

PocketC Help Central

Written by **Jeremy Dewey and Kevin Cao**

Converted to PDF format by Christopher Satola

Version 2.0 for WindowsCE

Concept

PocketC 's design is a mixture of a few modern language concepts. The language is C. Due to its invisible data type conversion feature, PocketC is also like a scripting language. String variables can be converted to a float or integers and back without any special function calls. It is similar to VB Script and Java Script.

Due to Windows CE operating system can be used with various CPUs, regular software that are compiled directly to the CPU instruction can only be used with that CPU devices. With our compiler, the source code is compiled into CPU-independent instructions. This allow user's application to be CPU-independent. We provide a free runtime module for PocketC developers. This runtime module is required for PocketC applications to run on various platforms without recompile. Its concept is similar to Java applets which are CPU/OS-independent.

PocketC is not another ANSI standard C compiler. If you are looking for an ANSI standard C compiler, we suggest you to take a look at gcc port for Windows CE.

We provide our custom function API which are designed to make programming easier and fun. If you have any good suggestions on new features, please post on the webboard or contact us through email.

Thank you.

Kevin Cao

Step 1. Write Program

Writing PocketC program is simple if you already know C. If you are new to programming, I suggest you pick up a beginner guide to C or read our language chapters. Our webboard is a good place to share your ideas or ask questions at. We have a descent size user community which mostly are software engineers, IT professionals, professors and students. Hopefully, you will join us in the future.

Once you install PocketC, you should be able to launch the program by Click on StartMenu-> Programs-> PocketC Development.

Now you are ready to write some code in the editor. Are you? :-)

Example:

```
main()
{
    string hello;
    hello = "hello!";
    puts1(hello); puts1(hello);
    alert("You say hi!"); alert("I say bye!");
    text(50,50,"hello world");
    graph_off();
}
```

In this example, we just demonstrated three methods of displaying text. Please click on the function names to find out what they do.

Step 2. Compile and Build

Now we have a program in the editor, the next step is to compile and build an application. Just click Build Menu -> Build.

If build process was successful, a message box will popup indicating the build was successful. Otherwise, an error message box will popup. After close the error message box, the editor will automatically jump to the error line.

Step 3. Run Program

After we successfully build a program, running the program is trivial. Just click Build Menu->Run. Your program will be launched. Just sit back, and observe your first PocketC program in action. You may also change the source code, and play around with with different functions in our function library.

Step 4. Distribute Program

PocketC applications requires the runtime module. We have an installer on the web for download. You can distribute the runtime module with your program freely. You may not charge money for our runtime module. Thanks.

End-users can launch PocketC runtime, and they will be able to see your program and launch it.

Features

Function Browser

When you have a relative large program, it will become more difficult to move around in your source code. We added a feature to ease the pain. Just click on File menu->Status. It will show you the file format, size and the list of function it contains. Just click on one of the function name in the list. The editor will take you to that particular function.

Auto Prototype

To help out our road coders, we added a feature called auto-function prototype. After you type a function name, and hit (character, the function prototype is displayed in the status window. Hit) character will remove the function prototype.

Multi-File Build

Some existing PocketC applications has over 2000 lines of code, and spread through 20 and more files. You can use Multi-File feature to help out. Just click Build Menu->Multifile Mode. It will ask you for a full path to the source file which contains the main() function. Now, you can edit any files and the compile will always compile the main file first.

Font Adjustment

You can also adjust the editor's font type and size as well in Help menu. Just click on Help menu->Preference. A dialog will popup with settings for font type, size, tab length. We even added a custom adjustable help system. In theory, you link another help file with the Documentation menu. In reality, WinCE help file must be in Windows directory. That is the limitation of WinCE help file. We cannot do much about it.

As you are writing code, the current line number is indicated on the status bar as well.

We update our software frequently to remove bugs and add new features. You can check the PocketC build time stamp by click on Help menu->About PocketC. The very first line will indicate the build time of your PocketC. When you are posting bugs, please remember to post your PocketC's build time as well. Sometime, the bug maybe already resolved. You just need to download a new version.

Some program examples

Generate Random Numbers

```
/* Demonstrate function rand() and random() */  
  
main()  
{  
float r;  
int i; int r2;  
for (i=0; i<100; i++)  
{  
r = rand();  
puts1(r);  
}  
puts1("Random (Range)");
```

```

for (i=0; i<10; i++)
{
r2 = random(10);
puts1(r2);
}
showconsole();
}

```

File enumeration

```

/* Demonstrate function fileenum() */
/* The program will display all contents of root directory */

```

```

#define FILE_ATTRIBUTE_DIRECTORY 0x00000010
main()
{
string filename;
string root;
showconsole();
root = "\\.";
filename = fileenum(1, "\\*.");
while (strlen(filename))
{
puts(filename);
if ((FILE_ATTRIBUTE_DIRECTORY & GetFileAttr(root+filename))
    == FILE_ATTRIBUTE_DIRECTORY)
puts1(" is a directory");
else
puts1(" is a file");
filename = fileenum(0, "");
}
}

```

File access

```

/* Demonstrate function fileread() and filewrite()*/
/* The program write the file testread.txt and read the contents */
/* Change include path if you installed PocketC in a different directory */

```

```

#include "pocketc\pcheader.h"
main()
{
string filename; int fh; int l; string textdata;
string textstring;
filename = "\\testread.txt";
textstring="1234567890123456789012345678901234567890\r\n"
    +"234567890123456789012345678901234567890";
fh=fileopen(filename,0,FILE_CREATE | FILE_WRITE);
if (fh!=-1)
{
filewrite(fh,textstring,strlen(textstring));
fileclose(fh);
}
fh=fileopen(filename,0, FILE_READ);
if (fh!=-1)
{
l=filegetlen(fh);
textdata=fileread(fh,260);
MessageBox(textdata,"test",1,1);
}
}

```

```

puts1(textdata);
fileclose(fh);
}
}

```

Window control focus

/ Demonstrate UI functions and focus */*

```

#include "/PocketC/pcheader.h"
main()
{
int iEvent, iControl;
createctrl("BUTTON", "Label 1", WS_VISIBLE|BS_PUSHBUTTON | WS_CHILD, 0,
10, 100, 100, 20, 2001);
createctrl("BUTTON", "Label 2", WS_VISIBLE|BS_PUSHBUTTON | WS_CHILD, 0,
10,130, 100, 20, 2000);
createctrl("EDIT","", WS_VISIBLE | WS_BORDER | ES_LOWERCASE,0,10,150, 100,
20,100);
wndshow(1000, 5);
wndshow(2000, 5);
guisetfocus(1000);
guisetfocus(2000);
do {
iControl = guigetfocus();
text(10, 200, "Event = " + (string)iEvent + " Control = " +
(string)iControl + " ");
iEvent = event(1);
} while (iEvent != PM_CHAR);
delgui(1000);
delgui(2000);
delgui(100);
clearg();
quit();
}

```

Database enumeration

/ Demonstrate database functions */*

/ This program will list all databases on the windows ce device */*

/ The output could be fairly large*/*

```

#define CEVT_I2 2
#define CEVT_UI2 18
#define CEVT_I4 3
#define CEVT_UI4 19
#define CEVT_LPWSTR 31
listdb();
checkdb(int oid);
main()
{
cursorwait(1);
showconsole();
listdb();
cursorwait(0);
}
listdb()
{
int ret; int oid;
ret = dbenum(1,0);

```

```

if (ret==0) return;
puts1("List all the databases");
puts1("Name-----Size-----NumRecords---");
while ((oid=dbenum(0,0))!=0)
{
puts1(dbname(oid) + "\t\t" + dbsize(oid) + "\t" + dbnrecs(oid));
checkdb(oid);
}
}
checkdb(int oid)
{
int ret; int cnt; int i; int reccnt; int j;
puts1("-----");
ret = dbopen(oid,"");
if (ret==0)
{
puts1(dbname(oid) + " open failed");
return;
}
reccnt = dbnrecs(oid);
for (j = 0; j < reccnt; j++)
{
cnt = dbrecpropcnt();
for (i=0; i < cnt; i++)
{
puts1("(" + dbrecproptype(i) + " " + dbrecpropval(i) + ")");
}
dbseek(8,1);
puts1("");
}
puts1("-----");
ret = dbclose();
}

```

Units converter

/* A simple unit conversion calculator */
/* written by a friend of Conan */
/* updated by Kevin Cao */

```

#define PM_CHAR 1
#define PM_BUTTONUP 5
#define PM_COMMAND 8
#define CLR_BCK_R 210
#define CLR_BCK_G 255
#define CLR_BCK_B 220
#define CLR_FGD_R 10
#define CLR_FGD_G 30
#define CLR_FGD_B 20
#define BETWEENBUTTONSX 33
#define BETWEENBUTTONSY 10
#define DISPLAYHEIGHT 40
#define FONTHEIGHT 16
#define FONTWIDTH 8
int ButtWidth,ButtHeight;
float Number;
float NumberPosition;
drawGUI()
{

```

```

int x,y,a,px,py;
setbkcolor(CLR_BCK_R,CLR_BCK_G,CLR_BCK_B);
setbrushattr(CLR_BCK_R,CLR_BCK_G,CLR_BCK_B);
setpixelattr(CLR_FGD_R,CLR_FGD_G,CLR_FGD_B);
setpenattr(0,3,CLR_FGD_R,CLR_FGD_G,CLR_FGD_B);
setfontattr("Arial",0,0,0,200,FontWidth,FontHeight);
clearg();
roundrect(BETWEENBUTTONSX,BETWEENBUTTONSY,screenx()-BETWEENBUTTONSX,BETWEEN
BUTTONSY+DISPLAYHEIGHT,5,5);
a=0;
for(y=0;y<4;y++)
{
for(x=0;x<5;x++)
{
px=(int)((x+1)*BETWEENBUTTONSX+(x*ButtWidth));
py=(int)((y+3)*BETWEENBUTTONSY+(y*ButtHeight)+DISPLAYHEIGHT);
roundrect(px,py,px+ButtWidth,py+ButtHeight,4,4);
if(x<3)
px=px+(int)((ButtWidth-FontWidth)*0.5);
py=py+(int)((ButtHeight-FontHeight)*0.5);
switch(a)
{
case 0: text(px,py,"1"); break;
case 1: text(px,py,"2"); break;
case 2: text(px,py,"3"); break;
case 3: text(px+25,py,"C->F"); break;
case 4: text(px+25,py,"F->C"); break;
case 5: text(px,py,"4"); break;
case 6: text(px,py,"5"); break;
case 7: text(px,py,"6"); break;
case 8: text(px+14,py,"Kg->Lb"); break;
case 9: text(px+14,py,"Lb->Kg"); break;
case 10: text(px,py,"7"); break;
case 11: text(px,py,"8"); break;
case 12: text(px,py,"9"); break;
case 13: text(px+6,py,"Cm->Inch"); break;
case 14: text(px+6,py,"Inch->Cm"); break;
case 15: text(px,py,"0"); break;
case 16: text(px,py,"."); break;
case 17: text(px-6,py,"Clr"); break;
case 18: text(px+2,py,"Kph->Mph"); break;
case 19: text(px+2,py,"Mph->Kph"); break;
}
a++;
}
}
}
DisplayNumber()
{
roundrect(BETWEENBUTTONSX,BETWEENBUTTONSY,screenx()-BETWEENBUTTONSX,BETWEEN
BUTTONSY+DISPLAYHEIGHT,5,5);
text(BETWEENBUTTONSX+10,(BETWEENBUTTONSY+DISPLAYHEIGHT-FontHeight-6),Number
);
}
ClearNumber()
{
Number=0;
NumberPosition=10;
}

```

```

}
AddToNumber(int Nu)
{
if(Nu==10) NumberPosition=(float)0.1; else
{
if(NumberPosition<1) // going to fraction
{
Number=Number+(Nu*NumberPosition);
NumberPosition=NumberPosition*(float)0.1;
}
else
Number=(Number*10)+Nu;
}
}
DoTask(int task)
{
switch(task) {
case 0:AddToNumber(1); break;
case 1:AddToNumber(2); break;
case 2:AddToNumber(3); break;
case 5:AddToNumber(4); break;
case 6:AddToNumber(5); break;
case 7:AddToNumber(6); break;
case 10:AddToNumber(7); break;
case 11:AddToNumber(8); break;
case 12:AddToNumber(9); break;
case 15:AddToNumber(0); break;
case 16:AddToNumber(10); break;
case 17:ClearNumber(); break;
case 20:{Number=Number*(float)-1;} break;
case 18:{Number=Number/(float)1.60933;} break;
case 19:{Number=Number*(float)1.60933;} break;
case 13:{Number=Number/(float)2.54;} break;
case 14:{Number=Number*(float)2.54;} break;
case 8:{Number=Number/(float)0.4536;} break;
case 9:{Number=Number*(float)0.4536;} break;
case 3:{Number=(Number*(float)1.8)+32;} break;
case 4:{Number=(Number-32)/(float)1.8;} break;
}
DisplayNumber();
}
KeyPressed(char kp)
{
switch (kp) {
case '0':DoTask(15); break;
case '1':DoTask(0); break;
case '2':DoTask(1); break;
case '3':DoTask(2); break;
case '4':DoTask(5); break;
case '5':DoTask(6); break;
case '6':DoTask(7); break;
case '7':DoTask(10); break;
case '8':DoTask(11); break;
case '9':DoTask(12); break;
case '.' :DoTask(16); break;
case 'c':DoTask(17); break;
case '-' :DoTask(20); break;
}
}

```

```

}
ButtonPushed(int x, int y)
{
int px,py,a;
if(x<BETWEENBUTTONSX) return;
if(y<((3*BETWEENBUTTONSY)+DISPLAYHEIGHT)) return;
px=(int)((float)(x-BETWEENBUTTONSX)/(ButtWidth+BETWEENBUTTONSX));
py=(int)((float)(y-(3*BETWEENBUTTONSY)-DISPLAYHEIGHT)/(ButtHeight+BETWEENB
UTTONSY));
if(x>((px+1)*BETWEENBUTTONSX)+((px+1)*ButtWidth)) return;
if(y>(int)((py+3)*BETWEENBUTTONSY)+((py+1)*ButtHeight)+DISPLAYHEIGHT)
return;
a=px+(5*py);
DoTask(a);
}
main()
{
int msg;
ClearNumber();
ButtWidth=(int)((float)(screenx()-(BETWEENBUTTONSX*6))*0.2);
ButtHeight=(int)((float)(screeny()-(BETWEENBUTTONSY*7)-DISPLAYHEIGHT)*0.25)
;
drawGUI();
DisplayNumber();
while(1)
{
msg = event(0);
if (msg == PM_CHAR) KeyPressed((char)key());
if (msg == PM_BUTTONUP) ButtonPushed(penx(),peny());
}
}

```

Language Introduction

PocketC is designed and implemented by Jeremy Dewey. It was a his personal project during the summer of 1997. We met when we were both hired by a tiny software company for summer interns. At that time, He had great interests on PalmPilot devices. After reading the book “C the Complete Reference” by Herbert Schildt, he started to programming PocketC after work. It is not an easy task to achieve since Palm-Pilot has merely 2 Meg RAM total. I also believe that is one of the reason why he did it. Every morning at work, he would show me the latest progress. It was a fun experience to see a tiny language getting developed slowly. At the end of the summer, PocketC is finally finished for Palm-Pilot. With his permission, I ported PocketC for Windows CE during the school.

I hope you enjoying our little language
Kevin Cao 10/09/1999

PocketC Data Type and Variables

Basic Data Type:

```

int
char
string
float

```

Identifier

PocketC identifier has the same rule as C indentifier. The first character must be a letter or an underscore and following characters must be either letters, digits or underscores

Variables

The following variable types are supported: int, float, char, string, pointer, and single-dimensional arrays.

Local variables must be declared before any code in a function.

Variable initializers are not yet supported. (e.g. `int x=5;` is not yet legal).

All variables, global and local, are initialized to zero or the empty string.

Variable names can be up to 31 characters long and are case sensitive.

Type	Name	
integer	int	1, 2, 5, -789, 452349
floating point	float	-1.2, 3.141592, 5.7e-4
characters	char	'a', 'b', '#'
strings	string	"Bob" "Katie" "Hello"

Variables are declared like this: *variable-type name[,name...];*

Here are a few examples:

```
int myInteger, row, column;
string name;
float pi;
char c, last, first;
pointer pi;
```

It is also possible to have an array of values. An array is a list of values that are stored in one variable. Arrays are declared like normal variables except that the variable name is followed by '[size]' where size is the number of item that the variable can hold. A declaration might look like this:

```
int values[10];
string names[7];
Of course, arrays and normal variables can be declared together:
int row, values[10], column;
string name, colors[8];
```

Local Variables

Variables that are declared inside a function are called local variables. Local variables are only referenced by the statements inside of the function. One restriction, You have to declare all local variables at the start of a function.

Global Variables

Opposite to local variables, global variables are know throughout the code and can be referenced by any port of the program. In addition, their values are kept through the entire program execution.

Pointers

A pointer is defined by the pointer type, not `int*`, for example. Importantly, pointers in PocketC are not typed. Instead, they take on the type of the data to which they point. Additionally, a pointer can refer to a function, and would be used as follows:

```
func(int x) { return 5*x; }
main() {
    pointer ptr;
    int result;
    ptr = func;
```

```

    result = (*ptr) (7);
    puts("5*7=" + result);
}

```

Additionally, pointer values are not addresses to actual PalmOS or WindowsCE memory. We will discuss more about pointers later on.

Assignment, Operators and Inc/Decrment

Assignment

Variable assignment is actually just another form of expression. Assignment is done in one of two ways for a normal variable:

name = expression

and for an array: *name[index-expression] = expression*

Here are a few examples:

```

int myInt, numbers[3];
string myString;
...
myInt = 8;
myString = "Animaniacs";
numbers[0] = myInt + 5;
numbers[2] = numbers[0] * 8;

```

However, since PocketC is loosely typed, any type of value can be assigned to any type of variable and the value will be automatically converted:

```

myString = 95; // The value of myString is now "95"
numbers[1] = "78"; // The value of numbers[1] is now 78;
numbers["2"] = "2"; // Another neat trick. numbers[2] is now 2

```

Now, what are all the operators that can be used in an expression, and what is their associativity? Good question.

Operators

The following table is in order of precedence, lowest first.

Operator	Assoc	Description
=	right	assigns the value of the expression on the right to the variable on the left. Evaluates to the expression on the right.
	left	logical 'or', evaluates to 0 if false, 1 if true
&&	left	logical 'and'
== != < <= > >=	left	relational operators. == (equal), != (not equal), <= (less than or equal), >= (greater than or equal). These evaluate to 1 if the expression is true, 0 otherwise
+ -	left	addition, subtraction (subtraction cannot be used with a string argument)
* / %	left	multiplication, division, modulus (cannot be used with strings, nor can modulus be used with floats)
- ! ++ -- ~ * [] () &	left	- (negation), ! (logical 'not'), ++ (increment), -- (decrement), ~ (bitwise neg), [] (array subscript), () (function pointer dereference), & (address of) (Of these, only the logical 'not' can be used with strings)

Notes: No shortcut logic is performed on the operands of || and && The compound assignment operators (+=, *=, etc.) are not supported. The comma and conditional operators (?:) are not supported.

Bitwise Operators

PocketC supports a full complement of bitwise operators. Bitwise operation is used for variable type int or char. You cannot use bitwise operations on float or string data types.

&	AND
	OR
^	Exclusive OR (XOR)
~	One's complement
>	Shift Right
<	Shift Left

Increment / Decrement

The ++ and – operators are special in that they must be placed before or after a variable and modify the value of the variable. The ++ increments the value of a variable by one, while the – decrements by one. The caveat is that if the ++/– is placed in front of the variable, the expression evaluates to the value of the variable after it is incremented/decremented. If it is placed after the variable, the expression evaluates to the variable's previous value.

Example:

```
int myInt;
...
myInt = 8;
puts(++myInt); // Prints "9" to the output form
myInt = 8;
puts(myInt++); // Prints "8" to the output form, but myInt is now 9
```

Automatic Casting

Just like in assignments statements, automatic conversion takes place in every part of an expression. If the two arguments to an operator are of different types, one of arguments will be promoted to the less strict type. The promotion order is char to int to float to string. So in the expression:

```
"Result is: " + 5;
```

The constant 5 is first promoted to a string, and the two strings are concatenated. This may have some undesirable side effects. For example, if you want to write an expression and result to the output form, you might do something like this:

```
puts("5 + 7 = " + 5 + 7); // Prints "5 + 7 = 57"
```

This probably wasn't the desired outcome. Instead, you would want the expression evaluated first, then concatenated to the string. The parentheses can be used to accomplish this:

```
puts("5 + 7 = " + (5 + 7)); // Prints "5 + 7 = 12"
```

One problem remains. Suppose you want to find the floating point value of a fraction of two integer.

```
puts("7 / 5 = " + (7 / 5)); // Prints "7 / 5 = 1"
```

This output is because both arguments are integers, so the result is also an integer. To solve this, we can cast one of them to a float:

```
puts("7 / 5 = " + ((float)7 / 5)); // Prints " 7 / 5 = 1.4"
```

This forces the integer 7 to a floating point number before dividing it by 5.

Declare Functions

Functions are the most important part of a program because they contain the actual instructions that make a program useful. All functions have a name and a parameter list (which may be empty) and are declared like the:

```
func-name([param-type param-name,...]) { statements }
```

Statements are discussed later, but for now, here are a few examples:

```
area(int width, int height) {  
    return width * height;  
}  
square(float x) {  
    return x * x;  
}  
five() {  
    return 5;  
}
```

There is one special function name which all programs must have: main. The main function is the function which is called first in your program. When the main function exits, the program terminates. The main function must be declared with no parameters:

```
// My Applet  
main() {  
    puts("Hello World");  
}
```

Functions can also have local variables, which are variables that can only be accessed within the function that declares them. Global variables, however, can be accessed from anywhere. Local variables are declared in the same way that global variables are except that they immediately follow the opening brace of a function:

```
// My Applet  
main() {  
    string localString;  
    localString = "Hello World";  
    puts(localString);  
}
```

Before we go any further, we need to talk a little bit about expressions.

Expressions

An expression is any number of constants, variables, and function calls connected by operators and parentheses.

A constant is any value that is directly entered into the program, such as: 5 5.4 'a' "String"

A value stored in a variable can be accessed by just typing its name: myInteger name

However, if that variable is an array, each value must be accessed individually by index. The valid indices for a given array are 0 to n-1 where n is the number of values in the array. So an array declared:

```
string names[4]  
can be accessed like so:  
names[0] = "first name";  
names[1] = "second name";  
names[2] = "third name";  
names[3] = "fourth name";
```

A function call consists of the name of a function, followed by an open paren, the parameter list, and a closing paren:

```

area(5, 7);
square(8.9);
clear();
text(30, 55, "Game Over");

```

These three basic elements can be combined with operators:

```

5 + 7 - area(12, 34);
square(5) * pi;
"Hello, " + "World";

```

Of course, function calls can have expressions in them as well:

```

area(6+3, 8*9);
area(8 * square(4), 7);

```

Statements

Statements are the individual parts that make up the body of a function. The following are the available statements:

Statement	Description
return;	Returns immediately from the current function (with a default return value of integer 0)
return expr;	Returns immediately from the current function, returning the value of the expression expr
if (expr) stmt	Evaluates the expression expr, if its result is true (non-zero or non-empty string), the statement stmt is executed, otherwise stmt is skipped, and execution continues
if (expr) stmtA else stmtB	Evaluates the expression expr, if its result is true (non-zero or non-empty string), the statement stmtA is executed, otherwise stmtB is executed
while (expr) stmt	The expression expr is evaluated. If it is true (non-zero or non-empty string), stmt is executed. The loop then begin again, evaluating expr and executing stmt until expr is no longer true. This means that stmt will never execute if expr is initially false
do stmt while (expr)	The same as while except that the statement stmt is executed before expr is evaluated. This guarantees that stmt will execute at least once
for (init;cond;iter) stmt	The initializer expression init is first evaluated. The condition expression cond is evaluated. If it is true, stmt is executed and the iterator expression iter is evaluated continuing the loop, otherwise the the for loop ends. Note: init is evaluated only once.
break;	Immediately exits from the directly enclosing while/do/for loop.
continue;	Immediately restarts the directly enclosing while/do/for loop. In a for loop, the iter expression is evaluated, followed by the cond expression and possibly the stmt
{ statements }	A brace followed by a list of statements, followed by another brace is considered a single statement
expression;	An expression followed by a semicolon is also considered to be a statement
switch (expr) { case const1: { statements } break; default: { statements } }	switch statement is a multiple-branch selection statement. It tests the value of an expression against a list of integer, character, float or string. When a match is found, the case block with that constant are executed. The break statement can be used at the end of a case statement. It will cause program to jump to the code after switch statement. Note: Regular ANSI C will not allow constant string in switch statement for branch selection. Constant string is a variable type for PocketC. So PocketC's switch statement can handle constant string.

Statement Examples:

return

Let's visit a previous example function to see how return works.

```
five() {  
    return 5;  
}
```

Since the return value of the function five is always 5, we can use the function any place we would normal put the constant 5.

```
puts("Five is " + five()); // Prints "Five is 5"
```

Also, since return causes the function to exit immediately, we could do this:

```
five() {  
    return 5;  
    puts("This won't print");  
}
```

and we would have the same effect.

if

```
if lessThan5(int x) {  
    if (x < 5)  
        puts("Less than five");  
    puts("Hello");  
}
```

If this function is called with a number less than 5, "Less than five" will be printed followed by the word "Hello", otherwise, only the word "Hello" is printed.

if ... then

```
if lessThan5(int x) {  
    if (x < 5)  
        puts("Less than five");  
    else  
        puts("Greater than or equal to five");  
}
```

If this function is called with a number less than 5, "Less than five" is printed, otherwise "Greater than or equal to five" is printed.

while

```
count() {  
    int x;  
    x = 5;  
    while (x > 0) {  
        puts(x);  
        x = x - 1;  
    }  
}
```

This bit of code will print the numbers from 5 to 1 counting backwards. Notice that braces were placed around the two lines of code in the while loop to make them act as a single statement.

do ... while

```
count() {
    int x;
    x = 6;
    do {
        x = x - 1; // could also be x-
        puts(x);
    } while (x > 0);
}
```

This bit of code (similar to the previous example) will print the numbers from 5 to 0 counting backwards. The zero is printed in this case because the expression $x < 0$ is not evaluated until after the loop

for

```
output() {
    string list[4];
    int index;
    list[0] = "Zero";
    list[1] = "One";
    list[2] = "Two";
    list[3] = "Three";
    for (index = 0 ; index < 4 ; index++)
        puts(list[index]);
}
```

This example will print out "ZeroOneTwoThree". When we dissect it we see that the array list is initialized first. We then reach the for loop. First, the initializer is evaluated, setting index to 0. Next, the condition is evaluated $index < 4$, which is true, so the body of the loop executes, printing "Zero". The iterator expression is then evaluated, increasing index by one. This continues until index is equal to 4, at which point the loop exits without executing the body again.

break

```
count() {
    int x;
    x = 5;
    while (x > 0) {
        if (x == 1)
            break;
        puts(x);
        x = x - 1;
    }
}
```

In this slightly more complex piece of code, the counting goes on as it normally would, printing out "5432". However, when x reaches 1, break is executed, breaking out of the while loop early, before the 1 gets printed.

continue

```
count() {
    int x;
    x = 6;
```

```

while (x > 1) {
    x--; // Do the subtraction first
    if (x == 3)
        continue;
    puts(x);
}
}

```

In this clearly contrived example, the output is "5421". When x reaches 3, the continue is executed, passing execution to the beginning of the loop, skipping over the puts.

switch statement

```

int    myInteger;
myInteger = 50;
switch (myInteger)
{
    case 40:
    case 30:    puts1("MyInteger is 40 or 30"); break;
    case 50:    puts1("MyInteger is 50");    break;
    default   :    puts("MyInteger is " + myInteger);
}

```

Note: if myInteger is 50, it will print MyInteger is 50
if myInteger is 40 or 30, it will print MyInteger is 40 or 30
Otherwise, it will print MyInteger is the actual MyInteger's value.

Include

Using the include keyword, it becomes possible to write programs in smaller chunks. The contents of the included file are functionally inserted into the line containing the include keyword.

Note:

For WinCE, include must contain the full path name. WinCE doesn't support relative path.

Example:

File MyFunctions.pc

```

times5(int x) {
    return x*5;
}

```

Another memo:

```

// My Program
#include "MyFunctions.pc"
main() {
    int y;
    y = times5(7);
    puts(y); // Prints 35
}

```

The compiler sees this as:

```

// My Program
times5(int x) {
    return x*5;
}
main() {
    int y;
    y = times5(7);
    puts(y); // Prints 35
}

```

```
}
```

Special Characters

There are two ways to add special characters to a string. The first is by appending them by number, such as:

```
str = "Here is a neat little square: " + (char)149;
```

The other method is through using escape sequences. The following escape sequences are supported:

Escape sequence	\\	\'	\"	\n	\r	\t	\x
Interpretation	\	'	"	newline	carriage return	tab	character specified by the following two hex digits. Example: '\x95' is the block character (decimal 149)

So, to create a string that contains a quote:

```
str = "She said \"I'm sorry,\" but it was too late...";  
puts(str); // Prints: She said "I'm sorry," but it was too late...
```

PocketC Windows CE 's source code is in Unicode. In order to make a unicode text file, you need to following a few rules. The first two bytes must be 0xFF and 0xFE.

sample code for write unicode properly:

```
test_writeunicode(string data)  
{  
int handle;  
char byte1;  
char byte2;  
byte1 = 0xFF;  
byte2 = 0xFE;  
handle = fopen("unicode.txt",1,FILE_CREATE|FILE_READWRITE);  
writebyte(handle,byte1);  
writebyte(handle,byte2);  
fwrite(handle,data,strlen(data)*2);  
fclose(handle);  
}
```

Pointers and Arrays

Note: Pointers are an advanced topic, which should be dealt with after the user is familiar with all the other programming concepts.

All variables are stored at some address in memory. A pointer is a variable which refers to another variable by containing that variable's address.

There are two primary operators which are used with pointers, * and &. The * operator dereferences the pointer. A dereferenced pointer acts just like the data to which it points. The & operator returns the address of a given variable. To illustrate:

```
pointer p, q;  
int i;  
main() {  
i = 5;  
p = &i; // Assign the address of 'i' to the pointer 'p'  
// now, typing '*p' is the same as typing 'i'
```

```

puts(*p); // Print the value of 'i'
*p = 7;   // Assign 7 to 'i'
q = p;    // Assign the value of 'p', which is the address of 'i', to 'q'
          // now, typing '*q' is the also the same as typing 'i'
// Things not to do
p = 8;    // BAD! Don't assign a constant value to a pointer
*i = 9;   // BAD! Don't try to dereference a non-pointer
}

```

A pointer can also be used to take the address of a function (but NOT a built in function!). Unlike variables, however, the & operator is NOT used. Calling a function through a pointer is a little tricky. First, the code looks ugly. Second, no error checking can be done on the parameters, so type conversions are not done and the number of arguments is not confirmed. For example:

```

func(int x) { return 5*x; }
main() {
    int result;
    pointer ptr;
    ptr = func; // Take the address of a function
    result = (*ptr)(5); // call the function (ugly)
    // Things not to do
    result = (*ptr)("5"); // this won't work, since the string
                          // isn't converted to an integer
    result = (*ptr)(5,7); // this will compile, but will result
                          // in stack corruption because the
                          // wrong number of arguments are used
}

```

Pointers and arrays

Pointers and arrays are fairly similar. Pointers can use the [] operator, and an array variable (when not used with []) results in the address of the first element. For example:

```

int array[5];
pointer p;
main() {
    p = array; // Assign the address of the first element of
              // 'array' to 'p'
    *p = 7;    // Assign 7 to array[0]
    p[1] = 8;  // Assign 8 to array[1]
}

```

This enables the pointers to arrays to be passed as function parameters. This also allows the user to implement their own version of two-dimensional arrays. By creating an array of pointers, each of which is a pointer to an array (or part of one), a two-dimensional array can be simulated.

```

int array[100];
pointer twod[10]; // after init(), this can be treated
                  // like at 10x10 matrix
init() {
    int i;
    for (i=0;i<10;i++)
        twod[i]=array + i*10; // Pointer arithmetic
}
main() {
    int x, y;
    init();
    for (x=0;x<10;x++)

```

```

for (y=0;y<10;y++)
    twod[x][y]=x * y; // Sets array[x*10 + y] = x*y
}

```

Pointer arithmetic

Pointer values can be used in a limited number of expressions. You can add and subtract from a pointer (and, thus, can use the increment and decrement operators as well). When you add 1 to a pointer, the pointer points to the next value in memory. Similarly, when you subtract 1 from a pointer, the pointer points to the previous value in memory. Caution should be used when using pointer arithmetic, because dereferencing an invalid memory location will cause an error during run time.

Databases

Introduction

A WinCE database is like a big spreadsheet table. In a regular spreadsheet, you have multiple columns and rows. In the WinCE database, Row is called record and columns are called properties. Each record can contain 0 or more properties. Each property is defined by a standard datatype, like integer or string.

Figure 1. A sample WinCE database

100	John Smith	11 Broadway	Cleveland
101	Nicky Taylor	1 Sunny Drive	MountainView
102	Blues Brother	532 194RD PL NE	Seattle

You probably know something about SQL if you are using database functions. Microsoft announced the SQL support for WindowsCE. Right now, you have to write functions to manipulate the data in your database.

Memory on your WinCE machine is very limited. Before you start working on your database, take a moment to design the database layout to minimize the storage requirement.

Create a database

PocketC Database APIs are supposed to be simple and easy to use. If you have the time or luxury, go search on <http://msdn.microsoft.com/> for WindowsCE database functions. You will thank us for the simple database interface in PocketC. In this section, we are going to learn how to create, open and close a database.

Create a database

Once you have a clear idea how the database or “spreadsheet” is going to look like, it is time to create the database with the **dbcreate** function. It takes two arguments:

Database name: The name can be up to 32 characters long. If you pass a longer name, the first 32 characters will be treated as the database name. All databases will be placed in the \Database folder.

Database type: an integer that can be used later on for search database of certain type. You can decide the number for your own database. Don’t assume the number you pick is used only for your database. Other database might use the same number.

Return value: if **dbcreate** is successful, it returns an integer. This integer is the object identifier for your database. It can be used to open the database you just created. see **dbopen** for detail.

When can dbcreate fail?

The WinCE database storage is running out of space.

Another database is already using the name

When it fails, the return value is 0. You can call **getlasterror** to obtain the error information.

Open a database

After a database is created successfully, you can open it now. A good programming practice is to check the return value of **dbcreate** before call **dbopen**. If you don’t check, your program might run on one user’s machine,

and fail on the other user's machine. For example: User B has a database called "ToDoList", and your program tried to create a database with the same name.

*Each WinCE database can be distinguished from each other by using its object id or its name. **dbopen** is the function you need to open a database. It takes two parameters:

Database Object ID: An integer which identify the database. It can be obtained by using **dbcreate** or **dbenum**.

Database name: The name can be up to 32 characters long. If you pass a longer name, the first 32 characters will be treated as the database name. All databases will be placed in the \Database folder.

You can specify either object id or name to open the database. If you know the function name only, just use 0 for the object ID. If you know the function object ID only, just use an empty string "" for the database name parameter.

Return value: If the database is opened, the return value is the object id of this database. Otherwise, the return value is 0.

When can dbopen fail?

the operating system doesnot have enough memory to open the database

the database doesnot exist with the OID or name you provided.

Note: PocketC Database API only supports opening one database at a time. If you want to open a second database, you must close the current database first.

Close Database

Since we can only open one database at one time, close a database is pretty simple. Just call **dbclose()**.

Return value can be 1 or 0. 1 means the database is closed. 0 means the function failed to close the database.

Section One Summary

In Section One, we learned how to create, open and close a database on WindowsCE.

dbcreate(string name, int dbtype) – create a new database. it needs a name and a type.

dbopen(int oid, string name) – open a database by its object id or name

dbclose() – close current opening database.

It is time to write a few lines of PocketC code to review what you have learned. After you create a new database, the big question is that how can you tell if your database is actually created? Please read Section Two to find out how!

Enumerate database

In this section, we are going to find out all the database on your WindowsCE device! You will probably be surprised by the number of databases.

Enumerate Database

dbenum is the function can be used to enumerate all the databases. It takes two parameters:

int bFirst: you must use 1 for parameter bFirst if you are using **dbenum** to get the first database. After that, you just keep using 0 until **dbenum** returns 0.

int databasetype: Do you remember **dbcreate** takes a database type parameter? You can use it here to specify what type of database you want to enumerate through. If you want to see all the database regardless of their type, just use 0 for database type.

Return value:

dbenum returns one database object id each time it is called. Once all of the databases are enumerated, **dbenum** returns 0. You will need the object id to find out more about the database.

Database Attributes

Now we have the database's object id, we can find out the database's name, physical storage size in bytes and number of records.

string dbname(int oid) – returns database name in a string. If failed, return an empty string ""

int dbsize(int oid) – returns database physical storage size in bytes. If failed, returns 0

in dbnrecs(int oid) – returns total records in the database. If failed, return 0

For above three functions, if parameter oid is 0, the function will return current open database's attributes.

Show me the CODE!

Here is a short segment of PocketC source code to list all the database.

```

listdb()
{
    int ret;    int oid;
    ret = dbenum(1,0);
    if (ret==0) return;
    puts1("List all the databases");
    puts1("Name-----Size-----NumRecords---");
    while ((oid=dbenum(0,0))!=0)
    {
        puts1(dbname(oid) + "\\t\\t" + dbsize(oid) + "\\t" + dbnrecs(oid));
    }
}

```

Output:

```

List all the databases
Name-----Size-----NumRecords---
testdb    456    2
\RecycleInfo    480    1
\Categories Database    1744    20
Contacts Database    380    0
Tasks Database    380    0
Appointments Database    380    0
\DesktopPositions    1444    11
\EventNotifications    600    5
\UserNotifications    380    0

```

The first database in the list is called testdb, which is created using PocketC. We will show you how to write actual data to the database in the next section.

***Note:** Did you notice that some database with a backslash afront of their names? These databases are the Windows CE system databases. Don't modified them unless you know what you are doing. Btw, Don't forget to add a backslash, otherwise, \D will be converted into some character, and dbopen will fail. You need to call **dbopen(0,"\\DesktopPositions")**.

Read database contents

Don't you want to find out what is in those databases listed by our code? I know I want to. In this section, we are going to learn how to dump the entire database out to the console screen.

The concept is fairly simple. We are going to go through each record in the database, and display the record properties one by one. The number of the record can be found by using [dbnrecs\(int oid\)](#). The tricky part is to find out the record properties.

int dbrecpropent() – returns number of properties in the current record. If failed, return -1.

int dbrecproptype(int index) – returns the property type by using its index. Windows CE database supports following data types.

```

#define CEVT_I2 2
#define CEVT_UI2 18
#define CEVT_I4 3
#define CEVT_UI4 19
/*Not Supported */ #define CEVT_FILETIME 64
#define CEVT_LPWSTR 31
/*Not Supported */ #define CEVT_BLOB 65
CEVT_I2
A 16-bit signed integer.
CEVT_I4
A 32-bit signed integer.
CEVT_LPWSTR
A null-terminated string.
CEVT_UI2

```

A 16-bit unsigned integer.
CEVT_UI4
 A 32-bit unsigned integer.

We currently don't support CEVT_FILETIME and CEVT_BLOB. PocketC users will not be able to access CEVT_FILETIME and CEVT_BLOB properties. I am working on a solution, and open for suggestions as well. It is possible to reuse the existing time function to obtain the CEVT_FILETIME.

dbrecpropval(int index) – returns the property value by using its index. Possible return type is int and string.

It seems we almost have everything we needed. We can find out the property count of the current record, the property type and value. When we call dbopen to open a database, it automatically set the current record to the first record in the database. We need a function that allow us to move current record position.

dbseek(int seektype, int dwValue) – seek the record depends on the seek type and seek value. Returns record object id. If failed, return 0.

```
#define CEDB_SEEK_CEOID 0x00000001
#define CEDB_SEEK_BEGINNING 0x00000002
#define CEDB_SEEK_END 0x00000004
#define CEDB_SEEK_CURRENT 0x00000008
```

Again, here is the list of seek type we support at this moment.

Seek Type	Seek Value
CEDB_SEEK_CEOID	Seek until finding an object that has the specified object identifier. The dwValue parameter specifies the object identifier. This type of seek operation is very efficient.
CEDB_SEEK_BEGINNING	Seek until finding the record at the specified position from the beginning of the database. The dwValue parameter specifies the number of records to seek.
CEDB_SEEK_END	Seek backward for the specified number of records from the end of the database. The dwValue parameter specifies the number of records.
CEDB_SEEK_CURRENT	Seek backward or forward from the current position of the seek pointer for the specified number of records. The dwValue parameter specifies the number of records from the current position. The function seeks forward if dwValue is a positive value, or backward if it is negative. A forward seek operation is efficient.

Show me the CODE!

This function dumps the contents of a database.

```
checkdb(int oid)
{
    int ret;    int cnt; int i; int reccnt; int j;
    puts1("-----");
    ret = dbopen(oid, "");
    if (ret==0)
    {
        puts1(dbname(0) + " open failed");
        return;
    }
    reccnt = dbnrecs(0);
    for (j = 0; j < reccnt; j++)
    {
        cnt = dbrecpropcnt();
        for (i=0; i < cnt; i++)
        {
            puts("(" + dbrecproptype(i) + " " + dbrecpropval(i) + ")");
        }
        dbseek(8,1);
    }
}
```

```

    puts1("");
}
puts1("-----");
ret = dbclose();
}

```

Database Output for a database with name “\Categories Database”

\Categories Database 1744 20

```

(2 0)(2 1)(31 Business)(2 0)(2 0)(2 0)
(2 1)(2 1)(31 Competition)(2 0)(2 0)(2 0)
(2 2)(2 1)(31 Favorites)(2 0)(2 0)(2 0)
(2 3)(2 1)(31 Gifts)(2 0)(2 0)(2 0)
(2 4)(2 1)(31 Goals/Objectives)(2 0)(2 0)(2 0)
(2 5)(2 1)(31 Holiday)(2 0)(2 0)(2 0)
(2 6)(2 1)(31 Holiday Cards)(2 0)(2 0)(2 0)
(2 7)(2 1)(31 Hot Contacts)(2 0)(2 0)(2 0)
(2 8)(2 1)(31 Ideas)(2 0)(2 0)(2 0)
(2 9)(2 1)(31 International)(2 0)(2 0)(2 0)
(2 10)(2 1)(31 Key Customer)(2 0)(2 0)(2 0)
(2 11)(2 1)(31 Miscellaneous)(2 0)(2 0)(2 0)
(2 12)(2 1)(31 Personal)(2 0)(2 0)(2 0)
(2 13)(2 1)(31 Phone Calls)(2 0)(2 0)(2 0)
(2 14)(2 1)(31 Status)(2 0)(2 0)(2 0)
(2 15)(2 1)(31 Strategies)(2 0)(2 0)(2 0)
(2 16)(2 1)(31 Suppliers)(2 0)(2 0)(2 0)
(2 17)(2 1)(31 Time & Expenses)(2 0)(2 0)(2 0)
(2 18)(2 1)(31 VIP)(2 0)(2 0)(2 0)
(2 19)(2 1)(31 Waiting)(2 0)(2 0)(2 0)

```

Write data to a database

After going through the previous three sections, you should have enough knowledge to browse through any WindowsCE databases. It is time to learn how to write to your own database, or modify the existing databases.

To make learning interesting, we are going to create a stock portfolio database.

Figure 2. A sample stock portfolio. The first column is the index number for each company stock. The second column is the company name, the third is the company’s stock symbol and the last column is how many shares you have.

1001	Intel Corp	INTC	60
1002	Net Bank	NTBK	300
1003	QWest	QWST	50

The name of our sample stock is called “sample stock”, and the type for our database is 20000. You can use a different name and different value for the database type. Just remember not to use any existing database names.

Review Section One, you should be able to write following source code which will create and open the database.

Sample source code:

```

int dboid;
dboid = dbcreate("sample stock",20000);
if (dboid == 0)
{ alert("failed to open database"); return; }
dbopen(dboid,"");

```

Now we are almost ready to write data to the database.

First of all, we need to learn a few tricks with `dbrecwrite` function.

dbrecwriteprop(int rec_oid, int prop_type, int prop_id, void prop_value)

`int rec_oid` : each record in the database has its own object identifier. You can obtain the `rec_oid` by using **dbrecnow()**. If `rec_oid` is 0, a new record will be written.

`int prop_type` : each property has a property type. The property type can be

`CEVT_I2`

A 16-bit signed integer.

`CEVT_I4`

A 32-bit signed integer.

`CEVT_LPWSTR`

A null-terminated string.

`CEVT_UI2`

A 16-bit unsigned integer.

`CEVT_UI4`

A 32-bit unsigned integer.

`int prop_id` : each property can have its identification. The identification is simply an positive integer value which can be decided by the developer.

`void prop_value`: depends on the type you supplied in `prop_type`. If `prop_type` is `CEVT_I4` a 32-bit signed integer, you better have a integer for `prop_value`.

For example, I want to create a new record with one integer property. The property's ID is 1, and Value is 1000.

Here is the code:

```
int recoid;
recoid = dbrecwriteprop(0, CEVT_I4, 1, (int) 1000);
```

*Remember to put typecast around the property value parameter.

What about one more property for this record? The good thing we saved the record object id for the new record we just created. Now we can use `recoid` again in the next call to `dbrecwriteprop`.

The second property is a string, its ID is 2, and Value is "Intel Corp".

```
recoid = dbrecwriteprop(recoid, CEVT_LPWSTR, 2, (string) "Intel Corp");
```

**One more time, Remember to put typecast around the property value parameter. WinCE database doesnot support Float or Char Property Type. So if you have float or character values, just type cast them into string or int.*

Now, Let's say I want to change first property's value to 1001. Sounds difficult? Nah.

```
recoid = dbrecwriteprop(recoid, CEVT_I4, 1, (int) 1001);
```

Now the record should look like

1001	Intel Corp
------	------------

Review **dbrecwrite(...)**

Function **dbrecwrite** is a pretty useful function. Let us review what it just did for us.

Create a new record with one property

Update property in a record

Add property to a record

Since this function can do a many things at once, please be careful when you are using it. I am going to show you a sample code which write our stock data into one database.

Show me the CODE!

```
storedb()
{
    int recoid;
    /* line 1 */
    recoid = dbrecwrite(0, CEVT_I4, 1, (int) 1001);
```

```

    recoid =      dbrecwrite(recoid,CEVT_LPWSTR,2,(string)"Intel Corp");
    recoid =      dbrecwrite(recoid,CEVT_LPWSTR,3,(string)"INTC");
    recoid =      dbrecwrite(recoid,CEVT_I4,4,(int)60);
    /* line 2 */
    recoid =      dbrecwrite(0,CEVT_I4,1,(int)1002);
    recoid =      dbrecwrite(recoid,CEVT_LPWSTR,2,(string)"Net Bank");
    recoid =      dbrecwrite(recoid,CEVT_LPWSTR,3,(string)"NTBK");
    recoid =      dbrecwrite(recoid,CEVT_I4,4,(int)300);
    /* line 3 */
    recoid =      dbrecwrite(0,CEVT_I4,1,(int)1003);
    recoid =      dbrecwrite(recoid,CEVT_LPWSTR,2,(string)"QWest");
    recoid =      dbrecwrite(recoid,CEVT_LPWSTR,3,(string)"QWST");
    recoid =      dbrecwrite(recoid,CEVT_I4,4,(int)50);
}

```

Console Output :

```

(3 1003)(31 QWest)(31 QWST)(3 50)
(3 1002)(31 Net Bank)(31 NTBK)(3 300)
(3 1001)(31 Intel Corp)(31 INTC)(3 60)

```

***Note:** We created Intel Record first, and it is pushed one lower every time when a new record is written to the database.

Conclusion

Section Five... I hope you are not bored to death right now. If you made this far, you are capable of writing a simple database browser and you should know following basic concepts:

- Database is made up by many records
- Record is made up by many properties
- Property can be an integer or a string
- Each item in the database has its own identification number.
- Database has object id, Record has object id and Property has its own id
- PocketC can only open one database at a time

Before we go any further, I strongly suggest you write a small database browser, and create a database with your own data.

.... < a few hours later >

Now, here is a list of additional functions for you do play with.

dbrecdelprop(int recoid, int proptype, int propid);

parameters:

int recoid – record object id

int proptype – the property you want to delete's type

int propid – the property you want to delete's ID

Description:

Delete one property from the record

Return:

returns record id if successful. If function failed, return 0.

dbrecdel(int recoid)

parameters:

int recoid – record object id

Description:

Delete one record

Return:

1 indicates success. 0 indicates failure. The function could fail when you pass into a wrong or invalid record object id

dbrecnow()

Description:

Get current record object id. When you are iterating through the database, you can obtain the current record object id by call dbrecnow.

Return:

0 indicates failure. The function should return the record object id for the current reading/writing record dbdelete

dbdelete(int dboid)

parameters:

int dboid – database object id

Description:

Delete one database

Return: 1 indicates success. 0 indicates failure. The function could fail when you pass into a wrong or invalid database object id or another program is using the database at this moment.

Appendix – PocketC Database API

dbcreate(string dbname, int dbtype)

– create database with name and type

dbopen(int dboid, string dbname)

– open database by using database object id or name

dbclose()

– close database

dbdelete(int dboid)

– delete database

dbseek(int seektype, int seekvalue)

– seek database in various ways

dbrecnow()

– get current record object id

dbrecdelprop(int recoid, int proptype, int propid);

– delete property from a record

dbrecdel(int recoid)

– delete a record

dbrecwrite(int recoid, int proptype, int propid, void propvalue)

– write one property to the database

dbrecread(int proptype, int propid)

– read one specific property from the database

dbsize(int dboid)

– return database size

dbname(int dboid)

– return database name

dbnrecs(int dboid)

– return database record numbers

dbenum(int first, int dbtype)

– enumerate through databases with certain type

dbrecproptype(int index)

– find out property type

dbrecpropcnt()

– find out property count

dbrecpropval()

– find out property value

Funkcje API - opis

Console

```
clear()  
puts(string text)  
puts1(string text)  
gets(string prompt)
```

clear()
Clears the console window
Return: None

puts(string text)
Print a string to the output form, Does not append a newline,
[string text] : a short string
Return: None

puts1(string text)
Print a string to the output form, Automatically append a newline,
[string text] : a short string
Return: None

gets(string prompt)
Presents an input dialog with the given string as a prompt,
[string prompt] : a short string
Return: Returns a string if the user pressed OK, or zero if the user presses Cancel,

Draw

```
getpixelB(int x,int y)  
getbkmode()  
setbkmode(int iBkMode)  
settextangle(int x)  
settextdefault()  
setfontattr(string fontname, int italic, int underline,int strikeouts, int weight,int width, int height)  
textw(string text)  
keepscreen_on()  
keepscreen_off()  
screenx()  
screeny()  
texth[string text]  
clearg()  
text(int x, int y, string str)  
line(int x1, int y1, int x2, int y2)  
rect(int x1, int y1, int x2, int y2)  
roundrect(int x1, int y1, int x2, int y2,int cx,int cy)
```

```

circle(int x, int y, int r)
ellipse(int x1, int y1, int x2, int y2)
polygon(int * array, int nCount)
polyline(int * array, int nCount)
drawbitmap(string filename,int x,int y)
drawbitblt(string filename,int x,int y,int flag)
drawimage(string filename,int position_x,int position_y,int scale, int
max_w, int max_h, int flag)
setbrushattr(int R, int G, int B)
setpenattr(int style, int width, int R, int G, int B)
setbkcolor(int R, int G, int B)
settextcolor(int R, int G, int B)
setpixelattr(int R, int G, int B)
setpixel(int x,int y)

```

getpixelR(int x,int y)

getpixelG(int x,int y)

getpixelB(int x,int y)

Obtain color of a pixel

[int x] : horizontal x value

[int y] : vertical y value

Return: None

getbkmode()

this function returns the current background mix mode for a specified device context. The background mix mode of a device context affects text and dashed pens.

Return: Either OPAQUE or TRANSPARENT, which specifies the current background mix mode, indicates success. Zero indicates failure.

setbkmode(int iBkMode)

this function sets the background mix mode of the specified device context. The background mix mode is used with text, hatched brushes, and with non-solid pen styles.

[in iBkMode] : Specifies the background mode. It is either of the following values: OPAQUE TRANSPARENT

Return: If the function succeeds, the return value specifies the previous background mode.

settextangle(int x)

To set the angle at 10th degree accuracy of your text, For example, if you want to make your string looks like standing up, just passing value $900 = 90 \text{ Degree} * 10$,

[int x] : integer value for 10th of a degree

Return: None

settextdefault()

To set the text style back to the default state,

Return: None

setfontattr(string fontname, int italic, int underline,int strikethrough, int weight,int width, int height)

Set the current text drawing attributes, Italic,Underline, and Strikethrough can be either 0 or 1, width and height are the font size,

[string fontname] : example "Arial"

[int italic] : 0 or 1

[int underline] : 0 or 1

[int strikethrough] : 0 or 1

[int weight] : thin 100, normal 400, bold 700, extrabold 800, heavy 900

[int width] : font width

[int height] : font height

Return: None

textw(string text)

Precalculate the width of the text on screen without drawing it

[string text] : a string

Return: return the width

keepscreen_on()

Keep the screen graphics even if you switch to another application, Default is ON.

Return: None

keepscreen_off()

Turn off the screen keeper

Return: None

screenx()

Return screen width

Return: an integer value for screen width

screeny()

Return screen height, height value is offset by the menu height,

Return: an integer value for screen height

texth(string text)

Precalculate the height of the text on screen without drawing it

[string text] : a string

Return: return the height

clearg()

Clear the graphics form,

Return: None

text(int x, int y, string str)

Display a string str at locations (x,y),

[int x] : horizontal x value,

[int y] : vertical y value,

[string str] : text to be drawn on screen

Return: None

line(int x1, int y1, int x2, int y2)

Draws a line from (x1, y1) to (x2, y2),

[int x1] : initial point's x value

[int y1] : initial point's y value

[int x2] : end point's x value

[int y2] : end point's y value

Return: None

rect(int x1, int y1, int x2, int y2)

Draws a rectangle from (x1, y1) to (x2, y2)

[int x1] : initial point's x value

[int y1] : initial point's y value

[int x2] : end point's x value

[int y2] : end point's y value

Return: None

roundrect(int x1, int y1, int x2, int y2,int cx,int cy)

Draws a round rectangle from (x1, y1) to (x2, y2) with round corner (cx,cy),

[int x1] : initial point's x

[int y1] : initial point's y

[int x2] : end point's x
[int y2] : end point's y
[int cx] : cx is the width of the ellipse used to draw the rounded corners,
[int cy] : cy is the height of the ellipse used to draw the rounded corners
Return: None

circle(int x, int y, int r)

Draws a circle center at (x,y) with radius r
[int x] : horizontal x value for the center of the circle
[int y] : vertical y value for the center of the circle,
[int r] : radius for the circle
Return: None

ellipse(int x1, int y1, int x2, int y2)

Draws an ellipse from (x1, y1) to (x2, y2)
[int x1] : initial point's x value
[int y1] : initial point's y value
[int x2] : end point's x value
[int y2] : end point's y value
Return: None

polygon(int *array, int nCount)

polygon function draws a polygon on the screen. A ploygon is made up by a list of x and y values on a 2D space. The points are connected by a line. This function draws a polygon consisting of two or more vertices connected by straight lines. The current pen outlines the polygon, and the current brush fills it using the specified polygon fill mode.

[int *int_array] an array of integers which represents a series of x and y values. 0 and Even index is the x value, and odd index is the y value. For example: (0,1) (2,3) (4,5) (6,7) (8,9) Draw a polygon with number of points (greater or equal to 2)

[int nCount] indicates how long this array is.

Return: None

Example:

```
main()
{
int p[10];
int i;
p[0] = 100; p[1] = 100;
p[2] = 140; p[3] = 170;
p[4] = 300; p[5] = 200;
p[6] = 60; p[7] = 180;
p[8] = 30; p[9] = 100;
polygon(p,10);
}
```

polyline(int *array, int nCount)

polyline function draws a series of line segments by connecting the points in the specified array.

[int *int_array] an array of integers which represents a series of x and y values. 0 and Even index is the x value, and odd index is the y value. For example: (0,1) (2,3) (4,5) (6,7) (8,9)

[int nCount] indicates how long this array is.

Return: None

```
main()
{
int p[10];
int i;
p[0] = 100; p[1] = 100;
```

```

p[2] = 140; p[3] = 170;
p[4] = 300; p[5] = 200;
p[6] = 60; p[7] = 180;
p[8] = 30; p[9] = 100;
polyline(p,10);
}

```

drawbitmap(string filename,int x,int y)

draws a bitmap on screen at location x and y.

[string filename] : file path

[int x] : horizontal x value

[int y] : vertical y value

Return: None

drawbitblt (string filename,int x,int y,int flag)

Draw a bitmap on screen at x and y location with special flags *Note: with this function, you can draw bitmap on screen with different effects,

[string filename] : file path

[int x] : horizontal x value

[int y] : vertical y value

[int flag] : special draw flag Flags: 0 COPY, 1 AND, 2 INVERT(XOR), 3 PAINT(OR), 4 ERASE, 5 NOTSRCCOPY, 6 NOTSRCERASE, 7 MERGECOPY, 8 MERGEPAINT, 9 PATCOPY, 10 PATPAINT, 11 PATINVERT, 12 DSTINVERT, 13 BLACK, 14 WHITE

Return: None

drawimage (string filename,int position_x,int position_y,int scale, int max_w, int max_h, int flag)

Draw a graphic image file on screen at position_x and position_y location with special flags,

*Note: with this function, you can draw bitmap on screen with different effects and scales,

[string filename] : file path

[int position_x] : horizontal x value

[int position_y] : vertical y value

[int scale] : scale range (1-100)

[int max_w] : maximum weight

[int max_h] : maximum height

[int flag] : special draw flag Flags: 0 COPY, 1 AND, 2 INVERT(XOR), 3 PAINT(OR), 4 ERASE, 5 NOTSRCCOPY, 6 NOTSRCERASE, 7 MERGECOPY, 8 MERGEPAINT, 9 PATCOPY, 10 PATPAINT, 11 PATINVERT, 12 DSTINVERT, 13 BLACK, 14 WHITE

Return: None

setbrushattr(int R, int G, int B)

Set the brush color, it is used for rect, circle, ellipse's filling color,

[int R] : Red

[int G] : Green

[int B] : Blue, *Note: RGB red, green, blue (RGB) color based on the specified color values

Return: None

setpenattr(int style, int width, int R, int G, int B)

Set the pen attribute,

[int nStyle] : style can be solid (0), dash(1), or invisible (5),

[int nWidth] : an integer value that determines your pen's width,

[int R] : Red

[int G] : Green

[int B] : Blue, *Note: RGB red, green, blue (RGB) color based on the specified color values

Return: None

setbkcolor(int R, int G, int B)

Set the background color

[int R] : Red

[int G] : Green
[int B] : Blue, *Note: RGB red, green, blue (RGB) color based on the specified color values
Return: None

settextcolor(int R, int G, int B)

Set the text color

[int R] : Red

[int G] : Green

[int B] : Blue, *Note: RGB red, green, blue (RGB) color based on the specified color values

Return: None

setpixelattr(int R, int G, int B)

Set the pixel color attribute

[int R] : Red

[int G] : Green

[int B] : Blue, *Note: RGB red, green, blue (RGB) color based on the specified color values

Return: None

setpixel(int x,int y)

Set the pixel at location x and y

[int x] : horizontal x value

[int y] : vertical y value

Return: None

getpixelR(int x,int y)

Obtain color of a pixel

[int x] : horizontal x value

[int y] : vertical y value

Return: None

getpixelG(int x,int y)

Obtain color of a pixel

[int x] : horizontal x value

[int y] : vertical y value

Return: None

Event

```
killtimer(int timerid)
keyevent(char vkey,char cScan, int flag, int extra)
waitp()
getc()
getnotify()
flushevent()
event(int blocking)
postevent(int nEvent)
penx() mousex()
peny() or mousey()
cursorwait(int nState)
menu()
guiid()
key()
timerid()
settimer(int timerid,int timeElapse)
```

killtimer(int timerid)

Kill the timer ID, otherwise, you will receive a PM_TIMER every timeElapse milliseconds,

[int nTimerID] : timer identifier

Return: Return 1 or 0 indicating TRUE or FALSE

keyevent(char vkey,char cScan, int flag, int extra)

Synthesizes a keystroke. The system can use such a synthesized keystroke to generate a WM_KEYUP or WM_KEYDOWN message

[char vkey] : Specifies a virtual-key code. The code must be a value in the range 1 to 254.

[char cScan] : Specifies a hardware scan code for the key.

[int flag] : KEYEVENTF_KEYUP If specified, the key is being released. If not specified, the key is being depressed.

[int extra] : Specifies an additional 32-bit value associated with the key stroke.

Return: none

waitp()

Wait for a pen event PM_BUTTONUP

Return: None

getc()

Wait for a key is released, and return the previous key value

Return: Char value

getnotify()

when an event PM_COMMAND occurs, it contains a notification message. Window controls like ListBox, Button, ComboBox have its own set of notification messages. When a PM_COMMAND event occurs, you can call two functions to find out which windows control sends this message, and why. First call guid() to find out the window control's ID, second call getnotify() to find out the control's notification message.

Return: return the notification message.

flushevent()

Flush all events in the current event queue

Return: return none

event(int blocking)

check for events in the event queue. [Warning] add sleep(0) statement into your tight loop. You need to allow other processes to have time slice when you are in a tight event. If you don't put sleep(0) in a tight loop, you won't be able to switch tasks and battery power will be drained quickly.

[int blocking] : If blocking is zero, event returns immediately, otherwise it waits for an event,

Return:Events are as follows:PM_NONE [0],PM_CHAR[1],PM_MOUSEMOVE[2],
PM_MEMORYSHORT[3], PM_BUTTONDOWN[4], PM_BUTTONUP[5], PM_DBCLICK[6],
PM_PAINT[7], PM_COMMAND[8],PM_KEYUP[10],PM_KEYDOWN[9],PM_TIMER[11]

postevent(int nEvent)

Post your own event to the event queue, make sure they don't have the same value as the standard PocketC event values,

[int nEvent] : event identifier,

Return: None

penx()

Retrieve the x value of the previous pen event,

*Note: Any MouseMove, Button related messages, Mouse position can be tracked with mousex() and mousey() function

Return: integer value for x position

peny()

Retrieve the y value of the previous pen event,

*Note: Any MouseMove, Button related messages, Mouse position can be tracked with mousex() and mousey() function

Return: integer value for y position

cursorwait(int nState)

Show/hide sandbox waiting box on screen

[int nState] : state can be 1 or 0, 1 means show wait cursor, 0 means hide wait cursor

Return: None

menu()

Retrieve the ID value of the previous menu event, *Note: When user select a menu item, PM_COMMAND event will be passed back as the return value of event(int) function, You should use menu() to get the menu id which was selected by the user,

Return: integer value for menu identifier

guiid()

Retrieve the ID value of the previous graphic user interface,

Return: integer value for GUI identifier

key()

Retrieve the character written during the last event()

Return: key value

timerid()

Retrieve the timer id for the last PM_TIMER message

Return: integer value for Timer identifier

settimer(int timerid,int timeElapse)

Set a timer, You can set up multiple timer, Each timer has its own ID, and its time length, When you set a timer, remember to specifies the time out value, in milliseconds, When time out, in your event queue, you will receive a PM_TIMER (11) message,

[int nTimerID] : timer identifier

[int timeElapse] : time length in ms,

Return: If the function succeeds, the return value is an integer identifying the new timer, An application can pass this value, or the string identifier, if it exists, to the killtimer function to destroy the timer, If the function fails to create a timer, the return value is Zero.

File

```
writebyte(int handle, int v)
```

```
readint(int handle)
```

```
readchar(int handle)
```

```
readbyte(int handle)
```

```
fileenum(int first, string filename)
```

```
filecopy(string ExistingFilePath,string NewFilePath,int bFailIfExists)
```

```
filemove(string ExistingFilePath,string NewFilePath,int bFailIfExists)
```

```
fileopen(string filepath, int type, int flag)
```

```
filemodeget(int filehandle)
```

```
filemodeset(int filehandle, int filetype)
```

```
fileclose(int filehandle)
```

```
fileread(int filehandle,int counter)
```

```
filewrite(int filehandle,string data,int counter)
```

```
filegetlen(int filehandle)
```

```
filesetlen(int filehandle, int newlen)
```

```
fileseek(int filehandle,int offset,int from)
```

```
fileseekend(int filehandle)
```

```
fileflush(int filehandle)
```

```

filepos(int filehandle)
[OBSOLETE] CopyFile(string NewFilePath,string ExistingFilePath,int
bFailIfExists)
MoveFile(string ExistingFileName,string NewFileName)
CreateDirectory(string Dirname)
RemoveDirectory(string Dirname)
DeleteFile(string FileName)
GetFileAttr(string filepath)
OpenFileDialog(string FileFilter)
SaveFileDialog(string FileFilter)
writebytes(int handle, string s)
writechar(int handle, int v)
writechars(int handle, string s)
writeint(int handle, int v)

```

writebyte(int handle, int v)

writes to the output stream the eight low- order bits of the argument v.

[int handle] : a file or serial connection handle

[int v] : an integer value

Return: None

readint(int handle)

reads four input bytes and returns an int value

[int handle] : input handle for file or serial connections

Return: an integer

readchar(int handle)

reads an input char and returns the char value

[int handle] : input handle for file or serial connections

Return: an Unicode char read

readbyte(int handle)

reads and returns one input byte.

[int handle] : input handle for file or serial connections

Return: the 8-bit value read.

fileenum(int first, string filename)

The function enumerates through all files in one directory.

[int first] : use 1 for find the first file.

[string filename] : specifies a valid directory or path and filename, which can contain wildcard characters * and ?

Return: returns file name. If returns an empty string, that indicates no more files have been found.

filecopy(string ExistingFilePath,string NewFilePath,int bFailIfExists)

Copies an existing file to a new file. Make sure set the last parameter bFailIfExists . It specifies how this operation is to proceed if a file of the same name as that specified by NewFilePath already exists. If this parameter is TRUE and the new file already exists, the function fails. If this parameter is FALSE and the new file already exists, the function overwrites the existing file and succeeds.

[string existingfilepath] : file path, Filepath means entire full file path to the file including the file name, For example: “/PocketC/test,pc”

[string newfilepath] : file path, Filepath means entire full file path to the file including the file name, For example: “/PocketC/test,pc”

[int bFailIfExists] : , if a file of the same name as that specified by lpNewFileName already exists, If this parameter is 1 and the new file already exists, the function fails, If this arameter is 0 and the new file already exists, the function overwrites the existing file and succeeds

Return: Return 1 or 0, 1 means True, 0 means False

filemove(string ExistingFilePath,string NewFilePath,int bFailIfExists)

Moves an existing file to a new file.

[string existingfilepath] : file path, Filepath means entire full file path to the file including the file name, For example: “/PocketC/test,pc”

[string newfilepath] : file path, Filepath means entire full file path to the file including the file name, For example: “/PocketC/test,pc”

Return: Return 1 or 0, 1 means True, 0 means False

fileopen(string filepath, int type, int flag)

Open a file in unicode/ascii. You can create a new file or simply open one. Please use the flag correctly.

[string filepath] : file path, Filepath means entire full file path to the file including the file name, For example: “/PocketC/test,pc”,

[int type] : file type, type could be 0 or 1, 0 means ASCII, 1 means Unicode

[int flag] : please specifies the action to take when opening the file. You can combine options listed below by using the bitwise-OR (|) operator.

FILE_CREATE 0x00001000 /*create a new file. If the file exists already, it is truncated to 0 length.*/

FILE_NOTRUNCATE 0x00002000 /*Combine this value with modeCreate. If the file being created already exists, it is not truncated to 0 length. Thus the file is guaranteed to open, either as a newly created file or as an existing file. This might be useful, for example, when opening a settings file that may or may not exist already.*/

FILE_READ 0x00000000 /*Opens the file for reading only.*/

FILE_READWRITE 0x00000002 /*Opens the file for reading and writing.*/

FILE_WRITE 0x00000001/*Opens the file for writing only.*/

Return: Returns an integer as the File Handle if successful, otherwise -1, Remember to keep this handle value somewhere, Because you have to use this handle for the rest of file operations,

filemodeget(int filehandle)

Obtain file mode,

[int filehandle] : filehandle, It is used to indentify which file this function is targeted,

Return: Return file mode, ASCII or Unicode

filemodeset(int filehandle, int filetype)

set file mode ASCII or Unicode, Use with caution, Don't try to write unicode and ascii code into one file,

[int filehandle] : filehandle, It is used to indentify which file this function is targeted,

[int filetype] : type could be 0 or 1, 0 means ASCII, 1 means Unicode

Return: None

fileclose(int filehandle)

Close a file, Note: you have to keep the return value from fileopen() for closing that file,

[int filehandle] : filehandle, It is used to indentify which file this function is targeted,

Return: None

fileread(int filehandle,int counter)

reads (unbuffered) data from a file at the current file position,

[int filehandle] : filehandle, It is used to indentify which file this function is targeted,

[int counter] : The maximum number of bytes to be read from the file, For text mode files, carriage return-linefeed pairs are counted as single characters,

Return: The number of bytes transferred to the buffer, Note , the return value may be less than nCount if the end of file was reached,

filewrite(int filehandle,string data,int counter)

writes (unbuffered) data in a file to the current file position,

[int filehandle] : filehandle, It is used to indentify which file this function is targeted,

[string data] : data you want to write to the file,

[int counter] : The maximum number of bytes to write. String in PocketC is unicode, each character takes two bytes. Make sure when you save strings to a file, its byte size is the string length * 2. If you want to save to an ASCII file, the maximum byte size is string length* 1.

Return: If failed, return -1. If successful, it returns number of written bytes.

filegetlen(int filehandle)

retrieves the length of the file, Note: Obtains the current logical length of the file in bytes, not the amount,

[int filehandle] : filehandle, It is used to identify which file this function is targeted,
Return: The length of the file,

filesetlen(int filehandle, int newlen)

set the length of the file. Newlen value can be larger or smaller than the current length of the file, The file will be extended or truncated according to the newlen

[int filehandle] : filehandle, It is used to identify which file this function is targeted,

[int newlen] : Desired length of the file in bytes.

Return: None

fileseek(int filehandle,int offset,int from)

positions the current file pointer, Repositions the pointer in a previously opened file, The Seek function permits random access to a file's contents by moving the pointer a specified amount, absolutely or relatively, No data is actually read during the seek, When a file is opened, the file pointer is positioned at offset 0, the beginning of the file.

[int filehandle] : filehandle

[int offset] : Number of bytes to move the pointer,

[int from] : pointer movement mode, (FILE_BEGIN 0) Move the file pointer lOff bytes forward from the beginning of the file, (FILE_CURRENT 1) Move the file pointer lOff bytes from the current position in the file, (FILE_END 2) Move the file pointer lOff bytes from the end of the file, Note that lOff must be negative to seek into the existing file; positive values will seek past the end of the file,

Return: If the requested position is legal, Seek returns the new byte offset from the beginning of the file,

fileseekend(int filehandle)

sets the value of the file pointer to the logical end of the file, fileseekend() is equivalent to fileseek(filehandle,0, FILE_END),

[int filehandle] : filehandle, It is used to identify which file this function is targeted,

Return: Return: The length of the file in bytes,

fileflush(int filehandle)

flushes any data yet to be written, Forces any data remaining in the file buffer to be written to the file,

[int filehandle] : filehandle, It is used to identify which file this function is targeted,

Return:

filepos(int filehandle)

obtains the current value of the file pointer, which can be used in subsequent calls to Seek,

[int filehandle] : filehandle, It is used to identify which file this function is targeted,

Return: current value of the file pointer

[OBSOLETE] CopyFile(string NewFilePath,string ExistingFilePath,int bFailIfExists)

copyFile function copies an existing file to a new file. Make sure set the last parameter bFailIfExists . It specifies how this operation is to proceed if a file of the same name as that specified by NewFilePath already exists. If this parameter is TRUE and the new file already exists, the function fails. If this parameter is FALSE and the new file already exists, the function overwrites the existing file and succeeds.

[string existingfilepath] : file path, Filepath means entire full file path to the file including the file name, For example: "/PocketC/test,pc"

[string newfilepath] : file path, Filepath means entire full file path to the file including the file name, For example: "/PocketC/test,pc"

[int bFailIfExists] : , if a file of the same name as that specified by lpNewFileName already exists, If this parameter is 1 and the new file already exists, the function fails, If this parameter is 0 and the new file already exists, the function overwrites the existing file and succeeds

Return: Return 1 or 0, 1 means True, 0 means False

MoveFile(string ExistingFileName,string NewFileName)

the MoveFile function renames an existing file or a directory(including all its children),

[string existingfilepath] : file path, Filepath means entire full file path to the file including the file name, For example: "/PocketC/test,pc"

[string newfilepath] : file path, Filepath means entire full file path to the file including the file name,

Return: Return 1 or 0, 1 means True, 0 means False

CreateDirectory(string Dirname)

the CreateDirectory function creates a new directory,

[string dirpath] : dir path, Filepath means entire full dir path, For example: “/MyFolder”

Return: Return 1 or 0, 1 means True, 0 means False

RemoveDirectory(string Dirname)

the RemoveDirectory function deletes an existing empty directory,

[string dirpath] : dir path, Filepath means entire full dir path, For example: “/MyFolder”

Return: Return 1 or 0, 1 means True, 0 means False

DeleteFile(string FileName)

the DeleteFile function deletes an existing file

[string existingfilepath] : file path, Filepath means entire full file path to the file including the file name, For example: “/PocketC/test.pc”

Return: Return 1 or 0 or -1, 1 means True, 0 means False, -1 means files not dot exist

GetFileAttr(string filepath)

get the file’s attribute

[string filepath] : file path

Return: File attribute.

FILE_ATTRIBUTE_READONLY 0x00000001

FILE_ATTRIBUTE_HIDDEN 0x00000002

FILE_ATTRIBUTE_SYSTEM 0x00000004

FILE_ATTRIBUTE_DIRECTORY 0x00000010

FILE_ATTRIBUTE_ARCHIVE 0x00000020

FILE_ATTRIBUTE_NORMAL 0x00000080

FILE_ATTRIBUTE_TEMPORARY 0x00000100

FILE_ATTRIBUTE_COMPRESSED 0x00000800

FILE_ATTRIBUTE_OFFLINE 0x00001000

OpenFileDialog(string FileFilter)

Use to initialize an Open or Save As common dialog box, After the user closes the dialog box, the system returns information about the user’s selection, In our case, the selected file path, If user hits cancel, the return string has string len = 0, if you don’t want to add any fill filter, just call OpenFileDialog(“”)

[string text] : a short string, The first string in each pair is a display string that describes the filter (for example, “Text Files”), and the second string specifies the filter pattern (for example, “*.TXT”), To specify multiple filter patterns for a single display string, use a semicolon to separate the patterns (for example, “*.TXT;*.DOC;*.BAK”), A pattern string can be a combination of valid filename characters and the asterisk (*) wildcard character, Do not include spaces in the pattern string, Filter strings are separated with character ‘|’, Filter Format Example: “PocketC Source (*.pc)*.pc” Make sure it has a semicolon at the end, The operating system does not change the order of the filters, It displays them in the File Types combobox in the order specified in FileFilter, “

Return: return filepath user just selected

SaveFileDialog(string FileFilter)

Use to initialize an Open or Save As common dialog box, After the user closes the dialog box, the system returns information about the user’s selection, In our case, the selected file path, If user hits cancel, the return string has string len = 0, if you don’t want to add any fill filter, just call SaveFileDialog(“”)

[string text] : a short string, The first string in each pair is a display string that describes the filter (for example, “Text Files”), and the second string specifies the filter pattern (for example, “*.TXT”), To specify multiple filter patterns for a single display string, use a semicolon to separate the patterns (for example, “*.TXT;*.DOC;*.BAK”), A pattern string can be a combination of valid filename characters and the asterisk(*) wildcard character, Do not include spaces in the pattern string, Filter strings are separated with character ‘|’, Filter Format Example: “PocketC Source (*.pc)*.pc” Make sure it has a semicolon at the end, The operating system does not change the order of the filters, It displays them in the File Types combo box in the order specified in FileFilter,

Return: return filepath user just selected

writebytes(int handle, string s)

writes a string to the output stream. For every character in the string s, taken in order, one byte is written to the output stream

[int handle] : file or serial connection stream

[string s] : output string

Return: None

writechar(int handle, int v)

writes a char value, which is comprised of two bytes.

[int handle] : output handle for file or serial connection

[int v] : value

Return: None

writetext(int handle, string s)

writes every character in the string s, to the output stream, in order, two bytes per character

[int handle] : output handle for file or serial connections

[string s] : output string

Return: None

writeint(int handle, int v)

writes an int value, which is comprised of four bytes, to the output stream

[int handle] : output handle for file or serial connections

[int v] : value

Return: None

GUI

wndmove(int handle, int x, int y, int width, int height)

wndshow(int handle, int flag)

menudel(int handle, int flag, int id)

menuins(int handle, int id_after, int flag, int id_new, string name)

menu_on()

menu_off()

lbrstcnt(int nID)

lbdelstr(int nID, int nIndex)

lbgetcnt(int nID)

lbgetcur(int nID)

lbsetcur(int nID, int nIndex)

lbgettxt(int nID, int nIndex)

lbfndstr(int nID, string str)

sendmsg(int nID, int nMsg, anytype wparam, anytype lparam)

createctrl(string strCtrl, string strName, int nStyle, int nExStyle, int xpos, int ypos, int width, int height, int nID)

cbaddstr(int nID, string str)

cbinsstr(int nID, int nIndex, string str)

cbrstcnt(int nID)

cbdelsr(int nID, int nIndex)

cbgetcnt(int nID)

cbgetcur(int nID)

cbsetcur(int nID, int nIndex)

cbgettxt(int nID, int nIndex)

cbfndstr(int nID, int findindex string str)

cbselstr(int nID, int nStart, string str)

guisetfocus(int guid)

guietfocus()

delgui(int id)
delallgui(int from, int to)
cbxset(int id, int state)
cbxget(int id)
editget(int id)
editset(int id, string str)
lbaddstr(int nID, string str)
lbinsstr(int nID, int nIndex, string str)
menupop(string strmenu)
menupopex(string strmenu, int x, int y)

wndmove(int handle, int x, int y, int width, int height)

Move a control or window to a new position/size. Units is in pixel.

[int handle] : window/control identifier

[int x] : position x

[int y] : y position

[int width] : width

[int height] : height

Return: If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

wndshow(int handle, int flag)

Show/hide a control on screen. Remark* the flag value can be many other values. In most cases, SW_SHOW and SW_HIDE would be enough. If you care to investigate all possible values, Microsoft documentation on Win32 API programming would be the only way to go.

[int handle] : window identifier

[int flag] : indicates how the window state. SW_SHOW = 5 SW_HIDE = 0

Return: If the window was previously visible, the return value is nonzero. If the window was previously hidden, the return value is zero.

menudel(int handle, int flag, int id)

delete a menu item from the menu bar. You can delete a menu by its position or its id.

[int handle] : the main menu handle, must be 0 right now.

[int flag] : menu flag. MF_BYCOMMAND or MF_BYPOSITION.

[int id] : menu id or position depend on the flag.

Return: none

menuins(int handle, int id_after, int flag, int id_new, string name)

insert a menu into current existing menu bar. Right now you can only insert menu under Program menu.

[int handle] : the main menu handle, right now it must be 0.

[int id_after] : the new menu is inserted behind the menu with id_after.

[int flag] : menu flag. See control documentation.

[int id_new] : the new menu item's id.

[string name] : the new menu name

Return: none

menu_on()

Show menu bar

Return: None

menu_off()

Hide menu bar

Return: None

lbrstent(int nID)

Reset the contents of a list box

[int nID] : user control ID

Return: None

lbdelstr(int nID, int nIndex)

Delete a string from the list box

[int nID] : user control ID

[int nIndex] : the zero-based index of the string to be deleted,

Return: A count of the strings remaining in the list, The return value is LB_ERR -1 if nIndex specifies an index greater than the number of items in the list,

lbgetcnt(int nID)

returns the number of strings in a list box,

[int nID] : user control ID

Return: number of strings in a listbox

lbgetcur(int nID)

get current selection in a listbox

[int nID] : user control ID

Return: return current selection string's zero-based index value, LB_ERR -1 if none is selected,

lbsetcur(int nID, int nIndex)

set current selection in a listbox

[int nID] : user control ID

Return: return LB_ERR -1 is an error occurred,

lbgettxt(int nID, int nIndex)

Get a string from a list box

[int nID] : user control ID

[int nIndex] : the zero-based index of the string to be deleted,

Return: return the string is at zero-based index location,

lbfindstr(int nID, string str)

Find a string from a list box

[int nID] : user control ID

[int findindex] : Zero-based index of the item before the first item to be searched. When the search reaches the bottom of the list box, it continues from the top of the list box back to the item specified by the indexStart parameter. If indexStart is -1, the entire list box is searched from the beginning

[string str] : a string

Return: return the founded string's zero-based index location, LB_ERR -1 if none is founded,

sendmsg(int nID, int nMsg, anytype wparam, anytype lparam)

Sendmsg sends a specific message to a window control, and it will not return until the message has been processed.

[int nID] : window control ID

[int nMsg] : window control message

[anytype wparam] : message dependent value

[anytype lparam] : message dependent value

Return: return value is depend on the specific message is sent

createctrl(string strCtrl, string strName, int nStyle, int nExStyle, int xpos, int ypos, int width, int height, int nID)

Create a window control. All of that work, you just created one control. If you want to create user interface quickly, Visual Form Buddy or PCForms would be make UI design much easier.

[string strCtrl] : Following control names are acceptable. "BUTTON" "COMBOBOX" "EDIT" "LISTBOX" "SCROLLBAR" "STATIC"

[string strName] : the control title.

[int nStyle] : each control has its own specific styles. For each control, consult the control documentat

[int nExStyle] : Extra styles.

[int xpos] : x position

[int ypos] : y position

[int width] : control width

[int height] : control height
[int nID] : control ID.
Return: -1 indicates failed. Otherwise, it returns the windows ID.

cbaddstr(int nID, string str)

Add a string to the combobox
[int nID] : user control ID
[string str] : a short string
Return: zero based index value of the added string in the combobox

cbinsstr(int nID, int nIndex, string str)

Insert a string to the combobox at a specific position
[int nID] : user control ID
[int nIndex] : zero based index value
[string str] : a short string
Return: zero based index value of the inserted string in the combobox

cbrstent(int nID)

Reset the contents of a combobox
[int nID] : user control ID
Return: None

cbdelstr(int nID, int nIndex)

Delete a string from the combobox
[int nID] : user control ID
[int nIndex] : the zero-based index of the string to be deleted,
Return: A count of the strings remaining in the list, The return value is LB_ERR -1 if nIndex specifies an index greater than the number of items in the list,

cbgetcnt(int nID)

returns the number of strings in a combobox,
[int nID] : user control ID
Return: number of strings in a combobox

cbgetcur(int nID)

get current selection in a combobox
[int nID] : user control ID
Return: return current selection string's zero-based index value, LB_ERR -1 if none is selected,

cbsetcur(int nID, int nIndex)

set current selection in a combobox
[int nID] : user control ID
Return: return LB_ERR -1 is an error occurred,

cbgettxt(int nID, int nIndex)

Get a string from a combobox
[int nID] : user control ID
[int nIndex] : the zero-based index of the string to be deleted,
Return: return the string is at zero-based index location,

cbfindstr(int nID, int findindex string str)

Find a string from a combobox.
[int nID] : user control ID
[int findindex] : Zero-based index of the item before the first item to be searched. When the search reaches the bottom of the list box, it continues from the top of the list box back to the item specified by the indexStart parameter. If indexStart is -1, the entire list box is searched from the beginning
[string str] : a string
Return: return the founded string's zero-based index location, LB_ERR -1 if none is founded,

cbsselstr(int nID, int nStart, string str)

Select a string if the string is found in the combobox

[int nID] : user control ID

[int nStart] : Specifies the zero-based index of the item preceding the first item to be searched. When the search reaches the bottom of the list, it continues from the top of the list back to the item specified by the indexStart parameter. If indexStart is -1, the entire list is searched from the beginning.

[string str] : a short string

Return: return the founded string's zero-based index location, LB_ERR -1 if none is founded.

guisetfocus(int guiid)

Set input focus to that GUI control.

[int nID] : user control ID

Return: Returns the previous window control's ID. We reserved 0 for the main window, 1 for console window

guigetfocus()

Check which GUI control has the input focus right now,

Return: returns GUI identifier

delgui(int id)

Delete User Interface control from the window

[int nID] : user control ID

Return: None

delallgui(int from, int to)

Delete gui control from the window with ID in range less than TO and greater than FROM

[int nIDfrom] : user control ID starting value

[int nIDto] : user control ID end value

Return: None

cbxset(int id, int state)

Set check box's state,

[int nID] : user control ID

[int nState] : checkbox state 1 means checked, 0 means unchecked

Return: None

cbxget(int id)

Get check box's current state

[int nID] : user control ID

Return: return check box's state, 1 means checked, 0 means unchecked

editget(int id)

Get current editbox's string

[int nID] : user control ID

Return: return edit box's string value

editset(int id, string str)

Set edit box's value by giving the gui's id and the string value

[int nID] : user control ID

[string str] : a short string

Return: None

lbaddstr(int nID, string str)

Add a string to the listbox

[int nID] : user control ID

[string str] : a short string

Return: zero based index value of the added string in the listbox

lbinsstr(int nID, int nIndex, string str)

Insert a string to the listbox at a specific position

[int nID] : user control ID

[int nIndex] : zero based index value

[string str] : a short string

Return: zero based index value of the inserted string in the listbox

menupop(string strmenu)

Create a popup menu on screen where the pen tabs. The input is a very simple string that presents the entire menu item list.

[string strmenu] : For each menu item, the string representation is "MenuName|MenuID". The name and the ID is separated by character '|'. A menu separator is "--|0". For example: "Chicago|40020|Cleveland|40021" For more, see Chapter2.

Return: return user's selection, 0 if none

menupopex(string strmenu, int x, int y)

Create a popup menu on screen where the pen tabs. The input is a very simple string that presents the entire menu item list.

[int handle] : the main menu handle, right now it must be 0.

[int id_after] : the new menu is inserted behind the menu with id_after.

[int flag] : menu flag. See control documentation.

[int id_new] : the new menu item's id.

[string name] : the new menu name

[int x] : screen horizontal position

[int y] : screen vertical position

Return: return user's selection, 0 if none

Math

log10(float x)

exp(float x)

rand()

random(int n)

floor(float x)

ceil(float x)

abs(float x)

cos(float x)

sin(float x)

tan(float x)

acos(float x)

asin(float x)

atan(float x)

cosh(float x)

sinh(float x)

tanh(float x)

pow(float x, float y)

sqrt(float x)

log(float x)

log10(float x)

Returns log base 10 of x,

[float ft] : a float value

Return: a float value

exp(float x)

Returns e

[float ft] : a float value

Return: x,

rand()

Returns a random float between 0 and 1,

Return: a float value

random(int n)

Returns a random int between 0 and n-1

[int n] : an integer

Return: a integer value

floor(float x)

Returns a floating point value representing the largest integer that is less than or equal to x, There is no error return,

[float ft] : a float value

Return: a integer value

ceil(float x)

Returns a floating point value representing the smallest integer that is greater than or equal to x, There is no error return,

[float ft] : a float value

Return: a integer value

abs(float x)

Returns an absolute value

[float ft] : a float value

Return: a integer value

cos(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

sin(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

tan(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

acos(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

asin(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

atan(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

cosh(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

sinh(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

tanh(float x)

Returns the expected trigonometric value, using radians,

[float ft] : a float value

Return: a float value

pow(float x, float y)

Returns x

[float x] : a float value

[float y] : a float value

Return: y,

sqrt(float x)

Returns square root of x,

[float ft] : a float value

Return: a float value

log(float x)

Returns natural log of x,

[float ft] : a float value

Return: a float value

Registry

```
regwint(int root,string keypath,string keyname, int ival)
regwstr(int root,string keypath,string keyname, string sval)
regdelkey(int root, string keypath)
regdelval(int root, string keypath)
regrint(int root,string keypath,string keyname)
regrstr(int root,string keypath,string keyname)
```

regwint(int root,string keypath,string keyname, int ival)

Write integer to registry database,

[int root] : root is defined as following define statement, {define HKEY_CLASSES_ROOT 0, define HKEY_CURRENT_USER 1, define HKEY_LOCAL_MACHINE 2 , define HKEY_USERS 3}

[string keypath] : keypath is defined with double backslash

Sample: regstr(0,"pcapp\\Shell\\Open\\Command","default"); ,

[string keyname] : keyname,

[int value] : key value

Return: return 0 for success, Otherwise fail,

regwstr(int root,string keypath,string keyname, string sval)

Write string to registry database,

[int root] : root is defined as following define statement, {define HKEY_CLASSES_ROOT 0, define HKEY_CURRENT_USER 1, define HKEY_LOCAL_MACHINE 2 , define HKEY_USERS 3}

[string keypath] : keypath is defined with double backslash

Sample: `regstr(0,"pcapp\\Shell\\Open\\Command","default");` ,
[string keyname] : keyname,
[string str] : key value
Return: return 0 for success, Otherwise fail,

regdelkey(int root, string keypath)

Function deletes a subkey

[int root] : root is defined as following HKEY_CLASSES_ROOT 0, HKEY_CURRENT_USER 1, HKEY_LOCAL_MACHINE 2 , HKEY_USERS 3
[string keypath] : SubKey string
Return: return 0 if it succeed, otherwise fail.

regdelval(int root, string keypath)

Removes a named value from the specified registry key.

[int root] : root is defined as following HKEY_CLASSES_ROOT 0, HKEY_CURRENT_USER 1, HKEY_LOCAL_MACHINE 2 , HKEY_USERS 3}
[string ValueName] : string that names the value to remove
Return: return 0 if it succeed, otherwise fail.

regrint(int root,string keypath,string keyname)

Read integer from registry database

[int root] : root is defined as following define statement, {define HKEY_CLASSES_ROOT 0, define HKEY_CURRENT_USER 1, define HKEY_LOCAL_MACHINE 2 , define HKEY_USERS 3}
[string keypath] : keypath is defined with double backslash
Sample: `regstr(0,"pcapp\\Shell\\Open\\Command","default");` ,
[string keyname] : keyname
Return: integer

regstr(int root,string keypath,string keyname)

Read string from registry database

[int root] : root is defined as following define statement, {define HKEY_CLASSES_ROOT 0, define HKEY_CURRENT_USER 1, define HKEY_LOCAL_MACHINE 2 , define HKEY_USERS 3}
[string keypath] : keypath is defined with double backslash
Sample: `regstr(0,"pcapp\\Shell\\Open\\Command","default");` ,
[string keyname] : keyname
Return: string

Serial

```
commclrerr(int handle, void v)
commsetbrk(int handle)
commclrbrk(int handle)
commesc(int handle, int ext)
commgetmask(int handle)
commsetmask(int handle,int mask)
commwait(int handle, int nblock)
rawiopen(int baud,string settings,int timeout)
seropen(int baud, string settings, int timeout)
OBSOLETE serdata()
OBSOLETE serrecv()
OBSOLETE sersend(char byte)
OBSOLETE serclose()
seropenex(string comport, int baud, string settings, int timeout)
```

commclerrr(int handle, void v)

this function retrieves information about a communications error and reports the current status of a communications device. The function is called when a communications error occurs, and it clears the error flag of the device to enable additional input and output (I/O) operations.

[int handle] : Handle to the communications device

[int void] : reserved. Must use 0 for now.

Return: A 32-bit variable to be filled with a mask indicating the type of error. This parameter can be one or more of the following error codes:

CE_BREAK–The hardware detected a break condition.

CE_FRAME The hardware detected a framing error.

CE_IOE An I/O error occurred during communications with the device.

CE_MODE The requested mode is not supported, or the hFile parameter is invalid. If this value is specified, it is the only valid error.

CE_OVERRUN A character-buffer overrun has occurred. The next character is lost.

CE_RXOVER An input buffer overflow has occurred. There is either no room in the input buffer, or a character was received after the end-of-file (EOF) character.

CE_RXPARITY The hardware detected a parity error.

CE_TXFULL The application tried to transmit a character, but the output buffer was full.

commsetbrk(int handle)

this function suspends character transmission for a specified communications device and places the transmission line in a break state until the commclbrk function is called.

[int handle] : Handle to the communications device

Return: Nonzero indicates success. Zero indicates failure

commclbrk(int handle)

this function restores character transmission for a specified communications device and places the transmission line in a nonbreak state

[int handle] : serial communication handle

Return: Nonzero indicates success. Zero indicates failure

commesc(int handle, int ext)

this function directs a specified communications device to perform an extended function.

[int handle] : communication handle

[int ext] : Specifies the code of the extended function to perform. It can be one of the following values:

SETIR –Sets the serial port in infrared (IR) mode.

CLRIR–Sets port to normal serial mode.

CLRDTTR–Clears the DTR (data-terminal-ready) signal.

CLRRTS–Clears the RTS (request-to-send) signal.

SETDTTR–Sends the DTR (data-terminal-ready) signal.

SETRTS–Sends the RTS (request-to-send) signal.

SETXOFF–Causes transmission to act as if an XOFF character has been received.

SETXON–Causes transmission to act as if an XON character has been received.

SETBREAK – Suspends character transmission and places the transmission line in a break state until the commclbrk function is called (or commesc is called with the CLRBREAK extended function code). The SETBREAK extended function code is identical to the commsetbrk function. Note that this extended function does not flush data that has not been transmitted.

CLRBREAK–Restores character transmission and places the transmission line in a nonbreak state. The CLRBREAK extended function code is identical to the commclbrk function.

Return: Nonzero indicates success. Zero indicates failure. To get extended error information

commgetmask(int handle)

this function retrieves the value of the event mask for a specified communications device. The commgetmask function uses a 32-bit mask variable to indicate the set of events that can be monitored for a particular communications resource. A handle to the communications resource can be specified in a call to the commwait function, which waits for one of the events to occur. To modify the event mask of a communications resource, use the commsetmask function.

[int handle] : serial connection handle

Return: The return value can be a combination of the following values:

EV_BREAK – a break was detected on input

EV_CTS – The CTS (clear-to-send) signal changed state

EV_DSR – The DSR (data-set-ready) signal changed state

EV_ERR – a line-status error occurred. Line-status errors are CE_FRAME, CE_OVERRUN, and CE_RXPARITY

EV_RING – a ring indicator was detected

EV_RLSD – the RLSD(receive-line-signal-detect) signal changed state.

EV_RXCHAR – a character was received and placed in the input buffer

EV_TXEMPTY – The last character in the output buffer was sent.

commsetmask(int handle,int mask)

Specifies a set of events to monitor for a communications device

[int handle] : serial connection handle

[int mask] : specifies the events to be enabled. A value of 0 disable all events.

[int mask] : This parameter can be a combination of the following values:

EV_BREAK – a break was detected on input

EV_CTS – The CTS (clear-to-send) signal changed state

EV_DSR – The DSR (data-set-ready) signal changed state

EV_ERR – a line-status error occurred. Line-status errors are CE_FRAME, CE_OVERRUN, and CE_RXPARITY

EV_RING – a ring indicator was detected

EV_RLSD – the RLSD(receive-line-signal-detect) signal changed state.

EV_RXCHAR – a character was received and placed in the input buffer

EV_TXEMPTY – The last character in the output buffer was sent.

Return: Nonzero indicates success. Zero indicates failure

commwait(int handle, int nblock)

Waits for the data. If nblock is 0, it returns immediately. When the data is in the buffer, the main event will receive an event message PM_COMM (12). If nblock is 1, the function will not return until there is data in the buffer.

[int handle] : connection handle

[int nblock] : value 0 or 1

Return: None

rawiropen(int baud,string settings,int timeout)

Open a serial port for infraRed. It searches through user's registry, and use correct port for infraRed transfer.

[int baud] : integer value represents the speed of the connection, (300-56000),

[string settings] : a 4 characters string represents the connection setting, { bits/char (6,7,8), parity (N,E,O), stop bits (1,2) , flow control (N, NoFlowControl X-software, C-CTS, R-RTS) }

[int timeout] : integer value for timeout, It is ignored at this point, Description: ""open a serial port, *Note: Port number is default to ""COM:1:"" for simplicity"

Return: 1 means for success, 0 means fail

seropen(int baud, string settings, int timeout)

Open a serial port, *Note: Port number is default to "COM:1:" for simplicity, Due to user requests, an addition function has been added for other port accesses, Please use seropenex if you have multiple ports

[int baud] : integer value represents the speed of the connection, (300-56000),

[string settings] : a 4 characters string represents the connection setting, { bits/char (6,7,8), parity (N,E,O), stop bits (1,2) , flow control (N, NoFlowControl X-software, C-CTS, R-RTS) }

[int timeout] : integer value for timeout, It is ignored at this point, Description: "open a serial port, *Note: Port number is default to "COM:1:" for simplicity

Return: 0 means fail, any positive return value means successful.

OBSOLETE serdata()

Check if there is data available in the buffer

Return: return 1 if data is waiting, 0 otherwise,

OBSOLETE serrecv()

Receive a byte, right now the data type is a character,
Return: return a character value

OBSOLETE sersend(char byte)

Send a byte,
[char byte] : a character to be send
Return: None

OBSOLETE serclose()

Close serial port,
Return: None

seropenex(string comport, int baud, string settings, int timeout)

Open the serial port, *Note: for Infrared Port: please obtain the port number from the registry key \\HKEY_LOCAL_MACHINE\\COMM\\IrDA\\Port, The Port number may not be the same on all devices.

[string strPort] : "COM1:" "COM2:" "COM3:" or any port you want to use,
[int baud] : integer value represents the speed of the connection, (300-56000),
[string settings] : a 4 characters string represents the connection setting, { bits/char (6,7,8), parity (N,E,O), stop bits (1,2) , flow control (N, NoFlowControl X-software, C-CTS, R-RTS) }

[int timeout] : integer value for timeout, It is ignored at this point
Return: 1 means for success, 0 means fail

System

runapptime(string path, int month, int day, int hour, int min)

exec(string action, string path, string parameter)

recentdoc(string path)

shortcut(string lnkpath, string filepath)

debugout(string debugstring)

getcurpath()

graph_on()

MessageBox(string text, string caption, int icon, int buttons)

about(string text)

confirm(string text)

alert(string text)

title(string text)

sleep(int len)

loadres(int resourceid)

showhelp(string filepath)

quit()

showabout()

graph_off()

runapptime(string path, int month, int day, int hour, int min)

Run application at specific time in the future, Parameter path is the application location,

For example: "\\pocketc\\pceditor.exe" or "/pocketc/pceditor.exe" *Note: current implementation is just a simple translation of the actual Win32 shell function,

[string filepath] : file path to the executable,

[int month] : month (1-12)

[int day] : day

[int hour] : hour, (0-24)

[int min] : minutes, (0-60)

Return: None

exec(string action, string path, string parameter)

Execute a file or an application from PocketC applets

[string action] : action, It can be “open”

[string filepath] : file path to the executable

[string parameter] : executable’s parameter

Return: None

recentdoc(string path)

Add document to recent document menu, Parameter path, pathname for the document,

For example: recentdoc(“\\windows\\alarm2,wav”);

[string filepath] : full file path to the file

Return: None

shortcut(string lnkpath,string filepath)

Add a shortcut to executable or file,

For example: shortcut(“\\windows\\Programs\\PocketC,lnk”,

“\\pocketc\\pceditor,exe”); shortcut(“\\windows\\Desktop\\PocketC,lnk”, “\\pocketc\\pceditor,exe”);

[string lnkpath] : shortcut file location,

[string filepath] : full file path to your target file or execute

Return: None

debugout(string debugstring)

Send a string through OutputDebugString (win32 function) for debug purpose only, you can catch these strings by using HPC trace or other debug utility that monitor OutputDebugString,

[string debugstring] : debug message to be caught by debug utilities that monitor OutputDebugString functions,

Return: None

getcurpath()

Obtain current PocketC applet’s file location

Return: return a string

graph_on()

Switches to the graphics window

Return: None

MessageBox(string text,string caption,int icon,int buttons)

Buttons style can be 1= OKCANCLE, 2=YESNO, 3=YESNOCANCEL.

[string text] : a short string

[string caption] : Messagebox’s caption

[int icon] : Icon identifier, Information 1, Asterisk 2, and Hand 3,

[int buttons] : button layout identifier,

Return: Return value -1 means Cancel, 0 means NO, 1 means Yes,

about(string text)

Set about box’s contents, example string: “MyAppName\r\nMyName”

[string text] : a short string

Return: None

confirm(string text)

Pops up an alert dialog with the given text and Yes/No buttons,

[string text] : a short string

Return: Returns 1 for Yes, 0 for No,

alert(string text)

Pops up an alert dialog with the given text,

[string text] : a short string

Return: None

title(string text)

Give your applet a name! So it will appear on the taskbar on the bottom. It will also keep your application single instance. When a new PocketC application launches, it will check if any existing same application running. If it is, it will terminate itself, and switch to the running application. It is important feature to have for PPC users.

[string text] : a short string

Return: None

sleep(int len)

a thread can relinquish the remainder of its time slice by calling this function with a sleep time of zero milliseconds
[int len] : : Specifies the time, in milliseconds, for which to suspend execution. A value of zero causes the thread to relinquish the remainder of its time slice to any other thread of equal priority that is ready to run. If there are no other threads of equal priority ready to run, the function returns immediately, and the thread continues execution. A value of (-1) INFINITE causes an infinite delay

Return: None.

loadres(int resourceid)

first, define resource by using preprocessor #resource. The format is #resource id "valid file path". During runtime use function loadres(int resourceid) to obtain the resource file path. The resource file is compiled with the application

[int resourceid] : Resource ID.

Return: valid file path to the resource file

showhelp(string filepath)

launch help file

[string filepath] : a valid filepath to a help file.

Return: 1 for success, 0 for fail.

quit()

Exit your applet gracefully,

Return: None

showabout()

Popup about window.

Return: None

graph_off()

Popup console window, the appearance of the graphics is not preserved if keepscreen_off() is called,

Return: None

Sound

beep(int soundtype)**wave(string wavfile,int flag)****beep(int soundtype)**

Generates a system sound,

[int soundtype] : between 0 and 3, Available sounds are default 0, asterisk 1, exclamation 2, hand 3

Return: None

wave(string wavfile,int flag)

Play a wave file, SND_ASYNC The sound is played asynchronously and sndPlaySound returns immediately after beginning the sound, To terminate an asynchronously played sound, call wave with flag set to -1, SND_LOOP The sound plays repeatedly until wave is called again with the flag parameter set to -1, SND_NOSTOP If a sound is currently playing, wave immediately returns FALSE, without playing the requested sound, SND_SYNC The sound is played synchronously and wave does not return until the sound end

[string wavfile] : full filepath to a wavfile
[int flag] : integer flag value for how to play the wave file SND_SYNC 0, SND_ASYNC 1,SND_LOOP 2,SND_NOSTOP 3,SND_STOP -1
Return: None

String

strgetc(string str, int index)
string strsetc(string str,int index, char ch)
strlen(string str)
strmid(string str, int first, int len)
strleft(string str, int len)
strright(string str, int len)
strupr(string str)
strrvrs(string)
strspn(string src,string charset)
strchr(string,char)
strrchr(string str,char ch)
strlwr(string str)
hex(int ival)
format(float fval, int precision)
islower(char ch)
isupper(char ch)
isalphanum(char ch)
isalpha(char ch)
strncmp(string1,string2,int count)

strgetc(string str, int index)

Get a character at index position from a string, index starts at 0, For example: strgetc(“Hello”, 4); it will return character ‘o’;

[string str] : a short string ,
[int index] : zero based index value
Return: Returns a character,

string strsetc(string str,int index, char ch)

Returns a string value with modified character at index position,

For example: string retstr; retstr = strsetc(“Hello”,4,’e’); retstr will have string value “Helle”,

[string str] : a short string ,
[int index] : zero based index value ,
[int ch] : a character
Return: Returns a modified string

strlen(string str)

Returns the length of a string,

[string str] : a short string
Return: return string length

strmid(string str, int first, int len)

Returns a string which consists of len characters from the original string starting at first character, (e.g, strmid(“Hello”, 1, 3) returns “ell”)

[string str] : a string
[int first] : start index position
[int len] : length
Return: return a substring

strleft(string str, int len)

Returns the len leftmost characters from the string,

[string str] : a short string

[int len] : integer length

Return: return a substring

strright(string str, int len)

Returns the len rightmost characters from the string,

[string str] : a short string

[int len] : integer length

Return: return a substring

strupr(string str)

Returns the original string in all uppercase,

[string str] : a short string

Return: return an uppercased string

strrvrs(string)

Reverse string order

[string str] : a short string

Return: returns the input string in reverse order

strspn(string src,string charset)

find a substring in a string, Returns an integer value specifying the length of the initial segment of string that consists entirely of characters not in strCharSet,

[string str] : a short string

[string chset] : a set of character to search for,

Return: If string begins with a character that is in strCharSet, the function returns 0, No return value is reserved to indicate an error,

strchr(string,char)

scan a string for the first occurrence of a character,

[string str] : a short string

[char ch] : character to search for

Return: return -1 if not found, otherwise char index value in the string

strrchr(string str,char ch)

scan a string for the last occurrence of a character,

[string str] : a short string

[char ch] : character to search for

Return: return -1 if not found, otherwise char index value in the string

strlwr(string str)

Returns the original string in all lowercase,

[string str] : a short string

Return: return an lowercased string

hex(int ival)

Convert an integer to a hex representations string

[int iVal] : an integer

Return: return value in hex string

format(float fval, int precision)

Format a float value to a string

[float fval] : a float value

[int precision] : decimal places.

Return: return a string

islower(char ch)

Routines returns true if c is lower case

[char ch] : a character

Return: return 0 means false, 1 means true

isupper(char ch)

Routines returns true if c is upper case

[char ch] : a character

Return: return 0 means false, 1 means true

isalnum(char ch)

Routines returns true if c is a particular representation of an alphabetic or numeric character

[char ch] : a character

Return: return nonzero means it is a alphabetic or numeric character, 0 means it is not,

isalpha(char ch)

Routines returns true if c is a particular representation of an alphabetic character

[char ch] : a character

Return: return nonzero means it is a alphabetic character, 0 means it is not,

strncmp(string1,string2,int count)

compare characters of two strings at first count characters, string1, string2 Strings to compare, count Number of characters to compare,

[string str1] : a short string

[string str2] : a short string

[int count] : number of characters to compare

Return: return 0 means equal, -1 means str1 less than str2, 1 means str1 greater than str2

Time

gethour()**getday()****getmonth()****getyear()****ticks()****getsec()****getmin()****gethour()**

Return current hour

Return: return current hour

getday()

Return current day

Return: return current day

getmonth()

Return current month number

Return: return current month number

getyear()

Return current year number, ex: 1998

Return: return current year number, ex: 1998

ticks()

The number of clock ticks since last reset

Return: return the number of clock ticks since last reset

getsec()

Return current second

Return: return current second

getmin()

Return current minute

Return: return current minute

Database

```

dbenum(int bFirst, int dbtype)
dbname(int oid)
dbsize(int oid)
dbnrecs(int oid)
dbrecpropcnt()
dbrecproptype(int index)
dbrecpropval(int index)
dbseek(int seektype, int dwValue)
dbrecwrite(int rec_oid, int prop_type, int prop_id, void prop_value)
dbrecread(int proptype, int propid)
dbrecdelprop(int recoid, int proptype, int propid)
dbrecdel(int recoid)
dbrecnow()
dbdelete(int dboid)
dbcreate(string name, int dbtype)
dbopen(int oid, string name)
dbclose()

```

dbenum(int bFirst, int dbtype)

enumerate all the databases.

[int bFirst] : Use 1 to get the first database. After that, just keep using 0 until dbenum returns 0.

[int dbtype] : use 0 if you want to see all database. If you want to see one type of database, use that type's id.

Return: dbenum returns one database object id each time it is called. Once all of the databases are enumerated, dbenum returns 0. You will need the object id to find out more about the database.

dbname(int oid)

find out database name base on its OID

[int oid] : Object ID

Return: database name as a string

dbsize(int oid)

find out database physical storage size in bytes.

[int oid] : database object ID

Return: returns database physical storage size in bytes. If failed, returns 0

dbnrecs(int oid)

find out database records count.

[int oid] : Database Object ID

Return: return total records in the database. If failed, return 0

dbrecpropcnt()

find out the number of properties in the current record.

Return: returns number of properties in the current record. If failed, return -1.

dbrecproptype(int index)

each property has a type. CEVT_I2 A 16-bit signed integer. CEVT_I4 A 32-bit signed integer. CEVT_LPWSTR A null-terminated string. CEVT_UI2 A 16-bit unsigned integer. CEVT_UI4 A 32-bit unsigned integer.

[int index] : index position of one property

Return: returns the property type by using its index

dbrecpropval(int index)

find out property value by using its index

[int index] : index position for the property

Return: possible return type is int and string for the property value

dbseek(int seektype, int dwValue)

Seek the record depends on the seek type and seek value

[int seektype] : determine the type of seek operation

[int dwValue] : input value depend on the seek operation.

Return: Returns record object id. If failed, return 0.

dbrecwrite(int rec_oid, int prop_type, int prop_id, void prop_value)

you can obtain the rec_oid by using dbrecnow(). If rec_oid is 0, a new record will be written.

[int rec_oid] : Each record in the database has its own object identifier.

[int prop_type] : each property has a property value.

[int prop_id] : each property can have its own identification. This id is simply an positive integer value which can be decided by the developer.

[void prop_value] : depends on the type you supplied in prop_type. If prop_type is a CEVT_I4 a 32 bit signed integer, you better have an integer for prop_value.

Return: an integer which is record object id for the record you just wrote in.

dbrecread(int proptype, int propid)

read one specific property from the database

[int proptype] : property type

[int propid] : property object id

Return: return the value for the property. You can cast the return valuet into the type of value you wanted.

dbrecdelprop(int recoid, int proptype, int propid)

delete one property from the record.

[int recoid] : record object id

[int proptype] : record property type

[int propid] : the property ID.

Return: returns record id if successful. If function failed, return 0.

dbrecdel(int recoid)

delete one record

[int recoid] : record object ID

Return: 1 indicates success. 0 indicates failure. The function could fail when you pass into a wrong or invalid record object id

dbrecnow()

get current record object id. When you are iterating through the database, you can obtain the current record object id by call dbrecnow.

Return: 0 indicates failure. The function should return the record object id for the current reading/writing record

dbdelete(int dboid)

Delete one database

[int dboid] : database object id

Return: 1 indicates success. 0 indicates failure. The function could fail when you pass into a wrong or invalid database object id or another program is using the database at this moment.

dbcreate(string name, int dbtype)

create a new database. For detail information check out detailed database documentation online

[string name] : database name

[int dbtype] : database type. The database type is up to developer to define.

Return: if dbcreate is successful, it returns an integer. This integer is the object identifier for your database. It can be used to open the database you just created. see dbopen for detail.

dbopen(int oid, string name)

you can specify either object id or name to open the database. If you know the function name only, just use 0 for the object ID. If you know the function object ID only, just use an empty string "" for the database name parameter.

[int oid] : Object ID which identify one database. Database Object ID: An integer which identify the database. It can be obtained by using dbcreate or dbenum.

[string name] : database name.: The name can be up to 32 characters long. If you pass a longer name, the first 32 characters will be treated as the database name. All databases will be placed in the \Database folder.

Return: If the database is opened, the return value is the object id of this database. Otherwise, the return value is 0.

dbclose()

close database

Return: 1 means the database is closed. 0 means the function failed to close the database

Memory

free(pointer ptr)

settype(pointer ptr, int size, char type)

typeof(pointer ptr)

memcpy(pointer dest, pointer src, int size)

malloc(int size)

free(pointer ptr)

Releases the memory of the block pointed to by ptr which was previously allocated by malloc(). When the program exits, all blocks that are still allocated will automatically be freed to prevent memory leaks.

[pointer ptr] : memory pointer obtained by using malloc(int size) function.

Return: None.

settype(pointer ptr, int size, char type)

Set the memory block variable type. Use this function only on memory allocated by malloc().

[pointer ptr] : pointer to memory allocated by malloc(int size) function.

[int size] : memory size.

[char type] : the type of the variable. ('i' for int, 'f' for float, 'c' for char, 's' for string)

Return: Returns 0 if an error occurred

typeof(pointer ptr)

Find out the type of dynamic allocated variable's type

[pointer ptr] : pointer to memory allocated by malloc(int size)

Return: returns the type of the value pointed to by ptr, ('i' for int, 'c' for char, 'f' for float, 's' for string, 'o' for other)

memcpy(pointer dest, pointer src, int size)

Copies the data from the block of size size pointed to by src to the block pointed to by dest. The types of the destination values is not preserved. (i.e. if dest points to a string, and src points to an int, the memory pointed to by dest will be changed to an int)..

[pointer desk] : destination memory pointer

[pointer src] : source memory pointer

[int size] : memory size that needs to be copied.

Return: None

malloc(int size)

allocates a block of size values of type [int], but of undefined value.

[int size] : size of memory block

Return: Returns a pointer to the block or 0 on failure.

New Table of Contents

PocketC Help Central	1
Written by Jeremy Dewey and Kevin Cao	1
Concept	1
Step 1. Write Program.	1
Step 2. Compile and Build	1
Step 3. Run Program	2
Step 4. Distribute Program	2
Features	2
Some program examples	2
Generate Random Numbers	2
File enumeration.	3
File access	3
Window control focus.	4
Database enumeration	4
Units converter	5
Language Introduction	8
PocketC Data Type and Variables	8
Identifier	8
Variables	8
Local Variables	9
Global Variables.	9
Pointers	9
Bitwise Operators	10
Increment / Decrement	11
Automatic Casting	11
Expressions.	12
Statements	13
Include	16
Special Characters	17
Pointers and Arrays	17
Pointers and arrays	18
Pointer arithmetic	19
Databases.	19
Introduction	19
Create a database	19
Create a database	19
Open a database	19
Close Database	20
Enumerate database	20
Enumerate Database	20
Database Attributes.	20
Read database contents.	21
Write data to a database	23
Conclusion	25
Appendix – PocketC Database API	26
Funkcje API - opis	27
Console.	27
Draw	27
Event	32
File	34
GUI.	39
Math	44
Registry	46
Serial	47
System	50
Sound.	52

String	53
Time	55
Database	56
Memory	58