessage("Error retrieving data");
  //Exit function or possibly application
end;
```

If rows$ is greater than 0, data was found in the table. Data$ is a list containing all of the elements retrieved from the **SqlExec** command. The first "row" in the list contains the name of each column. To navigate through the data, you could do something like the following:

```
for(row_cnt$, 1, row$)
  for(col_cnt$, 1, cols$)
    goto(data$, col_cnt$ - 1);
    data_name$ = data$;
    goto(data$, (row_cnt$ * cols$) - 1 + (col_cnt$ - 1));
    data_value$ = data$;
    ShowMessage(data_name$ + " = " + data_value$);
  end;
end;
```

Finally, you need to make sure you close the database. To do this, simply call the following:

```
SqlClose(db$);
```

A demo called **SqlDemo.ppl** comes with PPL, demonstrating some more SQLite commands, such as creating tables and inserting data. For a comprehensive list of the commands supported by SQLite, check out this web site: http://www.sqlite.org. Remember that PPL currently supports version 2.8.x of SQLite, so information specific to version 3.x of SQLite don't apply here.

PPL Game Programming - Part 1 - Game code structure.

PPL comes with a very powerful set of gaming functions called the GameAPI. The GameAPI comes in two different flavors. The first is the standard edition which does not contain the physic engine and the particles engine. You can create very nice games without them as well. In this first article of the series, we will concentrate on how games are handled in PPL.

Designing a game is a long and tedious process. You have to lay down your plan well in advance before even writing a single line of code. When it comes time to write the game you are often presented with low-level functions where you have to create custom routines to handle your particular kind of game. Writing a game engine requires good knowledge and time. With PPL, you can cut this step, saving days, weeks even months of hard work. The GameAPI will offer plenty of power for any type of 2D game programming.

The first step when you write a game with PPL is to start with a solid code structure that you will use as your starting template for all your future projects, unless you use the Game Level Editor that comes with the PIDE, the code generated by the GLE (Game Level Editor) has the same basic code structure as you will have here.

Let's review our code structure:

```
#include "GameAPI.ppl"

func mainproc(hWnd$, Msg$, wParam$, lParam$)
  ok$ = true;

  case (Msg$)

    WM_CLOSE:
      ShutGameAPI(hWnd$);

    WM_KEYDOWN:
      g_KeyEvent(wParam$, True);

    WM_KEYUP:
      g_KeyEvent(wParam$, False);

  end;

  return (ok$);

end;

func GameProc(hWnd$, Msg$, wParam$, lParam$)
  case (Msg$)
    WM_PAINT:
      G_Clear(0);
```

```
      RenderSprites;

    WM_TIMER:
      if (g_key.vkA$)
        PostMessage(hWnd$, WM_CLOSE, 0, 0);
      end;

  end;
  return (true);
end;

func WinMain
  h$ = newform({GAPI}, {GAPIClass}, &mainproc);
  ShowWindow(h$, SW_SHOW);

  InitGameAPIEx(h$, &GameProc, 240, 320, false, 5, 60);
  ShowFPS(true, G_RGB(255, 255, 255));

  return (true);
end;
```

At the first line of code we include the GameAPI library into our project. This is where most of the GameAPI constants are defined. Some really handy functions are defined in as well. The mainproc function is where all GameAPI events are handled. The first event we will handle by default is the **WM_CLOSE**, which is triggered when the GameAPI main form is closed. Here we need to shutdown the GameAPI by calling **ShutGameAPI(hWnd$)**. It is generally here that you will free all global objects. Sprites are freed by the function automatically. If you load surfaces manually, it is a good place to free them. Next we handle the **WM_KEYDOWN** and **WM_KEYUP** events. When a key is pressed (hardware keys on the PocketPC device), a keydown is triggered, then when the key is released, the keyup event is triggered. This code is pretty standard, a special function is called to set the g_key$ structure values, then you can easily check to see which key is being pressed and it supports multiple key presses too.

Next, every game as to have a main code function. At every internal cycle this function will be called. This is where you will handle game specific events like painting and timer. The painting can be handled manually using the **WM_PAINT** event or if you set the **G_AutoDraw(True)** right after the **InitGameAPIEx()** line, PPL will handle the drawing of sprites for you. If you use manual painting, the whole screen as to be repainted, that is why we clear the screen with **G_Clear(0)**. Zero is the color of the background, black in this case. The **RenderSprites()** function will paint all sprites on screen with the correct layer order and everything. The **WM_TIMER** is called every game code cycle. We will see later how to change the cycle rate with the **SetAISpeed()** function. Here is a good place to check for key pressed. In our case if the hardware A key is pressed, we send a close message to the main game form.

The WinMain function is where PPL will start executing instructions for this program. Here we need to create a new form, display it using the **ShowWindow()** function. Next we need to initialize the GameAPI and the sound engine. **InitGameAPIEx()** will do all this work for us. We need to tell it what

form will be used for the game display (h$), which main game code function to use (&GameProc), the resolution of the game (240x320 QVGA), the fullscreen parameter as to be false. Next is the cycle rate at which to call the **WM_TIMER** event in the main game function (&GameProc). Every 5 milliseconds PPL will try to trigger the **WM_TIMER** event. The last parameter is the maximum frames per second to display. 60 is a good generic value, it's smooth, gives time to the main game code to be executed and won't slow down the game by trying to draw unnecessary frames. Next we want to display the FPS (frames per second) information on screen. You might want to turn this off when your game is finished and you are ready to distribute it.

Finally we return a value of true to tell PPL to keep the application alive that it hasn't been closed.

This is a very basic game code structure. We will get into more details with sprites and their internal functions, pixel-perfect collision detection, sprite's mass, friction and velocity, particles and so much more as the series evolve. See you next month and happy game creation.

Shortcut keys in the PIDE

By now you've opened up PIDE and used it for a while. Hopefully everyone has noticed the keyboard shortcuts placed in the menus. Here are a few that are very helpful, but not so obvious.

In PIDE's Game Level Editor try <shift>+<Mouse Click> while a sprite is selected, this will Create a new sprite copying the original sprites properties.

In PIDE's Visual Form Builder:

- The <ESC> key will select the Form Control.
- If you need a little help, F1 will bring up VFB Help and <CTRL>+F1 will bring up the MSDN help for the selected control.
- Hold down the <CTRl> key and move the mouse over the form. This will show dashed lines from the mouse position to the ruler to help align controls on the form.
- Arrow keys can move a control by 1 pixel at a time to get things lined up perfectly.
- When editing controls that can use a list (GroupBox, ListBox, TabControl, etc...) select "Caption" in the properties and then press F5. You can enter values to preload into these control.
- If you need to select a color for the properties, use F4.
- If you need to select a filename for the properties, use F3
- If you need to select a true/false value for properties, <CTRL>+T will enter true and <CTRL>+F will enter false.

For those of you that are not keyboard jockies, right click on the properties to bring up the context menu with the options for the properties.

Here is a reference card for those who could use it

PIDE Editor ShortCut Keys:

```
<CTRL>-N           New
<CTRL>-O           Open
<CTRL>-S           Save file
<CTRL>-P           Print
<CTRL>-Z           Undo
<SHIFT><CTRL>-Z    Redo
<CTRL>-X           Cut
<CTRL>-C           Copy
<CTRL>-V           Paste
<CTRL><DEL>        Delete
<SHIFT><CTRL>-C    Comment Code
<CTRL>-R           RGB color
<CTRL>-D           Format Code
<CTRL>-G           Goto Line Number
<CTRL>-F           Find
<SHIFT><CTRL>-F    Find in Files
<CTRL>-F3          Find Again
<CTRL>-H           Replace
```

```
<CTRL>-F11          Find Definition
<SHIFT>-F11         Open Selected File
<SHIFT>-F8          Line Profile Result
<CTRL>-F7           Run
<CTRL>-F9           Dedicated Run
F7                  Compile
F5                  Debug
F10                 Step Over
F11                 Step Into
<CTRL>-F10          Run to Cursor
<SHIFT>-F5          Stop
F9                  Toggle Breakpoints
<CTRL>-B            Breakpoint Window
<CTRL>-F7           Watches Window
<CTRL>-F12          File Manager
F12                 Visual Form Builder
<CTRL>-G            Procedures List
F4                  Goto Map...
<SHIFT><CTRL><DEL> Clear a controls code.
F1                  Help
<CTRL>-F1           MSDN Help
```

PIDE VFB ShortCut Keys:

```
F1                  VFB Help
<CTRL>+F1           MSDN help on selected control
F3                  Select FileName for properties
F4                  Select Color for properties
F5                  Edit List of values for control
<ESC>               Focus on the Form Control
<CTRL>              Show Alignment lines
<CTRL>+T            Enter True for properties
<CTRL>+F            Enter False for properties
<Arrow Keys>        Move selected control by one pixel.
```

PIDE Game Level Editor ShortCut Keys:

```
<Shift>+<Click>    create a copy of the selected sprite.
```

Time to say goodbye

Thank you for your interest in PPL. We hope to see you in the forums and let's make it a source of good information, a library for PPL enthusiasts. If you any comments or questions, please visit our forums or contact us via support@arianesoft.ca

If you want to contribute to this newsletter with articles, tips, suggestions, please contact us, we will be happy to include them.

Regards,
Alain Deschenes
President
ArianeSoft Inc.
www.arianesoft.ca