PPL

Pocket Programming Language
for the PocketPC and PC.

Written by Alain Deschenes
© ArianeSoft Inc.

User Agreement

1. SOFTWARE LICENSE

(a) License Grant. Upon your acceptance of this Software License Agreement ArianeSoft grants you a non-exclusive, non-transferable (except as provided below), limited license to install and use a copy of the Software on your compatible computer, up to the Permitted Number of computers. The Permitted Number of computers shall be delineated at such time as you elect to purchase the Software. During the evaluation period , hereinafter defined, only a single user may install and use the software on one computer.

(b) Backup and Archival Copies. You may make one backup and one archival copy of the Software, provided your backup and archival copies are not installed or used on any computer and further provided that all such copies shall bear the original and unmodified copyright, patent and other intellectual property markings that appear on or in the Software. You may not transfer the rights to a backup or archival copy unless you transfer all rights in the Software as provided under Section 3.

(c) Home Use. You, as the primary user of the computer on which the Software is installed, may also install the Software on one of your home computers. However, the Software may not be used on your home computer at the same time as the Software is being used on the primary computer.

(d) Key Codes. Prior to your purchase and as part of the registration for the thirty (30) -day evaluation period, as applicable, you will receive an evaluation key code. You will receive a purchase key code when you elect to purchase the Software. The purchase key code will enable you to activate the Software beyond the initial evaluation period. You may not relicense, reproduce or distribute any key code except with the express written permission of ArianeSoft.

(e) Title. Title to the Software is not transferred to you. Ownership of all copies of the Software and of copies made by you is vested in ArianeSoft, subject to the rights of use granted to you in this Software License Agreement. As between you and ArianeSoft, documents, files, generated program code and schemas that are authored or created by you via your utilization of the Software, in accordance with its Documentation and the terms of this Software License Agreement, are your property.

(f) Reverse Engineering. Except and to the limited extent as may be otherwise specifically provided by applicable law, you may not reverse engineer, decompile, disassemble or otherwise attempt to discover the source code, underlying ideas, underlying user interface techniques or algorithms of the Software by any means whatsoever, directly or indirectly, or disclose any of the foregoing, except to the extent you may be expressly permitted to decompile under applicable law, if it is essential to do so in order to achieve operability of the Software with another software program, and you have first requested ArianeSoft to provide the information necessary to achieve such operability and ArianeSoft has not made such information available. ArianeSoft has the right to impose reasonable conditions and to request a reasonable fee before providing such information. Any information supplied by ArianeSoft or obtained by you, as permitted hereunder, may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is substantially similar to the expression of the Software. Requests for information should be directed to the ArianeSoft Customer Support.

(g) Other Restrictions. You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software to third parties except to the limited extent set forth in Section 3. You may not copy the Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this Software License Agreement must contain the same copyright, patent and other intellectual property markings that appear on or in the Software. You may not modify, adapt or translate the Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the Software; knowingly take any action that would cause the Software to be placed in the public domain; or use the Software in any computer environment not specified in this Software License Agreement. You will comply with applicable law and ArianeSoft's instructions regarding the use of the Software. You agree to notify your employees and agents who may have access to the Software of the restrictions contained in this Software License Agreement and to ensure their compliance with these restrictions. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL INDEMNIFY AND HOLD HARMLESS ARIANESOFT FROM ANY 3RD PARTY SUIT TO THE EXTENT BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE IN YOUR USE. WITHOUT LIMITATION, THE SOFTWARE IS NOT INTENDED FOR USE IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS OR AIR TRAFFIC CONTROL EQUIPMENT, WHERE THE FAILURE OF THE SOFTWARE COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE.

(h) License Metering. ArianeSoft has a built-in license metering module that helps you to avoid any unintentional violation of this Software License Agreement. ArianeSoft may use your internal network for license metering between installed versions of the Software.

(k) Software Activation. ArianeSoft may use your internal network and internet connection for the purpose of transmitting license-related data entered by the user at the time of installation or registration to an ArianeSoft-operated license server and validating the authenticity of the license-related data in order to protect ArianeSoft against software piracy.

2. INTELLECTUAL PROPERTY RIGHTS

Acknowledgement of ArianeSoft's Rights. You acknowledge that the Software and any copies that you are authorized by ArianeSoft to make are the intellectual property of and are owned by ArianeSoft and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of ArianeSoft and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You acknowledge that ArianeSoft retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that ArianeSoft's ownership rights extend to any images, photographs, animations, videos, audio, music, text and applets incorporated into the Software and all accompanying printed materials. You will take no actions which adversely affect ArianeSoft's intellectual property rights in the Software. Trademarks shall be used in accordance with accepted trademark practice, including identification of trademark owners' names. Trademarks may only be used to identify printed output produced by the Software, and such use of any trademark does not give you any right of ownership in that trademark. PPL and ARIANESOFT are trademarks of ArianeSoft. Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, WindowsCE, Windows 95, Windows 98, Windows NT, Windows 2000 and Windows XP are trademarks of Microsoft. Except as expressly stated above, this Software License Agreement does not grant you any intellectual property rights in the

Software.

3. LIMITED TRANSFER RIGHTS

Notwithstanding the foregoing, you may transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer each of this Software License Agreement, the Software and all other software or hardware bundled or pre-installed with the Software, including all copies, updates and prior versions, and all copies of font software converted into other formats, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; (c) the receiving party secures a personalized key code from ArianeSoft; and (d) the receiving party accepts the terms and conditions of this Software License Agreement and any other terms and conditions upon which you legally purchased a license to the Software. Notwithstanding the foregoing, you may not transfer education, pre-release, or not-for-resale copies of the Software.

4. PRE-RELEASE PRODUCT ADDITIONAL TERMS

If the product you have received with this license is pre-commercial release or beta Software (Pre-release Software), then this Section applies. To the extent that any provision in this Section is in conflict with any other term or condition in this Software License Agreement, this Section shall supersede such other term(s) and condition(s) with respect to the Pre-release Software, but only to the extent necessary to resolve the conflict. You acknowledge that the Software is a pre-release version, does not represent final product from ArianeSoft, and may contain bugs, errors and other problems that could cause system or other failures and data loss. CONSEQUENTLY, THE PRE-RELEASE SOFTWARE IS PROVIDED TO YOU AS-IS, AND ARIANESOFT DISCLAIMS ANY WARRANTY OR LIABILITY OBLIGATIONS TO YOU OF ANY KIND, WHETHER EXPRESS OR IMPLIED. You acknowledge that ArianeSoft has not promised or guaranteed to you that Pre-release Software will be announced or made available to anyone in the future, that ArianeSoft has no express or implied obligation to you to announce or introduce the Pre-release Software, and that ArianeSoft may not introduce a product similar to or compatible with the Pre-release Software. Accordingly, you acknowledge that any research or development that you perform regarding the Pre-release Software or any product associated with the Pre-release Software is done entirely at your own risk. During the term of this Software License Agreement, if requested by ArianeSoft, you will provide feedback to ArianeSoft regarding testing and use of the Pre-release Software, including error or bug reports. If you have been provided the Pre-release Software pursuant to a separate written agreement, your use of the Software is governed by such agreement. You may not sublicense, lease, loan, rent, distribute or otherwise transfer the Pre-release Software. Upon receipt of a later unreleased version of the Pre-release Software or release by ArianeSoft of a publicly released commercial version of the Software, whether as a stand-alone product or as part of a larger product, you agree to return or destroy all earlier Pre-release Software received from ArianeSoft and to abide by the terms of the license agreement for any such later versions of the Pre-release Software.

5. LIMITED WARRANTY AND LIMITATION OF LIABILITY

Limited Warranty and Customer Remedies. ArianeSoft warrants to the person or entity that first purchases a license for use of the Software pursuant to the terms of this Software License Agreement that (i) the Software will perform substantially in accordance with any accompanying Documentation for a period of ninety (90) days from the date of receipt, and (ii) any support services provided by ArianeSoft shall be substantially as described in section 6 of this agreement. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the Software, if any, are limited to ninety (90) days. ArianeSoft's and its suppliers' entire liability and your exclusive remedy shall be, at ArianeSoft's option, either (i) return of the price paid, if any, or (ii) repair or replacement of the Software that does not meet ArianeSoft's Limited Warranty and which is returned to ArianeSoft with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse or misapplication. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

(a) No Other Warranties and Disclaimer. THE FOREGOING LIMITED WARRANTY AND REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ARIANESOFT OR ITS SUPPLIER'S BREACH OF WARRANTY. ARIANESOFT AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, AND FOR ANY WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT WHICH THE SAME CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO YOU IN YOUR JURISDICTION, ARIANESOFT AND ITS SUPPLIERS MAKE NO WARRANTIES, CONDITIONS, REPRESENTATIONS OR TERMS, EXPRESS OR IMPLIED, WHETHER BY STATUTE, COMMON LAW, CUSTOM, USAGE OR OTHERWISE AS TO ANY OTHER MATTERS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ARIANESOFT AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, INFORMATIONAL CONTENT OR ACCURACY, QUIET ENJOYMENT, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

(b) Limitation Of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ARIANESOFT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ARIANESOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ARIANESOFT'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS SOFTWARE LICENSE AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT. Because some states and jurisdictions do not allow the exclusion or limitation of liability, the above limitation may not apply to you. In such states and jurisdictions, ArianeSoft's liability shall be limited to the greatest extent permitted by law.

6. TERM AND TERMINATION
This Software License Agreement may be terminated (a) by your giving ArianeSoft written notice of termination; or (b) by ArianeSoft, at its option, giving you written notice of termination if you commit a breach of this Software License Agreement and fail to cure such breach within ten (10) days after notice from ArianeSoft. Upon any termination of this Software License Agreement, you must cease all use of the Software, destroy all copies then in your possession or control and take such other actions as ArianeSoft may reasonably request to ensure that no copies of the Software remain in your possession or control.

Last updated: 2004-03-11

Support

You can reach us via email at the following addresses. Your email will be answered within the next 48 hours.

ArianeSoft Inc.

Discussion Forums:

www.arianesoft.ca

www.pocketmatrix.com

Email:

support@arianesoft.ca

References

This document has been designed to help understand PPL and to program with it. However it is not the scope of this file to introduce you to programming neither writing Windows (tm) Interfaces. However we will give you great links to help you understand more.

**** Microsoft Developer Network (MSDN) ****

Learn the C language

What is Object-oriented

Learn the C++ language

Startup sequence & custom loading screen

When PPL starts it does a few things in the background in a specific order. Here is the list of steps PPL goes through before exiting.

```
1.      Try to execute Default.ppl
2.      Try to execute Autorun.ppl
3.      If a program was passed as an argument to PPL, run it.
4.      Try to execute main.ppl
```

Custom loading screen

When PPL programs are ran, the compiling information is usually obtained in a little window at the bottom right of your screen. However this can be customized using a special program that you can write in PPL. This PPL file must be included in the folder where the compilation will occur and must be named LOAD.PPL.

The Load.PPL program must contain a procedure named Update. This procedure will be called whenever needed by PPL.

Example:

```
proc update(Status$, Message$, Position$, Max$)
  case (status$)
    LS_INIT:
      ShowMessage("INIT "+message$);
    LS_UPDATE:
      ShowMessage("UPDATE " +message$+" "+ position$ + "," + max$);
    LS_SHUT:
      ShowMessage("SHUT "+message$);
  end;
end;

func winmain
  return (true);
end;
```

Possible messages passed to the Update procedure are:

LS_INIT: Start compiling/executing. You should create a window here. The Message$ variable is set.

LS_UPDATE: Update compiling information. The Message$ variable is set. The Position$ and Max$ variables are set with the current line and maximum number of lines in the file compiled.

LS_SHUT: When all is done, you can free your window here.

Syntax

PPL uses a relatively simple syntax that is very close to the C language syntax. PPL is made of some of the easiest features found in a few programming languages including C, Pascal, Basic and Fortran. But we've added more and made PPL one of the easiest languages to understand and use.

PPL code structure:

```
<verb>;
<verb>(<expression>);
<variable> [, <variable>...] = <expression> [, <expression>...];
```

Examples:

```
ShowMessage("Hello World!");
ShowMessage(10+10);
a$ = 10;
a$, b$ = 10, 20;
Beep;
Beep();
```

PPL supports many types of tokens including:

`"string"`	String
`{string}`	Multi-level string
`'C'`	Ascii value of character
`'string'`	Evaluated string
`10`	Decimal value
`-10`	Negative value
`10.2345`	Value
`0xE10B`	Hex value
`0101`	Octal value (starts with 0)
`#9`	Character if value.
`#0xC`	Character of hex value.

Example:

```
A$ = "Line 1" + #13#10 + "Line 2";
B$ = 10 + 'A';      // = 75
C$ = B$ + 0x0012AC0;
```

Only these verbs are allowed to start a PPL code line:

```
IF (<expression>)
ELSE
ELSE IF (<expression>)
END;
PROC <expression> [(<var list,…>)]
FUNC <expression> [(<var list,…>)]
NPROC <expression>
NFUNC <expression>
FORWARD
PUBLIC
PRIVATE
REPEAT
UNTIL (<expression>);
WHILE (<expression>);
BREAK;
```

```
CONTINUE;
RETURN [(<expression>,…)];
CASE (<expression>)
<value> :
FOREACH (list)
LOCAL (<var list,…>);
GLOBAL (<var list,…>);
PUBLIC (<var list,…>);
PRIVATE (<var list,…>);
&<variable> = <variable>;
&<variable> [, &<variable>...] = <expression> [, <expression>...];
<variable> = <expression>;
<variable> [, <variable>...] = <expression> [, <expression>...];
<variable> ++ ;
<variable> -- ;
<function call> [(<arguments,…>)];
<procedure call> [(<arguments,…>)];
#INCLUDE <filename>
#GLOBAL
#LOCAL
#LIBRARY
#DEFINE <name>
#IFDEF <name>
#IFNDEF <name>
#ELSE
#ELSEIF <name>
#ENDIF
#DECLARE <name> <path> <name>
#DECLAREAPI <name> <path> <name>
#DEBUG
#NOLINK
#NOPPC
#CLASS
#ENDCLASS
#INHERITED
#OBJECT
```

OBJECT ORIENTED PROGRAMMING

PPL is capable of doing object-oriented programming. While the scope of this document is not to explain what object-oriented programming and how to use it, we will concentrate solely on how PPL does object-oriented programming.

To define a new class, use the #class compiler directive. The first parameter can either be the new class name or a class name of a class the new class will inherit from. Class definition must always be ended with the #endclass compiler directive.

#class <inherit class> <class name>
#endclass

Example:

```
#class oldclass
#endclass

#class oldclass myclass
#endclass
```

In this example, myclass inherits from oldclass.

Now it's time to add variables to the class. Variables within a class can be public or private. Public variables are accessible outside the class definition while private variables are not. After declaring the scope of your variables, you can initialize them. PPL will do the initialization as soon as an object of that class is created.

Example:

```
#class oldclass
  Public(z$);
  Private(t$);
  z$ = 20;
  t$ = 20;
#endclass

#class oldclass myclass
  Public(x$, y$);
  Private(internal$);
  x$ = 10;
  y$ = 20;
  internal$ = 30;
#endclass
```

The internal$ and t$ variables won't be accessible outside the scope of the each class's definition. X$, Y$ and Z$ will be available.

Why don't we create an object using the myclass class? To do this use the #object compiler directive. The first parameter is the variable name that will hold the object and the second parameter is the name of the class.

Example:

```
#object myclass o$
```

Using variables of the class from the o$ variable is very simple. Simply seperate the object variable from the class variable using a dot.

Example:

```
o.x$ = 60;
ShowMessage(o.z$);
```

While it is good the create a class with variables, no class can be useful without logic. PPL is very flexible in the way you want to define class's methods. Methods are just regular PROC's or FUNC's in PPL. You can also use NPROC and NFUNC. You can define the method's body within the class definition directly or you can forward the declaration to define the body later on. It is also very important to note that methods in PPL can be private or public. By default methods are private. If you need to declare a public method, place the PUBLIC keyword in front of the method definition within the class definition. The keyword PRIVATE is also supported to make your code more obvious.

Example:

```
#class oldclass
  Public(z$);
  Private(t$);
  z$ = 20;
  t$ = 20;

  proc test (a$)
    ShowMessage("a$ = ", a$, "t$ = ", t$);
  end;
#endclass

#class oldclass myclass
  Public(x$, y$);
  Private(internal$);
  x$ = 10;
  y$ = 20;
  internal$ = 30;

  forward proc mytest(b$);
  public forward func avg;
```

```
#endclass

proc myclass.mytest(b$)
  internal$ = b$ + x$;
  ShowMessage("Internal$ = ", internal$);
end;

func myclass.avg
  mytest(10);                  // This call is permitted. Inside class
definition.
  return ((x$ + y$ + z$) / 3);
end;

proc main
  #object myclass o$
  ShowMessage(o.avg);
  o.mytest(10);                  // This call is invalid, it is a private
procedure.
  FreeObject(o$);                  // Free the object from memory
end;
```

Objects can also be copied between one another and objects can point to others quite easily. PPL will copy objects automatically for you. However it is very important to note that you have to create an object with the same class before doing any assignments. Since method calls are connected at compile time within the compiler, PPL needs to know the object type before. Using the classname will tell PPL that the corresponding object variable is of a type <classname> but will not create the object in memory.

Example:

```
#object myclass o$
myclass(o2$);

o.y$ = 50;
o2$ = o$;
o2.x$ = 100;
```

Objects passed as parameters are always passed as reference to the original pointer.

Example:

```
proc Test (obj$)
  Obj.x$ = 10;
  Obj.CallFunc;
end;

proc main
  #object myclass o$;
  Test(o$);
  FreeObject(o$);
end;
```

To create a new object that is pointing to another, follow this:

Example:

```
#object MyOtherClass o$
MyOtherClass(o2$)
```

```
o.y$ = 50;
&o2$ = o$;
o2.x$ = 100;      // This will change o.x$, since o2$ is pointing to o$.
```

PPL also supports a constructor and a destructor. The Create procedure is the constructor and the Destroy procedure is the destructor. The Create procedure has to be declared as private and as an NPROC and the Destroy as a regular private PROC procedure without any parameters.

Example:

```
#class myclass
  Public(x$, y$);

  nproc create
    x$ = args$[0];
    y$ = args$[1];
  end;
  proc destroy
    ShowMessage("destroyed!");
  end;
#endclass

proc main
  #object myclass o$(10, 20);
  ShowMessage(o.x$);                    // result 10
  ShowMessage(o.y$);                    // result 20
  FreeObject(o$);                          // Free object from memory
end;
```

NB: It is very important to note that objects created with PPL won't be freed automatically just like normal variables. This is due to the fact that multiple objects can be created using the same variable. It is then imperative to free the objects from memory manually using the FreeObject() or Free() function.

When typecasting an object be very carefull, if the variable is not an already created object, it will be created automatically.

Example:

```
proc testing
  MyClass(o$);
  o.Test(10);
end;

proc main
  MyClass(o$);
  testing;
end;
```

In this case o$ is created as a MyClass object but in testing, since o$ is not global the typecasting will create a new object o$ instead.

The object-oriented syntax is very close to the regular PPL syntax and has been done this way to maintain simplicity.

Advanced Object-Oriented Programming

PPL allow for more advanced object-oriented programming. There will be times where you will need to create multiple objects and reference to them later by their pointer instead. This topic will cover object pointers and list of objects and arrays of objects too.

Object pointers:

Object pointers are obtained the same way any other variable pointers can be obtained. Assigning an object pointer to a variable is quite easy. The variable will not be converted to an object automatically. However assigning a pointer to another variable to then use as an object requires an extra step.

```
#object myclass o$
```

This will create a new object with the class "myclass".

```
p$ = &o$;
```

p$ will now contain the pointer (address location) of object o$.

Let's say class myclass as variable "name$" publically declared.

```
ShowMessage(p.name$);
```

Is not valid because, p$ is just an ordinary value at this point. However by making the p$ variable an object with the NewObject () function, PPL will be able to access the "name$" variable.

```
NewObject(p$, "myclass", &o$);
```

This will make p$ an object of class "myclass" and will make p$ point to o$. This could be achieve by doing the following:

```
p$ = o$;
```

However in a case where we need a list of objects or an array of objects, it will become very handy.

Another alternative to NewObject() is the following:

```
myclass(p$);
pointer$ = &o$;
&p$ = pointer$;
```

This code will first declare p$ as a myclass class type without creating a memory block for storing the class. Then we set pointer$ to the memory location of object o$. We then assign the p$ memory location to what is contained in pointer$.

Arrays of Objects:

An array of objects is simply an array of integer values containing pointers to objects.

```
dim(l$, 10);
for (i$, 0, 9)
  #object myclass o$;
  o.value$ = i$;
  l$[i$] = &o$;
end;

for (i$, 0, 9)
  Writeln(l$[i$].value$);
  FreeObject(l$[i$]);
end;
```

**** Note that l$[i$].value has a different syntax than normal array of structures. You need to access the array just like a normal array and then pass the public proc/func or public variable you need to access.**

List of Objects:

Much like an array of objects, a list of objects involves linked-list and the information stored into each list element is a full reference object. You can directly use the object from the linked-list element.

```
List(l$);
for (i$, 0, 9)
  #object myclass o$;
  o.value$ = i$;
  Add(l$, o$);
end;

ForEach(l$, p$)
  Writeln(p.value$);
  FreeObject(p$);
end;
```

Type casting objects:

PPL offers a nice mecanism to type cast objects to different classes. The best approach is to typecast the object on a single line but you can also temporarely typecast the object in an expression.

```
#object myclass o$;
otherclass(o$).ProcCall (10, 20);
a$ = otherclass(o$).variable$;
ShowMessage(otherclass(o$).funccall);

otherclass(o$);
o.ProcCall(10, 20);
a$ = o.variable$;
ShowMessage(o.funccall);
```

NB: It is very important to note that objects created with PPL won't be freed automatically just like normal variables. This is due to the fact that multiple objects can be created using the same variable. It is then imperative to free the objects from memory manually using the FreeObject() function.

Strings

Strings in PPL are very easy to use. PPL will take care of memory allocation for you and will free them up as needed. The garbage collection mechanism of the interpreter is very flexible and will save you a lot of time.

Strings definition:

```
"this is a string"
{this is a string that can hold pretty much anything including "double-quotes"
and {this also} !}
'the result of 10 + 10 = {10+10}'
```

We often use the {} strings in a case where we need to run a code string:

```
Run({ShowMessage("Hello World!");});
```

You can concatenate strings together using the + operator like this:

```
ShowMessage("Pocket "+"Programming "+"Language");
```

Strings are made up of a series of bytes and can be used as an array of bytes within PPL. PPL supports unicode characters through two commands, Wide() and Char(). It is recommended not to work with unicode strings within PPL since they are not an internal type supported by PPL. The two functions are used to convert values from and to Windows API calls.

Example:

```
S$ = "ABCDEFG";
```

```
//remember, when referenced this way, PPL returns the element of the string
// as its byte value, not its string value
ShowMessage(s$[0]);  //Displays 65
ShowMessage(s$[1]);  //Displays 66
ShowMessage(s$[3]);  //Displays 68
```

While accessing string characters one by one is useful, PPL also allows you to grab parts of a string using the same technique as above but with two element identifiers instead (commonly referred to as a "substring" operation in other languages):

Example:

```
s$ = "ABCDEFG";
a$ = s$[0, 3];      // Grabs the first 3 characters. a$ = "ABC";
a$ = s$[3, 0];      // Grabs the last 3 characters. a$ = "EFG";
a$ = s$[2,3];      // Grabs 3 characters from third character. Remember strings
are 0 index based. a$ = "CDE";
```

There are two ways to retrieve the character equivalent of a single element in a string:

Example:

```
s$ = "ABCDEFG";
ShowMessage(s$[0,1]);     //Displays "A"
ShowMessage(Chr(s$[2]));  //Displays "C"
```

Operators on strings:

Strings are also supported if used with the following mathematical operators:

```
+
-
*
/
+=
-=
*=
/=
```

** A division on a string requires the correct amount of variables assignments. Ex: "ABC" / 3, needs three variables before the = sign.

Example:

```
s$ = "ABC" + "DEF";            // result of s$ is "ABCDEF"
s$ = "ABCDEF" - "BC";            // result of s$ is "ADEF"
s$ = "ABC" * 3;                  // result of s$ is "ABCABCABC"
a$, b$, c$ = "ABC" / 3;            // result of a$ is "A", b$ is "B" and c$ is
"C"
s$ = "ABC";
s$ += "DEF";                  // result of s$ is "ABCDEF"
s$ -= "BC";                   // result of s$ is "ADEF"
a$, d$, e$, s$ /= 3;             // a$ = "A", d$ = "D", e$ = "E", s$ = "F"
s$ *= 4;                        // result of s$ is "FFFF"
s$ = 'The result of 10+10 = {10+10}'; // result of s$ is "The result of 10+10 =
20"
```

Concatenating strings:

Concatenating two strings or values together is very easy to do with the % operator.

Example:

```
s$ = 10 % 20;                    // result of s$ is "1020"
s$ = 10 % "TEST";                    // result of s$ is "10TEST"
```

Control characters in strings:

You can also include control characters in any string using the \ operator.

Example:

```
s$ = LoadStr("\\My Documents\\MyFile.txt", c$);
```

Here is a list of supported character controls:

\\\	
\t	Tab
\n	CRLF
\r	CR
\l	LF
\"	"
\}	}
\0	BLANK

Evaluated strings:

Evaluated strings are like normal strings, except that enclosed codes between {} are evaluated by the interpreter and there value inserted in the result string. Character controls are also supported in evaluated strings.

Example:

```
a$ = "cool";
ShowMessage('It is {a$} when 10+20={10+20}');      // It is cool when 10+20=30
```

Any valid PPL code can be enclosed within {}.

Numbers

PPL supports numbers of 8 bytes (double size) maximum. This should be enough for any application type.

1.7E +/- 308 (15 digits) are the minimum and maximum range of a double size value.

Examples:

```
ShowMessage(10 + 10 / 2);
ShowMessage(82732983.289372 / 3);
ShowMessage(abs(-3));
I$++;
ShowMessage(I$ AND Z$);
```

Precedence of operators

PPL, when determining how to perform calculations, works according to pre-defined rules. These rules may be overridden by the use of parenthesis ().

The priority given to the various operators, from highest to lowest, are

```
   / NOT DIV %
   * MOD
   + - | ^ &
   ==  <>  <  <=  >  >= << >> SHL SHR
   AND OR XOR
```

The operators are always evaluated left to right.

OPERATORS (+ - / *)

Operators on numbers can also be used with strings or any other types in PPL. Strings are also supported in multiple operators.

The + operator will try to convert the values to numeric and try to add them together. If one of the values cannot be converted the operator will concatenate the two values as a string.

Example:

```
i$ = i$ + 10;
i$++;
i$ = "This is a " + "string!";      // Result of i$ is "This is a string"
i$ = "ABCDEF" - "BC";                  // Result of i$ is "ADEF"
i$ = "ABC" * 3;                        // Result of i$ is "ABCABCABC"
i$ = "ABC" / 3;                        // Result is "A", "B", "C"
i$ = 10 / 2;                        // Result of i$ is 5
i$--;                                    // Result of i$ is 4
i$ += 10;                              // Result of i$ is 14
i$ -= 10 + 20;                        // Result of i$ is -24
i$ = "10" + "A"                        // Result of i$ is "10A"
i$ = "10" + "20"                        // Result of i$ is 30

s$ = "ABC" + "DEF";                   // result of s$ is "ABCDEF"
s$ = "ABCDEF" - "BC";                  // result of s$ is "ADEF"
s$ = "ABC" * 3;                        // result of s$ is "ABCABCABC"
a$, b$, c$ = "ABC" / 3;                  // result of a$ is "A", b$ is "B" and
c$ is "C"
s$ = "ABC";
s$ += "DEF";                        // result of s$ is "ABCDEF"
s$ -= "BC";                         // result of s$ is "ADEF"
a$, d$, e$, s$ /= 3;                   // a$ = "A", d$ = "D", e$ = "E", s$ = "F"
s$ *= 4;                              // result of s$ is "FFFF"

s$ = 10 + "TEST";                        // result of s$ is "10TEST"
s$ = 10 + "20";                        // result of s$ is 30
```

DIV MOD

DIV computes the quotient and the remainder of two integer values.
The result of the modulus operator (MOD) is the remainder when the first operand is divided by the second.
Example:

```
i$ = 10 div 3;
ShowMessage(i$);

if (i$ mod 10 == 0)
  ShowMessage("Can be divided by 10");
end;
```

& | ^ ~

&
The bitwise-AND operator compares each bit of its first operand to the corresponding bit of its second operand. If both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.
|
The bitwise-inclusive-OR operator compares each bit of its first operand to the corresponding bit of its second operand. If either bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

^
The bitwise-exclusive-OR operator compares each bit of its first operand to the corresponding bit of its second operand. If one bit is 0 and the other bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.
The one's complement operator, sometimes called the "bitwise complement" or "bitwise NOT" operator, produces the bitwise one's complement of its operand. The operand must be of integral type. This operator performs usual arithmetic conversions; the result has the type of the operand after conversion.

Example:

```
i$ = i$ & j$;
i$ = i$ | j$;
i$ = i$ ^ j$;
i$ &= 10;
i$ |= 10;
i$ ^= 10;
y$ = 2;
y$ = ~y$;
```

%
Concatenate two values into a string

Example:

```
i$ = 10 % 20;                 // Result of i$ is "1020"
i$ = "A" % "B"                 // Result of i$ is "AB"
i$ = "10" % 20                 // Result of i$ is "1020"
```

Notes

- Both values are converted to strings before being concatenated together

See Also: CONCAT

Main and WinMain procedures

The Main and WinMain procedures are always executed the first when the program is being executed. If no Proc or Func is defined you don't need to create a Main or a WinMain procedure.

The only difference between the Main and the WinMain procedures is that WinMain is a function returning a boolean value and it's execution, PPL doesn't remove the program from memory if the return value is true. The program will still be active and all calls to a window procedure will be forwarded to the program's window's function or handled by PPL. Be carefull about using the WinMain procedure without creating a window first. PPL will keep the program in memory and active. Without a window it is almost impossible to detect if a program is running or not, unless you use the program manager.

Example:

```
Proc CreateWindow
  ...
End;

Func WinMain
  CreateWindow;
  Return (true);
End;
```

To terminate a program that is running in memory, simply send a WM_TERMINATE message and PPL will automatically catch it.

PostMessage(NULL, WM_TERMINATE, 0, AppHandle$);

The AppHandle$ variable contains the handle of the application to terminate. The WM_TERMINATE is a special message processed by PPL.

Pointers

PPL handles pointers almost like C handles them. Since the nature of PPL is to be very user friendly, pointers are generally used in extreme cases. This doesn't mean that there is less pointer functions available to you. PPL comes with a variety of powerful pointer functions and pointer operators.

Let's review a few of the pointer functions and operators. In order to create a new memory allocation to hold bytes of information the New function needs to be used. The Free function is used to free the memory allocation.

The New function initialize the memory location with zeroes when it is created.

Example:

```
New(var$, 1024);      // Assign 1024 bytes of free memory to var$ and
                        initialize with 0's.
Free(var$);            // Free memory assigned by var$
```

Pointers in PPL are handled strictly using bytes. Strings are a series of bytes and not of WideChar type like the C language under some compilers.

To get the address where a variable is located in memory, use the & operator. The & operator will return a integer value on the stack with the address pointed by a variable. You can later on convert this address back to a variable with the @ operator. The @ operator translate an integer value into an actual value from the variable is points to.

Example:

```
New(var$, 1024);
S$ = "THIS IS A TEST!";
Memcpy (var$, s$, 5);
Memcpy (&var$+5, &s$+5, length(s$)-5);
ShowMessage(var$);
Free(var$);

S$ = "LET'S SEE, PPL HAS POINTERS TOO!";
ShowMessage(@(&s$+12));
ShowMessage(s$[12]);
```

You can resize a memory allocation that has already been created with the Resize function.

Example:

```
New(var$, 11);
Memcpy(var$, "TEST STRING", 11);
ShowMessage(var$);
Resize(var$, 22);
MemCpy(var$, "THIS IS A TEST STRING", 21);
ShowMessage(var$);
Free(var$);
```

You can use different functions to access memory location by either byte, short, int or double value. Short is 2 bytes, Int is 4 bytes and Double is 8 bytes.

Example:

```
New(var$, 15);
SetInt(var$, 150374);
ShowMessage(GetInt(var$));
SetByte(&var$+4, 15);
ShowMessage(GetByte(&var$+15));
SetShort(&var$+5, 1024);
```

```
ShowMessage(GetShort(&var$+5));
SetDouble(&var$+7, 23937.3893);
ShowMessage(GetDouble(&var$+7));
Free(var$);
```

Another nice feature of PPL is that variables can point to other variables. PPL is very flexible with pointers and you can also have many variables pointing to many variables.

Example:

```
A$ = 10;
ShowMessage("a$ is equal to 10");
ShowMessage(a$);
&b$ = a$;
ShowMessage("b$ should now point to a$");
ShowMessage(a$+","+b$);
A$ = 20;
ShowMessage("b$ and a$ are now 20");
ShowMessage(a$+","+b$);
B$ = 30;
ShowMessage("b$ and a$ are now 30");
ShowMessage(a$+","+b$);
C$ = 40;
&a$ = c$;
ShowMessage("a$ points to c$ and b$ points to a$ and are all 40");
ShowMessage(a$+","+b$+","+c$);
C$ = 50;
ShowMessage("a$, b$ and c$ are now 50");
ShowMessage(a$+","+b$+","+c$);
```

You have to be careful if you delete (clear) a variable that is pointed by other variables, PPL will not remove the links. Therefore using a variable that is pointing to a non-existant variable will produce unexpected results.

PPL will delete the variables pointed by the variables that you delete (clear), this can lead to memory location problems if you are pointing to a variable that has already been deleted. Make sure the variables you point to are either globals or are within the same program scope.

Arrays

Sometimes it's necessary to use arrays to store information within memory. It is also useful to retrieve information from a memory allocation returned by a function (Windows API often).

To create an array you will need to use the Dim function with the Clear or Free function to free the array from memory. Arrays can be of any given size up to a maximum of 8 elements. Array elements are made of double values (8 bytes).

Example:

```
Dim(var$, 10, 10, 10);
Var$[4, 4, 4] = "THIS IS A STRING";
Var$[5, 5, 5] = 9273.38;
Var$[6, 6, 6] = 10;
Free(var$);
```

This example shows you how flexible PPL can be. Strings can be stored in arrays very easily. PPL recognise strings and store the address where the string is located in the array element. Therefore to retrieve the string, you need to use the @ operator.

Example:

```
Dim(var$, 10, 10, 10);
Var$[4, 4, 4] = "THIS IS A STRING";
```

```
ShowMessage(@Vars$[4, 4, 4]);
Free(var$);
```

Strings can also be used as arrays in PPL. The only difference is that strings are arrays of bytes.

Example:

```
S$ = "ABCDEFG";
Dim(var$, length(s$));
While (I$<=length(s$))
  Var$[I$] = s$[I$];
End;
Free(var$);
```

You don't have to free or clear variables because PPL's garbage collection takes care of this for you but it is a very good habit to keep if you use other programming languages. If you assign a variable that has not been cleared in PPL, the old memory used by the variable will be freed by PPL for you.

Example:

```
Dim(var$, 10);
Var$[3] = 3;
Var$ = "NEW VALUE";      // var$ is not an array anymore but a string   now. PPL
frees the old memory used by var$ before creating a new one.
```

In a case where you would need to copy an array to another variable, PPL simplifies the task greatly by allowing you to assign an array to another variable.

Example:

```
Dim(a$, 10);
a$[3] = 3;
b$=a$;
b$[4] = 4;
ShowMessage(b$[3]);
ShowMessage(b$[4]);
free(a$, b$);
```

You can also use the **SDIM**() statement to define an array with a specific size.

SDIM (var$, SIZE, [DIMENSIONS...]);

Example:

```
SDim(a$, TBYTE, 10, 10);           // Can only be byte values. Size is 100.
a$[5,5] = 30;
a$[2,2] = "Hello";                 // This is not valid. You cannot store
strings pointers into tbyte size elements.
```

What about looping through all array's elements, I hear you ask? Well, there is nothing easier with PPL. The ForEach() statement is very powerful and support many different variable types. The only thing you need to add is an extra variable that will be set with the array's element's address every loop.

Example:

```
dim(a$, 5);
i$ = 0;
ForEach(a$, addr$)
  addr$ = i$;
  i$++;
end;

// Another way for doing this

dim(a$, 5);
i$ = 0;
ForEach(a$, addr$)
  a$[i$] = i$++;
end;
```

Matrix

Data elements of different types can be stored into a series of elements called a matrix.

The elements contained in a matrix can be an arbitrary mix of elements or matrices. A matrix is represented by a list of elements in brackets, separated by commas. Elements can have any integer, double-precision floating point or string value.

```
a$ = [10, 20, 30, "String1", "String2", 2983.9392893, 92378.823637];
```

Matrices can be nested to any depth, i.e. you can have matrices within matrices within matrices and so on to any depth (until you run out of memory). Brackets are used to construct matrices out of a list of expressions. These expressions can be constant or evaluated at run-time. e.g.

```
a$ = [10, 20, [sin(0.2), "String", [500, 600]], 30];
```

Multiple operations can be done on matrices, including adding matrices together, subtracting, multiplying and dividing. Many other operators also process matrices. Here is a list of all operators that process matrices:

```
+ - * / @ << >> asl asr shl shr and or xor == <> < > <= >=
```

Here are a few examples on how to use matrices with operators:

```
a$ = [10, 20] + [10, 20];            // Result is a matrix with [20, 40]
a$ = [10, 20] + [10, 20, 50, 60];      // Result is a matrix with [20, 40, 50,
60]
a$ = [10, 20] + 5;                   // Result is a matrix with [15, 25]
a$ = [10, 20];
a$ = a$ * [2, 4, 8];                   // Result is a matrix with [20, 80, 0]
a$ = [10, 20] == [10, 25];             // Result is a matrix with [1, 0]
```

You can also perform various operations on matrices using the integrated matrix functions provided with PPL:

```
mcount (matrix$) -> count$
mtype (matrixelement$) -> type$
madd (matrix$, value$) -> newmatrix$
mdel (matrix$, start$, count$) -> newmatrix$
mmid (matrix$, start$, count$) -> newmatrix$
```

Structures

PPL also support a very powerful and flexible type of variables that is called a structure. A structure can be made of many elements of any size. Structures are very often used in the Windows or WindowsCE API and are quite easy to use in PPL. To define a structure you need to use the Struct function. The struct function is very flexible, it allows you define elements with any values you want and any size.

Example:

```
Struct (r$, "Left", "Top", "Right", "Bottom");
```

```
HWnd$ = Newform("Window", "MyWindowClass", NULL);
GetWindowRect(HWnd$, &r$);
ShowMessage(r.left$+", "+r.top$+", "+r.right$+", "+r.bottom$);
CloseWindow(HWnd$);
Free(r$);
```

The tint is a predefined value that makes the element a integer value of 4 bytes in size. There is also tbyte (1 byte), tshort (2 bytes) and tdouble (8 bytes) that can be used. The default size is tint. But you can also use your own size in bytes. In the case of a user defined size, the data hold can be used as a regular string container or for any other purpose.

Example:

```
Struct (a$, "s", 50, "y", tbyte, "t", tshort);
a.s$ = "STRING";
a.y$ = 10;
a.t$ = 2938;
ShowMessage(a.s$);
Free(a$);
```

In a case where it is necessary to copy a struct to another struct, PPL has been designed to simplify this task.

Example:

```
Struct (a$, "a", 10, "b", tint);
a.a$ = "STRING";
a.b$ = 10;
c$ = a$;
ShowMessage(c.a$);
free(a$, c$);
```

Can you also define a single variable to be of a certain type or length. By using the **TYPE** function you can specify a variable's size and use it just like a structure's element.

Example:

```
Type(i$, tint);
i$ = 10;
ShowMessage(i$);

Type(b$, tbyte);
b$ = 255;
ShowMessage(b$);
```

Indexing Structures

It is also possible to use structure variables with an index offset just like arrays. In this case the index value becomes the field number of the structure. The indexing is zero based, just like arrays.

Example:

```
struct(a$, "str1", "str2", "str3");
a.str1$ = "String #1";
a$[1] = "String #2";
ShowMessage(@a$[1]);      // "String #2"
```

Linked-List

One of the most powerfull and flexible variable type in PPL is the Linked-List type. Linked-List variables are handled like a simple variable that can hold different variable types, except that each list element is stored in memory and handled by PPL internally. A Linked-List variable needs to be defined prior to being used. Then new elements can be added, deleted or inserted. Memory is allocated dynamically as the elements are created or deleted.

Example:

```
List(l$);

Add(l$, "Element 1");
Add(l$);
l$ = "Element 2";

First(l$);
ShowMessage(l$);
Next(l$);
ShowMessage(l$);
```

Each element can be of any type, including an array or a structure. At this point you should understand why Linked-List variable type is so powerfull and flexible.

Example:

```
List(l$);

Add(l$);
Dim(l$, 10, 10);
L$[5, 5] = 10;

Add(l$);
Struct(l$, "Item1", "Item2");
L.Item1$ = 1;
L.Item2$ = 2;

First(l$);
ShowMessage(l$[5,5]);
Next(l$);
ShowMessage(l.item1$);
```

You can process each elements of a list in a loop using the ForEach statement. ForEach will loop through all elements of the linked-list variable.

Example:

```
List(l$);
Strtolist("A;B;C;D;E;F;G", ";", l$);

ForEach(l$)
  ShowMessage(l$);
End;
```

Indexing Linked-List

It is also possible to use linked-list variables with an index offset just like arrays. In this case the index value becomes the list item index. The indexing is zero based, just like arrays.

Example:

```
list(l$);
Add(l$, 0, 1, 2, 3, 4, 5);
```

```
l$[2] = 20;
ShowMessage(l$[2]);      // 20
```

Files

File handling functions in PPL are pretty much the same as the ones in C. We've added two new functions, ReadString() and WriteString() which handles reading lines and writing lines to text file.

```
FOpen (filename$, mode$) -> filehandle$
FClose (filehandle$)
FRead (addr$, size$, count$, filehandle$) -> size$
FWrite (addr$, size$, count$, filehandle$) -> size$
FSeek (filehandle$, offset$, origin$) -> success$
** -1 as origin means, from end of file.
FTell (filehandle$) -> position$
ReadString (filehandle$) -> string$
WriteString (filehandle$, string$)
```

Windows API

In order to be complete, a programming language needs to access the standard Windows API functions from within the core of the system's dll files. PPL has been designed with this in mind and offers more than regular programming languages. PPL recognizes string values and converts them when needed to WideChar strings automaticly. Values coming from the API function will not be converted back to regular strings. You will have to do this yourself.

Example:

```
MessageBox(null, "Message", "Title", MB_OK);
GetWindowText(e$, &s$);
S$ = char(s$);
ShowMessage(s$);
```

PPL comes with a group of functions that simplifies creation and handling of regular Windows interface. The Newform function will create a new window on screen and handle most of the standard messages itself. NewControl will create a new control on an owner window. NewMenuBar will create a menu bar at the top of the screen. NewMenu and NewMenuItem allow you to create menus and menu items with a menu bar.

Example:

```
Func EditProc (hWnd$, Msg$, wParam$, lParam$)
  ok$ = true;
  case(Msg$)
    WM_KEYDOWN:
      if (wParam$ == VK_HOME)
        ShowMessage("HOME key pressed!");
      end;
  end;
  Return(ok$);
End;

Func WndProc (hWnd$, Msg$, wParam$, lParam$)
  ok $ = true;
  case(Msg$)
    WM_COMMAND:
      wmId$ = LOWORD(wParam$);
      wmEvent$ = HIWORD(wParam$);
      case(wmId$)
        401:
          PostMessage(hWnd$, WM_CLOSE, 0, 0);
        500:
          PostMessage(hWnd$, WM_CLOSE, 0, 0);
     End;
  End;
  Return(ok$);
End;
```

```
Proc WinMain
  f$ = NewForm("Window", "MyWindowClass", &WndProc);

  m$ = newmenubar(f$, 400);
  n$ = NewMenu(m$, "&File");
  NewMenuItem(n$, "E&xit", 401);

  b$ = NewControl(500, "BUTTON", "Close", 0, f$, 10, 10, 100, 50);
  e$ = NewControl(600, "EDIT", &EditProc, "", 0, f$, 10, 100, 150, 120);

  ShowWindow(f$, SW_SHOW);
End;
```

You should also review the EDIT.PPL source file that comes with the PPL package to get a grip of how it all works together.

Windows tm comes with a collection of .dll files that contains a good amount of procedures or functions that can be easily accessed from PPL by using the #declare and #declareapi compiler directives.

```
#declare SetRect "coredll.dll" SetRect 5 1
#declareapi GetWindowText "coredll.dll" GetWindowTextW 3 1
```

The first parameter is always the name you want to give the new procedure or function within PPL. The second one is the path of the .dllm the third is the name of the procedure function within the .dll, the fourth parameter is the number of input parameters and the fifth is the number of outputs.

The difference between #declare and #declareapi is that the #declareapi converts all PPL strings parameters to widechar strings to the windows procedure or function. Return widechar strings can also be converted with the char() function.

Example:

```
s$ = "Hello World!";
MessageBox(NULL, "Dialog", s$, MB_OK);

Len$ = GetWindowTextLength(hWnd$);
Dim(s$, len$);
GetWindowText(hWnd$, s$, len$);
s$ = char(s$);
ShowMessage(s$);
Free(s$);
```

IMPORTANT: Values passed to API functions should be in the right type. If you want to pass an integer value but it's stored internally as a string, you need to use the INT() function.

Example:

```
x$ = "10";
y$ = "20";

SendMessage(WM_USER + 10, 0, Int(x$), Int(y$));
```

The same thing is true when you want to pass a numerical value as a string. You need to use the STR() function.

Example:

```
SendMessage(WM_USER + 10, 0, STR(10), NULL);
```

Simplified Windows API

Here is list of the functions you will find in the SWAP.PPL library that you can include in your programs using the following line:

```
#include "swapi.ppl"
```

The functions provided by the SWAPI are very easy to use and should simplify programming in Windows a great deal.

CheckBoxes and Radio buttons:

proc Button_Set(button$, checked$)
Sets the checkbox or radiobutton check attribute.

func Button_Get(button$)
Retrieve the checkbox or radiobutton checked attribute.

Edit control:

func Edit_CanUndo(edit$)
Check if the edit control can undo.

func Edit_CharFromPos(edit$, x$, y$)
Convert a pixel position to a char index.

proc Edit_EmptyUndoBuffer(edit$)
Empty the undo buffer for the edit control.

func Edit_GetFirstVisibleLine(edit$)
Return the first visible line index of the edit control. The index is zero based.

func Edit_GetLimitText(edit$)
Return the text limit in character.

func Edit_GetLine(edit$, index$)
Return a line's text.

func Edit_Count(edit$)
Return the number of lines in an edit control.

func Edit_Modified(edit$)
Return wheter the edit control has been modified.

func Edit_GetPasswordChar(edit$)
Return the edit control password character.

func Edit_GetSelStart(edit$)
Return the edit control first character selection.

func Edit_GetSelEnd(edit$)
Return the edit control end character in the selection.

func Edit_GetSelLength(edit$)
Return the length in characters of the edit control selection.

proc Edit_LimitText(edit$, max$)
Set the edit control limit of characters that can be inputted.

func Edit_LineFromChar(edit$, pos$)
Return the line index number from a character position.

func Edit_LineIndex(edit$, line$)
Return the first character position of a line.

func Edit_LineLength(edit$, line$)
Return the length of a line.

proc Edit_LineScroll(edit$, x$, y$)
Scroll by X and Y lines the edit control.

func Edit_PosFromChar(edit$, charindex$)

func Edit_Get(edit$)
Return the edit control text.

proc Edit_Set(edit$, text$)
Set the edit control text.

proc Edit_ScrollCaret(edit$)
Scroll to the caret's position. Make sure it is in view.

proc Edit_Modify(edit$, modified$)
Set the modify flag of the edit control.

proc Edit_SetPasswordChar(edit$, char$)
Set the password character of the edit control.

proc Edit_SetReadOnly(edit$, readonly$)
Set the edit control read only flag.

proc Edit_SetSelStart(edit$, start$)
Set the edit control selection starting character.

proc Edit_SetSelEnd(edit$, end$)
Set the edit control selection ending character.

proc Edit_SetSelLength(edit$, length$)
Set the edit control selection length in character.

proc Edit_CopyToClipboard(edit$)
Copy select edit control text to clipboard.

proc Edit_CutToClipboard(edit$)
Cut selected edit control text to the clipboard.

proc Edit_PasteFromClipboard(edit$)
Paste the clipboard text replacing the edit control selection.

proc Edit_Undo(edit$)
Undo last change in edit control.

proc Edit_SelectAll(edit$)
Select all characters in edit control.

proc Edit_LoadFromList(edit$, slist$)
Set the text of an edit control from a list.

proc Edit_SaveToList(edit$, slist$)
Build a list variable from all the lines of an edit control.

proc Edit_LoadFromFile(edit$, Filename$)
Load the edit control text from a file.

proc Edit_SaveToFile(edit$, Filename$)
Save the edit control text to a file.

Menus and Menu Items:

proc Menu_Add(menu$, id$, caption$)
Add a new menu item to a menu.

proc Menu_Insert(menu$, before$, id$, caption$)
Insert a menu item before an item (at position before$) in the menu.

proc Menu_Check(menu$, id$, checked$)
Check the menu item or not.

func Menu_Create
Create a new menu.

proc Menu_Del(menu$, id$)
Delete a menu by it's id.

proc Menu_Destroy(menu$)
Destroy a menu.

proc Menu_DrawMenuBar(hwnd$)
Redraw the menu bar after adding or deleting menus to it.

proc Menu_Enable(menu$, id$, enabled$)
Enable or disable a menu item by its id.

func Menu_Checked(menu$, id$)
Return if the menu item is checked or not.

func Menu_Enabled(menu$, id$)
Return if the menu item is enabled or not.

proc Menu_Set(menu$, id$, caption$)
Set the menu item's caption.

func Menu_Get(menu$, id$)
Return the menu item's caption.

func Menu_CreatePopup
Create a popup menu.

proc Menu_TrackPopup(menu$, x$, y$)
Display popup menu.

Combobox:

proc ComboBox_Clear(combobox$)
Remove all items from the combobox list.

func ComboBox_Count(combobox$)
Return the number of items in the combobox list.

func ComboBox_Add(combobox$, text$)
Add a new item to the combobox list.

func ComboBox_Insert(combobox$, index$, text$)
Insert a new item in the combobox list.

func ComboBox_GetSel(combobox$)
Return the selected item.

proc ComboBox_SetSel(combobox$, index$)
Set the selected item by its index.

func ComboBox_Del(combobox$, index$)
Delete an item from the combobox list.

func ComboBox_Get(combobox$, index$)
Return a combobox list item caption.

proc ComboBox_Set(combobox$, index$, text$)
Set a combobox list item caption.

proc ComboBox_CopyToClipboard(combobox$)
Copy the selected edit portion of a combobox text to the clipboard.

proc ComboBox_CutToClipboard(combobox$)
Cut the selected combobox text to the clipboard.

proc ComboBox_PasteFromClipboard(combobox$)
Paste the clipboard text replacing the edit portion of a combobox selection.

proc ComboBox_Undo(combobox$)
Undo last change in edit portion of combobox.

proc ComboBox_LoadFromList(combobox$, slist$)
Set the combobox item list from a list variable.

proc ComboBox_SaveToList(combobox$, slist$)
Save the combobox item list to a list variable.

proc ComboBox_LoadFromFile(combobox$, Filename$)
Load the combobox list items from a file.

proc ComboBox_SaveToFile(combobox$, Filename$)
Save the combobox list items to a file.

Listbox:

proc ListBox_Clear(listbox$)
Clear all listbox items.

func ListBox_Count(listbox$)
Return the number of listbox items.

func ListBox_Add(listbox$, text$)
Add a new listbox item.

func ListBox_Insert(listbox$, index$, text$)
Insert a listbox item at (index$).

func ListBox_Del(listbox$, index$)
Delete the listbox item at index$.

func ListBox_GetSel(listbox$)
Return the selected listbox item.

func ListBox_GetSelCount(listbox$)
Return the number of items selected in the listbox.

proc ListBox_SetSel(listbox$, index$)
Set the selected listbox item.

func ListBox_Get(listbox$, index$)
Return a listbox item caption.

proc ListBox_Set(listbox$, index$, text$)
Set a listbox item caption.

proc ListBox_SelectAll(listbox$)
Select all listbox items.

proc Listbox_LoadFromList(listbox$, slist$)
Load listbox items from a list variable.

proc ListBox_SaveToList(listbox$, slist$)
Save the listbox items to a list variable.

proc ListBox_LoadFromFile(listbox$, filename$)
Load the listbox items from a file.

proc ListBox_SaveToFile(listbox$, filename$)
Save the listbox items to a file.

ListView:

nfunc ListView_SetColumnOrder

func ListView_Count(ListView$)
Return the number of items.

func ListView_Clear(ListView$)
Clear all items.

func ListView_GetSelCount(ListView$)
Return the number of items selected.

func ListView_GetColumnCount(ListView$)
Return the number of columns.

func ListView_GetSel(ListView$)
Return the selected item.

proc ListView_SetSel(ListView$, index$)
Set the selected item.

func ListView_IsSelected(ListView$, index$)
Return wheter the item is selected or not.

proc ListView_Select(ListView$, index$)
Select an item without deselecting the others.

func ListView_Get(ListView$, index$, item$)
Return the item caption.

func ListView_Set(ListView$, index$, item$, str$)
Set the item caption.

func ListView_AddColumn(ListView$, index$, str$, width$, fmt$)
Add a new column to the listview.

func ListView_DelColumn(ListView$, index$)
Delete a column from the listview.

proc ListView_Add(ListView$, text$)
Add a new item.

proc ListView_Insert(ListView$, index$, text$)
Insert a new item at (index$).

func ListView_Del(ListView$, index$)
Delete an item.

proc ListView_SelectAll(ListView$)
Select all items in the listview.

proc ListView_LoadFromList(ListView$, list$)
Load items from a list variable.

proc ListView_SaveToList(ListView$, list$)
Save items to a list variable.

proc ListView_LoadFromFile(ListView$, Filename$)
Load items from a file.

proc ListView_SaveToFile(ListView$, filename$)
Save items to a file.

ProgressBar:

func ProgressBar_SetRange(progressbar$, min$, max$)
Set the progress bar range of values.

func ProgressBar_GetRange(progressbar$, min$, max$)
Return the progress bar range of values.

func Progressbar_Get(progressbar$)
Return progress bar position.

proc Progressbar_Set(progressbar$, index$)
Set progress bar position.

proc Progressbar_SetStep(progressbar$, stepsize$)
Set the number of steps between each value.

func Progressbar_StepIt(progressbar$)

func Progressbar_StepDown(progressbar$, stepsize$)

TrackBar:

proc Trackbar_Clear(trackbar$)
Clear the trackbar selection.

proc Trackbar_SetRange(trackbar$, min$, max$)
Set the trackbar range of values.

proc Trackbar_GetRange(trackbar$, min$, max$)
Return the trackbar range of values.

proc Trackbar_SetTickFreq(trackbar$, tickfreq$)

proc Trackbar_SetSelStart(trackbar$, selstart$)
Set the trackbar selection start position.

proc Trackbar_SetSelEnd(trackbar$, selend$)
Set the trackbar selection end position.

func Trackbar_GetSelStart(trackbar$)
Return the trackbar selection start position.

func Trackbar_GetSelEnd(trackbar$)
Return the trackbar selection end position.

func Trackbar_Get(trackbar$)
Return trackbar position.

proc Trackbar_Set(trackbar$, index$)
Set the trackbar position.

proc Trackbar_StepUp(trackbar$, step$)

proc Trackbar_StepDown(trackbar$, step$)

UpDown:

func UpDown_GetBuddy(updown$)
Return buddy control the updown control is attached to.

proc UpDown_SetBuddy(updown$, buddy$)
Attach a buddy control to updown control.

func UpDown_Get(updown$)
Return the updown control position.

proc UpDown_Set(updown$, position$)
Set the updown control position.

proc UpDown_GetRange(updown$, min$, max$)
Return the updown control range of values.

proc UpDown_SetRange(updown$, min$, max$)
Set the updown control range of values.

TreeView:

func TreeView_Count(TreeView$)
Return the number of items in treeview.

func TreeView_Clear(TreeView$)
Clear all items from treeview.

func TreeView_GetSel(TreeView$)
Return selected item from treeview.

proc TreeView_SetSel(TreeView$, handle$)
Select an item from treeview.

func TreeView_IsSelected(TreeView$, handle$)
Return wheter the item is selected or not.

proc TreeView_Select(TreeView$, handle$)
Select an item without deselecting the others.

func TreeView_Get(TreeView$, handle$)
Return a treeview item caption.

func TreeView_Set(TreeView$, handle$, str$)
Set a treeview item caption.

func TreeView_Add(TreeView$, parenthandle$, text$)
Add a new treeview item.

proc TreeView_Insert(TreeView$, parenthandle$, text$)
Insert a treeview item before (parenthandle$).

func TreeView_Del(TreeView$, handle$)
Delete treeview item.

proc TreeView_SelectAll(TreeView$, parent$)
Select all treeview items.

func TreeView_First(TreeView$, parent$)
Return handle of first item in treeview root or from a parent item.

func TreeView_Next(TreeView$, handle$)
Return next item handle after handle$.

proc TreeView_LoadFromList(TreeView$, list$)
Load a items from a list variable.

proc TreeView_SaveToList(TreeView$, list$)
Save items to a list variable.

proc TreeView_LoadFromFile(TreeView$, Filename$)
Load items from a file.

proc TreeView_SaveToFile(TreeView$, filename$)
Save items to a file.

Rebar:

proc Rebar_Del(rebar$, band$)
Delete a band.

func Rebar_Count(rebar$)
Return number of bands.

proc Rebar_Add(rebar$, caption$)
Add a new band.

proc Rebar_Insert(rebar$, caption$, index$)
Insert a band at (index$).

StatusBar:

proc StatusBar_Set(statusbar$, caption$)
Set simple text of a status bar.

func StatusBar_Get(statusbar$)
Return simple test of a status bar.

Image and Icon:

proc Image_Set(image$, handle$)
Set an image handle.

func Icon_Set(icon$, handle$)
Set an icon handle.

ScrollBar:

proc ScrollBar_SetRange(scrollbar$, min$, max$)

Set scroll bar range of values.

proc ScrollBar_GetRange(scrollbar$, min$, max$)
Return scroll bar range of values.

func ScrollBar_Get(scrollbar$)
Return scroll bar position.

proc ScrollBar_Set(scrollbar$, index$)
Set scroll bar position.

Tab control:

proc Tab_Clear(tab$)
Clear all tabs from the tab control.

proc Tab_Del(tab$, index$)
Delete the tab at (index$).

func Tab_GetSel(tab$)
Return selected tab.

proc Tab_SetSel(tab$, index$)
Set the selected tab.

func Tab_Count(tab$)
Return the number of tabs.

proc Tab_Add(tab$, caption$)
Add a new tab.

proc Tab_Insert(tab$, caption$, index$)
Inser a new tab at (index$).

Image & Icon loading functions:

func Image_Load(filename$)
Load a bitmap image file and return its handle.

func Icon_Load(filename$)
Load an icon file and return its handle.

Conditional Compiling

PPL offers some very usefull compiler switches to allow conditional compiling. #IFDEF, #IFNDEF, #ELSE, #ELSEIF and #ENDIF will give you the ability to compile certain part of your code based on criterias you define.

Example:

```
#define demo

…

#ifdef demo
ShowMessage(This feature is locked!);
#elseif Shareware
ShowMessage(This feature will available once you buy the program!);
#else
Gotomap(Level2);
#endif
```

TRY / EXCEPT / FINALLY

```
try
  Statement1
except
  Statement2
finally
  Statement3
end;
```

PPL Assembler (PASM)

When it comes time to get real (raw) speed, PPL cannot fully give it to you because it is an interpreted language. But, there is a solution. The PPL Assembler is a multi-platform assembler language, included right into PPL, that compiles the code right to machine code to give you the fastest possible code execution. The PPL Assembler is easy to learn, can run on any machine PPL is supported without rewriting the code, you can call internal PPL functions directly from the assembly code and you can access PPL variables directly too.

To prepare PASM code you need to use the ASM() function. This function will analyse your code and translate it to the target's machine binary code. The function will then return a pointer to the binary code which can then be used with the CallASM() function and freed later on with the FreeASM().

A PASM code needs to have a label named MAIN at all time.

To do register indexing in PASM it's pretty simple. All register indexing must be in brackets [ ] and the register must be followed by a + or - sign.

Example:

```
:main
mov r0, [r1+8]
mov [r0+4], 10
```

The PPL assembler uses 6 registers which are listed here with their corresponding co-processor registers:

	ARM	INTEL	
R0	r0	eax	
R1	r1	ebx	
R2	r2	ecx	
R3	r3	edx	
SP	sp	esp	Stack pointer
SF	r12	ebp	Stack frame pointer

*** Most of these registers are not garanteed to keep there values when operands DIV, ROL and ROR are called.**

*** Some of the PASM operands are very complex and can produce extra operands using temporary registers. You should not expect a direct conversion to binary code output of the target processor since the PASM tries to achieve 100% compatibility between its supported platforms.**

BYTE and **WORD** value movements is also supported by the PASM. However extra code is generated by the PASM on the INTEL platform to fill the destination register or memory location with zeroes first. This is done to offer 100% compatibility with other supported platforms.

Example:

```
// Create a PPL variable of type int that we can use inside our PASM code.
// We cannot use regular PPL variables because they are of type double by
default.

new(a$, tint);
// Assign a value to the variable.
a$ = 10;

// Assemble the following PASM code.
code$ = asm (SMALL, true, {
:main
  mov r0, [a$]
:label
```

```
  add r0, 1
  cmp r0, 13
  savesp
  pplpush r0
  ppl ShowMessage
  jlt label
});

// Execute the PASM code.
if (code$)
  CallAsm(code$);
  // Free the PASM code from memory.
  FreeAsm(code$);
end;

// Free the PPL variable.
Free(a$);
```

Example 2:

```
// Working with PPL variables array

tdim(a$, tint, 100);
a$[2] = 10;

code$ = asm(SMALL, true, {
  mov r0, [a$, 8]           // a$[2]
  mov [a$, 4], 40           // a$[1] = 40
}

if (code$)
  CallAsm(code$);
  // Free the PASM code from memory.
  FreeAsm(code$);
end;

// Free the PPL variable array.
Free(a$);
```

Variables

Variables in PPL are defined at runtime as they are being used and can be cleared from memory at any time. PPL also uses no variable types, all the processing is done internally with only a minimal speed cost. PPL focus on easy of coding and flexibility instead of pure raw execution speed.

Variables are defined by two special character codes that are appended at the end of their name. $ and % are two symbols that defines local or global scope variables.

The $ is the local symbol and tell the interpreter to use the variable inside the current scope (procedure or main).

The % symbol defines global scope variables. These variables are never cleared from memory as long as the PPL program is still running. You can run multiple PPL code files within the same session and the global variables will not be erased. This behavior makes an excellent choice for passing values between different PPL programs.

PPL creates a few global variables when it is started.

Root%	The path where PPL is being run from.
Version%	Version number of current PPL build.
Hinstance%	The instance id of PPL.
NcmdShow%	
Argv%	Linked-list variable with all parameters passed to PPL.EXE.
LibPath%	Linked-list variable with paths to search with #include.

Thread% Current thread handle.
ThreadId% Current thread id.
Process% Current process handle.
ProcessId% Current process id.
Error% After each run or compile, PPL stores the error string into this variable. Runtime errors are also logged into this global variable. Be careful to store the value from this variable into a temporary variable because if you get out of the current scope (by calling another function or program), the error% variable will be cleared before the compiling.
Platform% This variable is set to PLATFORM_CE if the current running platform is a WINDOWS_CE machine, or 0 if it is running on a PC with Windows.
CS_DBLCLKS% Set to true during form initialization if you want a form to support double click functionality.

Some variables are also created at the program level everytime one starts.

AppName$ Name of the current program running.
AppPath$ Path of the current program running.
AppHandle$ Handle of the current program running.

Examples:

```
I$ = 10;
X$ = I$ + 20;
ShowMessage(I$);
A$, B$ = 10, 20;
ShowMessage(A$ + "," + B$);

I$ = "String";
ShowMessage(I$);

ShowMessage(root%);
```

You can clear variables from memory using the Clear function.

```
Clear(I$, X$);
```

In a case where you would like to use a variable name in a procedure that is already declared as global, you should use the local function. The local function will create a new variable inside the current scope of the procedure it is being called from.

Example:

```
Proc test
  Local (I$,X$);
  X$ = 30;
  ShowMessage(X$);
End;

Proc main
  Global(I$,X$);
  I$ = 10;
  X$ = 20;
  Test;
  ShowMessage(X$);
End;
```

You can define global variables anytime, anywhere. These variables will be available throughout the current program scope only.

Example:

```
Proc test
  Global(Y$);
  Y$ = 20;
  ShowMessage(X$);
End;

Proc main
  Global(X$);            // Make X$ global
  X$ = 30;
  Test;
  ShowMessage(Y$);
End;
```

You can obtain the size of a variable by using the Sizeof function.

Example:

```
S$ = "PPL STRING";
ShowMessage(sizeof(s$));
ShowMessage(length(s$));      // same thing when used on string
Dim(S$, 10);
ShowMessage(sizeof(S$));      // Size is 80. 10 x tdouble.
I$ = 10;
ShowMessage(sizeof(I$));      // Always 8 (double) when used with numeric
values.
```

The GetVar function shows nicely the flexibility of PPL, it search for a variable accessible within the current program scope. If the variable is not found, GetVar will create a new one automaticly.

Example:

```
S_1$ = "Hello World!");
ShowMessage(GetVar("s"+"_1$"));
```

You can also search a variable to see if it can be found from within the current scope or not. Use the VarExists function to do this.

Example:

```
If (VarExists("Z$"))
  ShowMessage("Exists!");
Else

  ShowMessage("Doesn't Exists!");
End;
```

You can increment and decrement variables values by simply adding a ++ or a - statement after a variable name.

Example:

```
I$ = 10;
I$++;
ShowMessage(I$);            // I$ is now 11.
I$--;
ShowMessage(I$);            // I$ is now 10.
ShowMessage(++I$);       // Display 11 and I$ is now 11.
ShowMessage(I$--);       // Display 11 and I$ is now 10.
```

You can also find out what type a variable is by using the VarType() function. Even if PPL allows for transparent variable type handling, internally it knows what value type a variable holds.

Variable Types:

_Numeric
_String
_Array
_Struct
_List

Example:

```
A$ = 10;
S$ = "String";

ShowMessage(VarType(a$)+", "+VarType(s$));
```

PPL also supports, what we will call, multiple variables assignment. The trick is to pass the variables separated by commas before the = operator. You have to make sure that you return enough values to satisfy the assignment of all variables. If you don't pass enough values, the remaining variables will be assigned a 0 value.

Example:

```
func test:2
  return (10, 20);
end;

proc main
  a$, b$, c$ = 10, 20, 30;
  a$, b$, c$ = test, 30;
end;
```

In a case where you would need to access a global variable within a proc or func that already has a local variable with the same name, all you need to do is prefix the variable name with Global. .

```
func mytest(g$)
  showmessage(g$); // Show 10
  showmessage(Global.g$) // Show 20
end;

proc main
  Global(g$);
  g$ = 20;
  mytest(10);
end;
```

LOCAL / GLOBAL

local (varlist,…);
global (varlist,…);

Make variables local to the current program scope. Variables can be defined as local or global inside a program in PPL. Global variables are accessible throughout the program but not the other programs. By making a variable local, it is only accessible to the current procedure or function scope.

Local and Global statements must be placed at the very beginning of a function or procedure.

Example:

```
Proc test
  Local (v$);
  v$ = 10;
  ShowMessage(v$);
End;

Proc main
  Global(v$);
  v$ = 20;
  ShowMessage(v$);
end;
```

void STATIC([any Var...])

Marks memory allocated to a variable so that it will not be freed when the variable is destroyed

Parameters

Var {in}
One or more variables whose memory you wish to retain once the variable is destroyed

Example:

```
func testalloc
  // Create a memory block of 1024 bytes. If this
  // variable is freed now, the memory block will be freed as well.
  new(s$, 1024);
  s$[0] = 1;
  s$[1] = 2;

  // After calling STATIC, the variable will be deleted
  // at the end of the proc but not it's allocated memory block.
  static(s$);

  //returns the pointer to the memory block allocated to s$
  return(&s$);
end;

proc main
  ptr$ = testalloc; // Return a pointer
  s$ = @ptr$;  // Assign pointer content of static variable to s$.
  ShowMessage(s$[0] % s$[1]); // Should display 12
  Free(ptr$);
end;
```

Notes:

- The PPL garbage collection system won't free memory marked with STATIC
- When declaring a variable STATIC, make sure to keep track of the memory block's address
- Memory marked with STATIC must be freed manually, or it will cause memory leaks

boolean ISNULL(string Variable)

Determines if *Variable* is a null string or not

Parameters

Variable {in}
A string that is possibly null

Return Value

ISNULL returns true if *Variable* is a null string, false otherwise

Example:

```
fn$ = GetFile("PPL Files (*.ppl)|*.PPL|PPC Files (*.ppc)|*.PPC|All Files (*.*)
|*.*");
if (isnull(fn$))
  Filename$ = fn$;
  LoadText;
end;
```

int SIZEOF(any Variable)

Returns the size of the variable as defined by the user.

Parameters

Variable {in}
  Memory location to find the size of

Return Value

SIZEOF returns the user defined size of *Variable*

Example:

```
new(s$, 1024);
&s$ = "This is string 1";
ShowMessage("sizeof: " + sizeof(s$)); //Displays 1024
ShowMessage("size: " + size(s$));         //Displays 1032
free(s$);
```

Notes:

- To get the actual amount of memory the variable occupies, use SIZE instead

See Also: SIZE, MEMSIZE

int SIZE(any Variable)

Returns the size in bytes occupied by *Variable* in memory.

Parameters

Variable {in}
  Memory location to find the size of

Return Value

SIZE returns the actual size of *Variable*

Example:

```
new(s$, 1024);
&s$ = "This is string 1";
ShowMessage("sizeof: " + sizeof(s$)); //Displays 1024
ShowMessage("size: " + size(s$));         //Displays 1032
free(s$);
```

See Also: SIZEOF, MEMSIZE

int ADDR(any Variable)

Returns the address in memory of *Variable*

Parameters

Variable {in}
  Variable you want to locate in memory

Return Value

ADDR returns the memory location of *Variable* as an integer

Example:

```
a$ = "Hello";
i$ = addr(a$);    // i$ is now the integer address of a$ where "Hello" is
stored.
ShowMessage(@i$); // Displays "Hello"
x$ = &a$;      // Same thing but store address in x$.
```

See Also: @, PTR

void CLEAR(any Variable, ...)

Resets the type and value of *Variable*

Parameters

Variable {in | out}
Variable can be one or more variables that you wish to reset

Example:

```
type(i$, TINT);
i$ = 10;
ShowMessage(VarType(i$) + ", " + i$);  //Displays "1, 10"
clear(i$);
ShowMessage(VarType(i$) + ", " + i$);  //Displays "0, 0"
```

Notes:

- CLEAR does not remove the variable from memory

See Also: EMPTY

void STRUCT(any Variable, struct Structure, [...])

Defines *Variable* as being of type *Structure*

Parameters

Variable {out}
Item that you wish to define as a structure

Structure {in}
Can either be a constant that has been #defined as a structure, or a set of fields separated by commas, or a set of field / size pairs separated by commas

Example:

```
#define TRect ("left", "top", "bottom", "right")

struct (rect$, TRect);

rect.left$ = 0;
rect.right$ = 100;
rect.top$ = 0;
rect.bottom$ = 10;

ShowMessage(rect.left$ + "\n" + rect.right$ + "\n" + rect.top$ + "\n" +
rect.bottom$);

struct (a$, "Field1", TByte, "Field2", TDouble, "Field3", 50);

a.Field1$ = 10;
a.Field2$ = 2983.2823;
a.Field3$ = "STRING";
```

```
ShowMessage(a.Field1$ + "\n" + a.Field2$ + "\n" + a.Field3$);
```

Notes:

- Internal types include: TBYTE, TSHORT, TWIDE, TINT, TUINT, TDOUBLE, TLONG

See Also: RESTRUCT

void DIM(any Variable, [int Dimensions...])

Dimension an array. Each array elements are of TDouble size (8 bytes).

Parameters

Variable {out}
Variable you wish to turn into an array

Dimensions {in}
One or more integers defining the size of each dimension of the array

Example:

```
Dim(a$, 10, 10, 10);
ShowMessage(a$[1, 1, 1]);
```

See Also: SDIM, TDIM, REDIM

void SDIM(array Variable, int ElementSize, [int Dimensions...])

Dimensions *Variable* as an array where each element is of size *ElementSize*

Parameters

Variable {out}
Variable you wish to turn into an array

ElementSize {in}
Size in bytes of each element in the array

Dimensions {in}
One or more integers defining the size of each dimension of the array

Example:

```
SDIM(a$, TBYTE, 10, 10, 10);
a$[1, 1, 1] = 50;
```

See Also: DIM, TDIM, REDIM

void TDIM(array Variable, [int Dimensions...])

Dimension *Variable* as an array where each element is the structure defined by *Variable*.

Parameters

Variable {in | out}
Structure you wish to turn into an array

Dimensions {in}
One or more integers defining the size of each dimension of the array

Example:

```
struct (a$, "x", "y");
TDIM(a$, 10);

a.x$[5] = 10;
a.y$[5] = 10;
```

Notes:

- The structure is maintained intact.

See Also: DIM, SDIM, REDIM

void REDIM(array Variable, [int Dimensions...])

Resize the dimensions of *Variable* while keeping the old values in place

Parameters

Variable {in | out}
  Array variable you wish to resize

Dimensions {in}
  One or more integers defining the size of each dimension of the array

Example:

```
dim(a$, 10);
a$[0] = 10;
a$[1] = 20;
a$[2] = 30;

redim(a$, 20);

ShowMessage(a$[0]);      // 10
ShowMessage(a$[1]);      // 20
ShowMessage(a$[2]);      // 30
```

See Also: DIM, SDIM, TDIM

any GETVAR(string VarName)

Gets or creates a reference to the specified variable

Parameters

VarName {in}
  Name of the variable you wish to retrieve

Return Value

GETVAR returns a reference to *VarName* if it exists, and if *VarName* doesn't exist GETVAR creates an instance of the variable and initializes it to 0

Example:

```
var1$ = "blah";
ShowMessage(var1$);                   //displays the message "blah"
ShowMessage(GetVar("var1$"));             //displays the message "blah"
ShowMessage(GetVar("var2$"));             //displays the message "0"
if(VarExists("var2$"))                   //displays the message "var2: 0"
  ShowMessage("var2: " + var2$);
else
  ShowMessage("No var2");
end;
if(VarExists("var3$"))                   //displays the message "No var3"
  ShowMessage("var3: " + var2$);       // because var3 hasn't been created
else                                     // in any way
  ShowMessage("No var3");
end;
```

See Also: VAREXISTS

boolean VAREXISTS(string VarName)

Determine if a variable exists or not

Parameters

VarName {in}

Name of the variable you wish to search for

Return Value
VAREXISTS returns true if *VarName* is defined, and false if *VarName* is not defined

Example:

See GETVAR for an example

See Also: GETVAR

boolean ISVALIDVAR(string VarName)

Determines if *VarName* is valid to use as a variable name or not

Parameters

VarName {in}
Name you wish to validate

Return Value
ISVALIDVAR returns true if *VarName* is a valid name for a variable, and false otherwise

Example:

```
IsValidVar(234);            //returns false
IsValidVar("A");             //returns false
IsValidVar("A$");       //returns true
```

Notes:

- Basically, a valid variable name is a string that ends in $ or %

See Also: VAREXISTS, GETVAR

int VARTYPE(any Variable)

Determines the type of a given variable

Parameters

Variable {in}
Item to determine the type of

Return Value
VARTYPE returns one of the following values: 0 (_Numeric), 1 (_String), 2 (_Array), 3 (_Struct), 4 (_List), 5 (_Matrix)

Example:

```
a$ = 10;
b$ = 5.5;
c$ = "This is a string";
d$ = 'a';
list(e$);

ShowMessage(VarType(a$) + ", " + VarType(b$) + ", " + VarType(c$) + ", " +
VarType(d$) + ", " + VarType(e$));
//The result will be a dialog displaying the string "0, 0, 1, 0, 4"
```

Notes:

- A variable that has not been assigned a value will default to type 0 (_Numeric)

See Also: ISVALIDVAR, LTYPE

void TYPE([Variables...], int Size)

Change the type of a *Variable*

Parameters
Variables {in | out}
One or more variables to change the type of

Size {in}
Size in bytes to make each of the variables; to convert to an internal type, use one of the constants listed in the Notes section

Example:

```
type(startpos$, endpos$, TINT);
PostMessage(e$, EM_GETSEL, &startpos$, &endpos$);
ShowMessage(startpos$);

type(a$, 50);
a$ = "NEW STRING";
```

Notes:

- The standard internal types are: TBYTE, TSHORT, TWIDE, TINT, TUINT, TDOUBLE, TLONG
- This does not improve performance, but it will improve memory efficiency
- Allows for easier porting of code from another language to PPL

See Also: VARTYPE

void LIST (any Variable)

Initializes a variable as a linked list

Parameters
Variable {out}
Item you wish to designate as a list

Example:

```
list(lst$);
i$ = 1;
while(i$ <= 10)
  add(lst$, i$);
  i$++;
end;
s$ = "";
first(lst$);
foreach(lst$)
  s$ = s$ + lst$ + ",";
end;
ShowMessage(s$);  //displays the string "1,2,3,4,5,6,7,8,9,10,"

goto(lst$, 5);
ins(lst$);
lst$ = 5.5;
s$ = "";
first(lst$);
foreach(lst$)
  s$ = s$ + lst$ + ",";
end;
ShowMessage(s$);  //displays the string "1,2,3,4,5,5.50,6,7,8,9,10,"
```

See Also: ADD, DEL, INS

void LCOPY(list From, list To)

Copy the contents of *From* to the variable *To*

Parameters

From {in}
The list to copy values from

To {in | out}
The list to copy values to

Example:

```
list(a$, b$);
add(a$, 1, 2, 3);
add(b$, "a", "b", "c");
first(a$);
goto(b$, 1);
copy(a$, b$);
s$ = listtostr(b$, ",", "", "");
ShowMessage(s$); //Displays "a,1,c"
lcopy(a$, b$);
s$ = listtostr(b$, ",", "", "");
ShowMessage(s$); //Displays "1,2,3"
```

Notes:

- The regular assignment operator = will not work with lists
- LCOPY copies **all** elements from the source list to the destination list. To copy a single element, use COPY

See Also: COPY

void EMPTY([any Variable...])

Frees the contents of the variable, but does not remove the variable's attributes (ex: if it's a list, it will still be a list)

Parameters

Variable {in | out}
One or more defined variables to clear the contents of

Example:

```
ShowMessage(VarType(l$));            //Displays 0, for an untyped variable
add(l$, "PPL", "IS", "COOL!");
ShowMessage(VarType(l$));            //Dispalys 4, for a list type variable
empty(l$);
ShowMessage(VarType(l$));            //Dispalys 4, for a list type variable
```

See Also: CLEAR

FREEOBJECT ([ObjAddress ...])

Free an object by it's address from memory. This function is useful when you have an array filled with object addresses.

Example:

```
#class myclass
  public(value$);

  nproc create
    value$ = args$[0];
  end;

  proc destroy
    ShowMessage("Destroy " + value$);
  end;
#endclass

proc main
```

```
  dim(l$, 20);
  for (i$, 1, 10)
    #object myclass o$(i$);
    l$[i$] = &o$;
  end;
  for (i$, 1, 10)
    freeobject(l$[i$]);
  end;
end;
```

CLASSES (List) -> count

Return a list of all the class names defined within the current application.

CLASSINHERIT (ClassName) -> InheritedClassName

Return the class name of the inherited class of (ClassName).

ASSIGN (ObjectVar, Address)

Assign a memory location (Address) to an object (ObjectVar).

Procedures and Functions

Functions must have a definition and should have a declaration. The function definition includes the function body the code that executes when the function is called.

A function declaration establishes the name and attributes of a function that is defined elsewhere in the program. A function declaration must precede the call to the function.

The compiler uses the prototype to compare the types of arguments in subsequent calls to the function with the functions parameters.

A function call passes execution control from the calling function to the called function. The arguments, if any, are passed by value to the called function. Execution of a return statement in the called function returns control and possibly a value to the calling function.

PROC / FUNC

```
proc Name ( parameters )
  Statement1
end;

func Name ( parameters )
  Statement1
end;
```

Procedures and functions are declared using the PROC or the FUNC statement. Each procedure or function must be terminated with an end statement. PROC and FUNC must not be finished by an ; operator. Functions must return a value using the return statement.

Example:

```
Func test (p1$, p2$)
  If (p1$ == 10)
    Return(false);
  End;
  ShowMessage(p1$ + p2$);
  Return(true);
End;

Proc test2
  ShowMessage("Test2");
End;

Proc main
  I$=20;
  If (Test(10, I$))
    ShowMessage("Worked!");
  End;
  Test2;
End;
```

The return function exits from the function and put return values on the stack before exiting.

You can return any variable types from a function, however the return value will be a pointer and not the actual variable. You will need to use the &a$ = myFunction; to assign a new pointer to an existing variable type that matches the variable returned from the function.

Example:

```
func teststr
  s$ = "TEST STRING";
  return (s$);
end;

func testarray
  dim(a$, 10);
  a$[1] = "TEST ARRAY!";
  return (a$);
end;

func teststruct
  struct(a$, "a", "b");
  a.a$ = "TEST STRUCT";
  return (a$);
end;

func testmatrix
  a$ = [10, "TEST MATRIX", 20];
  return (a$);
end;

proc main

  // Test string return
  s2$ = teststr;
  ShowMessage(s2$);

  // Test array return
  dim(s3$, 10);
  &s3$ = testarray;
  ShowMessage(@s3$[1]);

  // Test structure return
  struct(s4$, "a", "b");
  &s4$ = teststruct;
  ShowMessage(@s4.a$);

  // Test matrix return
  &a$ = testmatrix;
  ShowMessage(@a$[1]);
end;
```

The difference between a Proc and a Func is that a Func always returns one or more value on the stack upon exiting.

You can get the address where a procedure byte-code is stored by using the & operator in front of the procedure name.

Example:

```
Proc test;
  ShowMessage("test procedure!");
End;

Proc main
  ShowMessage(&test);
End;
```

In order to pass variable pointers to procedures or functions parameters, you must pass the variables with the & operator in front.

Example:

```
Proc Test (v$)
  V$ = "Now another value!";
End;

Proc Main
  S$ = "Value of s$";
  ShowMessage(s$);
  Test(s$);
  ShowMessage(s$);
  Test(&s$);
  ShowMessage(s$);
End;
```

You cannot pass structure's or array's elements as pointers in parameters.

Functions in PPL can also return multiple values. There is a special syntax to tell the compiler about such kind of functions. You need to add a : after the function definition followed by the number of values the function will return. The default number of return values for a normal function is always 1.

Example:

```
Func test (a$) : 2
  return (a$+1, a$+2);
end;

Func test (a$) : 3
  return (a$+1, a$+2, a$+3);
end;

Proc main
  a$, b$ = test (10);
  ShowMessage(a$, ", ", b$);
  a$, b$, c$ = test (10):2, 30;
  ShowMessage(a$, ", ", b$, ", ", c$);
end;
```

You can also call a specific function with a specific number of output by using the : operator after the call to the function followed by a numeric value. In the previous example, we explicitely call the function test that returns 2 output arguments in the second line of the main procedure.

To understand passing variables as pointers to a procedure or function, we will try to understand the following example code.

```
proc test2 (l2$);
  struct(l2$, "a", "b");      // Still directly points to L1$, creates structure
using L1$.
  l2.a$ = 10;                  // This is like doing L1.A$ but without the
global().
  l2.b$ = 20;
end;

proc test3 (l3$)
  test2(&l3$);                  // Physically points to L1$, pass L1$ pointer
end;

proc main
  test3(&l1$);                  // Pass pointer of L1$.
  ShowMessage(l1.a$);
end;
```

L1$ is our main variable here and we pass it's pointer to the test3 procedure. Test3 pass L3$ (which physically points to L1$) to the procedure Test2. In Test2, L2$ is pointing directly to L1$ and not L3$ since we passed L3$ as a pointer which was already pointing to L1$. By creating a structure using L2$ (remember that L2$ is pointing to L1$), therefore the structure element variables are created at the L1$ variable level, that is why L1.A$ in the main procedure exists.

RETURN (Arguments...)

Return from a procedure or a function

If you return from a function, you can pass an unlimited number of return values. The return operator supports multiple type of variables including:

Strings
Numbers
Arrays
Matrix
Objects

Example:

```
func test
  return(10);
end;

func test2:2
  return (10, 20);
end;

proc main
  a$ = test;
  a$, b$ = test2;
end;
```

RETURN does not support linked-list variable types. Only the current element value will be returned if you pass a linked-list as a return value. To return a list, you must use a list in the procedure or function parameters declaration.

Example:

```
proc testlist (l$)
  list(l$);
  add(l$, 10, 20, 30);
end;

proc main
  testlist(&l$);
  ForEach(l$)
```

```
    ShowMessage(l$);
  end;
end;
```

NPROC / NFUNC

nproc Name
  Statement1
end;

nfunc Name
  Statement1
end;

Procedures and functions, with an unknown number of input parameters, are declared using the NPROC or the NFUNC statement. Each procedure or function must be terminated with an end statement. NPROC and NFUNC must not be finished by an ; operator. Functions must return a value using the return statement.

A new list variable called ARGS$ will be created that will hold the parameter values passed to the it. The ARGS$ variable will only contain the variable pointers passed if they are anything else than a string or a numerical value.

Example:

```
nfunc test
  foreach(args$)
    t$ = t$ + args$;
  end;
  return (t$);
end;

proc main
  ShowMessage(test(10, 20, 30, 40, 50));
end;
```

Example:

```
nproc test
  struct(a$, "a", "b");
  &a$ = ARGS$[0];
  ShowMessage(a.a$);

  dim(b$, 10);
  &b$ = ARGS$[1];
  ShowMessage(b$[1]);
end;

proc main
  struct(a2$, "a", "b");
  a2.a$ = 10;
  dim(b2$, 10);
  b2$[1] = 20;
  test(a2$, b2$);
end;
```

FORWARD

Declare a function or procedure before it is defined

PPL offers a way, just like in C or Pascal, to declare a procedure or a function before it's actual body. This way you can access procedures or functions before they are defined.

Example:

```
forward proc test (x$)
```

```
proc test2
  test(10);
end;

proc test (x$)
  ShowMessage(x$);
end;

proc Main
  test2;
end;
```

OVERRIDE

Redefine an existing function or procedure

PPL provides the ability to overwrite functions or procedures that have already been defined. This allows you to redefine procedures or functions contained in a library without modifying the library itself.

Example:

```
proc test (a$, b$)
  ShowMessage(a$+b$);
end;

override proc test (a$, b$)
  ShowMessage(a$-b$);
end;

proc main
  Test(50, 20);      // result is 30
end;
```

{addr} GETPROC(HANDLE App, string Name)

Search for a procedure or function by it's name

The return value is the address where it is located in memory. The result will be null if PPL couldn't find it.

Parameters

App {in}
   Handle of the application to search. A value of **null** indicates the current application

Name {in}
   The name of the proc / func you are searching for

Return Value

GETPROC returns the address of the proc / func if found, or **null** otherwise

See Also: CURPROC

{return} CALL(HANDLE App, {addr} Address, [any Parameters...])

Execute a function or procedure using it's memory address

Parameters
App {in}
   Handle to the application where the function resides; use NULL for the current application

Address {in}
   Memory location of the function being called; can be retrieved using GETPROC

Parameters {in | out}
   One or more values or variables to be passed to the function

Return Value
CALL returns whatever the return value of the called function is (see the example for more clarification)

Example:

```
func test (x$, y$)
  return (x$+y$);
end;

proc main
  r$ = Call(NULL, &test, 10, 20);  //r$ = 30
  r$ = Call(NULL, GetProc(NULL, "test"), 30, 40); //r$ = 70
end;
```

See Also: GETPROC

Stack

The stack is local to each program running. There are no new stacks created for each procedure or function call. The stack is just like variables where all types are handled internally by the interpreter. PPL also comes with it's own functions to manipulate the stack.

void PUSH([any Value...])

Place one or more values on the stack

Example:

```
// Push and pull test

Push(10, 20, 30);
Push(10, 20, 30, 40, 50);
Pull(a$, b$, c$, d$, e$);
ShowMessage(a$+","+b$+","+c$+","+d$+","+e$);
Pull(a$, b$, c$);
ShowMessage(a$+","+b$+","+c$);
```

Notes:

- Values PUSHed on the stack are removed with PULL

See Also: PULL

void PULL([Var1...])

Remove values from the stack

Parameters

Var1 {in}
One or more variables to place the values into

Example:

See PUSH for an example

See Also: PUSH

int COUNTSTACK(void)

Returns the number of items on the current scope stack.

Parameters

None

Return Value

COUNTSTACK returns the item count on the stack currently in scope

See Also: COUNTSTACK, DUPSTACK, DROPSTACK, CLEARSTACK

void DUPSTACK(void)
Duplicate the last item on the stack.

Parameters
None

Return Value
None

See Also: COUNTSTACK, DROPSTACK, CLEARSTACK

void DROPSTACK(void)
Drop the last item on the stack.

Parameters
None

Return Value
None

See Also: COUNTSTACK, DUPSTACK, CLEARSTACK

void CLEARSTACK(void)
Clears the content of the current scope stack.

Parameters
None

Return Value
None

See Also: COUNTSTACK, DUPSTACK, DROPSTACK

Compiler Switches

In order to control some of the areas of the compiler, PPL offers a unique set of compiler switches that will greatly improve your coding time and effort.

It is important to note that compiler switches won't accept an ; at the end of the line.

#LIBRARY

Libraries are handled in PPL the same way header files (.h) are handled in C except that a #LIBRARY compiler switch needs to be used inside the code file. Library files are not compiled as a .ppc file.

Example:

```
#LIBRARY

proc libraryprocedure
  ShowMessage("This procedure is available to any ppl code you use it!");
end;
```

#DEFINE / #UNDEFINE

You can define values to be used with the conditional compiling switches like #IFDEF, #IFNDEF, #ELSE, #ELSEIF and #ENDIF by using the #DEFINE switch. Refer to the conditional compiling section for an example.

Defining values is done using the #DEFINE switch. You can pass any values to your define line until the EOL is reached. The ; is included inside the define value. It is a good practice to incorporate all your values inside the { } brakets.

Example:

```
#DEFINE def {null, "Message", "Title", MB_OK}
MessageBox(def);
```

You can also undefine something using the #UNDEFINE directive.

Example:

```
#UNDEFINE rect
```

You can declare external functions (from .dll) or api functions using the #DECLARE and #DECLAREAPI compiler switches. This switches are a little more complicated at first but are very easy to understand.

Tips:

If you want to append two or more defines within the same #define here is how you do it:

```
#define DEFINEA {"a", "b", tbyte}
#define DEFINEB {DEFINEA, {, "c", tshort, "d", tdouble"}}

struct(v$, DEFINEB);
v.a$ = 10;
v.c$ = 100;
```

#DECLARE / #DECLAREAPI / #C_DECLARE / #C_DECLAREAPI

#DECLARE *<pplname> <dllfile> <functioname> <input> <output>*
#DECLAREAPI *<pplname> <dllfile> <functioname> <input> <output>*

Import functions from external libraries. The standard call method is used for parameters (STDCALL). If your compiler doesn't support STDCALL you can use #C_DECLARE or #C_DECLAREAPI to use CDECL method instead.

Example:

#DECLARE *SetWindowText coredll.dll SetWindowTextW 3 1*

The SetWindowText parameter is the new name you want the function to have inside the PPL compiler. The coredll.dll parameter is the pathname to the .dll library. SetWindowTextW is the name of the function inside the library. 3 is the number of input parameters and 1 is the number of output parameters.

#GLOBAL

When you define values or declare external functions you need to tell the compiler is the scope of these new value or functions are to be global or not. If you don't issue a #GLOBAL compiler switch before the new defines or declares, the new values or functions will be removed from the compiler's tables after the current code being compiled is finished executing.

Example:

```
#LIBRARY
#GLOBAL

#define api "coredll.dll"
#declareapi SetWindowText api SetWindowTextW 3 1
```

#INCLUDE

Including a library inside a program is just as easy as it is in any other language. Use the #INCLUDE directive with the name of the file to include.

Example:

```
#INCLUDE "MyLib.ppl"
```

```
Proc Main
  ShowMessage(MyTestProc);
End;
```

Don't forget that the file you are including doesn't have to be a #LIBRARY. A #LIBRARY is a main program you compile that is not executed and only defines stuff.

One of the nice feature of PPL is that you don't need to use the #include very often. It is better to #define everything in a (#global, #library), program and then all programs being compiled after will inherit the defined values. This allow to keep programs very small and fast to compile.

#NOPPC

Tells the compiler not to automatically generate a .ppc file

#NOLINK

The PPL linker will remove all procedures declaration that are not being accessed by the program. But sometimes you need to keep all procedures, for example, if you use the GetProc() and Call() functions, use the **#NOLINK** directive in this case.

#IMPORT filename

Use the #import directive to import all COM enumerations from a .dll file or COM table definition file. All enumerations will become simple PPL defines.

#EXPLICIT

Forces the declaration of all variables prior to their use within a program

Example:

```
#explicit

local(s$, i$);

s$ = "PPL";
i$ = 10;
z$ = 20;                  // Crash here. Variable has not been defined.
```

Notes:

- Variables must be defined using one of the following: LOCAL, GLOBAL, PUBLIC or PRIVATE
- This only triggers validation at run time, not compile time

See Also: LOCAL / GLOBAL

Conditions and Loops

Just like any programming language, PPL supports condition evaluation and loops. However the End statement in PPL closes or finishes almost any loop statement or any conditional statements.

If / Else / End are used to evaluate values to execute conditional code.

Repeat / Until to loop until a condition is true.

While / End to loop while a condition is true.

Case / <expr>: / End to run conditional code on multiple conditions.

Break is used to exit a loop or a case statement.

Continue will go back to the beginning of a loop (next iteration).

Examples:

```
I$=20;
If (I$ == 10)
```

```
  ShowMessage("I$=10");
Else if (I$ <> 20)
  ShowMessage("I$ <> 20");
Else
  ShowMessage("I$ is unknown");
End;

I$=20;
B$=5;
Case(I$)
  10:
    ShowMessage("I$=10");
  20:
    ShowMessage("I$=20");
    Case(B$)
      5:
        ShowMessage("B$=5");
      10:
        ShowMessage("B$=10");
    End;
End;

I$=0;
Repeat
  I$=I$+1;
  If (I$>=500)
    Break;
Until(I$>=1000);


I$=0;
While(I$<=1000)
  I$++;
  If (I$>=500)
    Break;
End;
```

Conditions are evaluated using a series of special operators:

```
==, <>, <, >, <=, >=, AND, &, OR, |, XOR, NOT, (), ...

If (((I$ >= 10) AND (I$ <= 20)) OR (I$ == 30))
  ShowMessage("Between 10 and 20 or equals to 30");
End;
```

IF / ELSE

```
if ( expression )
  Statement1
[else
  Statement2]
end;
```

The if statement controls conditional branching. If the value of expression is nonzero, statement1 is executed. If the optional else is present, statement2 is executed if the value of expression is zero. An if statement must be terminated with an end statement even if it only as a one line statement.

Example:

```
If (I$ <> 10)
  ShowMessage("Not equal to 10!");
End;
```

```
If (I$ <= 10 or I$ >= 20)
  ShowMessage("Out of range!");
Else if (I$ == 15)
  ShowMessage("Right on target");
Else
  ShowMessage("Close");
End;
```

AND OR NOT

The logical AND operator returns 1 if both operands are nonzero; otherwise, it returns 0. Logical AND has left-to-right associativity. The logical OR operator returns 1 if either operand is nonzero; otherwise, it returns 0. Logical OR has left-to-right associativity.
The logical-negation (logical-NOT) operator produces the value 0 if its operand is true (nonzero) and the value 1 if its operand is false (0). The result has **int** type.
Example:

```
if (x$ == y$ and z$ == y$)
  ShowMessage(y$);
end;

if (not x$ or not y$)
  ShowMessage(x$);
end;
```

REPEAT / UNTIL / WHILE

```
repeat
  Statement1
  [break;]
  [continue;]
until ( expression );

while ( expression )
  Statement1
  [break;]
  [continue;]
end;
```

The while loop executes statement1 repeatedly until expression evaluates to zero. The test of expression takes place before each execution of the loop; therefore, a while loop executes zero or more times.
A while loop can also terminate when a break within the statement while the body is executed. Use continue to terminate the current iteration without exiting the while loop. continue passes control to the next iteration of the while loop.
Example:

```
i$ = 10;
Repeat
  i$++;
Until (i$>=10);

i$ = 0;
While (i$ <= 10)
  i$++;
end;
```

FOR

```
for (sourcevar, start, end, [increment])
  Statement1
  [break;]
  [continue;]
end;
```

The for loop will loop from start to end assigning the new value to variable sourcevar every iteration of the loop. The increment value is added or suctracted from the variable sourcevar value every iteration of the loop. The increment parameter is optional.

Example:

```
for (i$, 4, 10)
  ShowMessage(i$);
end;

for (i$, 10, 4, 2)
  ShowMessage(i$);
end;

for (i$, 10, 4, -1)
  ShowMessage(i$);
end;
```

FOREACH

foreach (sourcevar[, destvar])
Statement1
[break;]
[continue;]
end;

The foreach statement execute statement1 from the beginning of the list repeatedly until the end of the list. The variable sourcevar list pointer is always updated. If you specify a DestVar, you can read the value from the variable but if you set the DestVar value, the list won't be updated, it is always important to note that if you want to update the list element, use the SourceVar instead.

ForEach loops for lists can be used with only the sourcevar parameter, however if you want to use string values inside lists, it is better to pass the destvar and then use it with [index] ranges.

ForEach loops on arrays, the destvar will be set as a pointer to the current array element. Therefore you can get or set the value.

ForEach loops on matrices works the same as arrays. The DestVar is always set as a pointer to the current matrix element.

Example:

```
// Demo 1

List(l$);
Strtolist("First;Second;Third;Last", ";", l$);
ForEach (l$)
  ShowMessage(l$);
End;

// Working with strings index in a list
ForEach (l$, s$)
  ShowMessage(s$[0]+","+s$[1]);
  l$ = "New Value";      // Cannot use s$ here, it is not a pointer to the list
element.
end;

// Demo 2

#include "console.ppl"

proc winmain
  InitConsole;
  ShowConsole;

  dim(a$, 10, 10);
  foreach (a$, v$)
    v$ = i$;
    i$++;
  end;
  foreach (a$, v$)
    writeln(v$);
  end;
  return (true);
end;
```

FOREACHREV

foreachrev (sourcevar[, destvar])
Statement1
[break;]
[continue;]
end;

The foreachrev statement execute statement1 from the end of the list repeatedly until the first element of the list. The variable sourcevar list pointer is always updated. If you specify a DestVar, you can read the value from the variable but if you set the DestVar value, the list won't be updated, it is always important to note that if you want to update the list element, use the SourceVar instead.

ForEach loops for lists can be used with only the sourcevar parameter, however if you want to use string values inside lists, it is better to pass the destvar and then use it with [index] ranges.

ForEach loops on arrays, the destvar will be set as a pointer to the current array element. Therefore you can get or set the value.

ForEach loops on matrices works the same as arrays. The DestVar is always set as a pointer to the current matrix element.

Example:

```
// Demo 1

List(l$);
Strtolist("First;Second;Third;Last", ";", l$);
ForEachRev (l$)
  ShowMessage(l$);
End;

// Working with strings index in a list
ForEachRev (l$, s$)
  ShowMessage(s$[0]+","+s$[1]);
  l$ = "New Value";      // Cannot use s$ here, it is not a pointer to the list
element.
end;

// Demo 2

#include "console.ppl"

proc winmain
  InitConsole;
  ShowConsole;

  dim(a$, 10, 10);
  foreachRev (a$, v$)
    v$ = i$;
    i$++;
  end;
  foreachRev (a$, v$)
    writeln(v$);
  end;
  return (true);
end;
```

CASE

case (expression)
  value [, value, ...] :
    Statement1
  default:
   Statement2
end;

The case statement allows selection among multiple sections of code, depending on the value of expression. The case statement body consists of a series of values. The labeled statements are not syntactic requirements, but the case statement is meaningless without them. The values can be of pretty much any types supported by PPL.

Example:

```
I$ = 30;
Case (I$+10)
  10, 12, 13:
    ShowMessage("10");
  20, 21, 22:
    ShowMessage("20");
  default:
    ShowMessage("default");
End;

t$ = "boy";

case (t$)
  "toy":
    showmessage("Toy");
  "boy":
    showmessage("Boy");
end;
```

Using variables

There are two ways to use variables with the PPL Assembler. The first is to use PPL variables inside your assembly code. The second is to create temporary local variables using the assembler stack frame.

PPL Variables:

When using PPL variables you must aware that these variables needs to be created using the NEW(), STRUCT(), SDIM () or TDIM(). Regular PPL variables won't work in assembly code because their memory allocation is different from regular fixed sized memory assigned variables.

The PPL Assembler will translate the variable name to it's memory allocation pointer. This is what marks the difference between PPL variables and local assembler variables.

Example:

```
new(v$, 1024);

a$ = asm (1024, {
  mov r0, [v$]            // Move the value at address of v$ to r0.
  mov v$, 10            // Move 10 to address of v$.
});
```

Local Assembler Variables:

Local assembler variables are created using the VAR operand or when function parameter variables are used inside a PPL Assembler code. The variables memory allocation size is always 4 bytes. The use of local assembler variables is always indexed, therefore you cannot get the memory location of the variable on the stack frame. **Local variables declaration must always be inside a label or a function.**

Example:

```
a$ = asm(1024, {
  !myfunc (var1)
    var var2, var3
    mov var2, var1      // Move content of var1 into var2.
    add r0, var3            // Add var3 to the content of r0.
});
```

Mixing the two:

```
new(p$, TINT);
p$ = 10;

a$ = asm (1024, {
  !func (a)
    add [p$], a      // Add value of a (5) to value of p$ (10), store result in
p$.
  ret

  :main
    var Var1, Var2
    func(5)      // Result of p$ should be 15 now.
});

callasm(a$);

ShowMessage(p$);
```

ASM (Size, Code) -> PASMHandle

Assembles a code written in PASM form.

The size parameter will tell the assembler to allocate a specific amount of bytes to the code buffer. Make sure the size is aligned to 4 bytes on the PocketPC platform.

The Code parameter will hold the PASM code you want to assemble.

The return value is the location of the binary code in memory. If the code contains errors, the return value will be zero.

Predefined code buffer sizes:

SMALL	1k
MEDIUM	8k
LARGE	32k

CALLASM (PASMHandle, [Arguments...])

Calls a previously assembled PASM code buffer. Any arguments passed will be stored into an array named AARGS$ and the number of parameters will be stored in a variable called AARGSCOUNT$.

FREEASM (PASMHandle)

Free from memory a PASM code buffer.

!functionname ([param1, param2...])

Declares a function. The PASM will generate the necessary code for proper stack handling. The stack frame pointer will be updated and the stack pointer as well. A function must always be ended with a RET operand.

You can declare parameters when you define a function.

```
!function (parm1, parm2)
  mov r0, parm1
  sub r0, parm2
  ret

:main
  function(20, 10)
```

:label

This will declare a label. No internal code is generated. You can jump to any label at any position in your code.

**** NOTE: Labels are local to the function they are inside of.**

Example:

```
:main
   var Var1
   mov r0, 0

:loop
  add r0, 1
  cmp r0, 10
  jeq endloop
  jlt loop

:endloop
  mov Var1, r0
  savesp
  pplpush Var1
  ppl showmessage
```

VAR VarName

This operand will not generated any code but will allocate 8 bytes on the stack for variable storage which will be deallocated at function exit. A maximum of 128 variables can be used per function. The variables are initialized with 0 at the beginning of the code.

Example:

```
!function
  var Var1
  var Var2

  mov Var1, 10
  mov Var2, 20

  ret

:main
  jsr function
```

MOV [size] value1, value2

Move value2 to value1.

Possible syntaxes:

mov register1, register2	move value of register2 to register1
mov register1, [register2]	move value pointed by register2 to register1
mov [register1], register2	move value of register2 to pointer in register1
mov [register1], [register2]	move value pointed by register2 to pointer in register1

mov address, register1	mov value of register1 into pointer at address
mov address, value	mov value to pointer at address
mov [address], register1	mov value of register1 into pointer at address
mov [address], value	mov value to pointer at address
mov register1, value	mov value to register1
mov [register1], value	mov value to pointer in register1

The MOV operand also support BYTE and WORD size data movement.

Example:

```
new(v$, tbyte);
v$ = 10;

x$ = asm(100, {
  var Var1, Var2

  mov byte Var1, v$
  mov Var2, Var1
  add Var2, 5

  savesp
  pplpush Var2
  ppl showmessage            // Display 15
});

if (x$)
  callasm(x$);
  freeasm(x$);
end;

free(v$);
```

You can shift any register or address location by using the following syntax:

```
mov [r0-4], 10      // move 10 to pointer at location r0 - 4.
mov [a$+4], 20      // move 20 to pointer at location of variable a$ + 4.
```

The mov operand also support **BYTE** size and **WORD** size operations. It is imperative to not that byte or word size operations on registers is done on the full 32bits register, meaning that the whole register 32bits value is filled with zeroes first. This extra operation is only done on the Intel PC to keep 100% compatibility with the ARM (PPC) processors.

```
mov r0, 0xFFFFFFFF            // r0 = 0xFFFFFFFF
mov byte r0, 0xFFAAAAAA            // r0 = 0xFF000000

mov r0, 0xFFFFFFFF            // r0 = 0xFFFFFFFF
mov word r0, 0xFFFFAAAA            // r0 = 0xFFFF0000

mov r1, 0xFFFF                  // f1 = 0xFFFF0000
mov byte r0, r1                  // r0 = 0xFF000000
```

RET

Return from a subroutine. The PASM will generate the necessary stack cleanup code by itself.

ADD register, value

Add value to value in register and store the result in register.

<u>Possible syntaxes:</u>

add register1, register2	add value of register2 to register1
add register, value	add value to register
add register1, [register2]	add value at register2 address to register1
add [register1], register2	add value of register2 to value at address of register1
add [register], value	add value to value at address of register
add [register1], [register2]	add value at address of register2 to value at address of register1
add [register], [address]	add value at address to value at address of register
add [address], register	add value of register to value at address
add [address], value	add value to value at address
add [address1], [address2]	add value at address2 to value at address1

<u>Example:</u>

```
mov r0, 20            // r0 = 20
add r0, 10            // r0 = 30

mov [v$], 20      // v$ = 20
add [v$], 10      // v$ = 10
```

SUB register, value

Subtract value from value in register and store the result in register.

For possible syntaxes and examples, please refer to the ADD operand.

MUL register, value

Multiply value from value in register and store the result in register.

For possible syntaxes and examples, please refer to the ADD operand.

DIV register, value

Divide value by value in register and store the result in register.

For possible syntaxes and examples, please refer to the ADD operand.

ROL register, value

Rotate bits to the left by the number of times specified in value.

For possible syntaxes and examples, please refer to the ADD operand.

ROR register, value

Rotate bits to the right by the number of times specified in value.

For possible syntaxes and examples, please refer to the ADD operand.

AND value1, value2

For possible syntaxes and examples, please refer to the ADD operand.

XOR value1, value2

For possible syntaxes and examples, please refer to the ADD operand.

OR value1, value2

For possible syntaxes and examples, please refer to the ADD operand.

CMP value1, value2

Compare value1 to value2. This will set the compare flag to be used with the JSR or JMP operands.

For possible syntaxes and examples, please refer to the ADD operand.

SWP register1, register2

Swap the values of two registers. This operand only works on registers.

Possible syntaxes:

swp register1, register2	Swap value of register2 with value of register1

NEG value1

Reverse the sign of value1.

Possible syntaxes:

neg register	reverse sign of value in register
neg [register]	reverse sign of value at address of register
neg [address]	reverse sign of value at address

Example:

```
mov r0, -20             // r0 = -20
neg r0             // r0 = 20

mov [v$], -20      // v$ = -20
neg [v$]             // v$ = 20
```

JSR function

Call a function. You can also pass values to the function and stack frame will the handled by the PASM code generation.

JSREQ	// Call function if condition flag is equal
JSRNE	// Call function if condition flag is not equal
JSRGT	// Call function if condition flag is greater than
JSRGE	// Call function if condition flag is greater than or equal
JSRLT	// Call function if condition flag is less than
JSRLE	// Call function if condition flag is less than or equal

Example:

```
!function (Var1, Var2)
  saveppl
  pplpush Var1
  ppl showmessage            // Display 10

  saveppl
  pplpush Var2
  ppl showmessage            // Display 20

  ret
```

```
:main

  mov r0, 10
  cmp r0, 10
  jsreq function (10, 20)
```

JMP label

Goto a label.

JEQ	// Goto if condition flag is equal
JNE	// Goto if condition flag is not equal
JGT	// Goto if condition flag is greater than
JGE	// Goto if condition flag is greater than or equal
JLT	// Goto if condition flag is less than
JLE	// Goto if condition flag is less than or equal

Example:

```
:main
  mov r0, 0
:loop
  add r0, 1
  cmp r0, 10000
  jle loop
```

PUSH value

Push a value on the stack.

Possible syntaxes:

push register	Push value of register
push value	Push value
push [register]	Push value at address of register
push [address]	Push value at address

POP register

Pop a value from the stack and store it's value in a register.

Possible syntaxes:

pop register	; Pop a value from the stack into a register.
pop [register]	; Pop a value from the stack and store it into the address contained in register.
pop [address]	; Pop a value from the stack and store it into the address.

PUSHAD

Push all registers to the stack.

POPAD

Pop all registers from the stack.

SAVESP

When you want to call a PPL internal function that has an unlimited amount of parameters, you need to call this function before starting to pplpush any values on the PPL stack.

PPLPUSH value

Push a value on the PPL internal stack to be used with the PPL operand only.

For possible syntaxes and examples, please refer to the ADD operand.

Example:

```
pplpush 10      // Push 10 on PPL's stack.
pplpush 20      // Push 20 on PPL's stack.
ppl +            // Call the + function
pplpull      // Pull the result value and store it into r0
```

PPLPUSHSTR value

Push a string value on the stack. The value parameter represents the string's pointer value.

Example:

```
#include "console.ppl"

func WinMain;
  InitConsole;
  ShowConsole;

  sdim(StrVal1$, tbyte, 10);
  sdim(StrVal2$, tbyte, 10);
  sdim(StrVal3$, tbyte, 10);
  StrVal1$ = "First ";
  StrVal3$ = " Last";

  asmCall$ = asm(1024, {
#DEASM
    :main
      mov r0, StrVal2$
      mov BYTE [R0], 65
      mov BYTE [R0+1], 66
      mov BYTE [R0+2], 67
      mov BYTE [R0+3], 0

      SaveSP       // Save the stack frame pointer.
      pplpushstr StrVal1$      // Push StrVal1$ pointer on the stack and convert
it to a PPL string
      pplpushstr StrVal2$      // Push StrVal2$ pointer on the stack and convert
it to a PPL string
      pplpushstr StrVal3$      // Push StrVal3$ pointer on the stack and convert
it to a PPL string
      ppl Concat      // Call the CONCAT ppl internal function to concatenate
all strings on the stack pushed after the saved stack pointer.
      ppl Writeln      // Call the writeln console function.
  });

  callasm(asmCall$, 20, 30);
  writeln("Test3 " + StrVal1$);
  freeasm(asmCall$);

  return(true);
end;
```

PPLPULL

Pull a value from the PPL internal stack. Some PPL functions will return a value on the stack and this operand can pull it.

** **The result is always stored in the register r0.**

**** If the result is a string, it will be dereferenced by the PPL's garbage collector and therefore it is up to you to free this string from memory.**

Example #1: (values)

```
pplpush 10            // Push 10 on PPL's stack.
pplpush 20            // Push 20 on PPL's stack.
ppl +                  // Call the + function
pplpull            // Pull the result value and store it into r0
```

Example #2: (Strings)

```
#include "console.ppl"

func WinMain;
  InitConsole;
  ShowConsole;

  new(tstInt$,    tint);
  new(tstStrPtr$, tint);

  sdim(tstStr1$, tbyte, 20);
  sdim(tstStr2$, tbyte, 20);
  sdim(tstStr3$, tbyte, 20);

  tstStr1$ = "First, ";
  tstStr3$ = ", Last";

  asmCall$ = asm(1024, {
    var localVar
    :main
      mov  R0, tstStr2$
      mov  byte [R0], 'A'
      mov  byte [R0+1], 'b'
      mov  byte [R0+2], 'C'
      mov  byte [R0+3], 0

      savesp
      pplpushstr tstStr1$
      pplpushstr tstStr2$
      pplpushstr tstStr3$
      ppl  concat

      pplpull                        // Pull a string from the PPL's stack. The
string is removed from the garbage collector.

      mov  localVar, R0                  // Move address of string (R0) to a
local variable
      mov  tstStrPtr$, localVar      // Move that same address to the first 4
bytes (int) of tstStrPtr$
      pplpushstr [tstStrPtr$]            // Push address of string located in
tstStrPtr$
      ppl  writeln
  });

  callasm(asmCall$);
  writeln("PASM string in PPL: "+ @tstStrPtr$);

  free(@tstStrPtr$);            // Free the string returned by PPLPULL from
memory.
  freeasm(asmCall$);

  free(tstInt$, tstStrPtr$);
  free(tstStr1$, tstStr2$, tstStr3$);

  return(true);
end;
```

PPL FunctionName

Call a PPL internal function.

Example:

```
savesp
pplpush 10
ppl showmessage
```

DEBUG value

Output some information (value) to the debuglog directly from within the PASM.

Supported syntaxes:

debug "string"

debug 10

debug register

debug [register]

debug address

debug [address]

Example:

```
mov r0, 10
debug r0

mov r0, SF
debug [r0]

debug "Value of String$ is \"{[@string$]}\""
```

numeric ABS(numeric X)

Returns the absolute value of *X*

Parameters

X {in}
Value to make absolute

Return Value

ABS returns the absolute value of the input parameter

Example:

```
i$ = -1.25;
j$ = -10;
k$ = i$ + j$;
l$ = abs(i$) + abs(j$);
ShowMessage(k$ + "," + l$); //Displays "-11.25,11.25"
```

See Also: CHGSIGN

double ACOS(double X)

Returns the arccosine of *X*

Parameters

X {in}
Value to determine arccosine of

Return Value
ACOS returns the arccosine of *X* as a double

Example:

```
i$ = 0.5;
msg$ = "i$: " + i$ + "\n";
msg$ = msg$ + "acos: " + acos(i$) + "\n";
msg$ = msg$ + "asin: " + asin(i$) + "\n";
msg$ = msg$ + "atan: " + atan(i$) + "\n";
ShowMessage(msg$);
//Displays a dialog with the following:
// i$: 0.50
// acos: 1.047198
// asin: 0.523599
// atan: 0.463648
```

Notes:
- The range of the return value is from 0 to p radians
- If *X* is less than -1 or greater than 1, ACOS returns an indefinite (same as a quiet NaN)

See Also: ASIN, ATAN

double ASIN(double X)
Returns the arcsine of *X*

Parameters

X {in}
 Value to determine arcsine of

Return Value
ASIN returns the arcsine of *X* as a double

Example:

See ACOS for an example

Notes:
- The range of the return value is from -p/2 to p/2 radians
- If *X* is less than -1 or greater than 1, ASIN returns an indefinite (same as a quiet NaN)

See Also: ACOS, ATAN

double ATAN(double X)
Returns the arctangent of *X*

Parameters

X {in}
 Value to determine arctangent of

Return Value
ATAN returns the arctangent of *X* as a double. If *X* is 0, ATAN returns 0.

Example:

See ACOS for an example

Notes:
- The range of the return value is from -p/2 to p/2 radians

See Also: ACOS, ASIN

double ATAN2 (double Y, double X)
Returns the arctangent of y/x

Parameters

Y {in}
Numerator of operation

X {in}
Denominator of operation

Return Value
If both X and Y are 0, ATAN2 returns 0; otherwise, ATAN2 returns a value between -p and p radians, using the sign of X and Y to determine the quadrant

Notes:
- ATAN2 is well defined for every point other than the origin, even if *X* equals 0 and *Y* does not equal 0

See Also: ATAN

double COS(double X)
Returns the cosine of *X*

Parameters

X {in}
Value to determine cosine of

Return Value
COS returns the cosine of *X* as a double

Example:

```
i$ = 0.5;
msg$ = "i$: " + i$ + "\n";
msg$ = msg$ + "cos: " + cos(i$) + "\n";
msg$ = msg$ + "cosh: " + cosh(i$) + "\n";
msg$ = msg$ + "sin: " + sin(i$) + "\n";
msg$ = msg$ + "sinh: " + sinh(i$) + "\n";
ShowMessage(msg$);
//Displays a dialog with the following:
// i$: 0.50
// cos: 0.877583
// cosh: 1.127626
// sin: 0.479426
// sinh: 0.521095
```

Notes:
- If *X* >= 263, or *X* <= -263, a loss of significance in the return value occurs, in which case COS generates a _TLOSS error and returns an indefinite (same as a quiet NaN)

See Also: COS, COSH, SIN, SINH

double COSH(double X)
Returns the hyperbolic cosine of *X*

Parameters

X {in}
Value to determine hyperbolic cosine of

Return Value
COSH returns the hyperbolic cosine of *X* as a double

Example:

See COS for an example

Notes:

- If the result is too large, COSH returns HUGE_VAL

See Also: COS, COSH, SIN, SINH

double EXP(numeric X)

Find the exponential value of *X*

Parameters

X {in}
Floating point value

Return Value
EXP returns a double containing the exponential value of *X* if successful. On overflow, the function returns INF (infinite) and on underflow, EXP returns 0.

Example:

```
d$ = 2.5682195;
msg$ = "d$: " + d$ + "\n";
msg$ = msg$ + "exp: " + exp(d$) + "\n";
msg$ = msg$ + "log: " + log(d$) + "\n";
msg$ = msg$ + "log10: " + log10(d$) + "\n";
ShowMessage(msg$);
//Displays
// d$: 2.568220
// exp: 13.042581
// log: 0.943213
// log10: 0.409632
```

See Also: LOG, LOG10

double CEIL(numeric X)

Returns a double representing the smallest integer that is >= *X*

Parameters

X {in}
A floating point value

Return Value
CEIL returns a double

Example:

```
d$ = 2.5682195;
i$ = Floor(d$);
ShowMessage(i$); //Displays 2
i$ = Ceil(d$);
ShowMessage(i$); //Displays 3
```

See Also: FLOOR

double FLOOR(numeric X)
Returns a double representing the largest integer that is <= *X*

Parameters

X {in}
   A floating point value

Return Value
FLOOR returns a double

Example:

See CEIL for an example

See Also: CEIL

long ROUND(numeric Value)
Round returns the nearest whole number to *Value*

Parameters

Value {in}
   Number that you wish to round

Return Value
ROUND returns the nearest whole number

Example:

```
d$ = 2.5682195;
i$ = Round(d$);
ShowMessage(i$);  //Displays "3"
i$ = RoundEx(d$, 3);
ShowMessage(i$);  //Displays "2.568000"
```

Notes:
- Decimal values of .5 or more will be rounded up, and decimal values of .4 or less will be rounded down
- To round beyond the decimal point, use the ROUNDEX function

See Also: ROUNDEX

long TRUNC(double X)
Drops the decimal part of a floating point value

Parameters

X {in}
   Value to truncate

Return Value
TRUNC returns the whole number part of *X*

Example:

```
x$ = Trunc(10.25);
ShowMessage(x$);  //Displays "10"
```

See Also: ROUND, ROUNDEX

double LOG(numeric X)
Find the logarithm of *X*

Parameters

X {in}
  Floating point value

Return Value

LOG returns the logarithm of *X* if successful. If *X* is negative, LOG returns an indefinite (same as a quiet NaN). If *X* is 0, LOG returns INF (infinite).

Example:

See EXP for an example

See Also: EXP, LOG10

double LOG10(numeric X)

Find the log base 10 of *X*

Parameters

X {in}
  Floating point value

Return Value

LOG10 returns the log base 10 of *X* if successful. If *X* is negative, LOG10 returns an indefinite (same as a quiet NaN). If *X* is 0, LOG10 returns INF (infinite).

Example:

See EXP for an example

See Also: EXP, LOG

numeric POW(numeric X, numeric Y)

computes x raised to the power of y

Parameters

X {in}
  base value

Y {in}
  exponent

Return Value

POW returns the result of *X* raised to the power of *Y*. POW does not raise an error on overflow or underflow.

Specific returns values:

Values of x and y	Return Value of pow
x < > 0 and y = 0.0	1
x = 0.0 and y = 0.0	1
x = 0.0 and y < 0	INF

Example:

```
a$ = 5;
b$ = 3;
c$ = pow(a$, b$);
ShowMessage("exp(" + a$ + ", " + b$ + ") = " + c$);
//Displays exp(5, 3) = 125
```

Notes:

- POW does not recognize integral floating-point values greater than 264, such as 1.0E100.

int RAND(void)

Produces a pseudorandom integer

Return Value

RAND returns an integer in the range 0 to RAND_MAX

Example:

```
srand(1);
rand$ = rand();
random$ = random(100);
ShowMessage("rand$: " + rand$ + ", random$: " + random$);
//Displays rand$: 41, random$: 56 (your results might vary)
```

Notes:

- Use the SRAND function to seed the pseudorandom-number generator before calling RAND

See Also: SRAND, RANDOM

int RANDOM(double MaxValue)

Produces a pseudorandom value between 0 and *MaxValue*

Parameters

MaxValue {in}
The upper range to use when generating a random number

Return Value

RANDOM returns an integer between 0 and *MaxValue*

Example:

See RAND for an example

See Also: RAND, SRAND

void RANDOMSET(any Var, numeric Value1, numeric Value2)

Randomly assign *Var* to *Value1* or *Value2*

Paramters

Var {in | out}
Variable to hold the value of either *Value1* or *Value2*

Value1 {in}
First value for random draw

Value2 {in}
Second value for random draw

Example:

```
RandomSet(velocity$, Speed$, -Speed$);       // Randomly set velocity$ to Speed$
or -Speed$.
```

double SIN(double X)

Returns the sine of *X*

Parameters

X {in}
Value to determine sine of

Return Value

SIN returns the sine of *X* as a double

Example:

See COS for an example

Notes:

- If *X* is >= 263, or <= -263, a loss of significance in the result occurs, in which case the function generates a _TLOSS error and returns an indefinite (same as a quiet NaN)

See Also: COS, COSH, SINH

double SINH(double X)

Returns the hyperbolic sine of *X*

Parameters

X {in}
Value to determine hyperbolic sine of

Return Value

SINH returns the hyperbolic sine of *X* as a double

Example:

See COS for an example

Notes:

- If the result is too large, sinh returns ±HUGE_VAL

See Also: COS, COSH, SIN

double SQRT(double X)

Calculates the square root of *X*

Parameters

X {in}
Value to calculate the square root of

Return Value

SQRT returns the square root of *X* as a double

Example:

```
s$ = sqrt(5);
ShowMessage(s$); //Displays 2.236068
```

Notes:

If *X* is negative, SQRT returns an indefinite (same as a quiet NaN)

void SRAND (int Seed)

Sets the starting point for generating a series of pseudorandom integers

Parameters

Seed {in}
A value of 1 will reinitialize the random generator; any other value sets the generator to a random starting point

Example:

See RAND for an example

Notes:

- Calling RAND before any call to SRAND generates the same sequence as calling SRAND with *Seed* passed as 1

See Also: RAND, RANDOM

double TAN(double X)

Returns the tangent of X

Parameters

X {in}
Value to determine tangent of

Return Value

TAN returns the tangent of *X* as a double

Example:

```
i$ = 0.5;
msg$ = "i$: " + i$ + "\n";
msg$ = msg$ + "tan: " + tan(i$) + "\n";
msg$ = msg$ + "tanh: " + tanh(i$) + "\n";
ShowMessage(msg$);
//Displays a dialog with the following:
// i$: 0.50
// tan: 0.546302
// tanh: 0.462117
```

Notes:

- If *X* is >= 263, or <= -263, a loss of significance in the result occurs, in which case TAN generates a _TLOSS error and returns an indefinite (same as a quiet NaN)

See Also: TANH

double TANH(double X)

Returns the hyperbolic tangent of X

Parameters

X {in}
Value to determine hyperbolic tangent of

Return Value

TANH returns the hyperbolic tangent of *X* as a double

Example:

See TAN for an example

Notes:

- No error is thrown on a bad value

See Also: TAN

double CHGSIGN(double X)

Changes the sign of *X*

Parameters

X {in}

Number to reverse the sign of

Return Value

CHGSIGN returns a numerically equivallent value to *X*, but with the sign reversed

Example:

```
a$ = -3;
b$ = 5.642;
c$ = chgsign(a$); //c$ becomes 3
d$ = chgsign(b$); //d$ becomes -5.642
e$ = a$ + b$ + c$ + d$;
ShowMessage(e$); //Displays 0
```

See Also: ABS

double HYPOT(int X, int Y)

Calculates the length of the hypotenuse of a right triangle

Parameters

X {in}
Length of one side of a right triangle

Y {in}
Length of another side of a right triangle

Return Value

HYPOT returns the hypotenuse of the right triangle with sides *X* and *Y*; return value is a double

Example:

```
a$ = 10;
b$ = 24;
c$ = hypot(a$, b$);
ShowMessage("Side c: " + c$);
//Displays "Side c: 26"
```

Notes:

- A call to HYPOT is equivalent to the square root of (*X* * *X*) + (*Y* * *Y*)

unsigned long LROTL(unsigned long Value, int Shift)

Rotates *Value* to the left by *Shift* bits

Parameters

Value {in}
Number to shift the bits on; this should be an unsigned long

Shift {in}
Number of bits to shift *Value* by

Return Value

LROTL returns an unsigned long that is *Value* with its bits shifted left by *Shift* bits

Example:

```
val$ = 0x00001000;
i$ = lrotl(val$, 8);
i$ = lrotr(val$, 8);
```

Notes:

- LROTL "wraps" bits rotated off one end of value to the other end

See Also: LROTR

unsigned long LROTR(unsigned long Value, int Shift)

Rotates *Value* to the right by *Shift* bits

Parameters

Value {in}
Number to shift the bits on; this should be an unsigned long

Shift {in}
Number of bits to shift *Value* by

Return Value

LROTR returns an unsigned long that is *Value* with its bits shifted right by *Shift* bits

Example:

```
val$ = 0x00001000;
i$ = lrotl(val$, 8);
i$ = lrotr(val$, 8);
```

Notes:

- LROTR "wraps" bits rotated off one end of value to the other end

See Also: LROTL

unsigned int ROTL(unsigned int Value, int Shift)

Rotates *Value* to the left by *Shift* bits

Parameters

Value {in}
Number to shift the bits on; this should be an unsigned int

Shift {in}
Number of bits to shift *Value* by

Return Value

ROTL returns an unsigned int that is *Value* with its bits shifted left by *Shift* bits

Example:

```
val$ = 0x00001000;
i$ = rotl(val$, 8);
i$ = rotr(val$, 8);
```

Notes:

- ROTL "wraps" bits rotated off one end of value to the other end

See Also: ROTR

unsigned int ROTR(unsigned int Value, int Shift)

Rotates *Value* to the right by *Shift* bits

Parameters

Value {in}
Number to shift the bits on; this should be an unsigned int

Shift {in}

Number of bits to shift *Value* by

Return Value
ROTR returns an unsigned int that is *Value* with its bits shifted right by *Shift* bits

Example:

```
val$ = 0x00001000;
i$ = rotl(val$, 8);
i$ = rotr(val$, 8);
```

Notes:
- ROTR "wraps" bits rotated off one end of value to the other end

See Also: ROTL

(unsigned Value) << (int Shift)
Shifts *Value* left by *Shift* positions

Parameters

Value {in}
Number to shift bits on

Shift {in}
Number of positions to shift the bits

Return Value
<< returns an unsigned *Value* shifted by *Shift* positions

Example:

```
i$ = 10 << 2;
i$ = 10 shl 2;
```

Notes:
- SHL is syntactically equivalent to <<

See Also: >>

(unsigned Value) >> (int Shift)
Shifts *Value* right by *Shift* positions

Parameters

Value {in}
Number to shift bits on

Shift {in}
Number of positions to shift the bits

Return Value
<< returns an unsigned *Value* shifted by *Shift* positions

Example:

```
i$ = 10 << 2;
i$ = 10 shr 2;
```

Notes:
- SHR is syntactically equivalent to >>

See Also: <<

ASL

The bitwise shift operators shift their first operand left (<<) or right (>>) by the number of positions the second operand specifies. This operator works on signed value only.

Example:

```
i$ = -10 asl 2;
```

ASR

The bitwise shift operators shift their first operand left (<<) or right (>>) by the number of positions the second operand specifies. This operator works on signed value only.

Example:

```
i$ = -10 asr 2;
```

DWORD HIWORD(long Variable)

Retrieve the hi-order 16 bit value of a variable

Parameters

Variable {in}
  The variable to retrieve the information from

Return Value

HIWORD returns a 16 bit value

Example:

See MAKELONG for an example

See Also: MAKELONG, LOWORD

DWORD LOWORD(long Variable)

Retrieve the low-order 16 bit value of a variable

Parameters

Variable {in}
  The variable to retrieve the information from

Return Value

LOWORD returns a 16 bit value

Example:

See MAKELONG for an example

See Also: MAKELONG, HIWORD

long MAKELONG(int Low, int High)

Creates an unsigned 32-bit value by concatenating two specified 16-bit values

Parameters

Low {in}
  Low order 16 bit value

High {in}
  High order 16 bit value

Return Value

MAKELONG returns the two parameters concatenated together as a long value

Example:

```
type(Low$, High$, TINT);
type(Long$, TLONG);

Low$ = 10;
High$ = 20;

Long$ = MakeLong(Low$, High$);

ShowMessage(Low$ + "\n" + High$ + "\n" + LoWord(Long$) + "\n" + HiWord(Long$));
```

See Also: LOWORD, HIWORD

boolean VALIDINT(string Value)

Determines if *Value* contains valid numeric data

Parameters

Value {in}
string that might contain a numeric value

Return Value

VALIDINT returns true if *Value* contains numeric data, or false otherwise

Example:

```
s$ = "123";
t$ = "12ABA";

ShowMessage(ValidInt(s$) + "," + ValidInt(t$)); //Displays "1,0"
```

WRAP (Value, Min, Max, Around) -> NewValue

This function will wrap Value within the Min and Max delimiters. If you set Around to true, the NewValue will be wrapped around the delimiters but also cycled within.

Example:

```
i$ = 5;
i$ = Wrap(i$, 10, 30, True);      // i$ = 25;

i$ = 35;
i$ = Wrap(i$, 10, 30, True);      // i$ = 15;

i$ = 5;
i$ = Wrap(i$, 10, 30, True);      // i$ = 10;

i$ = 40;
i$ = Wrap(i$, 10, 30, True);      // i$ = 30;
```

string MID (string Source, int Index, int Length)

Returns the portion of *Source* starting at *Index* for a total of *Length* characters

Parameters

Source {in}
The group of characters you wish to retrieve your substring from

Index {in}
The position of the first character in the substring you wish to retrieve

Length {in}
The total size of the substring you want to capture

Return Value

MID returns a string

Example:

```
s$ = Mid("ABCDEF", 2, 3);
ShowMessage(s$);                    // Result is "CDE"

s$ = Mid("ABCDEF", 1, -1);
ShowMessage(s$);                    // Result is "BCDEF"
```

Notes:

- Remember that strings are 0 based
- Setting *Length* to -1 will copy the entire string following the character at position *Index*

See Also: INSERT, DELETE

int LENGTH (string Source)

Returns the number of characters in *Source*

Parameters

Source {in}
The group of characters you wish to determine the count of

Return Value

LENGTH returns a numeric value

Example:

```
ShowMessage(Length("ABCDEF"));            // Result is 6.
```

void DELETE (string Source, int Index, int Length)

Removes *Length* characters from *Source* starting at position *Index*

Parameters

Source {in | out}
The group of characters you want to dissect

Index {in}
The first character to remove

Length {in}
The total number of characters to remove

Example:

```
s$ = "ABCDEF";
Delete(s$, 2, 3);
ShowMessage(s$);              // Result is "ABF"
```

Notes:

- Remember that this will actually modify the *String* parameter, rather than returning the modified string

See Also: INSERT

void INSERT (string Substring, string Source, int Index)

Insert *Substring* into *Source* starting at position *Index*

Parameters

Substring {in}
The group of characters to be added

String {in | out}

The group of characters that will be expanded

Index {in}
The starting position for the insertion

Example:

```
s$ = "AEF";
Insert("BCD", s$, 2);
ShowMessage(s$);            // Result is "ABCDEF"
```

Notes:
- *String* will be directly modified by this function

See Also: DELETE, REPLACE

void REPLACE (string Source, string Find, string Replace)
Replace all occurences of *Find* with *Replace* in *Source*

Parameters
Source {in | out}
The group of characters that will be manipulated

Find {in}
The group of characters you are looking for

Replace {in}
The group of characters to replace *Find* with

Example:

```
x$ = "10203040506070";

REPLACE(x$, "0", "A");
ShowMessage(x$);  // Result is 1A2A3A4A5A6A7A
```

Notes:
- *Source* is directly modified by this function

See Also: DELETE, INSERT

void CHANGE(string Source, int Start, int Length, string Replace)
Alter a section of *Source* to contain the value in *Replace*

Parameters
Source {in | out}
The string you wish to manipulate

Start {in}
1st character in the source string to replace

Length {in}
Number of characters in the source string to replace

Replace {in}
String you want to insert into *Source*

Example:

```
a$ = "PPL WAS GREAT!";
Change(a$, 4, 3, "IS");
ShowMessage(a$);      // PPL IS GREAT!
```

See Also: STRIP

int POS(string Find, string Source)

Returns the position of the first character of *Find* within *Source*

Parameters

Find {in}
  The string you are searching for

Source {in}
  The string being searched

Return Value

POS returns an integer

Example:

```
i$ = Pos("BCD", "ABCDEF");
ShowMessage(i$);                  // Result is 1
```

Notes:

- Strings are zero based
- If *Find* is not found, the return value is -1

See Also: NPOS

int NPOS (string Find, string Source, int Start)

Like POS, but lets you specify the position to begin searching with *Start*

Parameters

Find {in}
  The string you are searching for

Source {in}
  The string being searched

Start {in}
  Position to begin the search

Return Value

NPOS returns an integer

Example:

```
i$ = NPos("B", "ABCDEFBFGR", 3);
ShowMessage(i$);                  // Result is 6
```

Notes:

- Strings are zero based
- If *Find* is not found, the return value is -1

See Also: POS

string CONCAT([any Items...])

Creates a string comprised of all the values in *Items*

Parameters

Items {in}
  One or more values to concatenate into a string

Return Value

CONCAT returns a string comprised of all the values in *Items*

Example:

```
s$ = Concat("A", "B", "C");
ShowMessage(s$); // Displays "ABC"
```

int ISALNUM(string Source)

Determines whether the value is alphanumeric ('a' .. 'z', 'A' .. 'Z', '0' .. '9') or not. A return value greater than 0 indicates that the value is alphanumeric.

Parameters

Source {in}
  The group of characters in question

Return Value

ISALNUM returns a 0 for non-alphanumeric values, a 1 for an alphanumeric string, and a value greater than 0 for an alphanumeric character

Example:

```
a$ = 'a';
b$ = 65;
c$ = '%';
d$ = "123abc";
e$ = "123^abc";

result$ = isalnum(a$);      // $result > 0
result$ = isalnum(b$);      // $result > 0
result$ = isalnum(c$);      // $result = 0
result$ = isalnum(d$);      // $result = 1
result$ = isalnum(e$);      // $result = 0
```

See Also: ISALPHA

int ISALPHA(string Source)

Determines whether the value is alpha ('a' .. 'z', 'A' .. 'Z') or not. A return value greater than 0 indicates that the value is alpha.

Parameters

Source {in}
  The group of characters in question.

Return Value

ISALPHA returns a 0 for non-alpha values, a 1 for an alpha string, and a value greater than 0 for an alpha character

Example:

```
a$ = 'a';
b$ = 65;
c$ = '%';
d$ = "123abc";
e$ = "123^abc";

result$ = isalpha(a$);      // $result > 0
result$ = isalpha(b$);      // $result > 0, because 65 is the ASCII for 'a'
result$ = isalpha(c$);      // $result = 0
result$ = isalpha(d$);      // $result = 0, because string contains numerics
(1,2,3)
result$ = isalpha(e$);      // $result = 0, because string contains non-alphas
(^)
```

See Also: ISALNUM

int ISCNTRL (string Source)

Determines whether the value is a control character (0x00 - 0x1F or 0x7F) or not. A return value greater than 0 indicates that the value is a control character.

Parameters

Source {in}
The group of characters in question.

Return Value

ISALPHA returns a 0 for non-alpha values, a 1 for an alpha string, and a value greater than 0 for an alpha character

Example:

```
a$ = 0;
b$ = 65;
c$ = 0x09;
d$ = "123abc";

result$ = iscntrl(a$);  // result$ > 0
result$ = iscntrl(b$);  // result$ = 0, because 65 is the ASCII for 'a'
result$ = iscntrl(c$);  // result$ > 0
result$ = iscntrl(d$);  // result$ = 0, because the string contains no control
chars
```

See Also:

int ISDIGIT (string Source)

Determines whether the value is a digit (0..9) or not. A return value greater than 0 indicates that the value is numeric.

Parameters

Source {in}
The group of characters in question.

Return Value

ISDIGIT returns a 0 for non-numeric values, a 1 for a numeric string, and a value greater than 0 for a numeric character

Example:

```
a$ = '0';
b$ = "0";
c$ = 0;
d$ = "123abc";
e$ = "12345";

result$ = isdigit(a$);  // result$ > 0
result$ = isdigit(b$);  // result$ = 1
result$ = isdigit(c$);  // result$ = 0, because c$ is a numeric field, not a
string containing a numeric value
result$ = isdigit(d$);  // result$ = 0, because the string contains alpha
characters
result$ = isdigit(e$);  // result$ = 1
```

See Also:

int ISLOWER (string Source)

Determines whether all alphas contained in the source are lower case ('a'..'z') or not. A return value greater than 0 indicates that all alphas are lower case.

Parameters
Source {in}
The group of characters in question.

Return Value
ISLOWER returns a 0 if at least one alpha is upper case, a 1 for a string where all alphas are lower case, and a value greater than 0 for a single character that's lower case

Example:

```
a$ = 'a';
b$ = 'A';
c$ = "ABC123";
d$ = "abc123";

result$ = islower(a$);  // result$ > 0
result$ = islower(b$);  // result$ = 0
result$ = islower(c$);  // result$ = 0, because at least one alpha character is
upper case
result$ = islower(d$);  // result$ = 1
```

See Also: ISUPPER

int ISPRINT(string Source)

Determines whether all characters contained in the source are within printable range ('a'..'z') or not. A return value greater than 0 indicates that all characters are printable.

Parameters
Source {in}
The group of characters in question.

Return Value
ISPRINT returns a 0 if at least one character is not printable, a 1 for a string where all characters are printable, and a value greater than 0 for a single character that's printable

Example:

```
a$ = 'a';
b$ = 12;
c$ = '%';
d$ = "123abc";
e$ = "123" + chr(12) + "abc";

result$ = isalnum(a$);      // result$ > 0
result$ = isalnum(b$);      // result$ = 0
result$ = isalnum(c$);      // result$ > 0
result$ = isalnum(d$);      // result$ = 1
result$ = isalnum(e$);      // result$ = 0
```

int ISPUNCT (string Source)

Determines whether all characters contained in the source are within printable range ('a'..'z'), but are neither spaces nor alphanumerics. A return value greater than 0 indicates that all characters meet the desired criteria.

Parameters
Source {in}
The group of characters in question.

Return Value
ISPUNCT returns a 0 if at least one character is either not printable, a space, or an alphanumeric; ISPUNCT returns a 1 for a string where all characters meet the criteria, and a value greater than 0 for a single character that meets the criteria

Example:

```
a$ = 'a';
b$ = 12;
c$ = '%';
d$ = "!.,*&";
e$ = "123" + chr(12) + "abc";

result$ = ispunct(a$);  // result$ = 0
result$ = ispunct(b$);  // result$ = 0
result$ = ispunct(c$);  // result$ > 0
result$ = ispunct(d$);  // result$ = 1
result$ = ispunct(e$);  // result$ = 0
```

See Also: ISSPACE, ISALNUM

int ISSPACE (string Source)

Determines whether all characters contained in the source are spaces or not. A return value greater than 0 indicates that all characters are spaces.

Parameters
Source {in}
The group of characters in question.

Return Value
ISSPACE returns a 0 if at least one character is not a space, a 1 for a string that contains all spaces, and a value greater than 0 for a single character that is a space

Example:

```
a$ = 'a';
b$ = chr(32);
c$ = "123 abc";
d$ = "    ";

result$ = isspace(a$);  // result$ = 0
result$ = isspace(b$);  // result$ > 0
result$ = isspace(c$);  // result$ = 0
result$ = isspace(d$);  // result$ = 1
```

int ISUPPER (string Source)

Determines whether all alphas contained in the source are upper case ('A'..'Z') or not. A return value greater than 0 indicates that all alphas are upper case.

Parameters
Source {in}
The group of characters in question.

Return Value
ISUPPER returns a 0 if at least one alpha is lower case, a 1 for a string where all alphas are upper case, and a value greater than 0 for a single character that's upper case

Example:

```
a$ = 'a';
b$ = 'A';
c$ = "ABC123";
d$ = "abc123";
```

```
result$ = isupper(a$);  // result$ = 0
result$ = isupper(b$);  // result$ > 0
result$ = isupper(c$);  // result$ = 1
result$ = isupper(d$);  // result$ = 0, because at least one alpha character is
lower case
```

See Also: ISLOWER

string UPPER(string Source)

Converts all alpha characters to upper case.

Parameters

Source {in}

The group of characters to convert.

Return Value

UPPER returns a string with all alpha characters converted to upper case ('A'..'Z').

Example:

```
s$ = "123abc";
u$ = upper(s$);
ShowMessage(u$);      // displays the string "123ABC"
```

See Also: LOWER

string LOWER(string Source)

Converts all alpha characters to lower case.

Parameters

Source {in}

The group of characters to convert.

Return Value

LOWER returns a string with all alpha characters converted to lower case ('a'..'z').

Example:

```
s$ = "123ABC";
u$ = lower(s$);
ShowMessage(u$);      // displays the string "123abc"
```

See Also: UPPER

string CHR (Int Value)

Returns the character equivalent of an integer.

Parameters

Value {in}

Integer to be converted.

Return Value

CHR returns a string representation of the integer.

Example:

```
i$ = 97;
ShowMessage(chr(i$));      // Display       a
ShowMessage(chr(65));      // Display      A
```

Notes:

- Works like the # operator, except it supports variables
- Only good for integers in the range of 0-255

See Also:

widestring WIDE (string Source)

Converts a regular single-byte string into WideString format.

Parameters

Source {in}
The group of characters to convert.

Return Value

WIDE returns the source string in WideString format.

Example:

```
s$ = "This is a normal string";
result$ = iswide(s$); //result$ will be false
w$ = wide(s$);
result$ = iswide(w$); //result$ will be true
```

Notes:

- The WideString is mainly used for Windows CE API calls
- PPL will convert all regular strings to widestrings when calling a Windows CE API function, but sometimes it is necessary to do it yourself
- PPL will not convert strings that are already in the widestring format

See Also: CHAR

string CHAR (widestring Source)

Converts a WideString string to a regular single-byte string.

Parameters

Source {in}
The group of characters to convert.

Return Value

CHAR returns the source string in regular single-byte format.

Example:

```
s$ = "This is a normal string";
w$ = wide(s$);
result$ = iswide(w$); //result$ will be true
s$ = char(w$);
result$ = iswide(s$); //result$ will be false
```

Notes:

- PPL will not convert a string that is already in the single-byte format

See Also: WIDE

{wide}string APICHAR (string Source)

Converts a string to the format necessary for the API calls of the currently running platform.

Parameters

Source {in}
The group of characters to convert.

Return Value
APICHAR returns a WideString on the PocketPC, and a single-byte formatted string on the PC.

Example:

```
XYText$ = "1,1";
SendMessage(StatusCtl$, SB_SETTEXT, 0, ApiChar(XYText$));
```

See Also: WIDE, CHAR

boolean ISWIDE (string Source)

Determines if the string passed in is defined as widechar or not.

Parameters
Source {in}
The group of characters in question.

Return Value
ISWIDE returns true if the first character defines the string as a widechar string, or false otherwise

Example:

See WIDE for an example

See Also: WIDE, CHAR

string DUP(string Source)

Duplicates a string in memory

Parameters
Source {in}
The group of characters to duplicate.

Return Value
DUP returns a copy of *Source* in a new memory location.

Example:

```
s$ = "This is string A";
s1$ = dup(s$);
replace(s1$, "A", "B");
ShowMessage(s$ + #10#13 + s1$);

//Messagebox displays "This is string A", a carriage return, and
//  "This is string B"
```

Notes:
- An exact copy of Source is created at a new memory location
- The duplicate string is a unique entity, so changes made to it will not be reflected in the original string

void SPRINTF(string Output, string FormatString, [any Arguments...])

Prints to *Output* a sequence of arguments formatted as the format argument specifies

Parameters

Output {out}
Variable to hold the formatted string

FormatString {in}
String containing printable text, as well as a combination of flags and variable types to be formatted and displayed

Arguments {in}
One or more variables containing values to be displayed according to *FormatString*; the number of *Arguments* should equal the number of flags in *FormatString*

Example.

```
sprintf(s$, "Characters: %c %c \n", 'a', 65);
//Characters: a A
sprintf(s$, "Decimals: %d %ld\n", 1977, 650000);
//Decimals: 1977 650000
sprintf(s$, "Preceding with blanks: %10d \n", 1977);
//Preceding with blanks:       1977
sprintf(s$, "Preceding with zeros: %010d \n", 1977);
//Preceding with zeros: 0000001977
sprintf(s$, "Some different radixes: %d %x %o %#x %#o \n", 100, 100, 100, 100,
100);
//Some different radixes: 100 64 144 0x64 0144
sprintf(s$, "floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
//floats: 3.14 +3e+000 3.141600E+000
sprintf(s$, "Width trick: %*d \n", 5, 10);
//Width trick:    10
sprintf(s$, "%s \n", "A string");
//A string
//
```

Notes:

FormatString in more detail:

- String that contains the text to be printed
- Optionally it can contain format tags that are substituted by the values specified in subsequent argument(s) and formatted as requested
- The number of format tags must correspond to the number of additional arguments that follows
- The format tags follow this prototype: %[flags][width][.precision][modifiers]type
- Type is the most significant tag and defines how the value will be printed

type	Output	Example
c	Character	a
d or i	Signed decimal integer	392
e	Scientific notation (mantise/exponent) using e character	3.9265e2
E	Scientific notation (mantise/exponent) using E character	3.9265E2
f	Decimal floating point	392.65
g	Use shorter %e or %f	392.65
G	Use shorter %E or %f	392.65
o	Signed octal	610
s	String of characters	sample
u	Unsigned decimal integer	7235
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (capital letters)	7FA
p	Address pointed by the argument	B800:0000
n	Nothing printed	

- Flags are optional and are as follows:

flag	meaning
-	Left align within the given width. (right align is the default).
+	Forces to preceed the result with a sign (+ or -) if signed type. (by default only - (minus) is printed).
blank	If the argument is a positive signed value, a blank is inserted before the number.
#	Used with o, x or X type the value is preceeded with 0, 0x or 0X respectively if non-zero. Used with e, E or f forces the output value to contain a decimal point even if only zeros follow. Used with g or G the result is the same as e or E but trailing zeros are not removed.

- Width is optional and designated as follows:

width	**meaning**
numberMinimum	number of characters to be printed. If the value to be printed is shorter than this number the result is padded with blanks. The value is never truncated even if the result is larger.
0number	Same as above but filled with 0s instead of blanks.
*	The width is not specified in the format string, it is specified by an integer value preceding the argument thas has to be formatted.

- Precision is optional and specified as follows:

.precision	**meaning**
.number	for d, i, o, u, x, X types: precision specifies the minimum number of decimal digits to be printed. If the value to be printed is shorter than this number the result is padded with blanks. The value is never truncated even if the result is larger.(if nothing specified default is 1). for e, E, f types: number of digits to be printed after de decimal point. (if nothing specified default is 6). for g, G types : maximum number of significant numbers to be printed. for s type: maximum number of characters to be printed. (default is to print until first null character is encountered). for c type : (no effect).

- Modifiers are optional and specified as follows:

modifier	**meaning (affects on how arguments are interpreted by the function)**
h	argument is interpreted as short int (integer types).
l	argument is interpreted as long int (interger types) or double (floating point types).
L	argument is interpreted as long double (floating point types).

See Also: PRINTF

string PRINTF(string FormatString, [any Arguments...])

Create a string with text and optional values formatted through *FormatString*

Parameters

FormatString {in}
String containing printable text, as well as a combination of flags and variable types to be formatted and displayed

Arguments {in}
One or more variables containing values to be displayed according to *FormatString*; the number of *Arguments* should equal the number of flags in *FormatString*

Return Value

PRINTF returns a string formatted according to *FormatString* and using values supplied through *Arguments*

Example:

```
s$ = printf("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
ShowMessage(s$); //Displays "floats: 3.14 +3e+000 3.141600E+000"
```

For more examples of using *FormatString*, as well as for details on all the functionality of *FormatString*, please see SPRINTF

See Also: SPRINTF

string LOADSTR(string Filename, int Sizevar)

Loads the contents of *Filename* into a string.

Parameters

FIlename {in}
String containing the name of the file to load.

Sizevar {out}
On return, contains the number of bytes read from *Filename*

Return Value

LOADSTR returns a string containing the contents of *Filename*.

Example:

```
s$ = LoadStr("\\My Documents\\My File.txt", sz$);
if (sz$ > 0)
  ShowMessage(s$);
end;
```

See Also: SAVESTR

void SAVESTR (string Filename, string Source, int Size)

Save *Size* number of bytes of *Source* to *Filename*.

Parameters

FIlename {in}
String containing the name of the file to load.

Source {in}
String containing the information you wish to save.

Size {in}
The number of characters you wish to save; use -1 to save the entire string.

Example:

```
SaveStr("\\My Documents\\MyFile.txt", "ABCDEF", -1);      // Writes 6 bytes to
the file.
SaveStr("\\My Documents\\MyFile.txt", "ABCDEF", 3);      // Writes first 3 bytes
to the file.
```

See Also: LOADSTR

string EXTRACTFILENAME(string Source)

Provides the file name portion of a fully qualified path.

Parameters

Source {in}
String containing a fully qualified file path.

Return Value

EXTRACTFILENAME returns the file name portion of *Source*.

Example:

```
f$ = "\\CF Card\\My Documents\\PPL\\test.ppl";
fn$ = extractfilename(f$);
ShowMessage(fn$);                  // displays the string "test.ppl"
```

See Also: EXTRACTFILEPATH, EXTRACTFILEDRIVE, EXTRACTFILEEXT

string EXTRACTFILEPATH(string Source)

Provides everything except the file name portion of a fully qualified path.

Parameters

Source {in}
String containing a fully qualified file path.

Return Value

EXTRACTFILEPATH returns the path portion of *Source*.

Example:

```
f$ = "\\CF Card\\My Documents\\PPL\\test.ppl";
```

```
fp$ = extractfilepath(f$);
ShowMessage(fp$);                  // displays the string "\CF Card\My
Documents\PPL\"
```

See Also: EXTRACTFILENAME, EXTRACTFILEDRIVE, EXTRACTFILEEXT

string EXTRACTFILEDRIVE (string Source)

Provides the root path location of a fully qualified path.

Parameters

Source {in}
  String containing a fully qualified file path.

Return Value

EXTRACTFILEDRIVE returns the root path portion of *Source*.

Example:

```
f$ = "\\CF Card\\My Documents\\PPL\\test.ppl";
fp$ = extractfilepath(f$);
ShowMessage(fp$);                  // displays the string "\CF Card\"
```

See Also: EXTRACTFILENAME, EXTRACTFILEPATH, EXTRACTFILEEXT

string EXTRACTFILEEXT(string Source)

Provides the extension of the file name portion of a fully qualified path.

Parameters

Source {in}
  String containing a fully qualified file path.

Return Value

EXTRACTFILEEXT returns the extension of the file name portion of *Source*.

Example:

```
f$ = "\\CF Card\\My Documents\\PPL\\test.ppl";
fp$ = extractfileext(f$);
ShowMessage(fp$);                  // displays the string ".ppl"
```

See Also: EXTRACTFILENAME, EXTRACTFILEPATH, EXTRACTFILEDRIVE

void ENCRYPT(string Input, long Length, string Key, boolean Encrypt)

Encrypt / decrypt some or all of a string

Parameters

Input {in}
  Group of characters to encrypt or decrypt

Length {in}
  Maximum number of characters to convert; can be -1 for entire String

Key {in}
  Text to act as key for conversion process

Encrypt {in}
  True to encrypt, false to decrypt

Example:

```
s$ = "HELLO WORLD!";
Encrypt(s$, -1, "MYKEY", True);
ShowMessage(s$);
Encrypt(s$, -1, "MYKEY", False);
ShowMessage(s$);
```

Notes:

- Length cannot be -1 if *Input* contains no characters

COMPRESS (Type, In, Out, Size) -> OutSize

Compresses a memory buffer (In) into another memory buffer (Out). The size parameter determines the size of the input buffer (In) in bytes. The function returns the number of bytes assigned to the output buffer (Out). The type parameter defines the compression technic to use.

Compression technics:

_RLE

RLE, or Run Length Encoding, is a very simple method for lossless compression. It simply replaces repeated bytes with a short description of which byte to repeat, and how many times to repeat it. Though simple and obviously very inefficient fore general purpose compression, it can be very useful at times (it is used in JPEG compression, for instance).

_HUFFMAN

Huffman encoding is one of the best methods for lossless compression. It replaces each symbol with an alternate binary representation, whose length is determined by the frequency of the particular symbol. Common symbols are represented by few bits, while uncommon symbols are represented by many bits. The Huffman algorithm is optimal in the sense that changing any of the binary codings of any of the symbols will result in a less compact representation. However, it does not deal with the ordering or repetition of symbols or sequences of symbols.

_LZ

There are many different variants of the Lempel-Ziv compression scheme. The Basic Compression Library has a fairly straight forward implementation of the LZ77 algorithm (Lempel-Ziv, 1977) that performs very well, while the source code should be quite easy to follow. The LZ coder can be used for general purpose compression, and performs exceptionally well for compressing text. It can also be used in combination with the provided RLE and Huffman coders (in the order: RLE, LZ, Huffman) to gain some extra compression in most situations.

See the COMPRESS.PPL demo provided with the PPL package for sample code.

UNCOMPRESS (In, Out, InSize)

Uncompress memory buffer (In) to output memory buffer (Out). The size of the input buffer must be provided in (InSize). See the Compress () function for more details about compression technics.

string TRIM(string Source)

Remove leading and trailing spaces from a string

Parameters

Source {in}
String to remove spaces from

Return Value

TRIM returns *Source* minus all leading and trailing spaces

Example:

See LTRIM for an example

See Also: LTRIM, RTRIM

string RTRIM(string Source)

Remove trailing spaces from a string

Parameters

Source {in}
String to remove spaces from

Return Value

RTRIM returns *Source* minus all trailing spaces

Example:

See LTRIM for an example

See Also: TRIM, LTRIM

string LTRIM(string Source)

Remove leading spaces from a string

Parameters

Source {in}
String to remove spaces from

Return Value

LTRIM returns *Source* minus all leading spaces

Example:

```
s$ = "    This is a string   ";
ShowMessage(ltrim(s$));   //Displays "This is a string   "
ShowMessage(rtrim(s$));   //Displays "    This is a string"
ShowMessage(trim(s$));    //Displays "This is a string"
```

See Also: TRIM, RTRIM

string LPAD(string Source, char Pad, int Count)

Pads *Source* with *Count* instances of *Pad*

Parameters

Source {in}
String you wish to pad

Pad {in}
Character to pad the string with

Count {in}
Number of instances of *Pad* to pad the string with

Return Value

LPAD returns a string with *Pad* repeated *Count* times plus *Source*

Example:

```
s$ = "PPL";
s$ = lpad(s$, '_', 10);            // s$ = "__________PPL"
```

See Also: RPAD

string RPAD(string Source, char Pad, int Count)

Pad string with a number (count$) of characters (character$) to the right. The return value is the new string.

Parameters

Source {in}
String you wish to pad

Pad {in}
Character to pad the string with

Count {in}
Number of instances of *Pad* to pad the string with

Return Value

RPAD returns a string with *Source* plus *Pad* repeated *Count* times

Example:

```
s$ = "PPL";
s$ = rpad(s$, '_', 10);            // s$ = "PPL__________"
```

See Also: LPAD

SOUNDEX (SoundEx, WordString, LengthOption, CensusOption) -> Value

A Soundex search algorithm takes a word, such as a person's name, as input and produces a character string which identifies a set of words that are (roughly) phonetically alike. It is very handy for searching large databases when the user has incomplete data.

The U.S. census has been making use of SoundEx codes to index surnames since the late 1800's. Those doing census lookups must use the same method to encode surnames as the census takers did when they generated the database. That means, for starters, our clever set of enhancements can't be used.

CensusOption SoundEx Code Returned w
0 Not census codes Enhanced SoundEx as documented here w
1 Normal census codes Used in all censuses including 1920 and beyond w
2 Special census codes Used intermittently in 1880, 1900, 1910 censuses w

CRC16 (String, Count) -> CrcValue

CRC-16 is an acronym for the 16 bit Cyclical Redundancy Check algorithm. CRC-16 generally refers to a specific 16 bit CRC formula sanctioned by the CCITT, an international standards body primarily concerned with telecommunications.

CRC calculations are done using a technique with the formidable name of "polynomial division". A block of data, regardless of how long, is treated as if each bit in the block is the coefficient in a long polynomial.

Example:

```
crc$ = CRC16("PPL SOFTWARE", 3);      // Only calculate the CRC-16 on the first
3 letters of the string.
```

CRC32 (String, Count) -> CrcValue

CRC-32 is an acronym for the 32 bit Cyclical Redundancy Check algorithm. CRC-32 generally refers to a specific 32 bit CRC formula sanctioned by the CCITT, an international standards body primarily concerned with telecommunications.

CRC calculations are done using a technique with the formidable name of "polynomial division". A block of data, regardless of how long, is treated as if each bit in the block is the coefficient in a long polynomial.

Example:

```
crc$ = CRC32("PPL SOFTWARE", 3);      // Only calculate the CRC-16 on the first
3 letters of the string.
```

HASH (HashAlgorithm, Buffer, Len) -> HashCode

Possible hashing algorithms:

DEFAULTHASH
RSHASH
JSHASH
PJWHASH
ELFHASH
BKDRHASH
SDBMHASH
DJBHASH
APHASH
MD5

Taken from http://www.partow.net

Hash functions are by definition and implementation pseudo random number generators (PRNG). From this generalization its generally accepted that the performance of hash functions and also comparisons between hash functions can be achieved by treating hash function as PRNGs.

Analysis techniques such a Poisson distribution can be used to analyze the collision rates of different hash functions for different groups of data. In general there is a theoretical hash function known as the perfect hash function for any group of data. The perfect hash function by definition states that no collisions will occur meaning no repeating hash values will arise from different elements of the group. In reality its very difficult to find a perfect hash function, in practice it is recognized that a perfect hash function is the hash function that produces the least amount of collisions for a particular set of data.

The problem is that there are so many permutations of types of data, some highly random, others containing high degrees of patterning that its difficult to generalize a hash function for all data types or even for specific data types. All one can do is via trial and error find the hash function that best suites their needs.

RS Hash Function
A simple hash function from Robert Sedgwicks Algorithms in C book. I've added some simple optimizations to the algorithm in order to speed up its hashing process.

JS Hash Function
A bitwise hash function written by Justin Sobel

PJW Hash Function
This hash algorithm is based on work by Peter J. Weinberger of AT&T Bell Labs.

ELF Hash Function
Similar to the PJW Hash function, but tweaked for 32-bit processors. Its the hash function widely used on most UNIX systems.

BKDR Hash Function
This hash function comes from Brian Kernighan and Dennis Ritchie's book "The C Programming Language". It is a simple hash function using a strange set of possible seeds which all constitute a pattern of 31....31...31 etc, it seems to be very similar to the DJB hash function.

SDBM Hash Function
This is the algorithm of choice which is used in the open source SDBM project. The hash function seems to have a good over-all distribution for many different data sets. It seems to work well in situations where there is a high variance in the MSBs of the elements in a data set.

DJB Hash Function
An algorithm produced by Daniel J. Bernstein and shown first to the world on the comp.lang.c newsgroup. Its efficient

as far as processing is concerned.

DEK Hash Function
An algorithm proposed by Donald E. Knuth in The Art Of Computer Programming Volume 3, under the topic of sorting and search chapter 6.4.

AP Hash Function
An algorithm produced by me Arash Partow. I took ideas from all of the above hash functions making a hybrid rotative and additive hash function algorithm based around four primes 3,5,7 and 11. There isn't any real mathematical analysis explaining why one should use this hash function instead of the others described above other than the fact that I tried to resemble the design as close as possible to a simple LFSR. An empirical result which demonstrated the distributive abilities of the hash algorithm was obtained using a hash-table with 100003 buckets, hashing The Project Gutenberg Etext of Webster's Unabridged Dictionary, the longest encountered chain length was 7, the average chain length was 2, the number of empty buckets was 4579.

```
/*
 **************************************************************************
 *                                                                        *
 *          General Purpose Hash Function Algorithms Library              *
 *                                                                        *
 * Author: Arash Partow - 2002                                            *
 * URL: http://www.partow.net                                             *
 * URL: http://www.partow.net/programming/hashfunctions/index.html        *
 *                                                                        *
 * Copyright notice:                                                      *
 * Free use of the General Purpose Hash Function Algorithms Library is    *
 * permitted under the guidelines and in accordance with the most current *
 * version of the Common Public License.                                  *
 * http://www.opensource.org/licenses/cpl.php                             *
 *                                                                        *
 **************************************************************************
*/
```

MD5 hashing code was taken from:

```
/*
 ***********************************************************************
 ** md5.h -- Header file for implementation of MD5                    **
 ** RSA Data Security, Inc. MD5 Message Digest Algorithm              **
 ** Created: 2/17/90 RLR                                              **
 ** Revised: 12/27/90 SRD,AJ,BSK,JT Reference C version               **
 ** Revised (for MD5): RLR 4/27/91                                    **
 **   -- G modified to have y&~z instead of y&z                       **
 **   -- FF, GG, HH modified to add in last register done             **
 **   -- Access pattern: round 2 works mod 5, round 3 works mod 3     **
 **   -- distinct additive constant for each step                     **
 **   -- round 4 added, working mod 7                                 **
 ***********************************************************************
 */

/*
 ***********************************************************************
 ** Copyright (C) 1990, RSA Data Security, Inc. All rights reserved. **
 **                                                                   **
 ** License to copy and use this software is granted provided that    **
 ** it is identified as the "RSA Data Security, Inc. MD5 Message      **
 ** Digest Algorithm" in all material mentioning or referencing this **
 ** software or this function.                                        **
 **                                                                   **
 ** License is also granted to make and use derivative works          **
 ** provided that such works are identified as "derived from the RSA **
 ** Data Security, Inc. MD5 Message Digest Algorithm" in all          **
 ** material mentioning or referencing the derived work.              **
 **                                                                   **
 ** RSA Data Security, Inc. makes no representations concerning       **
```

*** either the merchantability of this software or the suitability ***
*** of this software for any particular purpose. It is provided "as ***
*** is" without express or implied warranty of any kind. ***
*** ***
*** These notices must be retained in any copies of any part of this ***
*** documentation and/or software. ***
****
**/*

string REVERSE(string Source)
Reverse the order of a string.

Parameters
Source {in}
The string you want to manipulate

Return Value
REVERSE returns a string whose characters are in the opposite order of *Source*

Example:

```
s$ = "WELCOME";
ShowMessage(Reverse(s$));      // Result is: EMOCLEW
```

string SWAPCASE(string Source)
Reverse the case of every alpha in *Source*

Parameters
Source {in}
The group of characters you wish to manipulate

Return Value
SWAPCASE returns a string whose alpha characters are the opposite case of the ones in *Source*

Example:

```
s$ = "Welcome";
ShowMessage(SwapCase(s$));      // Result is: wELCOME
```

See Also: CAPITALIZE

string CAPITALIZE(string Source)
Set all the alpha characters in *Source* to lower case, then set the first character to upper case if it's an alpha.

Parameters
Source {in}
The group of characters you wish to capitalize

Return Value
CAPITALIZE returns a string where the first letter is upper case and the rest of the letters are lower case

Example:

```
s$ = "welcome";
ShowMessage(Capitalize(s$));      // Result is: Welcome
```

See Also: SWAPCASE

int STRIP(string Source, string Characters)
Remove characters from a string.

Parameters

Source {in | out}
  The string you wish to manipulate

Characters {in}
  The list of characters you want to remove from *Source*

Return Value
STRIP returns the number of characters that are removed from *Source*

Example:

```
a$ = "ABCDEFG";
Strip(a$, "BDF");
ShowMessage(a$);      // ACEG
```

See Also: CHANGE

double INT (string Value)
converts a string value to a double data type

Parameters
Value {in}
  string to be converted

Return Value
INT returns a double variable representation of *Value*

Example:

```
x$ = "25";
y$ = "30";

ShowMessage(x$ + y$);                   // Result is 55
ShowMessage(Int(x$) + Int(y$));      // Result is 55
```

Notes:

- This function is very useful when used with API function calls, because PPL won't automatically convert the values to integers when making the calls.

See Also: STR

string STR (double Value)
Converts a double type value to a string value.

Parameters
Value {in}
  Double value to be converted

Return Value
STR returns a string variable representation of *Value*

Example:

```
x$ = 10;

ShowMessage(str(x$)+"TEST");            // Result will be "10TEST".
ShowMessage(x$+"TEST");                  // Result will be "10TEST".
```

See Also: INT, FSTR

string FSTR (double Value)
Format a double value using a comma as a thousands separator.

Parameters
Value {in}
Double value to be converted

Return Value
FSTR returns a string containing the formatted value

Example:

```
ShowMessage(fstr(1032));              // Display 1,032
ShowMessage(fstr(10320));              // Display 10,320
ShowMessage(fstr(103203));              // Display 103,203
ShowMessage(fstr(1024938.3945));        // Display 1,024,938.3945
```

See Also: STR

float VMAX(any Var)
Returns the maximum bound of a variable

Parameters

Var {in}
Variable to determine the maximum bound of

Return Value
See Notes for details on what VMAX returns

Example:

```
DIM(a$, 10);
ShowMessage(VMax(a$));            // Returns 10.
DIM(a$, 10, 10);
ShowMessage(VMax(a$));            // Returns 100.
s$ = "HELLO";
ShowMessage(VMax(s$));            // Returns 5.
struct(r$, "field1", "field2");
ShowMessage(VMax(r$));            // Returns 2.
```

Notes:
- For number variable it returns : 1.7E308
- For string variable it returns: length of the string
- For structure variable it returns: number of fields
- For array variable it returns: number of elements

boolean NEW(any Variable, int SizeInBytes)
Allocates memory for *SizeInBytes* bytes that *Variable* will point to.

Parameters
Variable {in}
The variable you wish to allocate memory for

SizeInBytes {in}
The number of bytes you wish to reserve

Return Value
NEW returns true if the memory was allocated, or false otherwise.

Internal types:

TBYTE

TSHORT
TWIDE
TINT
TUINT
TDOUBLE
TLONG

Example:

```
if(New(a$, 100))
  &a$ = "HELLO";
  ShowMessage(a$);  //Displays HELLO in a message box
  free(a$);
end;
```

See Also: RESIZE, FREE

int MEMSIZE(any Address)

Provides the size in bytes of a memory block in the heap.

Parameters

Address {in}
  Memory location to find the size of

Return Value

MEMSIZE returns the size in bytes of the memory location pointed to by *Address*

Example:

```
new(s$, 1024);
&s$ = "This is string 1";
//Displays 16, the length of the data stored in s$
ShowMessage("Length of data: " + length(s$));
//Displays 1032, the size allocated to s$
ShowMessage("Size of memory location: " + memsize(s$));
free(s$);
```

See Also: SIZEOF, SIZE

void RESIZE (any Variable, int NewSizeInBytes)

Resize the allocation of a variable pointer to *NewSizeInBytes* bytes.

Parameters

Variable {in}
  The variable you wish to reallocate memory for

NewSizeInBytes {in}
  The number of bytes you wish to resize to

Example:

```
if (New(a$, 100))
  &a$ = "HELLO";
  ShowMessage(a$);
  resize(a$, 6);
  free(a$);
end;
```

Notes:

- The original data in memory is not erased or lost.

See Also: NEW, FREE

void FREE (any Address)

Free a memory location.

Parameters

Address {in}
  The location pointing to the memory you wish to free

Example:

```
if (New(a$, 100))
  &a$ = "HELLO";
  ShowMessage(a$);  //Displays "HELLO"
  free(a$);
end;
```

Notes:

- This function can also be used with objects.

See Also: NEW

{address} MALLOC(int Size)

Allocates *Size* bytes of memory, setting each byte to zero

Parameters

Size {in}
  Number of bytes to allocate

Return Value

MALLOC returns the address of the starting location of the allocated memory

Example:

```
m$ = malloc(1024);
memset(m$, 'A', 100);
free(m$);
```

See Also: MEMSET, FREE

boolean MEMSET(any Address, char Value, int SizeInBytes)

Set *SizeInBytes* bytes of memory location *Address* to *Value*

Parameters

Address {in}
  The variable you wish to reallocate memory for

Value {in}
  The character you wish to set each byte to

SizeInBytes {in}
  The number of bytes to set

Return Value

MEMSET returns true if the memory was modified, or false otherwise

Example:
See MALLOC

Notes:

- You can easily clear a memory location range using this function.

See Also: MALLOC, FREE

@ value

Converts an address value to a variable content value.

Example:

```
dim(a$, 10);
a$[5] = "This is a string";      // Stores the pointer of the string into a$[5].
s$ = @a$[5];                  // Converts address found in a$[5] to a string a
store it into s$
ShowMessage(s$);                 // Display "This is a string"
```

See Also: PTR

{contents} PTR(any Address)

Converts an address value to a variable content value.

Parameters

Address {in}
Pointer to the memory you wish to retrieve the contents of

Return Value

PTR returns the contents of the memory designated by *Address*

Example:

```
dim(a$, 10);
a$[5] = "This is a string";      // Stores the pointer of the string into a$[5].
s$ = ptr(a$[5]);                 // Converts address found in a$[5] to a string
a store it into s$
ShowMessage(s$);                 // Display "This is a string"
```

See Also: @

{pointer} MEMMOVE(any Src, any Dest, int Count)

Copies *Count* bytes from *Src* to *Dest*

Parameters

Src {in}
Memory location to copy information from

Dest {in | out}
Memory location to copy information to

Count {in}
Number of bytes to copy

Return Value

MEMMOVE returns a pointer to *Dest*

Example:

```
new(s$, 1024);
&s$ = "This is string 1";
new(t$, 1024);
&t$ = "My name is bob 2";
memmove(s$, t$, 14);
ShowMessage(t$); //Displays "This is string 2"
free(s$);
free(t$);
```

Notes:

- If some regions of the source area and the destination overlap, memmove ensures that the original source bytes in the overlapping region are copied before being overwritten

See Also: MEMCPY

{pointer} MEMCHR(any Buffer, char C, int Count)

Looks for the first occurrence of *c* in the first *count* bytes of *buffer*

Parameters

Buffer {in}
  Memory location of information you wish to search

C {in}
  Character you're searching for

Count {in}
  Number of bytes to search

Return Value

MEMCHR returns a pointer to the first match, or **null** if no match is found

Example:

```
new(s$, 1024);
&s$ = "This is a very long string with no letter before y";
p$ = memchr(s$, 's', 10);
//This block displays "s is a very long string with no letter before y"
if(p$ != null)
  ShowMessage(@p$);
else
  ShowMessage("Character not found");
end;
p$ = memchr(s$, 'x', length(s$));
//This block displays "Character not found"
if(p$ != null)
  ShowMessage(@p$);
else
  ShowMessage("Character not found");
end;

free(s$);
```

See Also: MEMCMP

{pointer} MEMCPY(any Dest, any Src, int Count)

Copies *Count* bytes from *Src* to *Dest*

Parameters

Src {in}
  Memory location to copy information from

Dest {in | out}
  Memory location to copy information to

Count {in}
  Number of bytes to copy

Return Value

MEMCPY returns a pointer to *Dest*

Example:

```
new(s$, 1024);
&s$ = "This is string 1";
new(t$, 1024);
&t$ = "My name is bob 2";
memcpy(t$, s$, 14);
ShowMessage(t$); //Displays "This is string 2"
free(s$);
free(t$);
```

Notes:

- If the source and destination overlap, this function does not ensure that the original source bytes in the overlapping region are copied before being overwritten

See Also: MEMMOVE

int MEMCMP(any Buf1, any Buf2, int Count)

The memcmp function compares the first *Count* bytes of *Buf1* and *Buf2*

Parameters

Buf1 {in}
   First memory location to compare

Buf2 {in}
   Second memory location to compare

Count {in}
   Number of bytes to compare

Return Value

MEMCMP returns 0 if the two buffers are identical for *Count* bytes, < 0 if *Buf1* is less than *Buf2*, or > 0 if *Buf1* is greater than *Buf2*

Example:

```
new(s$, 1024);
&s$ = "This is string 1";
new(t$, 1024);
&t$ = "This is string 2";
d$ = memcmp(s$, t$, 14);
ShowMessage(d$);  //Displays 0
d$ = memcmp(s$, t$, length(s$));
ShowMessage(d$);  //Displays -1 (s$ < t$)
free(s$);
free(t$);
```

See Also: MEMCHR

int FILL(any Container, [any Items...])

Populate *Container* with the values of *Items*

Parameters

Container {in | out}
   variable of a supported data type that you wish to populate

Items {in}
   one or more values to populate *Container* with

Return Value

FILL returns an integer containing the number of items that *Container* was populated with

Example:

```
struct(s$, "a", "b", "c");
Fill(s$, 10, 20, 30);
ShowMessage(s.a$+","+s.b$+","+s.c$);

dim(a$, 10);
Fill(a$, 10, 20, 30);
ShowMessage(a$[0]+","+a$[1]+","+a$[2]);
```

Notes:
Supported variable types are Structure, Array, List, Matrix and String

See Also: ADD

void SETINT({addr} Address, int Value)

Assigns *Value* as 4 bytes of data to the memory address specified in *Address*

Parameters

Address {out}
Memory location to write *Value* to

Value {in}
4 byte value to be written

Example:

```
type(i$, TINT);
type(l$, TLONG);

l$ = 0;
i$ = 10;

setint(&l$ + 2, i$);
ShowMessage(l$);  //Displays 655360
i$ = getint(&l$ + 2) + 5;
ShowMessage(i$);  //Displays 15
```

Notes:

- This is a specialized variant of the POKE function.

See Also: GETINT

int GETINT({addr} Address)

Get 4 bytes of data at memory location *Address*

Parameters

Address {in}
Memory location to retrieve data from

Return Value

GETINT returns an integer (4 bytes) starting at memory location *Address*

Example:

```
type(i$, TINT);
type(l$, TLONG);

l$ = 0;
i$ = 10;

setint(&l$ + 2, i$);
ShowMessage(l$);  //Displays 655360
i$ = getint(&l$ + 2) + 5;
ShowMessage(i$);  //Displays 15
```

See Also: SETINT

void SETBYTE({addr} Address, byte Value)

Assigns *Value* as 1 byte of data to the memory address specified in *Address*

Parameters

Address {out}

Memory location to write *Value* to

Value {in}
1 byte value to be written

Example:

```
type(b$, TBYTE);
type(l$, TLONG);

l$ = 0;
b$ = 10;

setbyte(&l$ + 2, b$);
ShowMessage(l$);  //Displays 655360
b$ = getbyte(&l$ + 2) + 5;
ShowMessage(b$);  //Displays 15
```

See Also: GETBYTE

byte GETBYTE({addr} Address)

Get 1 byte of data from memory location *Address*

Parameters

Address {in}
Memory location to retrieve data from

Return Value

GETBYTE returns a byte

Example:

```
type(b$, TBYTE);
type(l$, TLONG);

l$ = 0;
b$ = 10;

setbyte(&l$ + 2, b$);
ShowMessage(l$);  //Displays 655360
b$ = getbyte(&l$ + 2) + 5;
ShowMessage(b$);  //Displays 15
```

See Also: SETBYTE

void SETSHORT({addr} Address, short Value)

Assigns *Value* as 2 bytes of data to the memory address specified in *Address*

Parameters

Address {out}
Memory location to write *Value* to

Value {in}
2 byte value to be written

Example:

```
type(s$, TSHORT);
type(l$, TLONG);

l$ = 0;
s$ = 10;
```

```
setshort(&l$ + 2, s$);
ShowMessage(l$);  //Displays 655360
s$ = getshort(&l$ + 2) + 5;
ShowMessage(s$);  //Displays 15
```

See Also: GETSHORT

short GETSHORT({addr} Address)

Get 2 bytes of data from memory location *Address*

Parameters

Address {in}
  Memory location to retrieve data from

Return Value

GETBYTE returns a short

Example:

```
type(s$, TSHORT);
type(l$, TLONG);

l$ = 0;
s$ = 10;

setshort(&l$ + 2, s$);
ShowMessage(l$);  //Displays 655360
s$ = getshort(&l$ + 2) + 5;
ShowMessage(s$);  //Displays 15
```

See Also: SETSHORT

SETDOUBLE({addr} Address, double Value)

Assigns *Value* as 8 bytes of data to the memory address specified in *Address*

Parameters

Address {out}
  Memory location to write *Value* to

Value {in}
  8 byte value to be written

Example:

```
type(d$, TDOUBLE);
type(v$, TDOUBLE);

v$ = 0;
d$ = 1.5;

setdouble(&v$, d$);
ShowMessage(v$);  //Displays 655360
d$ = getdouble(&v$) + 5;
ShowMessage(d$);  //Displays 15
```

See Also: GETDOUBLE

double GETDOUBLE({addr} Address)

Get 8 bytes of data from memory location *Address*

Parameters

Address {in}
  Memory location to retrieve data from

Return Value
GETDOUBLE returns a double

Example:

```
type(d$, TDOUBLE);
type(v$, TDOUBLE);

v$ = 0;
d$ = 1.5;

setdouble(&v$, d$);
ShowMessage(v$);  //Displays 655360
d$ = getdouble(&v$) + 5;
ShowMessage(d$);  //Displays 15
```

See Also: SETDOUBLE

any PEEK({addr} Address, int Size)
Retrieve *Size* amount of bytes from the memory location *Address*

Parameters
Address {in}
Memory location to retrieve data from

Size {in}
Number of bytes to retrieve

Return Value
PEEK returns a numeric value if *Size* equates to TBYTE, TSHORT, TINT or TDOUBLE. Otherwise, PEEK returns a series of bytes

Example:

```
a$ = "ABCDEFG";
ShowMessage(Peek(&a$ + 2, TBYTE));      // Display ascii value of  'C'.
```

See Also: POKE

void POKE ({addr} Address, any Value, int Size)
Writes *Size* amount of bytes from *Value* to memory location *Address*

Parameters
Address {in}
Memory location to write data to

Value {in}
Variable containing bytes to be written

Size {in}
Number of bytes to write

Example:

```
a$ = "ABCDEF";
Poke(&a$ + 2, 'A', TBYTE);
ShowMessage(a$);            // Display "ABADEF"
```

Notes:
- Support for tbyte, tshort, tint and tdouble is provided

See Also: PEEK

void FREECARRAY(int Count, array Elements)

Free strings contained in a C array of strings

Parameters

Count {in}
  Number of arrays to free

Elements {in}
  Pointer to array

Example:

```
FreeCArray(10, @MyArray$);
```

See sql.ppl in the Lib directory for details on using FREECARRAY

DWORD TICK(void)

Returns the number of milliseconds that have elapsed since the OS was started

Return Value
TICK returns the number of milliseconds as a DWORD value

Example:

```
ShowMessage("The system has been running for " + Round(Tick / 1000) + "
seconds");
```

Notes:

- The resolution of the system timer is based on the OEMs setting. Check with the OEM for details
- The time will wrap around to zero if the system is run continuously for 49.7 days

void RUN(string Code)

Executes ppl code

Parameters

Code {in}
  The PPL code you wish to execute

Example:

```
code$ = "Global(s$);";
code$ = code$ + "s$ = 10;";
code$ = code$ + "ShowMessage(s$);";

Run(code$);
```

Notes:

- Check the global variable ERROR% for errors reported by the parser/compiler

See Also: RUNEX

void RUNEX(string Code)

Executes PPL code with runtime error checking activated

Parameters

Code {in}
  The PPL code you wish to execute

Example:

```
code$ = "Global(s$);";
code$ = code$ + "s$ = 10;";
code$ = code$ + "ShowMessage(s$);";

RunEx(code$);
```

Notes:

- By allowing PPL to check for runtime errors you get a better error description when a runtime error occurs
- Check the global variable ERROR% for errors reported by the parser/compiler

See Also: RUN

{pointer} EVAL(string Expression, int Scope, boolean FreeCode)

Evaluate an expression string within the same program scope as the calling program

Parameters

Expression {in}
  Code that is to be evaluated

Scope {in}
  Variable scope level to apply to evaluated code

FreeCode {in}
  Tells PPL whether or not to release the code once the EVAL statement is complete

Return Value

EVAL returns a pointer to the code generated by the EVAL statement. If *FreeCode* is false, you should keep track of this handle

Example:

```
s$ = 10;
Eval("ShowMessage(s$)", -1, true);
```

Notes:

- The scope parameter represents the variable scope level to use.

value	**meaning**
-1	Current procedure/function variables scope level. Uses the local variables storage.
-2	Current application global variables scope level. Uses the current application global variables storage.
-3	Own variables scope. You own eval variables allocation obtain with AllocEval().

void RUNFILE(string Filename)

Execute a ppl or ppc file

Parameters

Filename {in}
  Path and name of a PPL or PPC file

Example:

```
RunFile("\\My Documents\\test2.ppl");
```

Notes:

- Check the global variable ERROR% for errors reported by the parser/compiler

See Also: RUNFILEEX

void RUNFILEEX(string Filename)

Execute a .ppl or .ppc program file with runtime error checking activated

Parameters

Filename {in}
Path and name of a PPL or PPC file

Example:

```
RunFileEx("\\My Documents\\test2.ppl");
```

Notes:

- By allowing PPL to check for runtime errors you get a better error description when a runtime error occurs
- Check the global variable ERROR% for errors reported by the parser/compiler

See Also: RUNFILE

void COMPILE(string CodeOrFile)

Compiles a ppl code string or a ppl file

Parameters

CodeOrFile {in}
This can either be a variable containing PPL code, an explicit string, or the path and name of a PPL file

Example:

```
Compile("\\My Documents\\test2.ppl");
ShowMessage(Error%);
//If there were any errors during compilation, this will display them
```

Notes:

- Check the global variable ERROR% for errors reported by the parser/compiler

See Also: MAKEEXE

void MAKEEXE(string Filename)

Creates an .exe file (self-running) out of a ppl or ppc file

Parameters

Filename {in}
Path and name of a PPL or PPC file

Example:

```
MakeExe("\\My Documents\\test2.ppl");
ShowMessage(Error%);
//This will display any errors that occurred during compilation
```

Notes:

- Check the global variable ERROR% for errors reported by the parser/compiler

See Also: COMPILE

void SHOWMESSAGE(string Message, ...)

Displays a modal dialog with the text of each parameter concatenated together

Parameters

Message {in}
Text to display to the user

Example:

```
ShowMessage("Hi there end user");            //Dialog shows "Hi there end user"
username$ = "Fred";
ShowMessage("Hello, ", username$);            //Dialog shows "Hello, Fred"
```

See Also: DEBUG

void DEBUG(string Message)

Output a string to the debuglog.txt log file

Parameters

Message {in}
Text to write to the file

Example:

```
Debug("An error has occurred");
```

Notes:

- The default path is either the location of PPL if running a script or the directory of the executable if running a compiled .exe
- The path to the debug file can be changed with #DEBUGFILE compiler directive

See Also: SHOWMESSAGE

HANDLE FINDAPP(string AppName)

Find the application identified by *AppName*

Parameters

AppName {in}
Name of the application you are trying to locate

Return Value

FINDAPP returns a handle to the application if found, or **null** otherwise

Example:

```
app$ = FindApp("MyApplication.ppl");
if(app$ == null)
      ShowMessage("MyApplication not found");
else
      KillApp(app$);
end;
```

Notes:

- This function only works with PPL applications

See Also: KILLAPP

void KILLAPP(HANDLE AppHandle)

Terminates the application associated with *AppHandle*

Parameters

AppHandle {in}
Handle of the application to terminate

Example:

See FINDAPP for an example

Notes:

- Only use this to terminate another PPL application
- To terminate the calling application, use EXIT

See Also: FINDAPP, EXIT

void FREEAPP(HANDLE AppHandle)

Free an idle application from memory

Parameters

AppHandle {in}
Handle of the application to free

Example:

```
app$ = FindApp(AppName$);
Eval("FreeApp(" + app$ + ");", -1, true);
```

Notes:

- Only call this function within the EVAL function
- Only use this to terminate another PPL application
- To terminate the calling application, use EXIT

See Also: FINDAPP, EXIT, EVAL

string APPNAME(HANDLE AppHandle)

Get an application's name

Parameters

AppHandle {in}
Handle of the application you want the name of

Return Value

APPNAME returns the name of the application associated with *AppHandle*

Example:

See APPLICATIONS for an example

Notes:

- To get the name of the currently running application, use the global variable **AppName$**

See Also: APPLICATIONS

long APPSIZE(HANDLE AppHandle)

Get the size of the reserved code bytes for the application associated with *AppHandle*

Parameters

AppHandle {in}
Handle of the application in question

Return Value

APPSIZE returns a long containing the size in bytes of the reserved code bytes

See Also: APPBUFFER

{address} APPBUFFER(HANDLE App)

Returns the location address of the start of byte codes of an application

Parameters

App {in}
Handle of application to retrieve the address of

Return Value

APPBUFFER returns the starting location of the buffer for *App*

Example:

```
h$ = FindApp(name$);
if(h$ <> null)
  buf$ = AppBuffer(h$);
end;
```

See Also: FINDAPP

HWND APPFORM(HANDLE App)

Returns a handle to the main form of an application

Parameters

App {in}
Handle of application

Return Value

APPFORM returns the handle of *App*'s main form

Example:

```
h$ = FindApp(name$);
if(h$ <> null)
  hwnd$ = AppForm(h$);
  PostMessage(hwnd$, WM_CLOSE, 0, 0);
end;
```

See Also: FINDAPP

void DELAY(long Milliseconds)

Temporarily suspend program execution

Parameters

Milliseconds {in}
Amount of time to pause execution, expressed in milliseconds

void RESET(boolean HardReset)

Reset the PocketPC

Parameters

HardReset {in}
If true, the function will perform a hard reset of the device; false will perform a soft reset

string DEVICESERIAL(int ID)

Return the device's unique serial number

Parameters

ID {in}
2000 or 2002, depending on the OS of the device

Example:

```
ShowMessage(DeviceSerial(2000));
```

Notes:
2002 should actually work for 2002, 2003 and 2003SE

See Also: DEVICE

{string | numeric} DEVICE(int ID)

Returns processor information about a specific *ID* field

Parameters

ID {in}
Item to retrieve information about. See table in Notes for details

Return Value

DEVICE returns a string or numeric value, depending on *ID*; see table in Notes for details

Notes:

ID	Information	Data Type
0	Version	Numeric
1	Process Core	String
2	Core Revision	Numeric
3	Processor Name	String
4	Processor Revision	Numeric
5	CatalogNumber	String
6	Vendor	String
7	InstructionSet	String
8	Clock Speed	String
9	Platform Name	String
10	Device Vendor Name	String

Example:

```
for(i$, 0, 10)
  s$ = s$ + device(i$) + #13#10;
end;
ShowMessage(s$);
```

See Also: DEVICESERIAL

void WAITCURSOR(boolean Visible)

Set the WindowsCE wait cursor

Parameters

Visible {in}
If true, the wait cursor will be displayed, otherwise it will be hidden

Example:

```
if(LoadingData$ == true)
  SetWaitCursor(true);
else
  SetWaitCursor(false);
end;
```

HALT

Halt the current program execution right away.

{fileptr} FOPEN(string Filename, string Mode)

Opens a file on the PocketPC for manipulation

Parameters

Filename {in}
Path and name of file you wish to alter

Mode {in}
How you wish to open the file; common options include "w" (write), "r" (read), and "a" (append)

Return Value

FOPEN returns a handle to the file

Example:

```
f$ = fopen("\\My Documents\\test.txt", "w");
writestring(f$, "This is a line of text");
fclose(f$);

f$ = fopen("\\My Documents\\test.txt", "r");
s$ = readstring(f$);
ShowMessage(s$);            //displays the message "This is a line of text"
fclose(f$);
```

Notes:

- "r" opens the file as read only
- "w"rite opens a file for writing, destroying the file if it already exists
- "a"ppend opens the specified file and places the pointer at the end of the file; if the file doesn't already exist, it is created

See Also: FCLOSE

void FCLOSE(HANDLE FileHandle)

Closes a file that was opened using the FOPEN function

Parameters

FileHandle {in}
Handle returned from a call to FOPEN

Example:
See FOPEN for an example.

See Also: FOPEN

long FREAD(any Variable, int Size, int NumItems, HANDLE File)

Allows you to read from a file that has been opened with the FOPEN command

Parameters

Variable {out}
Item that stores the information read in

Size {in}
Bytes per item read in

NumItems {in}
Number of items to attempt to retrieve from the file

File {in}
Handle returned from a call to FOPEN

Return Value

FREAD returns the number of bytes actually read in from the file.

Example:

```
fp$ = fopen(AppPath$ + "test.txt", "w+"); // r+ allows for reading and writing
of file, but file must exist

//This writes one record the length of s$ to file fp$
// the resulting file should have the string
// "This is a string" stored in it
s$ = "This is a string";
fwrite(s$, sizeof(s$), 1, fp$);

fseek(fp$, 0, SEEK_SET);

//This creates a variable 5 bytes long and reads
// 1 "record" back from the file
sdim(s$, TBYTE, 5);
fread(s$, sizeof(s$)-1, 1, fp$);

ShowMessage(s$); //This will display the message "This" (since we only read the
first 4 bytes)

fclose(fp$);
```

See Also: FWRITE

long FWRITE (any Variable, int Size, int NumItems, HANDLE File)

Writes information to a file

Parameters

Variable {in}
Information to write to the file

Size {in}
Bytes per item to write out

NumItems {in}
Number of items to write to file

File {in}
Handle returned from a call to FOPEN

Return Value

FWRITE returns the number of bytes actually written to the file

Example:

See FREAD for an example

See Also: FREAD

void FSEEK(HANDLE File, long Offset, int Origin)

Moves the file pointer (if any) associated with stream to a new location

Parameters

File {in}
Handle retrieved by a call to FOPEN

Offset {in}
Distance in bytes from Origin to position the file pointer

Origin {in}
Position to start seek from. Can be one of the following values:
SEEK_CUR - Current position of file pointer
SEEK_END - End of file
SEEK_SET - Beginning of file

Notes:

- The next operation on the stream takes place at the new location
- On a stream open for update, the next operation can be either a read or a write
- The pointer can be positioned beyond the end of the file; fseek clears the end-of-file indicator and negates the effect of any prior ungetc calls against stream
- When a file is opened for appending data, the current file position is determined by the last I/O operation, not by where the next write would occur; if no I/O operation has yet occurred on a file opened for appending, the file position is the start of the file
- For streams opened in text mode, fseek has limited use, because carriage return-linefeed translations can cause fseek to produce unexpected results; the only fseek operations guaranteed to work on streams opened in text mode are:
  Seeking with an offset of 0 relative to any of the origin values.
  Seeking from the beginning of the file with an offset value returned from a call to ftell.
- Also in text mode, CTRL+Z is interpreted as an end-of-file character on input; in files opened for reading/writing, fopen and all related routines check for a CTRL+Z at the end of the file and remove it if possible; this is done because using fseek and ftell to move within a file that ends with a CTRL+Z may cause fseek to behave improperly near the end of the file

See Also: FTELL

string READSTRING(HANDLE FilePtr)

Reads the current line of the file associated with *FilePtr*

Parameters

FilePtr {in}
Handle to the file you wish to read from

Return Value

READSTRING returns the line of text read from *FilePtr*

Example:

```
file$ = "\\My Documents\\test.txt";

fp$ = fopen(file$, "w");
writestring(fp$, "This is some text");
fclose(fp$);

fp$ = fopen(file$, "r");
str$ = readstring(fp$);
fclose(fp$);

ShowMessage(str$);  //displays the message "This is some text"
```

See Also: WRITESTRING

void WRITESTRING(HANDLE FilePtr, string Text)

Writes a line of text to the file associated with *FilePtr*

Parameters

FilePtr {in}
Handle to the file you wish to write to

Text {in}
String you wish to write to the file

Example:

See READSTRING for an example

See Also: READSTRING

long FTELL(HANDLE FilePtr)

Provides the current position of *FilePtr*

Parameters

FilePtr {in}
  File to find position in

Return Value
FTELL returns the position of *FilePtr* as a long

Example:

```
fp$ = fopen("\\My Documents\\test.txt", "r");
s$ = readstring(fp$);
pos$ = ftell(fp$);

ShowMessage("Line two of file test.txt is " + pos$ + " characters into the
file");
```

Notes:

- The position within the file is zero based

See Also: FSEEK

long FEOF(HANDLE FilePtr)

Determines if *FilePtr* is at the end of the file or not

Parameters

FilePtr {in}
  File to determine state of

Return Value
FEOF returns non-zero if an attempt has been made to read past the end of *FilePtr*, otherwise FEOF returns zero

See Also: FSEEK, FTELL

long FERROR(HANDLE File)

Test for a read / write error on *File*

Parameters

File {in}
  Handle retrieved by a call to FOPEN

Return Value
FERROR returns the number of the error that was generated, or 0 for no errors

Example:

```
f$ = fopen("\\My Documents\\test.txt", "r");
fread(s$, 1, 1, f$);
if(ferror(f$) <> 0)
  ShowMessage("Error reading file");
end;
```

Notes:

- If an error has occurred, the error indicator for the stream remains set until the stream is closed or rewound, or until clearerr is called against it

See Also: FREAD, FWRITE

int FFLUSH(HANDLE FilePtr)

Writes the contents of the buffer for *FilePtr* to the physical file

Parameters

FilePtr {in}
File to flush

Return Value

On success, FFLUSH returns 0. Otherwise, FFLUSH returns EOF.

Example:

```
fp$ = fopen(AppPath$ + "test.txt", "w+");

s$ = "This is a string";
fwrite(s$, sizeof(s$), 1, fp$);

i$ = fflush(fp$);
if(i$ == 0)
  ShowMessage("File flushed successfully")
end;

fclose(fp$);
```

Notes:

- FFLUSH also returns 0 if the file has no buffer or if it has been opened read only
- If FFLUSH returns EOF, data may have been lost due to a write failure. When setting up a critical error handler, it is safest to turn buffering off with the SETVBUF function.

See Also: FWRITE

int FGETPOS(HANDLE File, POS Var)

Get the current position of an open file

Parameters

File {in}
Handle retrieved by a call to FOPEN

Var {out}
Variable to hold the current file-position indicator

Return Value

FGETPOS returns 0 if successful, or nonzero if it fails

Notes:

- Use FSETPOS to move to the file position stored by FGETPOS
- The value of *Var* is stored in an internal format and is intended for use only by FGETPOS and FSETPOS

See Also: FSETPOS

int FSETPOS(HANDLE File, POS Var)

Sets the current position of an open file

Parameters

File {in}
Handle retrieved by a call to FOPEN

Var {in}
File-position indicator as retrieved by FGETPOS

Return Value

FSETPOS returns 0 if successful, or nonzero if it fails

Notes:

- The function clears the end-of-file indicator and undoes any effects of ungetc on stream
- After calling fsetpos, the next operation on stream may be either input or output

See Also: FGETPOS

string GETFILE(string Filters, [string InitialDir])

Provides the user with a dialog for retrieving a file name

Parameters

Filters {in}
A series of extension / description combinations for various file types

InitialDir {in}
The directory to start in when the dialog is displayed; if omitted, the dialog will start in \My Documents

Return Value

GETFILE returns the full path and file name if a file is selected, or 0 otherwise

Example:

```
fn$ = GetFile("ADOCE Files (*.cdb)|*.cdb");
if(fn$ == 0)
  ShowMessage("No file name selected");
else
  //Do something with selected file
end;
```

See Also: PUTFILE

string PUTFILE(string Filters, [string InitialDir])

Parameters

Filters {in}
A series of extension / description combinations for various file types

InitialDir {in}
The directory to start in when the dialog is displayed; if omitted, the dialog will start in \My Documents

Return Value

PUTFILE returns the full path and file name if a file is selected, or 0 otherwise

Example:

```
fn$ = PutFile("ADOCE Files (*.cdb)|*.cdb");
```

```
if(fn$ == 0)
  ShowMessage("No file name provided");
else
  //Do something with selected file
end;
```

Notes:

- The main differences between GETFILE and PUTFILE are the title (GETFILE says "Open..." and PUTFILE says "Save As..."), and the fact that PutFile starts with the keyboard being displayed

See Also: GETFILE

boolean FILEEXISTS(string FileName)

Determine whether or not a particular file or directory exists

Parameters

FileName {in}
String containing a possibly valid path and / or file name

Return Value

FILEEXISTS returns true if *FileName* is found, or false otherwise

Example:

```
//On PocketPC
if(FileExists("\\My Documents\\test.txt"))
  ShowMessage("We have a file");
else
  ShowMessage("Sorry, no file");
end;

//On PC
if(FileExists("c:\windows"))
  ShowMessage("Probably Windows 9x or Windows XP");
else
  ShowMessage("Maybe Windows NT or Windows 2000");
end;
```

HANDLE NEWFORM(string Title, string ClassName, {pointer} ProcHandle)

Creates a new GUI form for a PPL application

Parameters

Title {in}
The text you wish to be displayed in the title bar of the application

ClassName {in}
Used to identify this window in a call to ENUMWINDOWS

ProcHandle {in}
Address of the callback function that will be used to handle all of the system calls that this new form will receive

Return Value

NEWFORM returns a handle to the newly created window

Example:

```
form$ = NewForm("My Title", "MyWindowClass", &WndProc);
menu$ = NewMenu(form$, "File", 400);
ShowWindow(form$, SW_SHOW);
```

Notes:

- If *ProcHandle* is NULL, the default message handler will be used, but you will not be able to trap any events for the form

- Keep track of the return handle so you can add controls to the form

See Also: NEWFORMEX, NEWDLG

HANDLE NEWFORMEX(string Title, string ClassName, long ExStyles, long Styles, int Left, int Top, int Width, int Height, {pointer} ProcHandle)

Creates a new GUI form for a PPL application using extended creation information

Parameters

Title {in}
The text you wish to be displayed in the title bar of the application

ClassName {in}
Used to identify this window in a call to ENUMWINDOWS

ExStyles {in}
One or more extended window styles. The values can be ORd together - for example, WS_EX_CAPTIONOKBTN | WS_EX_WINDOWEDGE. The available extended styles are listed in the PIDE when you create a new form, or you can find detailed descriptions on MSDN

Styles {in}
One or more window styles. See *ExStyles* for a more detailed description.

Left {in}
Upper left X coordinate of the form

Top {in}
Upper left Y coordinate of the form

Width {in}
Width in pixels of the form

Height {in}
Height in pixels of the form

ProcHandle {in}
Address of the callback function that will be used to handle all of the system calls that this new form will receive

Return Value

NEWFORMEX returns a handle to the newly created window

Example:

See CodeEditor.ppl in the RUNTIME\VFB subdirectory under the PPL install for an example

Notes:

- If *ProcHandle* is NULL, the default message handler will be used, but you will not be able to trap any events for the form
- Keep track of the return handle so you can add controls to the form

See Also: NEWFORM, NEWDLG

HANDLE NEWDLG(string Title, string ClassName, {pointer} ProcHandle, int Width, int Height)

Creates a new GUI form for a PPL application using dialog properties

Parameters

Title {in}
The text you wish to be displayed in the title bar of the dialog

ClassName {in}
Used to identify this window in a call to ENUMWINDOWS

ProcHandle {in}
Address of the callback function that will be used to handle all of the system calls that this new form will receive

Width {in}
Width in pixels of the form

Height {in}
Height in pixels of the form

Return Value
NEWDLG returns a handle to the newly created window

Example:

See SHOWMODAL for an example

Notes:
- If *ProcHandle* is NULL, the default message handler will be used, but you will not be able to trap any events for the form
- Keep track of the return handle so you can add controls to the form
- Use the SHOWMODAL function to display a window created with NEWDLG

See Also: NEWFORM, NEWFORMEX

long INHERITED(HANDLE hWnd, long Msg, long wParam, long lParam)
Allow your application to process the specified message

Parameters

hWnd {in}
Handle of the window that recieved the message

Msg {in}
Message that was recieved

wParam {in}
Optional data related to the message

lParam {in}
Optional data related to the message

Return Value
INHERITED returns the value generated by the default processing of *Msg*

Example:

```
func ListProc(hWnd$, Msg$, wParam$, lParam$)
  ok$ = true;
  case (Msg$)
    WM_LBUTTONDOWN:
      ok$ = inherited(hWnd$, Msg$, wParam$, lParam$);

      Struct (shrg$, SHRGINFO);

      shrg.cbSize$ = sizeof (shrg$);
      shrg.hwndClient$ = hWnd$;
      shrg.ptDownx$ = LOWORD(lParam$);
      shrg.ptDowny$ = HIWORD(lParam$);
      shrg.dwFlags$ = SHRG_RETURNCMD;

      if (SHRecognizeGesture(&shrg$) == GN_CONTEXTMENU)
```

```
        id$ = TrackPopupMenuEx(n$, TPM_LEFTALIGN | TPM_RETURNCMD, LOWORD
(lParam$), HIWORD(lParam$), hWnd$, NULL);
        SendMessage(f$, WM_COMMAND, MakeLong(0, id$), 0);
        ReleaseCapture;
        SetForegroundWindow(f$);
        SetFocus(l$);
      end;

  end;
  return (ok$);
end;
```

Notes:

- Assign the return value to ok$ so that the default windows processing does not happen again once the function has completed

long SHOWMODAL(HANDLE Dialog, HANDLE FocusControl, boolean FullScreen)

Show *Dialog* as a modal window, requiring input before other windows can be active again

Parameters

Dialog {in}
Handle of the dialog to show, created by a call to NEWDLG

FocusControl {in}
Handle of the control on *Dialog* that should recieve focus when the form is first shown

FullScreen {in}
Is the dialog full screen or not

Return Value

SHOWMODAL returns the value of the first button pressed that has an ID less than 100

Example:

```
//Code excerpted from the PocketPC version of the PPL Editor
FindDialog$ = NewDlg("Find...", "PPLFindForm", &FindDialogProc, 176, 121);
BUTTON101$ = NewControl(1, "BUTTON", NULL, "OK", WS_TABSTOP|WS_VISIBLE,
FindDialog$, 92, 84, 64, 24);
LABEL102$ = NewControl(102, "STATIC", NULL, "Search for:", WS_VISIBLE,
FindDialog$, 8, 8, 80, 16);
SearchCB$ = NewControl(107, "COMBOBOX", NULL, "",
CBS_DROPDOWN|CBS_NOINTEGRALHEIGHT|WS_VISIBLE|WS_TABSTOP|WS_VSCROLL, FindDialog$,
8, 24, 156, 100);
SendMessage(SearchCB$, CB_INITSTORAGE, 0, 0);
BUTTON104$ = NewControl(2, "BUTTON", NULL, "Cancel", WS_TABSTOP|WS_VISIBLE,
FindDialog$, 20, 84, 64, 24);
MatchCase$ = NewControl(105, "BUTTON", NULL, "Match case",
BS_AUTOCHECKBOX|WS_VISIBLE|WS_TABSTOP, FindDialog$, 8, 48, 96, 16);
WHOLEWORD$ = NewControl(106, "BUTTON", NULL, "Whole word",
BS_AUTOCHECKBOX|WS_VISIBLE|WS_TABSTOP, FindDialog$, 8, 64, 96, 16);

m1000$ = NewMenu(FindDialog$, "File", 1000);
NewMenuItem(m1000$, -1, "Exit", 1001);

ShowModal(FindDialog$, SearchCB$, false);

DestroyWindow(FindDialog$);
```

Notes:

- In order for SHOWMODAL to work properly, at least one control must have an ID less than 100
- Setting the parameter to fullscreen forces the application to resize the window to fit the screen

See Also: NEWDLG

HANDLE NEWFONT(HANDLE Window, string FontName, int Height, boolean Bold, boolean Italic, boolean Underline)

This will create a new font to be used with the Windows API.

Parameters

Window {in}
Handle of the window that the font will be associated with

FontName {in}
Name of the font family

Height {in}
Approximate desired height of the font. This value will be compared against the available sizes of the requested font and the closest one will be selected

Bold {in}
Whether or not the font should be bold

Italics {in}
Whether or not the font should be italicized

Underline {in}
Whether or not the font should be underlined

Return Value

NEWFONT returns the handle of the newly created font

Example:

```
f$ = NewForm("Editor", "EditorClass", &WndProc);
e$ = NewControl(1000, "EDIT", &EditProc, "", WS_BORDER | WS_VISIBLE | WS_HSCROLL
| WS_VSCROLL | ES_WANTRETURN | ES_MULTILINE | ES_AUTOVSCROLL | ES_AUTOHSCROLL,
f$, r.left$, r.top$, r.right$ - r.left$, r.bottom$ - r.top$ - SBHeight$);

fnt$ = NewFont(f$, "Tahoma", 10, false, false, false);
SetFont(e$, fnt$);
```

See Also: SETFONT

void SETFONT (HANDLE Window, HANDLE Font)

Set the default font for *Window*

Parameters

Window {in}
Handle of the window or control to recieve the new font

Font {in}
Handle of the new font, returned by a call to NEWFONT

Example:

See NEWFONT for an example

See Also: NEWFONT

void DESTROYWINDOW(HANDLE WindowHandle)

Removes the window associated with *WindowHandle* from memory

Parameters

WindowHandle {in}

Handle of the window to destroy

Example:

See SHOWMODAL for an example

See Also: NEWFORM, NEWDLG

void SHOWWINDOW(HANDLE FormHandle, long ShowCommand)

Manipulate the visibility of a form created with NEWFORM

Parameters

FormHandle {in}
The value returned from a call to NEWFORM

ShowCommand {in}
How to display the form; most common values are SW_SHOW and SW_HIDE

Example:

Please see NEWFORM for an example of using this function

Notes:

- For more detailed information on the *ShowCommand* options, check the definition of ShowWindow on msdn.microsoft.com
- Remember that doing a ShowWindow with mode set to SW_HIDE on all of your windows is NOT the same as shutting down the application

.**See Also:** NEWFORM, SHOWMODAL

void HANDLEMESSAGE(void)

Process the first message in the Windows message queue

See Also: PROCESSMESSAGES

void PROCESSMESSAGES(void)

Process all messages on the Windows message queue

See Also: HANDLEMESSAGE

void REGISTERCLASS(string ClassName, {pointer} ProcHandle)

Associates *ClassName* with *ProcHandle*. All windows created with class *ClassName* will route their messages through *ProcHandle*

Parameters

ClassName {in}
The class you wish to associate with *ProcHandle*

ProcHandle {in}
Pointer to the procedure to associate with *ClassName*

Example:

```
Func WndProc (hWnd$, Msg$, lParam$, wParam$)
  case (msg$)
    WM_PAINT:
         // Do something...
  end;
  return (true);
end;

func Main
  RegisterClass("MyClass", &WndProc);
  f$ = NewForm("MyWindow", "MyClass", NULL);
  ShowWindow(f$, SW_SHOWNORMAL);
  return (true);
end;
```

See Also: NEWFORM, UNREGISTERCLASS

void UNREGISTERCLASS(string ClassName)

Disassociate *ClassName* with it's current custom event handler

Parameters

ClassName {in}
  Class to unregister

See Also: REGISTERCLASS

void SETUSERDATA(HANDLE Window, any Value)

Assign a user string or integer value to *Window*

Parameters

Window {in}
  Valid window handle that you wish to associate *Value* with

Value {in}
  Value to associate with *Window*

Example:

```
//Assume that Form100$ is a valid window handle
SetUserData(Form100$, "This is a string to store");
s$ = GetUserData(Form100$);
ShowMessage(s$); //Displays "This is a string to store"
```

See Also: GETUSERDATA

any GETUSERDATA(HANDLE Window)

Retrieve a user string or integer value that has been assigned to *Window*

Parameters

Window {in}
  Handle of the window to retrieve user data from

Return Value

GETUSERDATA returns a string or integer value

Example:

See SETUSERDATA for an example

See Also: SETUSERDATA

DEFWINDOWPROC (hWnd, Message, wParam, lParam)

The DefWindowProc function calls the default window procedure to provide default processing for any window messages that an application does not process. This function ensures that every message is processed. DefWindowProc is called with the same parameters received by the window procedure.

HANDLE NEWMENU(HANDLE Window, string Caption, long ID)

Creates a new top level menu item for *Window*

Parameters

Window {in}
Handle of the window to create a menu item for

Caption {in}
Text to display for menu item

ID {in}
Identifier for menu item when responding to system calls

Return Value
NEWMENU returns a handle to the newly created menu

Example:

```
See NEWMAINMENU for an example
```

See Also: NEWMENUITEM

void NEWBUTTON(HANDLE Window, HANDLE Image, int ButtonIndex, long ID)
Create a new button on the main menu

Parameters

Window {in}
Handle of the window to create the button for

Image {in}
Handle of the graphic that stores the button images

ButtonIndex {in}
Position of the image within the graphic. This number is 0 based, and the images must be square. Image height is used as the determining factor for ratio. In other words, an image of 240 width and 24 height will contain 10 button images. To create a separator on the menu, pass -1 to this parameter

ID {in}
Identifier for menu item when responding to system calls

Example:

```
See NEWMAINMENU for an example
```

Notes:
- ImageHandle must be loaded using either LoadImage or SHLoadDIBitmap

See Also: NEWMENU

HANDLE NEWSUBMENU(HANDLE Menu, int InsertMenuId, string Caption, long ID)
Create a new submenu off of *Menu*

Menu {in}
Handle of the menu to create a new submenu for

InsertMenuId {in}
-1 to add the submenu to the end of the menu. A valid ID in this parameter will place the new submenu before the menu item with ID *InsertMenuId*

Caption {in}
Text to display for submenu

ID {in}
Identifier for submenu when responding to system calls. Set to 0 if submenu doesn't need to respond to any actions

Example:

See NEWMENU for an example

See Also: NEWMENU, NEWBUTTON

void NEWMENUITEM(HANDLE Menu, int InsertMenuId, string Caption, long ID)

Create a new item for *Menu*

Parameters

Menu {in}
Handle of the menu to create a new item for

InsertMenuId {in}
-1 to add the menu item to the end of the menu. A valid ID in this parameter will place the new menu item before the menu item with ID *InsertMenuId*

Caption {in}
Text to display for menu item

ID {in}
Identifier for menu item when responding to system calls

Example:

See NEWMENU for an example

See Also: NEWMENU, NEWSUBMENU

string MENUCAPTION(HANDLE MainMenu, long MenuID)

Returns the caption of a menu item

Parameters

MainMenu {in}
Handle to the main menu of the form containing the menu item in question

MenuID {in}
ID of the menu item to retrieve the caption from

Return Value

MENUCAPTION returns a string containing the text of the specified menu item

Example:

```
m$ = MainMenu(Form100$);
ShowMessage(MenuCaption(m$, 1001));
```

See Also: MAINMENU

HANDLE MAINMENU(HANDLE Window)

Retrieve the handle of the menu associated with *Window*

Parameters

Window {in}
Handle of the window to create a menu item for

See Also: NEWMAINMENU

void NEWMAINMENU(HANDLE Window)

Creates a blank menu bar for the specified window

Parameters

Window {in}
Handle of the window to create a menu bar for

Example:

```
func CreateMenu(form$)
  NewMainMenu(form$);
  m$ = NewMenu(form$, "File", 1000);
  NewMenuItem(m$, -1, "Exit", 1001);
  o$ = NewSubMenu(m$, -1, "SubFile", 0);
  NewMenuItem(o$, -1, "Sub1", 1002);

  i$ = SHLoadDIBitmap(root% + "tb.bmp");

  if(i$ <> 0)
    NewButton(form$, i$, -1, 0);
    NewButton(form$, i$, 5, 401); // New
    NewButton(form$, i$, 7, 402); // Open
    NewButton(form$, i$, 12, 403); // Save
  end;
end;
```

See Also: NEWMENU, MAINMENU

HANDLE NEWCONTROL(long ID, string ClassName, {pointer} ProcHandle, string Text, long Styles, HANDLE ParentWindow, int Left, int Top, int Width, int Height)

Add a new control of type *ClassName* to *ParentWindow*

Parameters

ID {in}
Value the application will use when sending messages to this control

ClassName {in}
Type of control; for example, a listbox would contain the value "LISTBOX"

ProcHandle {in}
Pointer to a user defined procedure for handling the control's messages; if NULL, the control's messages will be sent through the application's default message handler

Text {in}
If the control contains text, this parameter will set that text; for example, in a combo box this will set the text displayed in the combo before the user selects anything

Styles {in}
A long that represents the properties that you wish to set for this control; for example, WS_VISIBLE means the user will be able to see the control on the form; check MSDN for more specifics on this property based on the control you wish to create

ParentWindow {in}
Handle of the window that will contain this control; normally will be the form the control is on, but it could also be the handle to a group box or other type of control that can act as a container control

Left {in}
Upper X coordinate of control

Top {in}
Upper Y coordinate of control

Width {in}
Horizontal size of control

Height {in}
Vertical size of control

Return Value

If successful, NEWCONTROL returns a handle to the newly created control; otherwise, NEWCONTROL returns 0

Example:

```
LISTBOX101$ = NewControl(101, "LISTBOX", NULL, "",
WS_VISIBLE|WS_TABSTOP|LBS_STANDARD, FORM100$, 8, 8, 208, 144);
```

For more examples, create a new form in the Visual Form Builder, add some controls to it, then select Create Source from the Form menu

See Also: NEWFORM, NEWFORMEX

void SETSELSTART(HANDLE Ctl, long StartPos)

Sets the position of the first character in the selected portion of text of an Edit or Memo control

Parameters

Ctl {in}
Handle of the control to select text in

StartPos {in}
Position within *Ctl* to start selection

Example:

```
//Supose you have an Edit control with the text "This is an edit control"
SetSelStart(EDIT100$, 2);
SetSelLength(EDIT100$, 10);
s$ = GetSelText(EDIT100$);
ShowMessage(s$); //Displays "is is an e"
```

See Also: GETSELSTART, GETSELLENGTH, SETSELLENGTH

long GETSELSTART(HANDLE Ctl)

Get the position of the first character of selected text in *Ctl*

Parameters

Ctl {in}
Handle of the control that has selected text

Return Value
GETSELSTART returns the 0 based position of the selected text

Example:

```
i$ = GetSelStart(Edit101$);
j$ = GetSelLength(Edit101$);
s$ = GetText(Edit101$);
sel$ = Mid(s$, i$, j$);
ShowMessage("Selected Text: " + sel$);

//Of course, you can always use GETSELTEXT to accomplish the same thing
```

See Also: SETSELSTART, GETSELLENGTH, SETSELLENGTH

void SETSELLENGTH(HANDLE Ctl, long Length)

Set the length of the selected area of text in an Edit or Memo control

Parameters

Ctl {in}
Handle of Edit or Memo control

Length {in}
Number of characters to mark as selected

Example:

See SETSELSTART for an example

See Also: GETSELSTART, SETSELSTART, GETSELLENGTH

long GETSELLENGTH(HANDLE Ctl)

Get the number of characters currently selected in *Ctl*

Parameters

Ctl {in}
Handle of the control with selected text

Return Value

GETSELLENGTH returns the length of the selected text in *Ctl*

Example:

See GETSELSTART for an example

See Also: GETSELSTART, SETSELSTART, SETSELLENGTH

void SETSELTEXT(HANDLE Ctl, string Text)

Sets the value of the currently highlighted section of *Ctl* to *Text*

Parameters

Ctl {in}
Handle of the edit or memo control whose selected text is to be changed

Text {in}
Value to replace the selected text with

Example:

```
//Supose you have an Edit control with the text "This is an edit control"
SetSelStart(EDIT100$, 2);
SetSelLength(EDIT100$, 10);
SetSelText(EDIT100$, "$$$$$$$$$$");
s$ = GetText(EDIT100$);
ShowMessage(s$); //Displays "Th$$$$$$$$$$dit control"
```

See Also: SETSELSTART, SETSELLENGTH

string GETSELTEXT(HANDLE Ctl)

Retrieve the currently highlighted text in an edit or memo control

Parameters

Ctl {in}
Handle of the edit or memo control whose highlighted text you wish to retrieve

Return Value

GETSELTEXT returns a string containing the highlighted text

Example:

See SETSELSTART for an example

See Also: GETSELSTART, GETSELLENGTH

void SETTEXT(HANDLE Ctl, string Text)
Assign the caption of *Ctl* to *Text*

Parameters

Ctl {in}
  Handle of control to set the caption for

Text {in}
  Contents of the caption

Example:

See GETTEXT for an example

Notes:
If *Ctl* is a Label, SETTEXT will set the caption of the label to *Text*
If *Ctl* is an Edit or Memo control, SETTEXT will set the contents of the control to *Text*

See Also: GETTEXT

string GETTEXT(HANDLE Ctl)
Retrieves the caption of an Edit based control

Parameters

Ctl {in}
  Handle of the control to retrieve the caption from

Return Value
GETTEXT returns a string which is dependent on the control type. See Notes for more details.

Example:

```
SetText(EDIT100$, "This is some text");
s$ = GetText(EDIT100$);
ShowMessage(s$);  //Displays "This is some text"
```

Notes:
- If *Ctl* is a Label, GETTEXT returns the caption of the label
- If *Ctl* is an Edit or Memo control, GETTEXT returns the contents of the control
- If *Ctl* is a Combobox, GETTEXT returns the currently selected item

See Also: SETTEXT

string GETTEXTLINE(HANDLE Ctl, int Index)
Retrieve the text at line *Index* of *Ctl*

Parameters

Ctl {in}
  Handle of an Edit control to retrieve text from; control should be created with the ES_MULTILINE property

Index {in}
  Row whose text you wish to retrieve

Return Value

GETTEXTLINE returns the text found at row *Index* in the specified Edit control. Remember that *Index* is 0 based.

Example:

```
Edit_Clear(Edit101$);
Edit_Set(Edit101$, "This is\na multiline\ncontrol.");
s$ = GetTextLine(Edit101$, 1);
ShowMessage(s$); //Displays "a multiline"
```

Notes:

- Edit_Clear and Edit_Set are functions found in swapi.ppl

See Also: GETTEXT, SETTEXT

long GETITEMINDEX(HANDLE Ctl)

Retrieve the index of the currently selected item in *Ctl*

Parameters

Ctl {in}
Handle of the desired control

Return Value

GETITEMINDEX returns the 0 based index of the selected item, or a -1 if no item is selected

Example:

```
if(GetItemIndex(ListBox101$) == -1)
  SetItemIndex(ListBox101$, 0);
end;
```

Notes:

- Supported control types are LISTBOX and COMBOBOX

See Also: SETITEMINDEX

SETITEMINDEX(HANDLE Ctl, long Index)

Specify the currently selected item in *Ctl*

Parameters

Ctl {in}
Handle of the desired control

Index {in}
0 based index of item that should be selected

Example:

```
if(GetItemIndex(ListBox101$) == -1)
  SetItemIndex(ListBox101$, 0);
end;
```

Notes:

- Supported control types are LISTBOX and COMBOBOX

See Also: GETITEMINDEX

void SETITEM(HANDLE Ctl, long Index, string Text)

Sets the item at position *Index* to *Text* for the specified control

Parameters

Ctl {in}
   Handle of the desired control

Index {in}
   0 based index of item that should be modified

Text {in}
   New string to display

Example:

```
if(GetItem(ListBox101$, 0) == "none")
  SetItem(ListBox101$, 0, "some");
end;
```

Notes:

- Supported control types are LISTBOX and COMBOBOX
- The string value will be converted automatically to the right format (WideChar or Single-byte Char) depending on the device PPL running from

See Also: GETITEM

string GETITEM(HANDLE Ctl, long Index)

Retrieve the item at position *Index* for the specified control

Parameters

Ctl {in}
   Handle of the desired control

Index {in}
   0 based index of item that should be retrieved

Return Value

GETITEM returns a string containing the text of the item at position *Index*

Example:

```
if(GetItem(ListBox101$, 0) == "none")
  SetItem(ListBox101$, 0, "some");
end;
```

Notes:

- Supported control types are LISTBOX and COMBOBOX
- The string value will be converted automatically to the right format (WideChar or Single-byte Char) depending on the device PPL running from

See Also: SETITEM

void ADD(list Variable, [any Elements...])

Adds one or more new pointers to a linked list

Parameters

Variable {in | out}
   The list that you wish to add items to

Elements {in | optional}
   One or more items you wish to add to the list

Example:

```
list(items$);
cnt$ = 1;
```

```
while (cnt$ <= 10)
      Add(items$);      // adds a new item to the list and moves the
                        // list's pointer to the newly added item
      items$ = cnt$;      // assigns a value to the new item
end;

Add(item$, 1, 2, 3, 4, 5, 6, 7);
```

Notes:

- A linked list can hold any type of item
- Each item of a list can be of a different type

See Also: LIST, DEL, INS

void DEL(list Variable)

Deletes the current item from *Variable*

Parameters

Variable {in | out}
   List to remove the item from

Example:

```
list(l$);
cnt$ = 1;
while(cnt$ <= 10)
      add(l$, cnt$);
      cnt$++;
end;
goto(l$, 4);
del(l$);

msg$ = "";
foreach(l$)
      msg$ = msg$ + l$ + ",";
end;
ShowMessage(msg$);      //Displays "1,2,3,4,6,7,8,9,10,"
```

See Also: LIST, ADD, INS

int DELALL(list Source, string Find, string Delimiter, int Field, int StructField, long Options)

Delete all list items that match the criteria specified

Parameters

Source {in}
   the list variable you want to do the search on.

Find {in}
   value you are looking for.

Delimiter {in}
   If the strings in Source are delimited, specifies the value used to split the strings

Field {in}
   Which segment of each string to search if Delimiter was specified (see example for more details)

StructField {in}
   If *Source* is a list of structures, *StructField* is the field in the structure to search; field indexes are 0 based

Options {in}
   Specifies advanced search criteria, and can be one or more of the following:

FO_CASESENSITIVE	The search is case sensitive.
FO_FIRSTPART	Make sure the string you are looking for is only found at the beginning of the string sequence.
FO_LASTPART	Only look for the last part of the string sequence.
FO_POS	The string you are looking for can appear anywhere within the string sequence.
FO_TRIM	Trim the string sequence before doing the search.
FO_RTRIM	Right trim the string sequence before doing the search.
FO_LTRIM	Left trim the string sequence before doing the search.

Example:

```
list(l$);
Add(l$, " KEY1 = 10", " KEY2 = 11", " KEY2 = 12");
ShowMessage(DelAll(l$, "key2", "=", 0, FO_TRIM));      // RESULT is 2
ForEach(l$)
  ShowMessage(l$);      //  KEY1 = 10
end;

list(l$);
Add(l$, "A,B,C", "B,C", "A,B,C", "D,E");
ShowMessage(DelAll(l$, "A", ",", 1, FO_POS));      // RESULT is 3
ForEach(l$)
  ShowMessage(l$);      // D,E
end;
```

void INS(list Variable, [any Elements...])

Inserts one or more items into Variable at the list's current position

Parameters

Variable {in | out}
   List that you wish to add items to

Elements {in | optional}
   One or more items to insert into the list

Example:

```
list(l$);
cnt$ = 1;
while(cnt$ <= 10)
  add(l$, cnt$);
  cnt$++;
end;
goto(l$, 5);
ins(l$, "5.4", "5.3", "5.2", "5.1");

msg$ = "";
foreach(l$)
  msg$ = msg$ + l$ + ",";
end;
ShowMessage(msg$);  //Displays "1,2,3,4,5,5.1,5.2,5.3,5.4,6,7,8,9,10,"
```

Notes:

- Items will be inserted in front of the element being pointed to. In other words, if the list's pointer is currently on element 4, the inserted item will become the new element 4
- If you insert items using the *Elements* parameter, they will be inserted into the list in the opposite order of how they are referenced in the INS function

See Also: LIST, ADD, DEL

boolean LMOVE(list Variable, int FromIndex, int ToIndex)

Move a list item from *FromIndex* to *ToIndex*

Parameters

Variable {in | out}
List to swap items in

FromIndex {in}
Position of item to swap

ToIndex {in}
Position to move item to

Return Value
LMOVE returns true if move was successful, or false otherwise

Example:

```
List(l$);
Add(l$, 10, 20, 30, 40);
ShowMessage(listtostr(l$, ",", "", ""));      // "10,20,30,40"
LMove(l$, 2, 1);
ShowMessage(listtostr(l$, ",", "", ""));      // "10,30,20,40"
```

See Also: LIST, ADD

void COPY(list From, list To)
Copy the current element in *From* to the current position in *To*

Parameters

From {in}
List to copy an item from

To {in | out}
List to copy an item to

Example:

See LCOPY for an example

Notes:
- COPY only copies a single element. To copy an entire list, use LCOPY

See Also: LCOPY

long COUNT(list Variable)
Retrieve the number of elements in *Variable*

Parameters

Variable {in}
The list to count the elements of

Return Value
COUNT returns the number of elements that exist in *Variable*

Example:

```
list(lst$);
strtolist("1,2,3,4,5,6", ",", lst$);
ShowMessage(count(lst$));  //displays the message "6"
```

See Also: LIST

int FIND(list Source, int Start, int End, var Value, string Delimiter, int Field, int

StructField, int Options)
Locate a string within a list of strings

Parameters

Source {in}
List to serarch

Start {in}
Index of first item to search; -1 will start from the beginning of the list

End {in}
Index of last item to search; -1 will search until the end of the list

Value {in}
A string or numeric representing the data you wish to look for

Delimiter {in}
If the strings in the list are delimited, this specifies what delimiter was used

Field {in}
If the strings in the list are delimited, Field determines which section of string to search

StructField {in}
If the list elements are structures, *StructField* is the element within the structure to search. The first element of a structure is 0, and so on.

Options {in}
One or more flags to further narrow the scope of searching. The following values can be combined to the Options parameter:

FO_CASESENSITIVE	The search is case sensitive.
FO_FIRSTPART	Make sure the string you are looking for is only found at the beginning of the string sequence.
FO_LASTPART	Only look for the last part of the string sequence.
FO_POS	The string you are looking for can appear anywhere within the string sequence.
FO_TRIM	Trim the string sequence before doing the search.
FO_RTRIM	Right trim the string sequence before doing the search.
FO_LTRIM	Left trim the string sequence before doing the search.
FO_NUMERIC	Find the numeric value of the list element.

Return Value

FIND returns the 0 based index of the first item that matches the criteria, otherwise it returns -1

Example:

```
list(l$);
Add(l$, " KEY1 = 10", " KEY2 = 11");
ShowMessage(Find(l$, -1, -1, "key2", "=", 0, 0, FO_POS));      //Displays 1
ShowMessage(Find(l$, -1, -1, "key2", "=", 0, 0, FO_TRIM)); //Displays 1
ShowMessage(Find(l$, -1, -1, "key2", "=", 0, 0, FO_CASESENSITIVE |
FO_TRIM));  //Displays -1


//List of structures are also supported by the Find function

list(l$);
add(l$);
struct(l$, "a", "b");
l.a$ = "KEY1 = 10";
l.b$ = "KEY2 = 3";

add(l$);
struct(l$, "a", "b");
l.a$ = "KEY1 = 9";
```

```
l.b$ = "KEY2 = 11";

ShowMessage(Find(l$, -1, -1, "key2", "=", 0, 1, FO_POS));  //Displays 0
ShowMessage(Find(l$, -1, -1, 11, "=", 1, 1, FO_NUMERIC | FO_TRIM));  //Displays
1
```

void NEXT(list Variable)

Move to the next element of *Variable*

Parameters

Variable {in | out}
  List that you wish to traverse

Example:

```
list(lst$);
Add(lst$, "Hi", "There", 1, 2, 3);
msg$ = "";
First(lst$);
for(i$, 1, 5)
      msg$ = msg$ + lst$ + " ";
      Next(lst$);
end;
ShowMessage(msg$);      //Displays "Hi There 1 2 3 "

msg$ = "";
Last(lst$);
for(i$, 1, 5)
      msg$ = msg$ + lst$ + " ";
      Prev(lst$);
end;
ShowMessage(msg$);      //Displays "3 2 1 There Hi "
```

See Also: PREV, FIRST, LAST

void PREV(list Variable)

Move to the previous element of *Variable*

Parameters

Variable {in | out}
  List that you wish to traverse

Example:

See NEXT for an example

See Also: NEXT, FIRST, LAST

void FIRST(list Variable)

Move to the first element of *Variable*

Parameters

Variable {in | out}
  List that you wish to traverse

Example:

See NEXT for an example

See Also: NEXT, PREV, LAST

void LAST(list Variable)
Move to the last element of *Variable*

Parameters

Variable {in | out}
  List that you wish to traverse

Example:

See NEXT for an example

See Also: NEXT, PREV, FIRST

boolean ISFIRST(list Variable)
Determines if *Variable* is pointing to the first element of the list or not

Parameters

Variable {in}
  List to enquire about

Return Value
ISFIRST returns true (1) if *Variable* is on the first element of the list, or false otherwise

Example:

```
strtolist("a;b;c;d;e;f", ";", lst$);
goto(lst$, 3);
ShowMessage("IsFirst: " + IsFirst(lst$) + ", IsLast: " + IsLast(lst$));
first(lst$);
ShowMessage("IsFirst: " + IsFirst(lst$) + ", IsLast: " + IsLast(lst$));
last(lst$);
ShowMessage("IsFirst: " + IsFirst(lst$) + ", IsLast: " + IsLast(lst$));

//The resulting three messages are displayed:
//IsFirst: 0, IsLast: 0
//IsFirst: 1, IsLast: 0
//IsFirst: 0, IsLast: 1
```

See Also: ISLAST, FIRST, LAST

boolean ISLAST(list Variable)
Determines if *Variable* is pointing to the last element of the list or not

Parameters

Variable {in}
  List to enquire about

Return Value
ISLAST returns true (1) if *Variable* is on the last element of the list, or false otherwise

Example:

See ISFIRST for an example

See Also: ISFIRST, FIRST, LAST

int LPOS(list Items)
Returns the current list item pointer index

Parameters

Items {in}
  Variable containing a list

Return Value

LPOS returns an integer containing the current position of the pointer to *Items*

Example:

```
list(items$);
cnt$ = 1;
while(cnt$ <= 10)
      Add(items$);
      items$ = cnt$;
      cnt$++;
end;

First(items$);
ShowMessage(items$);      //shows a message box with "1"

Goto(items$, 8);
ShowMessage(lpos(items$)); //shows a message box with "8"

Goto(items$, 4);
ShowMessage(items$);      //shows a message box with "5" (remember, lists are 0
based)
```

Notes:

- Linked lists are 0 based, so the return value will be between 0 and (# of items - 1)

See Also: GOTO

void GOTO(list Items, int Index)

Position list *Items* to element *Index*

Parameters

Items {in}
  Variable containing a list

Index {in}
  Position in the list to move to

Example:

See LPOS for an example

Notes:

- Linked lists are 0 based, so *Index* should be one less than the position of the element you wish to move to

See Also: LPOS

boolean ISLIST(any Variable)

Determines if *Variable* is a list or not

Parameters

Variable {in}
  Variable that might be a list

Return Value

ISLIST returns true if *Variable* is a variable of type List, or false otherwise

Example:

```
str$ = "This is a string";
list(lst$);

//This code will display the message "lst$ is a list"
if(IsList(str$))
  ShowMessage("str$ is a list");
else if(IsList(lst$))
  ShowMessage("lst$ is a list");
else
  ShowMessage("You have no lists");
end;
```

See Also: LIST

any POP(list Variable)

Retrieve the value of the item at the current position in *Variable* and decrement *Variable*'s position by one

Parameters

Variable {in}
  The list to POP a value from

Return Value

POP returns whatever the current element of *Variable* is

Example:

```
list(l$);
Add(l$, 10, 20, 30);
ShowMessage(pop(l$));      // 30
ShowMessage(pop(l$));      // 20
ShowMessage(pop(l$));      // 10
```

See Also: PREV, GOTO

int SPLIT(list Source, int Index, list Dest)

Split *Source* at the specified *Index* and store all items after *Index* in *Dest*.

Parameters

Source {in}
  The list you wish to divide

Index {in}
  Position of last element to leave in original list

Dest {in}
  List to place remainder of elements into

Return Value

SPLIT returns the number of list items transfered to the destination list.

Example:

```
list(a$);
Add(a$, "H", "G", "F", "E", "D", "C", "B", "A");
Split(a$, 3, b$);

ForEach (b$)
  ShowMessage(b$);      // E,D,C,B,A
end;

ForEach (a$)
```

```
  ShowMessage(a$);      // H,G,F
end;
```

void SORT(list Source, boolean Ascending, boolean CaseSensitive)

Sort list *Source* in ascending or descending order

Parameters

Source {in | out}
The list you wish sort

Ascending {in}
True to sort the list in ascending order, false to sort in descending order

CaseSensitive {in}
If true, the strings will be sorted in a case sensitive manner (ex: Armadillo would come before aardvark); if false, case is ignored (ex: if aardvark is first going in, it will be first going out)

Example:

```
list(a$);
Add(a$, "H", "G", "F", "E", "D", "C", "B", "A");
Sort(a$, true, true);
ForEach (a$)
  ShowMessage(a$);      // A,B,C,D,E,F,G,H
end;

Sort(a$, false, true);
ForEach (a$)
  ShowMessage(a$);      // H,G,F,E,D,C,B,A
end;
```

int LTYPE(list Items)

Determine the variable type of a particular item in a list

Parameters

Items {in}
List containing item(s) in question

Return Value

LTYPE returns one of the following values: 0 - Numeric, 1 - String, 2 - Array, 3 - struct, 5 - Matrix

Example:

```
list(a$);
Add(a$, "Hello World!", 10, 20);

ShowMessage(LType(a$[0]) + ", " + LType(a$[1])+ ", " + LType(a$[2]));
//The result will be a dialog displaying the string "1, 0, 0"
```

See Also: VARTYPE

long APPLICATIONS (list Apps)

Retrieve the handles of all currently running PPL applications

Parameters

Apps {in | out}
Variable to hold the handles of all running applications

Return Value

APPLICATIONS returns the count of *Apps*

Example:

```
list(apps$);
c$ = Applications(apps$);
msg$ = "The following applications are running:\n";
foreach(apps$)
  msg$ = msg$ + AppName(apps$) + "\n";
end;
ShowMessage(msg$);
```

See Also: APPNAME

long FORMS(list Forms, HANDLE App)

Parameters

Forms {in | out}
  Variable to hold the list of available forms

App {in}
  Handle of the application whose forms you wish to enumerate

Return Value
FORMS returns the count of *Forms*

Example:

```
// Display all the forms for an application
//    hwnd$ is a valid handle to an application

Forms(forms$, hwnd$);
msg$ = "";
foreach(forms$)
  g$ = GetText(forms$);
  msg$ = msg$ % g$ % "\n";
end;
ShowMessage("Form List\n" % msg$);
```

See Also: APPLICATIONS

void ARRAYTOLIST(array Source, list Dest)

Creates a new item in *Dest* for each element in *Source*

Parameters
Source {in}
  array of items to convert to a list

Dest {out}
  variable to use as a list

Example:

```
Dim(a$, 4);
Fill(a$, 1, 2, 3, 4);
ArrayToList(a$, l$);
i$ = 0;
foreach(l$)
  i$ = i$ + l$;
end;

ShowMessage(i$);      //displays the string "10" (1 + 2 + 3 + 4)
```

```
goto(l$, 1);
l$ = 5;
ListToArray(l$, a$);
i$ = 0;
cnt$ = 0;
while (cnt$ < 4)
  i$ = i$ + a$[cnt$];
  cnt$++;
end;

ShowMessage(i$);  //displays the string "13" (1 + 5 + 3 + 4)
```

See Also: LISTTOARRAY

void LISTTOARRAY(list Source, array Dest)

Adds an element to *Dest* for each item in *Source*

Parameters

Source {in}
 list of items to convert to an array

Dest {out}
 variable to use as an array

Example:

See ARRAYTOLIST for an example.

See Also: ARRAYTOLIST

int STRTOLIST(string Source, string Delimeter, list Dest)

Create list *Dest* whose elements are substrings of *Source*

Parameters

Source {in}
 String to convert to a list

Delimeter {in}
 Character that separates each element for the list

Dest {out}
 Variable to contain the newly created list

Return Value

STRTOLIST returns the number of items added to the list

Example:

```
strtolist("A;B;C;D;E;F", ";", lst$);
if (IsList(lst$))
  goto(lst$, 3);
  ShowMessage(lst$);  //displays the string "D"
end;
```

Notes:

- The variable used to store the list does not need to be initialized first
- If the string contains multiple delimiter types, use STRTOLISTEX

See Also: STRTOLISTEX, LISTTOSTR

int STRTOLISTEX(string Source, string Delimiters, list Dest)

Create list *Dest* whose elements are substrings of *Source*

Parameters

Source {in}
String to convert to a list

Delimeter {in}
A string of one or more characters that separate each element for the list

Dest {out}
Variable to contain the newly created list

Return Value

STRTOLISTEX returns the number of items added to the list

Example:

```
strtolistex("A;B,C;D,E;F", ";,", lst$);
if (IsList(lst$))
  goto(lst$, 3);
  ShowMessage(lst$);  //displays the string "D"
end;
```

Notes:

- The variable used to store the list does not need to be initialized first

See Also: STRTOLIST, LISTTOSTR

string LISTTOSTR(list Source, string Separator, string Beforedelim, string Afterdelim)

Creates a string containing every element of *Source*

Parameters

Source {in}
List whose elements are to be written to a string

Separator {in}
Character used to separate each element

Beforedelim {in}
Character to place before each element of the list

Afterdelim {in}
Character to place after each element of the list

Return Value

LISTTOSTR returns a string containing all of the elements of *Source* separated and surrounded by *Separator*, *Beforedelim* and *Afterdelim*

Example:

```
list(lst$);
add(lst$, "A", "B", "C", "D", "E", "F");

str$ = listtostr(lst$, ",", "", "");
ShowMessage(str$); //Displays "A,B,C,D,E,F"
str$ = listtostr(lst$, ";", "'", "'");
ShowMessage(str$); //Displays "'A';'B';'C';'D';'E';'F'"
```

See Also: STRTOLIST, STRTOLISTEX

int STRUCTTOLIST(struct Source, list Dest)

Creates a list containing an element for each element of *Source*

Parameters

Source {in}
  Structure to write to the list

Dest {in | out}
  List to place the structure elements in

Return Value
STRUCTTOLIST returns an integer with the number of elements in the structure

Example:

```
Struct(s$, "A", "B", "C");
s.a$ = 10;
s.b$ = "HELLO";
s.c$ = 20;
list(l$);
StructToList(s$, l$);
ForEach (l$)
  ShowMessage(l$);
end;
```

See Also: LISTTOSTRUCT, STRUCTDEFTOLIST

int LISTTOSTRUCT(list Source, struct Dest)
Copy the value of each element in *Source* to an element in *Dest*

Parameters

Source {in}
  List to copy values from

Dest {in | out}
  Structure to write elements to

Return Value
LISTTOSTRUCT returns the number of elements in the list

Example:

```
list(l$);
add(l$, 1, "BOB", 23);
struct(s$, "A", "B", "C");
ListToStruct(l$, s$);
ShowMessage(s.a$ + "," + @s.b$ + "," + s.c$);
```

See Also: STRUCTTOLIST, STRUCTDEFTOLIST

int STRUCTDEFTOLIST(struct Source, list Dest)
Copy all of *Source's* element names to *Dest*

Parameters

Source {in}
  Structure to copy element names from

Dest {in | out}
  List to copy element names to

Return Value
STRUCTDEFTOLIST returns an integer containing the number of elements in *Source*

Example:

```
Struct(s$, "A", "B", "C");
list(l$);
StructDefToList(s$, l$);
ForEach(l$)
  ShowMessage(l$);  //Displays "A", "B", "C"
end;
```

See Also: STRUCTTOLIST, LISTTOSTRUCT

long FUNCTIONS(list Items, boolean DLL)

Return a list of all functions loaded into PPL

Parameters

Items {in | out}
Variable defined as a list to store the function names

DLL {in}
If true, only functions that points to a .dll file will be listed; if false, only internal functions will be listed

Return Value

FUNCTIONS returns the number of elements in *Items*

Example:

```
list(l$);
functions(l$, false);
s$ = listtostr(l$, #13#10, "", "");
ShowMessage(s$);
```

See Also: VARIABLES

long VARIABLES(list Items, int Scope)

Return a list of all existing variable names within the specified scope

Parameters

Items {in | out}
Variable defined as a list to store the variable names

Scope {in}
Range of variables to retrive. Possible values:
0 = Local to current program or procedure
1 = Global to current program or procedure
2 = Global to PPL

Return Value

VARIABLES returns the number of elements in *Items*

Example:

```
list(l$);
variables(l$, 0);
s$ = listtostr(l$, #13#10, "", "");
ShowMessage(s$);
```

See Also: FUNCTIONS

int ENUMWINDOWS(list Windows)

Populates *Windows* with a handle for each top-level window present on the screen

Parameters

Windows {out}
A list that is populated with the handle of each enumerated window

Return Value

ENUMWINDOWS returns the number of windows enumerated

int ENUMFONTFAMILIES(list Fonts, hdc Context, string FamilyName)

Enumerates the fonts in *FamilyName* that are available on a specified device as denoted by *Context*

Parameters

Fonts {out}
A list that is populated with information about each enumerated font

Context {in}
The handle used to enumerate the fonts

FamilyName {in}
The family to enumerate fonts for; use **null** to get a font from each family

Return Value

ENUMFONTFAMILIES returns the number of fonts enumerated

Example:

```
f$ = GetForeGroundWindow;
dc$ = GetDC(f$);
ShowMessage(EnumFontFamilies(v$, dc$, NULL));
ReleaseDc(f$, 0);
ForEach (v$)
  restruct(v$, "logfont", "textmetric", "fonttype");
  ShowMessage(v.logfont$+","+v.textmetric$+","+v.fonttype$);
end;
```

MADD (matrix$, value$) -> newmatrix$

Appends a new value (value$) to the end of matrix (matrix$). The old matrix (matrix$) is freed from memory and the new one is returned by the function.

Example:

```
a$ = [10, 20];
a$ = madd(a$, 30);
a$ = madd(a$, "STRING");
a$ = madd(a$, [100, 200]);
```

MDEL (matrix$, start$, count$) -> newmatrix$

Delete elements from matrix (matrix$) starting at start$ for count$ elements. The old matrix (matrix$) is freed from memory and the new matrix is returned by the function.

Example:

```
a$ = [10, 20, 30, 40, 50];
a$ = mdel(a$, 1, 3);      // a$ = [10, 50]

b$ = [10, 20, 30, 40, 50];
b$ = mdel(a$, 0, 2);      // b$ = [30, 40, 50]
```

MCOUNT (matrix$, recursive$) -> count

Count the number of elements in a matrix, if you specify recursive$ as TRUE, all matrix elements will be recursively counted too.

MMID (matrix$, start$, count$) -> matrix$

Return a new matrix made up of elements of matrix (matrix$) starting at start$ for count$ elements.

Example:

```
a$ = [10, 20, 30, 40, 50];
a$ = mmid(a$, 1, 3);      // a$ = [20, 30, 40]

b$ = [10, 20, 30, 40, 50];
b$ = mmid$(b$, 0, 2);      // b$ = [10, 20]
```

MTYPE (matrixelement$) -> type

Return the type of a matrix element. Each matrix element is stored with a byte specifying it's type.

Possible return values are:

ET_NUMERIC 1
ET_STRING 2
ET_MATRIX 3
ET_END 4

Example:

```
a$ = [10, "STRING", [10, 20]];
ShowMessage(mtype(a$[0]));      // ET_NUMERIC
ShowMessage(mtype(a$[1]));      // ET_STRING
ShowMessage(mtype(a$[2]));      // ET_MATRIX
ShowMessage(mtype(a$[2][0]));      // ET_NUMERIC
```

MATRIXTOARRAY (Matrix, Array) -> Count

Convert a matrix to an array variable. The number of elements in the array must match the number of elements in the matrix.

Example:

```
a$ = [10, 20, 30];
dim(m$, 3);
matrixtoarray(a$, m$);
ShowMessage(m$[0],",",m$[1],",",m$[2]);
```

MATRIXTOSTRING (Matrix) -> String

Return the string equivalent of a matrix. The string can then be converted back to a matrix using the stringtomatrix function.

STRINGTOMATRIX (String) -> Matrix

Convert a string to a matrix. The string format must be like this:

[1, 2, [1, 2], {String}, 3, 4]

G_INIT (HANDLE hWnd, ptr DrawProc, int Width, int Height, int Orientation, int AISpeed, int FPSSpeed, boolean FullScreen)

Initialize the graphics engine

Parameters

hWnd {in}
Window to render images to

DrawProc {in}
Pointer to a custom drawing routine; if NULL, PPL will do all the drawing for you, but you will have no control over it

Width {in}
Width of the drawing area

Height {in}

Height of the drawing area

Orientation {in}
Display mode: portrait, landscape, or the inverted version of each. Use the following constants:
ORIENTATION_UNKNOWN
ORIENTATION_NORTH
ORIENTATION_WEST
ORIENTATION_SOUTH
ORIENTATION_EAST

AISpeed {in}

FPSSpeed {in}

FullScreen {in}
Always set to true

Notes:
The PPL GameAPI uses the GAPI (gx.dll) to function
Only fullscreen graphics are supported at this point
The DrawProc is called by two events. WM_PAINT, when painting is necessary and WM_TIMER when processing is needed. WM_TIMER is called less frequently then WM_PAINT. Processing usely involves, handling key presses, moving sprites, updating counters... WM_PAINT should be reserved for drawing elements (sprites, text...) to screen to help the framerate not to go down too much. Keep the DrawProc optimized as much as you can

The FPSSpeed specifies at own many milliseconds the screen will be redrawn. The default value of 15 is about 60 FPS and will not go over that. If you don't want to limit the fps, set this value to 0.

The AISpeed parameter specify the frequency at which the GameAPI executes the DrawProc. A value of -1 means no GameProc or SpriteProc are called at all. PPL is intelligent enough to reduce the FPS speed if it cannot keep up with the AISpeed, this should leave more processing power for the AI (WM_TIMER).

Please check the Simple2.ppl demo program for more information about the GameAPI function.

Example:

```
Width$ = GetSystemMetrics(SM_CXSCREEN);
Height$ = GetSystemMetrics(SM_CYSCREEN);

g_init (hWnd$, &Draw, Width$, Height$, ORIENTATION_WEST, 2, True);
```

See Also: G_SHUT, G_SUSPEND, G_RESUME

void G_SHUT (long hWnd)
Shut down the graphics engine

Parameters

hWnd {in}
Window that the graphics engine is associated with

See Also: G_INIT, G_SUSPEND, G_RESUME

void G_SUSPEND(void)
Suspend GameAPI activities (graphics and sound)

Notes:
- Use G_RESUME to resume activities

See Also: G_INIT, G_SHUT, G_RESUME

void G_RESUME(void)
Resume GameAPI activities (graphics and sound)

Notes:
- Only call after issuing a G_SUSPEND

See Also: G_INIT, G_SHUT, G_SUSPEND

G_SETBLEND (AlphaBlending)

Set the following graphic operations alpha blending value. The value must be between 0 and 255. A blend of 0xFF will remove blending operation.

void G_BEGINSCENE(void)
Prepare the screen for drawing

Notes:
- This call must be ended with a G_UPDATE when all drawing is done
- Be careful not to use this function within the WM_PAINT event
- PPL does this by default unless you set AutoDraw to false

See Also: G_UPDATE

void G_UPDATE(void)
Update the screen display

Notes:
- All processing (drawing) is done on a backbuffer, so the screen needs to be updated to display any changes you've made
- Every G_UPDATE call needs to be done after a G_BEGINSCENE call

See Also: G_BEGINSCENE

void G_SPEED(long Speed)
Specify the amount of time between calls to WM_TIMER

Parameters

Speed {in}
amount of time in milliseconds between WM_TIMER calls; a value of -1 stops all processing of GameProc and SpriteProc functions

Notes:
The processing (WM_TIMER) is done in the following order:

1. Calls the DrawProc specified in G_INIT
2. Calls the SpriteProc specified in the call to LOADSPRITE for each sprite

void G_AUTODRAW(boolean AutoDraw)
Specifies who has control of rendering sprites

Parameters

AutoDraw {in}
True to allow the GameAPI to handle sprite rendering; False to handle the rendering on your own

Notes:

The internal drawing sequence is the following:

1. Prepare screen for drawing
2. Clear screen with black
3. Draw sprites

4. Calculate FPS (If activated)
5. Draw the FPS counter (If activated)
6. Update screen

void G_FILLRECT (int Left, int Top, int Right, int Bottom, long Color)

Draw a filled rectangle with the specified color

Parameters

Left {in}
Upper X coordinate of rectangle

Top {in}
Upper Y coordinate of rectangle

Right {in}
Lower X coordinate of rectangle

Bottom {in}
Lower Y coordinate of rectangle

Color {in}
Color to fill rectangle with

Example:

```
G_FillRect(10, 10, 100, 100, G_RGB(50, 50, 50));
```

Notes:

- Use the function G_RGB to get a valid value for *Color*

See Also: G_DRAWRECT, G_RGB

void G_DRAWRECT(int Left, int Top, int Right, int Bottom, long Color)

Draw an outline of a rectangle with the specified color

Parameters

Left {in}
Upper X coordinate of rectangle

Top {in}
Upper Y coordinate of rectangle

Right {in}
Lower X coordinate of rectangle

Bottom {in}
Lower Y coordinate of rectangle

Color {in}
Color to draw rectangle with

Example:

```
G_DrawRect(10, 10, 100, 100, G_RGB(50, 50, 50));
```

Notes:

- Use the function G_RGB to get a valid value for *Color*

See Also: G_FILLRECT, G_RGB

void G_CIRCLE (int X, int Y, int Radius, long Color)

Draw an outline of a circle with the specified color

Parameters

X {in}
  horizontal position of center of circle

Y {in}
  vertical position of center of circle

Radius {in}
  radius of circle

Color {in}
  Color to draw circle with

Example:

```
G_Circle(50, 50, 15, G_RGB(50, 50, 50));
```

Notes:

- Use the function G_RGB to get a valid value for *Color*

See Also: G_RGB

G_LINE (int X, int Y, int X2, int Y2, long Color)

Draw a line from (*X*, *Y*) to (*X2*, *Y2*) using *Color*

Parameters

X {in}
  starting horizontal position of line

Y {in}
  starting vertical position of line

X2 {in}
  ending horizontal position of line

Y2 {in}
  ending vertical position of line

Color {in}
  Color to draw line with

Example:

```
G_Line(10, 10, 100, 100, G_RGB(255, 0, 0));
```

Notes:

- Use the function G_RGB to get a valid value for *Color*

See Also: G_RGB

void G_CLEAR (int Color)

Clears the surface using Color

Parameters

Color {in}
  Color to fill the surface with

See Also: G_RGB

long G_RGB (int Red, int Green, int Blue)

Returns a color with the specified amounts of *Red*, *Green* and *Blue*

Parameters

Red {in}
  A value between 0 and 255

Green {in}
  A value between 0 and 255

Blue {in}
  A value between 0 and 255

Example:

```
G_RGB(50, 50, 50);  //Returns a shade of grey
```

Notes:

- If all three intensities are zero, the result is black. If all three intensities are 255, the result is white.

long FPS(void)

Returns the frames per second the graphics engine is rendering at

Return Value
FPS returns the current frames per second as a long

See Also: AVGFPS, SHOWFPS

long AVGFPS(void)

Returns the average frames per second that the GameAPI can produce

Return Value
Returns the average frames per second as a long

See Also: FPS, SHOWFPS

void SHOWFPS(boolean Show, long Color)

Display the current frames per second on the screen

Parameters
Show {in}
  True to display FPS, False to hide it

Color {in}
  Color to render text in

Example:

```
ShowFPS(True, G_RGB(255, 255, 255));  //Display FPS with white text
```

Notes:

- Displayed each time a frame is drawn

See Also: FPS, AVGFPS

FONTINFO G_LOADVGAFONT(string Filename, int FontWidth, int FontHeight)

Load a font for use with VGA text out functions

Parameters

FileName {in}

Name of file containing font information

FontWidth {in}
Number of pixels wide each character is

FontHeight {in}
Number of pixels high each character is

Example:

```
&f$ = G_LoadVGAFont(root%+"MyFont.fnt", 8, 16);
g_textout(f$, "This is a test!", 0, 0, 0, 0);
G_FreeVGAFont(@f$);  // This line is very important, it frees the font data from
memory.
```

See Also: G_FREEVGAFONT

void G_TEXTOUT(FONTINFO FontData, string Text, int Alignment, int X, int Y, long Color)

Render *Text* to the screen

Parameters

FontData {in}
Handle to a font retrieved using G_LOADVGAFONT

Text {in}
Characters to print to the screen

Left {in}
Upper horizontal coordinate of clipping rectangle

Top {in}
Upper vertical coordinate of clipping rectangle

Right {in}
Lower horizontal coordinate of clipping rectangle

Bottom {in}
Lower vertical coordinate of clipping rectangle; set to -1 to have G_DRAWTEXTEX automatically calculate the correct height of the rectangle based on the amount of text and the value of *WordWrap*

WordWrap {in}
When set to True, G_DRAWTEXTEX will automatically wrap the text at the specified boundary based on location of spaces within text (in other words, a word won't be split across two lines)

Draw a string (text) at (x,y) using the font (fontdata) with the color (Color). Alignment can either be:

The fontdata is the content of a VGA font info. The default value is NULL, it will use the standard system vga font.

Hundreds of VGA font files can be found in the FONTS.ZIP file that came with the PPL archive.

Example:

```
font$ = g_LoadVgaFont("\\My Documents\\fontvga.fnt", 8, 16);
G_TextOut(font$, "This is a string!", DVT_NONE, 0, 0, G_RGB(255, 255, 255));
free(@font$);
```

G_TEXTOUTEX (FontData, Text, Left, Top, Right, Bottom, Color, WordWrap) -> bottom$

Draw text to the screen with clipping and word wrap

Parameters

Handle {in}
Handle to a font retrieved using G_LOADFONT

Text {in}
Characters to print to the screen

Left {in}
Upper horizontal coordinate of clipping rectangle

Top {in}
Upper vertical coordinate of clipping rectangle

Right {in}
Lower horizontal coordinate of clipping rectangle

Bottom {in}
Lower vertical coordinate of clipping rectangle; set to -1 to have G_DRAWTEXTEX automatically calculate the correct height of the rectangle based on the amount of text and the value of *WordWrap*

WordWrap {in}
When set to True, G_DRAWTEXTEX will automatically wrap the text at the specified boundary based on location of spaces within text (in other words, a word won't be split across two lines)

Return Value
G_DRAWTEXTEX returns the vertical position of the last line of text rendered

Draw a string (text) using the font (fontdata) with the color (Color). The text is clipped and wordwrap inside the rect (Left, Top, Right, Bottom). If you specify a -1 for the bottom parameter, the function automatically calculates the correct height for the rectangle based on the amount of text and word wrapping. A value of -2 will not draw the text but just calculate the bottom value. The function returns the calculated bottom of the rectangle.

The fontdata is the content of a VGA font info. The fontdata is the content of a VGA font info. The default value is NULL, it will use the standard system vga font.

Hundreds of VGA font files can be found in the FONTS.ZIP file that came with the PPL archive.

Example:

```
font$ = g_LoadVgaFont("\\My Documents\\fontvga.fnt", 8, 16);
G_TextOutEx(font$, "This is \na string that is wrapped!", 10, 10, 60, 40, G_RGB
(255, 255, 255), true);
free(@font$);
```

See Also: G_TEXTOUT, G_LOADVGAFONT

FONT G_LOADFONT(string FontName, int Size, long Color, long Style)
Load a truetype font to be used with the GameAPI

Parameters

FontName {in}
The name (ex: "Arial") of the desired font; it must be an installed font in the target OS

Size {in}
Point size for the desired font; if you need more than one point size, each size will have to be a separate font

Color {in}
Color the font is to be rendered in; this must be set at load time

Style {in}
A combination of one or more values indicating how the font is to be formatted; see Notes for details

Example:

```
SystemFont$ = g_loadfont("Tahoma", 24, RGB(255, 255, 255), FONT_BOLD +
FONT_ITALIC);
g_drawtext(SystemFont$,  "Hello World!", 10, 10);
g_freefont(SystemFont$);
```

Notes:
The values for *Style* are as follows:

- FONT_NORMAL
- FONT_BOLD
- FONT_ITALIC
- FONT_UNDERLINE
- FONT_STRIKEOUT

See Also: G_FREEFONT, G_DRAWTEXT

void G_FREEFONT([FONT Handle...])
Free one or more fonts from memory

Parameters

Handle {in}
  A list of one or more font handles separated by commas that you wish to free

Example:

See G_LOADFONT for an example

See Also: G_LOADFONT, G_DRAWTEXT

int G_DRAWTEXTEX(FONT Handle, string Text, int Left, int Top, int Right, int Bottom, boolean WordWrap)
Draw text to the screen with clipping and word wrap

Parameters

Handle {in}
  Handle to a font retrieved using G_LOADFONT

Text {in}
  Characters to print to the screen

Left {in}
  Upper horizontal coordinate of clipping rectangle

Top {in}
  Upper vertical coordinate of clipping rectangle

Right {in}
  Lower horizontal coordinate of clipping rectangle

Bottom {in}
  Lower vertical coordinate of clipping rectangle; set to -1 to have G_DRAWTEXTEX automatically calculate the correct height of the rectangle based on the amount of text and the value of *WordWrap*

WordWrap {in}
  When set to True, G_DRAWTEXTEX will automatically wrap the text at the specified boundary based on location of spaces within text (in other words, a word won't be split across two lines)

Return Value
G_DRAWTEXTEX returns the vertical position of the last line of text rendered

Example:

```
SystemFont$ = g_loadfont("Tahoma", 24, RGB(255, 255, 255), FONT_BOLD +
FONT_ITALIC);
g_DrawTextEx(SystemFont$, "Hello World!", 10, 10, 100, 60, false);
```

See Also: G_DRAWTEXT

void G_DRAWTEXT(FONT Handle, string Text, int X, int Y)

Draw text to the screen

Parameters

Handle {in}
Handle to a font retrieved using G_LOADFONT

Text {in}
Characters to print to the screen

X {in}
Starting horizontal position to write the text to

Y {in}
Starting vertical position to write the text to

Example:

See G_LOADFONT for an example

Notes:

- The text is not clipped, and it is not wrapped if it longer than the width of the screen

See Also: G_DRAWTEXTEX

int G_TEXTWIDTH(FONT Handle, string Text)

Retrieve the width of *Text*

Parameters

Handle {in}
Handle to a font retrieved using G_LOADFONT

Text {in}
String to calculate width of

Return Value

G_TEXTWIDTH returns the length of the string in pixels based on *Handle*

Example:

```
SystemFont$ = g_loadfont("Tahoma", 24, RGB(255, 255, 255), FONT_BOLD +
FONT_ITALIC);
length$ = g_TextWidth(SystemFont$, "Is this too long?");
if(length$ > 100)
  ShowMessage("Your string is too long");
else
  ShowMessage("Your string is the right length");
end;
```

See Also: G_FONTHEIGHT

int G_FONTHEIGHT(FONT Handle)

Retrieve the height of *Handle*

Parameters

Handle {in}
Handle to a font retrieved using G_LOADFONT

Return Value
G_FONTHEIGHT returns the font's height in pixels

Example:

```
SystemFont$ = g_loadfont("Tahoma", 24, RGB(255, 255, 255), FONT_BOLD +
FONT_ITALIC);
ShowMessage(g_fontheight(SystemFont$)); //Displays 39
```

See Also: G_TEXTWIDTH

void G_SETPIXEL(int X, int Y, long Color)

Change the *Color* of the pixel at coordinates *X*, *Y*

Parameters

X {in}
horizontal position of pixel to change

Y {in}
vertical position of pixel to change

Color {in}
new color for pixel

Example:

```
newcolor$ = g_rgb(50, 50, 50);
oldcolor$ = g_getpixel(10, 10);
if(oldcolor$ <> newcolor$)
  g_setpixel(10, 10, newcolor$);
end;
```

See Also: G_GETPIXEL

long G_GETPIXEL(int X, int Y)

Retrieve the color of the pixel at *X*, *Y*

Parameters

X {in}
horizontal position of pixel to retrieve

Y {in}
vertical position of pixel to retrieve

Return Value
G_GETPIXEL returns the color of the pixel as a long

Example:

See G_SETPIXEL for an example

See Also: G_SETPIXEL

void G_SETCLIPPING(int Left, int Top, int Right, int Bottom)

Define a clipping rectangle. Drawing operations performed after this call will be contained within the bounds of the rectangle

Parameters

Left {in}
Upper X coordinate of rectangle

Top {in}
Upper Y coordinate of rectangle

Right {in}
Lower X coordinate of rectangle

Bottom {in}
Lower Y coordinate of rectangle

Notes:

- Anything rendered outside the boundaries of the rectangle will be lost

long G_WIDTH(void)

Retrieve the width in pixels of the graphic display

Return Value
G_WIDTH returns the display's width as a long

See Also: G_HEIGHT

long G_HEIGHT(void)

Retrieve the height in pixels of the graphic display

Return Value
G_HEIGHT returns the display's height as a long

See Also: G_WIDTH

LOCKINFO G_LOCK(SURFACE Object)

Lock surface pixels for fast pixel access

Parameters

Object {in}
The surface you wish to lock for drawing; NULL will lock the active surface

Return Value
G_LOCK returns a pointer to a LOCKINFO structure

Example:

```
g_beginscene;
struct(lockinf$, LockInfo);
&lockinf$ = G_Lock(NULL);
pitch$ = lockinf.pitch$ >> 1;
for (x$, 10, 20, 1)
  for (y$, 10, 20, 1)
    poke(lockinf.pixels$ + y$ * pitch$ + x$, g_rgb(255, 255, 255), tbyte);
  end;
end;
G_Unlock(lockinf$, NULL, false);
g_update;
```

Notes:

- Each G_LOCK must be terminated by a G_UNLOCK
- G_LOCK must be called within the WM_PAINT message, or between calls to G_BEGINSCENE and G_UPDATE
- If running through the PIDE, the program must be run in Exclusive mode or Access Violation errors will be generated

See Also: G_UNLOCK, G_BEGINSCENE, G_UPDATE

void G_UNLOCK(LOCKINFO lf, SURFACE Object, boolean Discard)

Unlock surface pixels

Parameters

lf {in}
Pointer to a LOCKINFO structure, retrieved using G_LOCK

Object {in}
Surface for which the LOCKINFO structure was retrieved; use NULL for the active surface

Discard {in}
True to undo any changes made to the buffer, False to leave changes

Example:

See G_LOCK for an example

Notes:

- You should set Discard to true when you are only doing read accesses

See Also: G_LOCK

HDC G_GETDC(SURFACE Object)

Returns the Windows device context of the specified surface

Parameters

Object {in}
Surface to capture the device context of

Return Value

G_GETDC returns the device context of *Object*

Example:

```
surface$ = NewSurface(100, 100);
hdc$ = G_GETDC(surface$);
G_RELEASEDC(surface$, hdc$);
```

See screenshot.ppl for an example of using these commands

Notes:

- Using the return value of G_GETDC allows you to use the standard Windows drawing functions on the surface
- You must at some point follow a call to G_GETDC with a call to G_RELEASEDC

See Also: G_RELEASEDC

void G_RELEASEDC(SURFACE Canvas, HDC Handle)

Free a device context retrieved using G_GETDC

Parameters

Canvas {in}
The surface the device context is attached to

Handle {in}
The device context that is being freed

Example:

See G_GETDC for an example

Notes:

- You must release the device context of a surface after a call to G_GETDC

See Also: G_GETDC

void G_GAMELOOP(long Milliseconds)

Run the game loop for a specific amount of time

Parameters

Milliseconds {in}
Amount of time, in milliseconds, to run the game loop for; if -1, the game loop will run until the form is closed

void G_SETPROC(addr Proc)

Sets the procedure that will be used by the GameAPI

Parameters

Proc {in}
Address of the procedure to be used

Example:

```
func GameProc(hWnd$, Msg$, wParam$, lParam$)
  //Do stuff here
end;

G_SETPROC(&GameProc);
```

SETAISPEED (Speed, GravitySpeed, AutoMoveSpeed)

Set the gameapi WM_TIMER event triggering speed with (Speed), the default value is -1.

The GravitySpeed is the elapse in milliseconds that gravity processing is being called, default value is 5.

AutoMoveSpeed is the elapse at which the automatic movement processing is called (VelX and VelY), the default value is 5.

void GAMECOLLIDE(boolean Active)

Specify whether the GameAPI should trigger a WM_COLLIDE event when sprites collide

Parameters

Active {in}
If true, the GameAPI will send a WM_COLLIDE event to the main GameProc when sprites collide; otherwise, the event will have to be handled by the individual sprite procedures. The default for this setting is false

SURFACE NEWSURFACE(int Width, int Height)

Create a new surface for rendering

Parameters

Width {in}

How wide to make the new surface

Height {in}
How tall to make the new surface

Return Value

NEWSURFACE returns a handle to the newly created surface

Example:

```
surface$ = NewSurface(100, 100);
os$ = SetRenderTarget(surface$);
g_BeginScene;
g_textout(null, "Test", DVT_NONE, 0, 0, G_RGB(255, 255, 255));
g_Update;
SaveSurface(surface$, "\\My Documents\\Test");
SetRenderTarget(os$);
FreeSurface(surface$);
```

Notes:

- To start painting on the surface, use G_BEGINSCENE; when you are finished, use G_UPDATE to update the scene
- Use FREESURFACE to delete the surface
- The global variable Buffer% always points to the main screen surface

See Also: LOADSURFACE, FREESURFACE

SURFACE LOADSURFACE(string Filename, long TransparentColor)

Load a bitmap into a GameAPI surface

Paramters

Filename {in}
File containing the bitmap to load

TransparentColor {in}
Color in the image to use as a transparent color; a -1 means no transparency

Return Value

LOADSURFACE returns the handle to the newly created surface

Example:

```
LoadSurf$ = LoadSurface("\\My Documents\\background.bmp", -1);
FreeSurface(LoadSurf$);
```

Notes:

- Use the RGB function instead of the G_RGB function with the *TransparentColor* parameter

See Also: NEWSURFACE**,** FREESURFACE

void FREESURFACE(SURFACE Handle, [...])

Parameters

Handle {in}
One or more surfaces to be freed from memory

Example:

See LOADSURFACE for an example

Notes:

- Surface handles are created with NEWSURFACE or LOADSURFACE

See Also: LOADSURFACE, NEWSURFACE

void SAVESURFACE(SURFACE Handle, string Filename)

Save a surface to disk

Parameters

Handle {in}
Handle retrieved from a call to LOADSURFACE or NEWSURFACE

Filename {in}
Path and name of file to save the surface as

Example:

```
if(screenshot$ == true)
  SaveSurface(surface$, AppPath$ + "curscreen");
end;
```

Notes

- You don't need to pass the extension. The default .bmp extension is used automatically.

See Also: LOADSURFACE

SURFACE SETRENDERTARGET(SURFACE NewTarget)

Set the default drawing surface

Parameters

NewTarget {in}
Surface that will become the new drawing surface

Return Value

SETRENDERTARGET returns the handle to the current rendering surface

Example:

```
mysurface$ = NewSurface(100, 100);
old$ = SetRenderTarget(mysurface$);
// Do some stuff
SetRenderTarget(old$);
```

Notes:

- All drawing operations will be performed on the new surface
- To restore the main GameAPI surface use BUFFER% as *NewTarget*

void DRAWSURFACE(SURFACE Source, SURFACE Dest, int DestX, int DestY, int SourceX, int SourceY)

Copy an image starting at the specified position from *Source* to the specified position on *Dest*

Parameters

Source {in}

Dest {in}

DestX {in}

DestY {in}

SourceX {in}

SourceY {in}

Example:

Notes:

See Also: DRAWSURFACEEX

DRAWSURFACE2 (SurfaceHandle, TargetSurface, DestX, DestY, DestWidth, DestHeight, SourceX, SourceY, SourceWidth, SourceHeight)

Same as DrawSurfaceEx with less parameters.

void DRAWSURFACEEX (SURFACE Source, SURFACE Dest, int DestX, int DestY, int DestWidth, int DestHeight, int SourceX, int SourceY, int SourceWidth, int SourceHeight, int Angle, Alpha, Tint, TintLevel, Light, int OffsetX, int OffsetY, int MirrorX, MirrorY, ClipLeft, ClipTop, ClipRight, ClipBottom)

Parameters

Source {in}
Surface to copy image from

Dest {in}
Surface to copy image to

DestX {in}
Starting X position for image on *Dest*

DestY {in}
Starting Y position for image on *Dest*

DestWidth {in}
Width of area to draw to on *Dest*

DestHeight {in}
Height of area to draw to on *Dest*

SourceX {in}
Starting X position for image on *Source*

SourceY {in}
Starting Y position for image on *Source*

SourceWidth {in}

SourceHeight {in}

Angle {in}

Alpha {in}

Tint {in}

TintLevel {in}

Light {in}

OffsetX {in}

OffsetY {in}

MirrorX {in}

MirrorY {in}

ClipLeft {in}

ClipTop {in}

ClipRight {in}

ClipBottom {in}

Draw surface (surface) on surface (TargetSurface) at position (SourceX, SourceY) using the current size (DestWidth, DestHeight).

The target surface can be buffer% for the main screen buffer or any other surface pointer.

You can rotate the surface with the (angle) parameter and you can also change the blending of the surface by using (alpha).

If you specify a tint other than -1, the surface will be drawn with this color using the Tintlevel for level of tinting.

You can also specify the amount of light the surface should be painted with. Light ranges from 0 to 255, a value of -1 doesn't use light.

You can also only draw a portion of the source surface on the screen using (SourceX, SourceY, SourceWidth, SourceHeight).

OffsetX and OffsetY draw the surface scrolled either horizontally or vertically.

MirrorX and MirrorY invert the surface horizontally or vertically.

ClipLeft, ClipTop, ClipRight, ClipBottom, specifies the clipping rectangle for the surface.

NB: **This function should be used inside a DrawProc procedure or within a G_BEGINSCENE and G_UPDATE.**

Example:

```
      surface$ = LoadSurface("\\My Documents\\Bitmap.bmp", -1);
      DrawSurfaceEx(surface$, NULL, 10, 10, 30, 30, 0, 0, 0, 0, 0, 50, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0);
```

void COPYSURFACE(SURFACE Source, SURFACE Dest)

Copy the contents of *Source* to *Dest*

Parameters

Source {in}
Surface to copy from

Dest {in}
Surface to copy to

Example:

```
SourceSurf$ = NewSurface(100, 100);
DestSurf$ = NewSurface(100, 100);
SetRenderTarget(SourceSurf$);
G_LINE(10, 10, 50, 50, G_RGB(255, 0, 0));
CopySurface(SourceSurf$, DestSurf$);
```

Notes:

- *Dest* must be big enough to hold the contents of *Source*

See Also: CLONESURFACE

SURFACE CLONESURFACE(SURFACE Source)
Duplicate an existing surface

Parameters

Source {in}
Surface to duplicate

Return Value
CLONESURFACE returns a handle to the newly created surface

Example:

```
OrigSurf$ = NewSurface(100, 100);
NewSurf$ = CloneSurface(OrigSurf$);
```

Notes:
If the destination surface has already been created, use COPYSURFACE instead

See Also: COPYSURFACE

void SURFACEEFFECT(SURFACE Source, long Effect, long Value)
Apply a permanent effect to a surface

. The value parameter is only used with SE_TINT as a color or SE_FADE as a value between 0 and 255.

Parameters

Source {in}
Surface to apply the effect to

Effect {in}
Effect to apply. Valid values include:
SE_BLUR
SE_NEGATIVE
SE_GREYSCALE
SE_TINT
SE_FADE

Value {in}
If *Effect* is SE_TINT, *Value* is a color (use the G_RGB function); if *Effect* is SE_FADE, *Value* is between 0 and 255; Otherwise, *Value* is ignored

Example:

```
SurfaceEffect(surf$, SE_TINT, G_RGB(255, 0, 0));  //Apply a red tint to the
surface
```

int SURFACEWIDTH (SURFACE Handle)
Retrieve the width of the specified surface

Parameters

Handle {in}
Surface to find the width of

Return Value
SURFACEWIDTH returns the width of *Handle* in pixels

See Also: SURFACEHEIGHT

int SURFACEHEIGHT(SURFACE Handle)

Retrieve the height of the specified surface

Parameters

Handle {in}
Surface to find the height of

Return Value
SURFACEHEIGHT returns the height of *Handle* in pixels

See Also: SURFACEWIDTH

void SETCOLORMASK(SURFACE Handle, long Color)

Set the transparent color of a surface

Parameters

Handle {in}
Surface to set the transparent color on

Color {in}
Color to set the transparency to; -1 for no transparency

Example:

```
surf$ = NewSurface(100, 100);
SetColorMask(surf$, G_RGB(100, 100, 100));
```

See Also: COLORMASK

long COLORMASK(SURFACE Handle)

Retrieve the transparent color of a surface

Parameters

Handle {in}
Surface to set the transparent color on

Return Value
COLORMASK returns the current transparent color of *Handle*

See Also: SETCOLORMASK

void SETORIGINX(float X)

Set the screen X location in a 2D plane space

Parameters

X {in}
New horizontal coordinate for the screen's origin

Example:

```
if(OriginX <> 0)
  SetOriginX(0);
end;
```

Notes:

- This is very usefull for RTS or RPG games, where the whole game can be scrolled just by changing the screen's origin

See Also: ORIGINX, ORIGINY, SETORIGINY

void SETORIGINY(float Y)

Set the screen Y location in a 2D plane space

Parameters

Y {in}
New vertical coordinate for the screen's origin

Example:

```
if(OriginY <> 0)
  SetOriginY(0);
end;
```

Notes:

- This is very usefull for RTS or RPG games, where the whole game can be scrolled just by changing the screen's origin

See Also: ORIGINX, SETORIGINX, ORIGINY

float ORIGINX(void)

Return the screen's origin X location

Return Value

ORIGINX returns a float representing the screen's horizontal origin coordinate

Example:

See SETORIGINX for an example

See Also: SETORIGINX, ORIGINY, SETORIGINY

float ORIGINY(void)

Return the screen's origin Y location

Return Value

ORIGINY returns a float representing the screen's vertical origin coordinate

Example:

See SETORIGINY for an example

See Also: ORIGINX, SETORIGINX, SETORIGINY

void SETMAPLIGHT(int Light)

Set the screen light intensity level

Parameters

Light {in}
level of intensity for dynamic lighting; 256 is the default, which is total light; 0 is total darkness

Notes:

- All sprites that uses dynamic lighting will be of this default light intensity

void SETMAPCOLOR(long Color)

Set the map background color

Parameters

Color {in}
New color for the background

Notes:

- This color is only used when in autodraw mode

See Also: MAPCOLOR

long MAPCOLOR(void)

Return the map background color

Return Value

MAPCOLOR returns a long containing the current background color for the GameAPI

See Also: SETMAPCOLOR

SETGRAVITY (Gravity)

Set the world gravity. Normal gravity is around 0.1.

GRAVITY -> Gravity

Return the world's gravity.

SETFRICTION (Friction)

Set the world's friction. Normal air friction is around 0.00025.

FRICTION -> Friction

Return the world's friction.

void SETLAYER(long ID, long ZOrder, int X, int Y, int OffsetX, int OffsetY, boolean Visible, int AutoOffsetX, int AutoOffsetY, int AutoScrollX, int AutoScrollY)

Set all sprites with the layer (Id) ZOrder, Visible state and also move the sprites by OffSetX, OffSetY.

The X, Y and replaces the sprite's position. Leave these values to 0 if you don't want the position of the sprites to change.

The zorder value is only modified if it's not equal to zero.

AutoOffset's and AutoScroll's values are updated also. Leave to 0 not to update the sprites.

SETBORDER (Left, Top, Right, Bottom)

Set the screen borders for automatic sprite collision when sprite is elastic or it has looping. This function sets the BORDER$ global variable automatically.

In the SpriteProc, the wParam$ and lParam$ are left at zero while the Sprite$ variable is set with the sprite handle that collided with the borders.

In the GameProc, the wParam$ is set with the sprite handle that collided with the borders while lParam$ is set to zero.

SETBACKSPRITE (Sprite)

Set the sprite that PPL will use to render the background of the scene. PPL will automatically render the sprite isometrically or tile it, depending on the display style of your game.

SETCOLLISIONGRID (Pixels, Width, Height)

Sets the collision detection grid size. The collision detection between sprites is done through a grid where the cells are 32 pixels by 32 pixels by default. The default number of cells horizontally and vertically are 128. If your map is bigger than 128 * 32 by 128 * 32 you should consider raising these limits using the SetCollisionGrid() function. The bigger the number of pixels per cell, the less accurate the collision detection will be but the faster it will be.

The grid size also handles pixel locations less than 0. The possible range is always -GridSize/2 to GridSize/2.

SETGRIDCELLS (ID, Left, Top, Right, Bottom)

Set a sprite's collision Id in the collision grid. If a sprite touched the area within (Left, Top, Right, Bottom) and that the collision Id matches, a collision will occur. You can as many collision id's as you want in the cells.

DELGRIDCELLS (ID, Left, Top, Right, Bottom)

Delete the collision id (ID) stored in the collision grid cells within this range (Left, Top, Right, Bottom).

CLEARGRIDCELLS (Left, Top, Right, Bottom)

Delete all collision id's stored in the collision grid cells within this range (Left, Top, Right, Bottom).

int DISTANCE(int X, int Y, int X1, int Y1)

Calculate the distance in pixels between (X,Y) and (X1,Y1)

Parameters

X {in}
  horizontal position of first coordinate

Y {in}
  vertical position of first coordinate

X1 {in}
  horizontal position of second coordinate

Y1 {in}
  vertical position of second coordinate

Return Value

DISTANCE returns the distance between the two sets of coordinates

Example:

```
dist$ = Distance(10, 10, 100, 100);
ShowMessage("distance: " + dist$ + " pixels");  //Displays "distance: 180
pixels"
```

See Also: MIDDLE

void MIDDLE(int X, int Y, int X1, int Y1, int MidX, int MidY)

Calculates the mid-point between (*X*,*Y*) and (*X1*,*Y1*)

Parameters

X {in}
  horizontal position of first coordinate

Y {in}
  vertical position of first coordinate

X1 {in}
  horizontal position of second coordinate

Y1 {in}
  vertical position of second coordinate

MidX {out}
  variable to hold horizontal position of mid-point

MidY {out}
  variable to hold vertical position of mid-point

Example:

```
Middle(10, 10, 100, 100, MidX$, MidY$);
ShowMessage("Middle: " + MidX$ + ", " + MidY$);  //Displays "Middle: 55, 55"
```

See Also: DISTANCE

COS256 (x) -> result

SIN256 (x) -> result

ANGLE (x, y, x2, y2) -> angle

Return the angle in degree between two points.

ADJUSTXY (X, Y, MapWidth, MapHeight, Isometric)

Adjust the variable X and Y position to fit within the cells of MapWidth and MapHeight. You can specify true for the isometric parameter.

Example:

```
WM_MOUSEMOVE:
  x$ = wParam$;
  y$ = lParam$;
  AdjustXY(x$, y$, 32, 32, False);
  Map[x$, y$] = NOT Map[x$, y$];
```

boolean WAITFORINPUT(long WaitTime)

Wait for user input or a specified amount of time to pass

Parameters

WaitTime {in}
Amount of time, in milliseconds, to wait for user input; if -1, process will wait indefinitely

Return Value

WAITFORINPUT returns true if a key was pressed or the stylus was used; if *WaitTime* elapses, WAITFORINPUT returns false

Example:

```
result$ = WaitForInput(3000);
if(result$ == false)
  ShowMessage("You waited needlessly for 3 seconds");
else
  ShowMessage("Congratulations on doing something!");
end;
```

PARTICLE NEWPARTICLE(long ID, SPRITE Actor, int X, int Y, float AccX, float AccY, float VelX, float VelY, long MaxCycle, boolean Loop, float Fade, boolean RandomCycle, boolean Physic)

Creates a new particle

Parameters

ID {in}
This number identifies the particle as part of a certain group; some particle functions operate based off of this ID

Actor {in}
The sprite to associate with this particle

X {in}
Starting horizontal position of the particle

Y {in}
Starting vertical position of the particle

AccelarationX {in}
Horizontal speed to add to the particle each cycle

AccelarationY {in}
Vertical speed to add to the particle each cycle

VelX {in}
The base speed at which the particle will move along the X axis

VelY {in}
The base speed at which the particle will move along the Y axis

MaxCycle {in}
The maximum number of cycles that the particle will last

Loop {in}
If true, once *MaxCycle* has been reached the particle will return to it's original X,Y coordinate and repeat its life cycle; otherwise, the particle will be deleted after *MaxCycle*

Fade {in}
The percentage of alpha blending applied each cycle; a value of -1 will prevent the particle from fading

RandomCycle {in}
If true, the particle will start it's cycle at some random value; otherwise, the particle will start its cycle at 0

Physic {in}
If true, world physics such as gravity and friction will apply to the particle; otherwise, the particle will be oblivious of world physics

Return Value
NEWPARTICLE returns a handle to the newly created particle

Example:

void CLEARPARTICLES(long ID)
Delete all particles of a certain *ID*

Parameters

ID {in}
ID of the particle group you wish to delete; if ID is 0, all particles will be deleted

Example:

```
if(player_died$ == true)
  ClearParticles(0);
end;
```

See Also: DELPARTICLE

void PROCESSPARTICLES(long ID)
Cycle all particles associated with the specified *ID*

Parameters

ID {in}
ID of the particle group you wish to cycle; if ID is 0, all particles are processed

Example:

for detailed examples on using particles, see particles.ppl and motionblur.ppl in the Demos directory of the PPL install

See Also: RENDERPARTICLES

void RENDERPARTICLES(long ID)
Renders all particles associated with the specified ID on screen.

Parameters

ID {in}
ID of the particle group you wish to cycle; if ID is 0, all particles are processed

Example:

for detailed examples on using particles, see particles.ppl and motionblur.ppl in the Demos directory of the PPL install

See Also: PROCESSPARTICLES

COUNTPARTICLES (Id) -> Particles

Count the number particles that are associated with ID.

long PARTICLES(long ID, list Particles)
Retrieve a list of particles

Parameters

ID {in}
ID of the particle group you wish to list; a value of 0 will retrieve all particles

Particles {in | out}
Variable to hold the list of particles

Return Value
PARTICLES returns the number of particles that are in the list

Example:

```
cnt$ = Particles(0, lst$);
if(cnt$ > 0)
  foreach(lst$)
    //Do something with the particles
  end;
end;
```

void SETPARTICLE(long ParticleHandle, StartX, StartY, AccelerationX, AccelerationY, MaxCycle)

Set a specified particle (ParticleHandle) properties. You can specify a value of -1 in any of the property to keep the original value.

Example:

```
Particles(1, p$);
ForEach(p$)
  SetParticle(p$, 10, 10, -1, -1, -1);      // Set only the StartX and StartY
values.
end;
```

SPRITE NEWSPRITE(addr SpriteFunc)
Create a new, empty sprite

Parameters

SpriteFunc {in}
Address of a function to handle the sprite's WM_TIMER activity; if NULL, the default game handler will recieve all of the sprite's events

Return Value
NEWSPRITE returns a SPRITE object

Notes:
- The event used for *SpriteFunc* is WM_TIMER. Only processes like moving the sprite should be performed here - no drawing allowed
- WM_LBUTTONDOWN, WM_RBUTTONDOWN, WM_MBUTTONDOWN, WM_MOUSEMOVE and WM_LBUTTONUP, WM_RBUTTONUP, WM_MBUTTONUP events will be passed automatically to *SpriteProc* if the stylus is within the sprite's area
- The wParam$ parameter is the X position of the stylus and the lParam$ is the Y position when *SpriteProc* is called

See Also: LOADSPRITE, REPLACESPRITE

SPRITE LOADSPRITE(string Filename, long TransparentColor, int FrameCount, int AnimSpeed, addr SpriteFunc)
Create a sprite based off of the image in *Filename*

Parameters

Filename {in}
Fully qualified path to a bitmap file

TransparentColor {in}
RGB value of the color within the image that will be transparent; use -1 for no transparency

FrameCount {in}
How many frames of animation are contained in the image

AnimSpeed {in}
How many milliseconds pass before switching to the sprite's next frame of animation

SpriteFunc {in}
Address of a function to handle the sprite's WM_TIMER activity; if NULL, the default game handler will recieve all of the sprite's events

Return Value
LOADSPRITE returns a SPRITE object who's image was retrieved from *Filename*

Notes:
- The event used for *SpriteFunc* is WM_TIMER. Only processes like moving the sprite should be performed here - no drawing allowed
- SETSPRITEINDEX will change the currently displayed frame of animation
- SETSPRITEANIMSPEED sets the delay in milliseconds between frames
- WM_LBUTTONDOWN, WM_RBUTTONDOWN, WM_MBUTTONDOWN, WM_MOUSEMOVE and WM_LBUTTONUP, WM_RBUTTONUP, WM_MBUTTONUP events will be passed automatically to *SpriteProc* if the stylus is within the sprite's area
- The wParam$ parameter is the X position of the stylus and the lParam$ is the Y position when *SpriteProc* is called

Example:

```
sprite$ = LoadSprite("\\My Documents\\MySprite.bmp", G_RGB(100, 100, 100), 3,
150, NULL);
```

See simple2.ppl in the Demos folder for more information about how to use the sprite functions

See Also: NEWSPRITE, REPLACESPRITE

CLONESPRITE (FromSprite) -> Sprite

Clone the content of a sprite (FromSprite). The surface is pointing to the original's sprite's surface. Make sure you don't delete the original sprite while there are still clones of it existing since the image data is contained into the original sprite only.

void REPLACESPRITE(SPRITE Actor, string Filename, long TransparentColor, int FrameCount, int AnimSpeed, addr SpriteFunc)

Use an existing SPRITE object to store a new image

Parameters

Actor {in}
Sprite you wish to assign a new image to

Filename {in}
Fully qualified path to a bitmap file

TransparentColor {in}
RGB value of the color within the image that will be transparent; use -1 for no transparency

FrameCount {in}
How many frames of animation are contained in the image

AnimSpeed {in}
How many milliseconds pass before switching to the sprite's next frame of animation

SpriteFunc {in}
Address of a function to handle the sprite's WM_TIMER activity; if NULL, the default game handler will recieve all of the sprite's events

Example:

```
sprite$ = LoadSprite("\\My Documents\\MySprite.bmp", G_RGB(100, 100, 100), 3,
150, NULL);
ReplaceSprite(sprite$, "\\My Documents\\Sprite2.bmp", RGB(100, 100, 100), 5,
100, NULL);
```

Notes:

- You should use the RGB function instead of the G_RGB function with *TransparentColor*

See Also: LOADSPRITE

void SETSPRITESURFACE(SPRITE Actor, SURFACE Canvas, int FrameCount, int AnimationSpeed)

Change the surface which a sprite is associated with

Parameters

Actor {in}
The sprite to assign to a new surface

Canvas {in}
Surface to assign the sprite to

FrameCount {in}
Number of frames of animation for the sprite

AnimationSpeed {in}
Number of milliseconds between frames; set to 0 to stop the animation

Example:

```
b$ = LoadSurface("tile.bmp", g_rgb(0,0,0));
s$ = NewSprite(null);
SetSpriteSurface(s$, b$, 1, 0);
```

Notes:

- The surface is not copied, but rather the sprite is set to point to the new surface specified

See Also: NEWSURFACE, LOADSURFACE

void DELSPRITE(SPRITE Actor)

Delete specified sprite from the list and from memory

Parameters

Actor {in}
Sprite to delete

Example:

```
s$ = SpriteData(sprite$);
if(@s$ == "dead")
  DelSprite(sprite$);
end;
```

See Also: CLEARSPRITES

void SETSPRITEAISPEED(SPRITE Actor, long AiSpeed)

Control the speed in milliseconds at which the sprite's procedure is being called. A value of -1, will call the sprite's procedure at the same interval as the global game ai speed. The default value is -1.

Parameters

Actor {in}
Sprite to retrieve information on

SPRITEAISPEED (Sprite) -> AiSpeed

Return the sprite's ai speed.

int SPRITES(LIST Spr, long ID)

Build a list with all sprites loaded in memory

Parameters

Spr {out}
Variable to hold the list of sprites

ID {in}
Set to null to list all sprites, or set to a certain value to only list sprites whose ID property matches *ID*

Return Value

SPRITES returns the count of *Spr*

Example:

```
if(Sprites(s$, null) > 0)
  foreach(s$)
    //Do something to the sprite
  end;
end;
```

SETSPRITEOPTIONS (Sprite, Options)

Set all sprite options. Options are list below.

SO_LOOP	If the sprite is auto-moving (with VelX and VelY), it might be moved past screen edges, if this

happens, it will be looped to the other side of the screen.

SO_TIMER Tell the gameapi engine to call the sprite procedure for WM_TIMER events. The interval at which the event is call is defined by SetSpriteAISpeed(). WM_TIMER events won't be called unless this option is set in the sprite.

SO_BORDER Collision detection is done on borders of the screen.

SO_WORLD A world sprite is not processed by PPL, it is only drawn to the screen. Any gravity, friction, animation, automatic movement and timer procedure is not processed by the game engine for this sprite. Use this feature for world sprites that don't need any processing to give your game more speed.

SO_COLLIDE WM_COLLIDE events for the sprite's procedure are called only if this option is set in the sprite.

SO_FIXEDX Make a sprite independant of the map ORIGINX value. The sprite will never be scrolled and always remain at a fixed X position on the screen.

SO_FIXEDY Make a sprite independant of the map ORIGINY value. The sprite will never be scrolled and always remain at a fixed Y position on the screen.

SO_FIXED Make a sprite independant of the map ORIGINX and ORIGINY values. The sprite will never be scrolled and always remain at a fixed X,Y position on the screen.

SO_PAUSED The sprite will NOT be processed each cycle.

SO_OVAL Set the sprite's virtual shape for the physic engine to apply rolling physic, just like a ball. **The SO_BOUNCE shouldn't be mixed with this option. SO_OVAL is should only be used for sprites that have a mass and elasticity properties.**

SO_BOUNCE Will bounce off other sprites but no physics will be calculated. The movement speed of the sprite will stay the same on impact. **Don't use with SO_OVAL.**

SO_PIXELCHECK This makes the sprite pixel-perfect collision detection. PPL will use pixel-perfect detection instead of boundrects.

SO_CHECKCOLLIDE Tell PPL to check for collision even though the VelX and VelY values are at zero. PPL only checks for collision if the VelX or VelY value is not zero.

SO_ACCURATECHECK When collision check is done on a sprite, normal check is done only at the position the sprite will be moved to, however there might situations where you want to collision check to be done at every pixel during the sprite's movement.

SO_MIRRORX Invert a sprite's surface horizontally.

SO_MIRRORY Invert a sprite's surface vertically.

SO_PARENTCLIP Make the sprite clip it's surface to not go past the parent's clipping region.

SO_BLUR Blurs the sprite's surface.

SO_NEGATIVE Invert the sprite's surface colors.

SO_GREYSCALE Make the sprite's surface monochrome.

SO_ISOTILE When using the TileX and/or TileY property for a sprite, this will draw the tiles in an isometric way.

SO_TRANSPARENT The sprite will be displayed using a cheap transparent technic which can prove very usefull is some situations.

SO_PROCESSONLYVIEW The sprite processing (calling the sprite's procedure) will be done only when the sprite is in view and visible.

SO_NOFRICTION No global friction is applied to the sprite.

SO_CANCELVELX When a collision is detected, PPL automatically cancels the VelX of the source sprite. If you specify this option in the target sprite, PPL will cancel the VelX of the source sprite.

SO_CANCELVELY When a collision is detected, PPL automatically cancels the VelY of the source sprite. If you specify this option in the target sprite, PPL will cancel the VelY of the source sprite.

SO_HIDEANIMDONE When the sprite's animation frames are done, the sprite is hidden.

SO_LEFT Allow for collision detection only to the left of the sprite.

SO_TOP Allow for collision detection only to the top of the sprite.

SO_RIGHT Allow for collision detection only to the right of the sprite.

SO_BOTTOM Allow for collision detection only to the bottom of the sprite.

SO_KINETIC Apply kinetic energy force to colliding sprites. This option differs from SO_BOUNCE which makes the object bounce back in the reverse direction without any physics applied to them.
SO_VCOLLISION Bouncing objects will only collide vertically with the sprite, this will improve bouncing behavior in some cases. SO_BOUNCE must be used for this option to work.
SO_HCOLLISION Bouncing objects will only collide horizontally with the sprite, this will improve bouncing behavior in some cases. SO_BOUNCE must be used for this option to work.
SO_PLATFORM Sets the collision direction detection to a platform game style. If you are designing a platform game type, you will want your hero sprite to use this collision direction detection mecanism.

SPRITEOPTIONS (Sprite) -> options

Return the sprite's options.

ADDSPRITEOPTION (Sprite, Option)

Add a sprite's option to the sprite's option list.

DELSPRITEOPTION (Sprite, Option)

Remove a sprite's option from the sprite's options list.

void CLEARSPRITES(void)

Clear all sprites from memory

See Also: DELSPRITE

void DRAWSPRITE(SPRITE Object, SURFACE TargetSurface, int X, int Y, int Index, int Alpha, int Angle)

Draw a sprite at a particular position

Parameters

Object {in}
The sprite to render

TargetSurface {in}
Surface to render *Object* to; specify NULL to render to the main screen

X {in}
Horizontal position to start rendering at

Y {in}
Vertical position to start rendering at

Index {in}
Frame of animation to render for the sprite

Alpha {in}

Value to use for alpha blending; must be between 0 and 255

Angle {in}
The angle to rotate the sprite when drawing; leave at 0 for no rotation

See Also: MOVESPRITE

void SPRITE(Var MemLoc, SPRITE Actor)

Writes the information pointed to by *MemLoc* into a sprite structure

Parameters

MemLoc {in}
Memory location where sprite information is stored

Actor {out}
Variable of type TSprite that information is written to

Sprite Structure fields:

Y
W
H
INDEX
ALPHA
ORDER
TINT
TINTLEVEL
LIGHT
LIGHTRADIUS
ANGLE
VISIBLE
FRAMECOUNT
ANIMSPEED
DATA
COLLIDE
PROC

Parameter	**Description**
X	X position
Y position	
Width of the sprite	
Height of the sprite	
Frame index of sprite	Alpha blending to use
Layer order level	
Tint color	
Level of tinting to apply	
Light level the sprite diffuse	
Radius of light	
Angle of sprite rotation	
Wheter the sprite is visible or not on screen	
Number of frames the sprite image contains	
Milliseconds between each frame	
User data	
Check for sprite collisions or not	
Sprite function address	

Example:

```
struct (Sprite$, TSprite);
list(l$);

Sprites(l$);
foreach(l$)
  Sprite(l$, Sprite$);
  if (^Sprite.data$ <> "Done!")
```

```
    Sprite.x$++;
    Sprite.y$++;
    Sprite.data$ = "Done!";
  end;
end;
```

Notes:

- This allows you to manipulate the elements of a sprite directly
- Actor must be defined before the call to SPRITE; ideally, this would occur outside of the DrawProc / SpriteProc loops
- See simple2.ppl in the Demos folder for an example of SPRITE in action

void PROCESSSPRITES(boolean Sort, boolean Light)

Updates certain sprite properties when they are modified by hand

Parameters

Sort {in}
If you modify the *order* property of a sprite, set *Sort* to true

Light {in}
If you modify the *light* or *lightradius* properties of a sprite, set *Light* to true

Example:

See the Simple2.ppl demo for an example on how to use this function

- If you modify the X or Y values of sprite by hand (ex: sprite.x$ = 10;) and you are in isometric display, call PROCESSSPRITES
- If you add or remove the options SO_TOPMOST or SO_BACKGROUND from a sprite, call PROCESSSPRITES
- If you modify the X or Y values in non-isometric display and don't change the Order value or lighting, use ADJUSTSPRITERECT instead
- When using the PPL GameAPI, sprites are ordered before being displayed and light levels of each sprite are calculated when any of them moves
- If you use the Sprite() function you need to tell PPL to process the sprites because there were some changes applied
- You mainly use this function inside of the DrawProc or SpriteProc functions
- It is highly recommended that you use the designated PPL functions for manipulating a sprite's properties

See Also: ADJUSTSPRITERECT

void MOVESPRITE(SPRITE Actor, int X, int Y)

Move a sprite to a new location

Parameters

Actor {in}
The sprite that needs to be moved

X {in}
The horizontal position of the upper left corner of the sprite

Y {in}
The vertical position of the upper left corner of the sprite

Example:

See simple.ppl in the Demos folder for an example

See Also: RELMOVESPRITE

void RELMOVESPRITE(SPRITE Actor, int X, int Y)

Move a sprite relative to the parent sprite

Parameters

Actor {in}
  The sprite that needs to be moved

X {in}
  The horizontal position of the upper left corner of the sprite

Y {in}
  The vertical position of the upper left corner of the sprite

See Also: MOVESPRITE

void ADJUSTSPRITERECT(SPRITE Actor)

Updates a sprite's internal rectangle

Parameters

Actor {in}
  Sprite that needs its internal rectangle adjusted

Example:

```
&MySprite$ = Sprite$;
MySprite.x$ = 10;
MySprite.y$ = 30;
AdjustSpriteRect(Sprite$);
```

Notes:

- You must call ADJUSTSPRITERECT if you move a sprite manually (in non-isometric display) as opposed to using the MOVESPRITE function
- If you move a sprite manually in isometric display mode, you must call PROCESSSPRITES instead
- It is highly recommended that you use the designated PPL functions for manipulating a sprite's properties

See Also: MOVESPRITE

void SPRITEPOS(SPRITE Actor, int XVar, int YVar)

Find the position of a sprite

Parameters

Actor {in}
  Sprite to find the position of

XVar {out}
  Variable to hold the X coordinate of the sprite

YVar {out}
  Variable to hold the Y coordinate of the sprite

Example:

```
local(x$, y$);

SpritePos(sprite$, x$, y$);
```

See Also: SPRITEX, SPRITEY

int SPRITEX(SPRITE Actor)

Retrieve the current X coordinate of *Actor*

Parameters

Actor {in}
  Sprite to get the X coordinate of

Return Value
SPRITEX returns an integer

Example:

```
x$ = SpriteX(sprite$);
if(x$ < (G_WIDTH - SpriteWidth(sprite$)))
  x$++;
  SetSpriteX(sprite$, x$);
end;
```

Notes:
- This can also be retrieved directly: **ShowMessage**(Actor.X$);

See Also: SETSPRITEX, SPRITEY, SETSPRITEY

void SETSPRITEX(SPRITE Actor, int X)
Set the X coordinate of *Actor*

Parameters

Actor {in}
  Sprite to set the X coordinate of

X {in}
  New value for X coordinate

Example:

See SPRITEX for an example

Notes:
- This can also be set directly: Actor.X$ = SomeValue;

See Also: SPRITEX, SPRITEY, SETSPRITEY

void SETSPRITEY(SPRITE Actor, int Y)
Set the Y coordinate of *Actor*

Parameters

Actor {in}
  Sprite to set the Y coordinate of

Y {in}
  New value for Y coordinate

Example:

See SPRITEY for an example

Notes:
- This can also be set directly: Actor.Y$ = SomeValue;

See Also: SPRITEX, SETSPRITEX, SPRITEY

int SPRITEY(SPRITE Actor)
Retrieve the current Y coordinate of *Actor*

Parameters

Actor {in}
  Sprite to get the Y coordinate of

Return Value
SPRITEY returns an integer

Example:

```
y$ = SpriteY(sprite$);
if(y$ < (G_HEIGHT - SpriteHeight(sprite$)))
  y$++;
  SetSpriteY(sprite$, y$);
end;
```

Notes:
- This can also be retrieved directly: **ShowMessage**(Actor.Y$);

See Also: SPRITEX, SETSPRITEX, SETSPRITEY

int SPRITEWIDTH(SPRITE Actor)
Retrieve the width of the given sprite

Parameters

Actor {in}
  Sprite to retrieve the width of

Return Value
SPRITEWIDTH returns the width in pixels of *Actor*

Example:

See SETSPRITEWIDTH for an example

See Also: SETSPRITEWIDTH

int SPRITEHEIGHT(SPRITE Actor)
Retrieve the height of the given sprite

Parameters

Actor {in}
  Sprite to retrieve the height of

Return Value
SPRITEHEIGHT returns the height in pixels of *Actor*

Example:

See SETSPRITEHEIGHT for an example

See Also: SETSPRITEHEIGHT

void SETSPRITEWIDTH(SPRITE Actor, int Width)
Set the width of the sprite

Parameters

Actor {in}
  Sprite whose width must be set

Width {in}
  Value in pixels to assign to the width

Example:

```
if (SpriteWidth(sprite$) < 20)
  SetSpriteWidth(sprite$, 20);
end;
```

See Also: SPRITEWIDTH

void SETSPRITEHEIGHT(SPRITE Actor, int Height)

Set the height of the sprite

Parameters

Actor {in}
  Sprite whose height must be set

Height {in}
  Value in pixels to assign to the height

Example:

```
if (SpriteHeight(sprite$) < 20)
  SetSpriteHeight(sprite$, 20);
end;
```

See Also: SPRITEHEIGHT

void SHOWSPRITE(SPRITE Actor, boolean Visible)

Toggle the visibility of a sprite

Parameters

Actor {in}
  Sprite to show or hide

Visible {in}
  True to show sprite, or false to hide it

Example:

```
sprite$ = LoadSprite("c:\\data\\sprites\\ball.bmp", G_RGB(255, 0, 255), 1, 150,
NULL);

//Elsewhere in code
visible$ = SpriteVisible(sprite$);            //Get current state of sprite
ShowSprite(sprite$, not(visible$));            //Set sprite to opposite state
```

See Also: SPRITEVISIBLE

boolean SPRITEVISIBLE(SPRITE Actor)

Retrieve visibility of a sprite

Parameters

Actor {in}
  Sprite to check visibility of

Example:

See SHOWSPRITE for an example

See Also: SHOWSPRITE

void SETSPRITEVELX(SPRITE Actor, float Speed)

Set the sprite's movement speed on the x axis

Parameters

Actor {in}
  Sprite to set the velocity on

Speed {in}
  Desired x axis velocity

Example:

```
if(SpriteVelX(sprite$) < 0.5)
  SetSpriteVelX(sprite$, 0.5);
end;
```

See bounce.ppl in the Demos directory for details on using SetSpriteVelX

See Also: SPRITEVELX, SETSPRITEVELLIMITS

float SPRITEVELX(SPRITE Actor)

Retrieve *Actor*'s x axis velocity

Parameters

Actor {in}
  Sprite to retrieve value from

Return Value

SPRITEVELX returns the sprite's movement along the x axis

Example:

See SETSPRITEVELX for an example

See Also: SETSPRITEVELX, SETSPRITEVELLIMITS

void SETSPRITEVELY(SPRITE Actor, float Speed)

Set the sprite's movement speed on the y axis

Parameters

Actor {in}
  Sprite to set the velocity on

Speed {in}
  Desired y axis velocity

Example:

```
//Stop sprite's movement along the y axis
if(SpriteVelY(sprite$) > 0)
  SetSpriteVelY(sprite$, 0);
end;
```

See bounce.ppl in the Demos directory for details on using SetSpriteVelY

See Also: SPRITEVELY, SETSPRITEVELLIMITS

float SPRITEVELY(SPRITE Actor)

Retrieve *Actor*'s y axis velocity

Parameters

Actor {in}
 Sprite to retrieve value from

Return Value

SPRITEVELY returns the sprite's movement along the y axis

Example:

See SETSPRITEVELY for an example

See Also: SETSPRITEVELY, SETSPRITEVELLIMITS

void SETSPRITEVELLIMITS (SPRITE Actor, long Minimum, long Maximum)

Set *Actor*'s minimum and maximum velocity values

Parameters

Actor {in}
 Sprite to set velocity limits for

Minimum {in}
 Slowest speed the sprite can travel; sprite will never move slower than this value

Maximum {in}
 Fastest speed the sprite can travel; sprite will never move faster than this value

Example:

```
SetSpriteVelLimits(sprite$, 1, 100);
```

Notes:

- The default minimum and maximum values are 0; in this case no limits are applied

See Also: SETSPRITEVELX, SETSPRITEVELY

void SETSPRITEANGLE(SPRITE Actor, int Angle)

Rotate the given sprite

Parameters

Actor {in}
 Sprite to rotate

Angle {in}
 Number of degrees to rotate sprite; must be between 0 and 360

Example:

```
//Assign the spirte a random angle between 0 and 360
SetSpriteAngle(sprite$, Random(360));
```

Notes:

- SETSPRITEANGLE rotates the sprite within the bounding rectangle of the sprite; in other words, if the upper left corner of the sprite is at (10,10), it will still be at (10,10) after the rotation

- Each rotation is based on the original orientation of the sprite; in other words, if you call SETSPRITEANGLE (sprite$, 10) and then call SETSPRITEANGLE(sprite$, 45), the net result is that the sprite is rotated 45 degrees from its original state, not 55 degrees

See Also: SPRITEANGLE

int SPRITEANGLE(SPRITE Actor)

Retrieve the angle of the given sprite

Parameters

Actor {in}
 Sprite to get the angle of

Return Value

SPRITEANGLE returns a value between 0 and 360

Example:

```
if(SpriteAngle(sprite$) < 180)
  SetSpriteAngle(sprite$, 180);
end;
```

See Also: SETSPRITEANGLE

void SETSPRITEALPHA(SPRITE Actor, int Alpha)

Set the sprite alpha blending

Parameters

Actor {in}
 Sprite to set alpha blending for

Alpha {in}
 Alpha value - must be between 0 and 255

Example:

```
if(SpriteAlpha(sprite$) < 100)
  SetSpriteAlpha(sprite$, 100);
end;
```

See Particles.ppl in the Demos folder for details on using SetSpriteAlpha

See Also: SPRITEALPHA

int SPRITEALPHA(SPRITE Actor)

Retrieve the sprite's alpha blending

Parameters

Actor {in}
 Sprite to retrieve alpha blending value on

Return Value

SPRITEALPHA returns a value between 0 and 255

Example:

See SETSPRITEALPHA for an example

See Also: SETSPRITEALPHA

void SETSPRITEORDER(SPRITE Actor, int Order)

Set the Z-Order of the specified sprite

Parameters

Actor {in}
  Sprite to change the order of

Order {in}
  New Z-Order position for *Actor*

Example:

```
if(SpriteOrder(s1$) < SpriteOrder(s2$))
  SetSpriteOrder(s1$, SpriteOrder(s2$) + 1);
end;
```

Notes:

- Sprites are displayed by their Z-Order, with the lower value Z-Order sprites being rendered to the screen first

See Also: SPRITEORDER

int SPRITEORDER(SPRITE Actor)

Return the Z-Order of *Actor*

Parameters

Actor {in}
  Sprite to determine the order of

Example:

See SETSPRITEORDER for an example

See Also: SETSPRITEORDER

void SETSPRITETINT(SPRITE Actor, long Color)

Make the sprite completely opaque with the color specified

Parameters

Actor {in}
  Sprite to apply tint to

Color {in}
  Color to use for tint; use -1 to remove tint all tint information

Example:

```
if(SpriteTint(sprite$) <> -1)
  SetSpriteTint(sprite$, -1);
end;
```

See Also: SPRITETINT

long SPRITETINT(SPRITE Actor)

Return the sprite tint color

Parameters

Actor {in}
  Sprite to retrieve tint information from

Return Value
SPRITETINT returns the current tint color

Example:

See SETSPRITETINT for an example

See Also: SETSPRITETINT

void SETSPRITETINTLEVEL(SPRITE Actor, int Level)
Set the sprite tinting level

Parameters

Actor {in}
Sprite to apply tint level to

Level {in}
Value to assign to the tint level

Example:

```
tint$ = SpriteTint(sprite$);
if(tint$ == -1)
  SetSpriteTint(sprite$, G_RGB(255, 0, 0));
  SetSpriteTintLevel(sprite$, 10);
else
  tintLevel$ = SpriteTintLevel(sprite$);
  SetSpriteTintLevel(sprite$, tintLevel$ + 10);
end;
```

See Also: SPRITETINTLEVEL

int SPRITETINTLEVEL(SPRITE Actor)
Return the sprite tinting level

Parameters

Actor {in}
Sprite to retrieve tint level information from

Return Value
SPRITETINTLEVEL returns the current tint level

Example:

See SETSPRITETINTLEVEL for an example

See Also: SETSPRITETINTLEVEL

void SETSPRITEINDEX(SPRITE Actor, int Index)
Set the index of the frame to display for *Actor*

Parameters

Actor {in}
Sprite whose frame of animation you wish to change

Index {in}
New frame of animation to display

Example:

```
if(SpriteIndex(sprite$) > 0)
  SetSpriteIndex(sprite$, 0);
end;
```

See Also: SPRITEINDEX

int SPRITEINDEX(SPRITE Actor)

Return the current frame of animation for *Actor*

Parameters

Actor {in}
Sprite to retrieve current frame of animation from

Return Value

SPRITEINDEX returns a value between 0 and (# of frames of animation - 1)

Example:

See SETSPRITEINDEX for an example

See Also: SETSPRITEINDEX

void SETSPRITEANIMSPEED(SPRITE Actor, long Speed)

Set the delay between each frame of a sprite

Parameters

Actor {in}
Sprite to modify the animation speed on

Speed {in}
New delay in milliseconds; set to 0 to stop animation

Example:

```
if(SpriteAnimSpeed(sprite$) == 0)
  SetSpriteAnimSpeed(sprite$, Random(10000) + 1);
end;
```

See Also: SPRITEANIMSPEED

long SPRITEANIMSPEED(SPRITE Actor)

Retrieve the delay in milliseconds between each frame of *Actor*

Parameters

Actor {in}
Sprite to determine the animation speed of

Return Value

SPRITEANIMSPEED returns the delay in milliseconds between each frame

Example:

See SETSPRITEANIMSPEED for an example

See Also: SETSPRITEANIMSPEED

void SETSPRITEFRAMES(SPRITE Actor, int FirstFrame, int LastFrame, int Count, boolean SetNow)

Set a range of frames to automatically animate

Parameters

Actor {in}
Sprite to modify

FirstFrame {in}
First image in the sequence to play

LastFrame {in}
Last image in the sequence to play; if *LastFrame* is smaller than *FirstFrame*, the animation will play backwards

Count {in}
Number of times to play the animation; set to -1 to play continuously

SetNow {in}
If true, the new range of frames will be applied immediately; otherwise, the new animation will take effect when the current one has finished

Example:

```
if(CharacterAction$ == "jump")
  SetSpriteFrames(MySprite$, 10, 15, 1, false);
end;
```

Notes:

- The animation speed is determined by SETSPRITEANIMSPEED

See Also: SPRITEFIRSTFRAME, SPRITELASTFRAME

int SPRITEFIRSTFRAME(SPRITE Actor)

Retrieve the first frame of animation for *Actor*

Parameters

Actor {in}
Sprite to retrieve information about

Return Value
SPRITEFIRSTFRAME returns the first frame displayed when the sprite is animated

See Also: SPRITELASTFRAME

int SPRITELASTFRAME(SPRITE Actor)

Retrieve the last frame of animation for *Actor*

Parameters

Actor {in}
Sprite to retrieve information about

Return Value
SPRITELASTFRAME returns the last frame displayed when the sprite is animated

See Also: SPRITEFIRSTFRAME

void SETSPRITEDIRECTION(SPRITE Actor, int Angle, float Velocity)

Set the direction and speed at which a sprite should travel

Parameters

Actor {in}

Sprite to change parameters of

Angle {in}
Direction the sprite should move - a value between 0 and 360

Velocity {in}
Speed the sprite will move each cycle; if value is -1, the sprite's original velocity will be used

Example:

```
//If sprite is not moving, start
velocity$ = SpriteVelocity(sprite$);
if(velocity$ == 0)
  SetSpriteDirection(sprite$, Random(360), 0.5);
else
  SetSpriteVelocity(sprite$, 0);
  MoveSprite(sprite$, (G_WIDTH / 2) - (SpriteWidth(sprite$) / 2), (G_HEIGHT / 2)
- (SpriteHeight(sprite$) / 2));
end;
```

See Also: SPRITEDIRECTION, SETSPRITEVELOCITY, SPRITEVELOCITY

int SPRITEDIRECTION(SPRITE Actor)
Return the current direction (angle) of *Actor*

Parameters

Actor {in}
Sprite to retrieve direction of

Return Value
SPRITEDIRECTION returns a value between 0 and 360

Example:

```
angle$ = SpriteDirection(sprite$);
```

See Also: SETSPRITEDIRECTION, SETSPRITEVELOCITY, SPRITEVELOCITY

void SETSPRITEVELOCITY(SPRITE Actor, float Velocity)
Set the sprite's velocity while keeping the original direction of the sprite

Parameters

Actor {in}
Sprite to change parameters of

Velocity {in}
Speed the sprite will move each cycle

Example:

See SETSPRITEDIRECTION for an example

See Also: SETSPRITEDIRECTION, SPRITEDIRECTION, SPRITEVELOCITY

float SPRITEVELOCITY(SPRITE Actor)
Return the speed in pixels of the sprite displacement

Parameters

Actor {in}
Sprite to retrieve the speed of

Return Value
SPRITEVELOCITY returns the speed of *Actor* as a float

Example:

See SETSPRITEDIRECTION for an example

See Also: SETSPRITEDIRECTION, SPRITEDIRECTION, SETSPRITEVELOCITY

void SETSPRITECOLOR(SPRITE Actor, long Color)
Set the sprite's pixel color if no surface is defined

Parameters

Actor {in}
Sprite to set the pixel color of

Color {in}
RGB value to set the color to

Example:

See particles.ppl in the Demos folder for details on using SetSpriteColor

See Also: SPRITECOLOR

long SPRITECOLOR(SPRITE Actor)
Retrieve the sprite's pixel color

Parameters

Actor {in}
Sprite to get pixel color of

Return Value
SPRITECOLOR returns the RGB value of the sprite's defined pixel color

See Also: SETSPRITECOLOR

void SETSPRITEMASS(SPRITE Actor, float Mass)
Sets *Sprite's* mass as a percentage compared to other sprites

Parameters

Actor {in}
Sprite to apply *Mass* to

Mass {in}
Percentage to set

Example:

```
sm$ = SpriteMass(sprite$);
if(sm$ < .25)
  SetSpriteMass(sprite$, sm$ + .05);
end;
```

Notes:
Mass is applied with *Gravity* to the sprite

See Also: SPRITEMASS, SETGRAVITY, GRAVITY

float SPRITEMASS(SPRITE Actor)

Retrieve the percentage assigned to *Actor* for mass

Parameters

Actor {in}
Sprite to retrieve the *Mass* of

Return Value
SPRITEMASS returns a float

Example:
See SETSPRITEMASS for an example

See Also: SETSPRITEMASS, SETGRAVITY, GRAVITY

void SETSPRITEFRICTION(SPRITE Actor, float Friction)
Sets the amount of friction a sprite applies during collision

Parameters

Actor {in}
Sprite to apply *Friction* to

Friction {in}
Percentage to set

Example:

```
sf$ = SpriteFriction(sprite$);
if(sf$ < .25)
  SetSpriteFriction(sprite$, sf$ + .05);
end;
```

Notes:
When two sprites collide, the source sprite will be slowed down based on the target sprite's *Friction* value

See Also: SPRITEFRICTION

float SPRITEFRICTION(SPRITE Actor)

Parameters

Actor {in}
Sprite to retrieve *Friction* of

Return Value
SPRITEFRICTION returns a float

Example:
See SETSPRITEFRICTION for an example

See Also: SETSPRITEFRICTION

void SETSPRITEELASTICITY(SPRITE Actor, float Elasticity)
Sets the sprite's elasticity as a percent

Parameters

Actor {in}
Sprite to apply *Elasticity* to

Elasticity {in}
  Percentage to set

Example:

```
se$ = SpriteElasticity(sprite$);
if(se$ < .25)
  SetSpriteElasticity(sprite$, se$ + .05);
end;
```

Notes:

- Elasticity is applied when the sprite collides with another sprite
- The bigger the elasticity, the more rebound is applied to the sprite

See Also: SPRITEELASTICITY

float SPRITEELASTICITY(SPRITE Actor)

Retrieve a sprite's elasticity

Parameters

Actor {in}
  Sprite to retrieve *Elasticity* of

Return Value

SPRITEELASTICITY returns the sprite's elasticity as a percentage

Example:

See SETSPRITEELASTICITY for an example

See Also: SETSPRITEELASTICITY

void SETSPRITEPROC(SPRITE Actor, {addr} Func)

Set the function to be called before *Actor* is rendered to screen each frame

Parameters

Actor {in}
  Sprite to associate a function with

Func {in}
  Address of function

Example:

```
func SpriteProc(Sprite$, Msg$, wParam$, lParam$)
  //do some processing to the sprite here
end;

//elsewhere in code
SetSpriteProc(sprite$, &spriteproc);
```

See simple4.ppl for more details on what SpriteProc might be used for

See Also: SPRITEPROC

{addr} SPRITEPROC(SPRITE Actor)

Retrieve *Actor*'s current function

Parameters

Actor {in}

Sprite to retrieve the function of

Return Value
SPRITEPROC returns a pointer to *Actor*'s associated function

See Also: SETSPRITEPROC

void SETSPRITEDATA(SPRITE Actor, any Data)

Associates a value with *Actor* for user purposes

Parameters

Actor {in}
Sprite whose data parameter will be changed

Data {in}
User defined value to store with *Actor*

Example:

```
SetSpriteData(sprite$, "Cool Dude!");

//elsewhere in code
data$ = SpriteData(sprite$);
ShowMessage(@data$); //Displays "Cool Dude!"
```

See Also: SPRITEDATA

pointer SPRITEDATA(SPRITE Actor)

Retrieve the stored user data from *Actor*

Parameters

Actor {in}
Sprite whose data you wish to retrieve

Return Value
SPRITEDATA returns a pointer to the data element associated with *Actor*

Example:

See SETSPRITEDATA for an example

See Also: SETSPRITEDATA

SETSPRITECOLLIDE (Sprite, Collide)

Collide is an string identifying the possible colliding group of id's. The sprite will check for collisions with other sprites that are not within the collide list. The Collide parameter is a string and it's also stored in lowercase. When you need to verify the SpriteCollide, please make sure you compare it to a **lowercase** characters string.

Example:

```
SpriteA
      Id = "monster"
      Collide: "hero"

SpriteB
      Id = "mbullet"
      Collide: "hero"

SpriteC
      Id = "hero"
      Collide: "monster"
```

```
SpriteD
      Id = "hbullet"
      Collide: "monster"

SpriteA will collide with SpriteC and SpriteD only.
SpriteB will collide with SpriteC only.
SpriteC will collide with SpriteA and SpriteB only.
SpriteD will collide with SpriteA only.
```

A WM_COLLIDE event message is called in the SpriteProc and GameProc for every sprite it collides with.

Each collision is analyzed internally by PPL and some vital information is returned to the user through global variables:

T_Collide% Structure containing miscellanious information about the collision for the target sprite.
  T_Collide.Dir% Direction the source sprite hitted the target sprite at. D_TOP, D_LEFT, D_BOTTOM, D_RIGHT
  T_Collide.Angle% Angle between the source sprite and the target sprite.
  T_Collide.X% X axis representing the position where the source sprite touched the target sprite.
  T_Collide.Y% Y axis representing the position where the source sprite touched the target sprite.

S_Collide% Structure containing miscellanious information about the collision for the source sprite.
  S_Collide.Dir% Direction the target sprite hitted the source sprite at. D_TOP, D_LEFT, D_BOTTOM, D_RIGHT
  S_Collide.Angle% Angle between the source sprite and the target sprite.
  S_Collide.X% X axis representing the position where the source sprite touched the target sprite.
  S_Collide.Y% Y axis representing the position where the source sprite touched the target sprite.

In the SpriteProc, the wParam$ parameter is the sprite handle it is colliding with.

In the GameProc, the wParam$ parameter is the sprite handle of the actual sprite and the lParam$ parameter is the sprite handle it is colliding with.

See Simple4.ppl demo for an example on how to use the WM_COLLIDE event.

If you specify **BORDER** within your collide parameter, the engine will automatically trigger collision events on collision with the borders. Check SetBorder() function for more information on how to set the borders.

In the SpriteProc, the wParam$ and lParam$ are left at zero while the Sprite$ variable is set with the sprite handle that collided with the borders.

In the GameProc, the wParam$ is set with the sprite handle that collided with the borders while lParam$ is set to zero.

string SPRITECOLLIDE(SPRITE Actor)

Returns the string containing *Actor*'s collision information

Parameters

Actor {in}
  Sprite to retrieve collision information from

Return Value

SPRITECOLLIDE returns a string

See Also: SETSPRITECOLLIDE

void SETSPRITEID(SPRITE Actor, string ID)

Assign an ID to *Actor*

Parameters

Actor {in}

Sprite to assign a new ID to

ID {in}
ID to assign

Example:

```
SetSpriteID(MySprite$, "hero");
if(SpriteID(MySprite$) == "hero")
  g_ShowMessage("Hero!");
end;
```

Notes:

- The ID is used mainly for collision detection, but is available for user defined purposes as well
- The ID is stored in **lowercase**
- To unassign an ID, use NULL for the second parameter

See Also: SPRITEID

STRING SPRITEID(SPRITE Actor)

Return the ID assigned to *Actor*

Parameters

Actor {in}
Sprite to retrieve the ID for

Return Value

SPRITEID returns a string

Example:

See SETSPRITEID for an example

See Also: SETSPRITEID

SETSPRITELAYER (Sprite, LayerId)

Set the sprite layer id. Then you can use the SetLayer() function.

SPRITELAYER (Sprite) -> Id

Return the sprite's layer id.

SETSPRITEOFFSETX (Sprite, OffsetX)

Set the sprite's offset horizontal pixel position for surface scrolling. You can't scroll horizontally and vertically at the same time.

SPRITEOFFSETX (Sprite) -> OffsetX

Return the sprite's horizontal offset pixel position.

SETSPRITEOFFSETY (Sprite, OffsetY)

Set the sprite's offset vertical pixel position for surface scrolling. You can't scroll horizontally and vertically at the same time.

SPRITEOFFSETY (Sprite) -> OffsetY

Return the sprite's vertical offset pixel position.

SETSPRITEPHYSIC (Sprite, Shape, Mass, Elasticity, Friction)

This function provides an easier way to set the sprite's physic attributes.

SETSPRITEPARENT (Sprite, Parent)

Set the sprite (Sprite) a parent sprite (Parent). Sprite's location will always be relative to the screen's location. While moving the parent sprite, the children sprites will move with it.

SPRITEPARENT (Sprite) -> Parent

Return the sprite's parent sprite.

CLEARSPRITECHILDREN (Sprite)

Detach all sprite's children sprites.

SETSPRITECLIP (Sprite, Left, Top, Right, Bottom)

Set the clipping rectangle of the sprite.

void SETSPRITERECT (Sprite, Rect)

Set the sprite rectangle. The Rect parameter is a RECT type structure.

Example:

```
struct(r$, RECT);
r.left$ = 10;
r.top$ = 10;
r.right$ = 20;
r.bottom$ = 20;
SetSpriteRect(Sprite$, r$);
```

SPRITERECT (Sprite) -> Rect

Return a pointer to a RECT structure.

Example:

```
struct(r$, RECT);
&r$ = SpriteRect(Sprite$);
```

SETSPRITECOLLIDERECT (Sprite, X, Y, X2, Y2)

Set the sprite's collision rectangle. This can prove really useful in a case where you have a tree and only the bottom part of the tree can collide with other sprites.

HANDLE SPRITESURFACE(SPRITE Actor)

Returns the surface handle of a sprite

Parameters

Actor {in}
  Sprite whose surface you wish to retrieve

Return Value

SPRITESURFACE returns a handle to the sprite's surface

Example:

```
surface$ = SpriteSurface(sprite$);
SaveSurface(surface$, "\\My Documents\\sprite.bmp");
```

void SETSPRITETILEX(SPRITE Actor, int Cols)

Duplicate *Actor Cols* number of times along the X axis

Parameters

Actor {in}
  Sprite to duplicate

Rows {in}
  Number of times to duplicate sprite; each copy of sprite will be drawn 1 pixel to the right of the previous sprite

Example:

```
SetSpriteTileX(sprite$, 5); //Draws 5 sprites on the screen
```

Notes
SETSPRITETILEX does not create additional sprites; it simply renders the image of *Actor* on the screen multiple times

See Also: SPRITETILEX

int SPRITETILEX(SPRITE Actor)

Retrieve the number of times *Actor* is tiled

Parameters

Actor {in}
Sprite to retrieve information on

Return Value
SPRITETILEX returns the number of times *Actor* is duplicated along the X axis

Example:

```
if(SpriteTileX(sprite$) > 5)
  SetSpriteTileX(sprite$, 5);
end;
```

See Also: SETSPRITETILEX

void SETSPRITETILEY(SPRITE Actor, int Rows)

Duplicate *Actor Rows* number of times along the Y axis

Parameters

Actor {in}
Sprite to duplicate

Rows {in}
Number of times to duplicate sprite; each copy of sprite will be placed 1 pixel below the bottom of the previous sprite

Example:

```
SetSpriteTileY(sprite$, 5); //Draws 5 sprites on the screen
```

Notes
SETSPRITETILEY does not create additional sprites; it simply renders the image of *Actor* on the screen multiple times

See Also: SPRITETILEY

int SPRITETILEY(SPRITE Actor)

Retrieve the number of times *Actor* is tiled

Parameters

Actor {in}
Sprite to retrieve information on

Return Value
SPRITETILEY returns the number of times *Actor* is duplicated along the Y axis

Example:

```
if(SpriteTileY(sprite$) > 5)
  SetSpriteTileY(sprite$, 5);
end;
```

See Also: SETSPRITETILEY

SETSPRITEAUTOSCROLLX (Sprite, ScrollPercent)

Set the percentage of automatic scrolling for the sprite. The automatic scrolling is controlled by the sprite engine and is applied when the Origin is changing.

Example:

```
// Scroll the sprite only 2% of normal Origin's scrolling.
SetSpriteAutoScrollX(Sprite$, 0.02);
```

float SPRITEAUTOSCROLLX(SPRITE Actor)

Retrieve the percentage of autoscrolling in the X direction for *Actor*

Parameters

Actor {in}
Sprite to retrieve information on

Return Value

SPRITEAUTOSCROLLX returns a float indicating the percent by which *Actor* will scroll

Return the percentage assigned to automatic scrolling.

SETSPRITEAUTOSCROLLY (Sprite, ScrollPercent)

Set the percentage of automatic scrolling for the sprite. The automatic scrolling is controlled by the sprite engine and is applied when the Origin is changing.

Example:

```
// Scroll the sprite only 2% of normal Origin's scrolling.
SetSpriteAutoScrollY(Sprite$, 0.02);
```

SPRITEAUTOSCROLLY (Sprite) -> ScrollY

Return the percentage assigned to automatic scrolling.

SETSPRITEAUTOOFFSETX (Sprite, OffsetX)

Just like automatic scrolling, automatic offset will change the sprite's offset based on the Origin values. The value is only applied by the percentage speficied.

Example:

```
// Offset the sprite by only 2% of normal Origin's scrolling.
SetSpriteAutoOffsetX (Sprite$, 0.02);
```

SPRITEAUTOOFFSETX (Sprite) -> OffsetX

Return the percentage assigned to automatic offset.

SETSPRITEAUTOOFFSETY (Sprite, OffsetY)

Just like automatic scrolling, automatic offset will change the sprite's offset based on the Origin values. The value is only applied by the percentage speficied.

Example:

```
// Offset the sprite by only 2% of normal Origin's scrolling.
SetSpriteAutoOffsetY (Sprite$, 0.02);
```

SPRITEAUTOOFFSETY (Sprite) -> OffsetY

Return the percentage assigned to automatic offset.

SETSPRITEALTALPHA (Sprite, Alpha)

Set the alternate alpha blending. The alternate mode is triggered (only in isometric display mode) when a sprite with an alternate radius is displayed behing the sprite (Sprite) and is within the radius range. The sprite (Sprite) alpha blending is automatically changed by the game api engine.

SPRITEALTALPHA (Sprite) -> Alpha

Return the alternate alpha blending value of a sprite.

SETSPRITEALTINDEX (Sprite, Index)

Set the alternate image index of a sprite. The alternate mode is triggered (only in isometric display mode) when a sprite with an alternate radius is displayed behing the sprite (Sprite) and is within the radius range. The sprite (Sprite) image index is automatically changed by the game api engine.

SPRITEALTINDEX (Sprite) -> Index

Return the alternate image index of a sprite.

SETSPRITEALTRADIUS (Sprite, Radius)

Set the radius range (in pixels) of a sprite. When this sprite is moved behind a sprite with an alternate alpha blending or an alternate image index property set, the target sprite will be changed. This is only available in isometric display mode.

SPRITEALTRADIUS (Sprite) -> Radius

Return the alternate radius of a sprite.

COLLIDE (Sprite, X, Y, CollideX, CollideY) -> collision

This function checks to see if sprite (Sprite) is hitting anything at position (X, Y). The collision information is returned in S_Collide and S_Collide variables (see SetSpriteCollide). The function returns the sprite it collided with if there was a collision detected or NULL if none.

Example:

```
If (Collide (Player$, SpriteX(Player$), SpriteY(Player$)+4, cx$, cy$) <> NULL)
  CanJump$ = true;
end;
```

SPRITE SPRITEAT(int X, int Y, boolean OnScreen)

Returns a sprite (if available) touching position (*X*, *Y*)

Parameters

X {in}
   horizontal position to check

Y {in}
   vertical position to check

OnScreen {in}
   If *OnScreen* is true, SPRITEAT will only consider sprites that are on the screen, and return the sprite with the highest Z-Order; if *OnScreen* is false, all visible sprites will be considered and no Z-Order analysis will be done

Return Value

SPRITEAT returns the sprite that best meets the supplied criteria

Example:

```
sprite$ = SpriteAt(10, 10, true);
if(sprite$ <> null)
  ShowMessage("I found a sprite!");
end;
```

See Also: SPRITESAT

int SPRITESAT(int X, int Y, boolean OnScreen, LIST Spr)

Creates a list of all sprites touching point (*X*, *Y*)

Parameters

X {in}
horizontal position to check

Y {in}
vertical position to check

OnScreen {in}
If *OnScreen* is true, SPRITESAT will only consider sprites that are on the screen; if *OnScreen* is false, all visible sprites will be considered

Spr {out}
Variable that will contain the list of sprites

Return Value

SPRITESAT returns the number of elements contained in *Spr*

Example:

```
if(SpritesAt(10, 10, true, &lst$) > 0)
  foreach(lst$)
    //Do something with sprites
  end;
end;
```

See Also: SPRITEAT

SPRITE SPRITEATRECT(int Left, int Top, int Right, int Bottom, boolean OnScreen)

Find a sprite (if available) touching the rectangle specified by (*Left*, *Top*) to (*Right*, *Bottom*)

Parameters

Left {in}
Upper horizontal position of rectangle

Top {in}
Upper vertical position of rectangle

Right {in}
Lower horizontal position of rectangle

Bottom {in}
Lower vertical position of rectangle

OnScreen {in}
If *OnScreen* is true, SPRITEATRECT will only consider sprites that are on the screen, and return the sprite with the highest Z-Order; if *OnScreen* is false, all visible sprites will be considered and no Z-Order analysis will be done

Return Value

SPRITEATRECT returns the sprite that best meets the supplied criteria

Example:

```
sprite$ = SpriteAtRect(10, 10, 100, 100, true);
if(sprite$ <> null)
  ShowMessage("I found a sprite!");
end;
```

See Also: SPRITESATRECT

int SPRITESATRECT(int Left, int Top, int Right, int Bottom, boolean OnScreen, LIST Spr)

Find any sprite (if available) touching the rectangle specified by (*Left*, *Top*) to (*Right*, *Bottom*)

Parameters

Left {in}
Upper horizontal position of rectangle

Top {in}
Upper vertical position of rectangle

Right {in}
Lower horizontal position of rectangle

Bottom {in}
Lower vertical position of rectangle

OnScreen {in}
If *OnScreen* is true, SPRITESATRECT will only consider sprites that are on the screen; if *OnScreen* is false, all visible sprites will be considered

Spr {out}
Variable that will contain the list of sprites

Return Value

SPRITESATRECT returns the number of elements contained in *Spr*

Example:

```
if(SpritesAtRect(10, 10, 100, 100, true, &lst$) > 0)
  foreach(lst$)
    //Do something with sprites
  end;
end;
```

See Also: SPRITEATRECT

boolean SPRITEINVIEW(SPRITE Actor)

Determines whether *Actor* is within the screen visible area (that the user can see)

Parameters

Actor {in}
Sprite to evaluate

Return Value

SPRITEINVIEW returns true if the sprite is in the screen visible area, or false otherwise

Example:

```
if(SpriteInView(sprite$))
  ShowMessage("Now you see me");
else
  ShowMessage("Now you don't");
end;
```

OFFSETSPRITE (Sprite, X, Y)

Move sprite Sprite by X and Y pixels.

CALCPIXELCHECK (Sprite)

Recalculate pixel check masks for pixel perfect collision detection accuracy. Any manual change to the sprite's size, angle ... need to be followed by a DoPixelCheck() function.

Example:

```
struct(s$, TSPRITE);
&s$ = MySprite$;
s.angle$ = 100;
DoPixelCheck(s$);
```

long DELSPRITES (long SpriteID)

Delete all sprites with ID *SpriteID*

Parameters

SpriteID {in}
  ID of the sprites you wish to delete; set to **null** to delete all sprites

Return Value

DELSPRITES returns the number of sprites deleted

See Also: DELSPRITE, CLEARSPRITES

void PAUSE(boolean Paused)

Freeze any sprites that have been created, or unfreeze any sprites that are currently frozen

Parameters

Paused {in}
  True to freeze sprites, false to unfreeze them

Example:

```
Global(IsPaused$);

func PauseGame()
  if(IsPaused$)
    Pause(false);
    IsPaused$ = false;
  else
    Pause(true);
    IsPaused$ = true;
  end;
end;
```

Notes:

- "Freezing" a sprite entails: no animation, no timer processing, no collision detection and no SpriteAt functionality
- Any sprites created after calling PAUSE with true will not be frozen
- This function is useful when you need to bring up a screen to ask for user input

SETSPRITELIGHT (Sprite, Light)

Set the light intensity of a sprite. Use SetSpriteLightRadius for the radius of the light. The values ranges from 0 to 256.

SETSPRITELIGHTRADIUS (Sprite, Radius)

Set the sprite light radius.

SETSPRITETIMER (Sprite, Id, Interval, UserValue)

Set a timer specific to a sprite that will trigger at (interval) milliseconds. The event is triggered using the sprite's procedure with the WM_USERTIMER event. Each timer is identified by an ID. Each timer can have a user value assigned to them. The ID is passed to the wParam$ of the sprite's proc and the UserValue is passed to the lParam$.

KILLSPRITETIMER (Sprite, Id)

Delete a timer (id) from a sprite.

PAUSESPRITETIMER (Sprite, Id, Pause)

Pause a sprite's timer (id) or unpause it.

SPRITETIMER (Sprite, Id) -> Paused

Return wheter the sprite timer (id) is paused or not.

S_INIT (long Frequency, int BitsPerSeconds, boolean Stereo, int ModChannels, int WaveChannels)

Initializes the sound system

Parameters

Frequency {in}
Output rate in hz

BitsPerSecond {in}
Value can either be 8 or 16

Stereo {in}
Whether or not output should be in stereo

ModChannels {in}
Maximum number of channels for playing .mod files

WaveChannels {in}
Maximum number of channels for playing .wav files

Example:

```
S_INIT(44100, 16, True, 2, 8);
```

See Also: S_SHUT

void S_SHUT(void)

Shuts down the sound system and unloads all loaded .mod and .wav files

See Also: S_INIT

long LOADSOUND(string Filename, boolean Module)

Loads a .wav or .mod file into memory

Parameters

Filename {in}
Name of the .wav or .mod file to load

Module {in}
If true, LOADSOUND expects a .mod file

Return Value

LOADSOUND returns the channel that the sound was loaded to

Example:

```
// Load sound file from disk.
w$ = LoadSound(AppPath$ + "drum.wav", false);
m$ = LoadSound(AppPath$ + "ars.mod", true);

//Elsewhere in code
PlaySound(m$);
```

See Also: PLAYSOUND

void PLAYSOUND(long Sound)

Plays a previously loaded .mod or .wav file

Parameters

Sound {in}
ID of .mod or .wav file to play

Example:

See LOADSOUND for an example

See Also: LOADSOUND, STOPSOUND

void STOPSOUND(long ChannelID)

Stops the .mod or .wav associated with *ChannelID* from playing

Parameters

ChannelID {in}
Any valid channel created through the **LOADSOUND** function

Example:

See SOUNDSTATE for an example

See Also: PLAYSOUND, PAUSESOUND, RESUMESOUND

void PAUSESOUND(long ChannelID)

Pauses the .mod or .wav associated with *ChannelID*

Parameters

ChannelID {in}
Any valid channel created through the **LOADSOUND** function

Example:

See SOUNDSTATE for an example

See Also: STOPSOUND, RESUMESOUND

void RESUMESOUND(long ChannelID)

Continue a .mod or .wav file previously paused with **PAUSESOUND**

Parameters

ChannelID {in}
Any valid channel created through the **LOADSOUND** function

Example:

```
if(SoundState(channel$) == 2)
  ResumeSound(channel$);
end;
```

See Also: STOPSOUND, PAUSESOUND

int SOUNDSTATE(long ChannelID)

Returns the state of the specified channel as defined upon a successful call to **LOADSOUND**

Parameters

ChannelID {in}
Any valid channel created through the **LOADSOUND** function

Return Value

SOUNDSTATE returns an integer with the following possible values:
  0 = Stopped
  1 = Playing
  2 = Paused

Example:

```
s$ = SoundState(channel$);
Case(s$)
  0:
    PlaySound(channel$);
  1:
    PauseSound(channel$);
  2:
    StopSound(channel$);
end;
```

See Also: LOADSOUND, PLAYSOUND, STOPSOUND, PAUSESOUND

int VOLUME(long ChannelID)

Retrieve volume of specified .mod or .wav file

Parameters

ChannelID {in}
  Any valid channel created through the **LOADSOUND** function

Return Value

VOLUME returns the volume of *ChannelID*

Example:

```
v$ = Volume(channel$);
if(v$ < 64)
  SetVolume(channel$, v$ + 2);
end;
```

See Also: SETVOLUME

void SETVOLUME(long ChannelID, int Volume)

Adjust the playback volume of a .mod or .wav file

Parameters

ChannelID {in}
  Any valid channel created through the **LOADSOUND** function

Volume {in}
  Desired volume for the given channel; the range is 0 (no sound) to 64 (maximum volume)

Example:

See VOLUME for an example

See Also: VOLUME

long FREQUENCY(long ChannelID)

Return the frequency of the specified .mod or .wav object

Parameters

ChannelID {in}
  Any valid channel created through the **LOADSOUND** function

Return Value

FREQUENCY returns the frequency of *ChannelID*

Example:

```
if(Frequency(channel$) == 44100)
  SetFrequency(channel$, 22050);
end;
```

See Also: SETFREQUENCY

void SETFREQUENCY(long ChannelID, long Frequency)

Set the playback frequency for *ChannelID*

Parameters

ChannelID {in}
  Any valid channel created through the **LOADSOUND** function

Frequency {in}
  New frequency value

Example:

See FREQUENCY for an example

See Also: FREQUENCY

int PAN(long ChannelID)

Retrieve the pan value of the .wav or .mod file

Parameters

ChannelID {in}
  Any valid channel created through the **LOADSOUND** function

Return Value

PAN returns the pan value of *ChannelID*

Example:

```
if(Pan(channel$) <> 128)
  SetPan(channel$, 128);
end;
```

See Also: SETPAN

void SETPAN(long ChannelID, int Pan)

Set the pan value for *ChannelID*

Parameters

ChannelID {in}
  Any valid channel created through the **LOADSOUND** function

Pan {in}
  Value for panning; range is 0 (completely left speaker) to 255 (completely right speaker); use 128 for no panning

Example:

See PAN for an example

See Also: PAN

boolean LOOP(long ChannelID)

Determine whether the specified channel is looping

Parameters

ChannelID {in}
Any valid channel created through the **LOADSOUND** function

Return Value

LOOP returns true if the channel is set to loop, or false otherwise

Example:

```
if(not(Loop(channel$)))
  SetLoop(channel$, true);
end;
```

See Also: SETLOOP

void SETLOOP(long ChannelID, boolean Loop)

Specify whether *ChannelID* should loop or not

Parameters

ChannelID {in}
Any valid channel created through the **LOADSOUND** function

Loop {in}
True to loop the sample in *ChannelID*, or false to only play it once

Example:

See LOOP for an example

See Also: LOOP

HANDLE CREATECOMOBJECT (string ID)

Create an instance of a COM object

Parameters

ID {in}
CLSID or PROGID of COM object to create

Return Value

CREATECOMOBJECT returns a handle to the COM object if successful, or a 0 otherwise

Example:

For a comprehensive example of the COM functions, see the com.ppl example in the Demos folder of the install

```
obj$ = CreateCOMObject("ADOCE.Connection.3.1");
if(obj$ <> 0)
  //Do stuff with object here
  FreeCOMObject(obj$);
else
  ShowMessage(COMERROR%);
end;
```

Notes:
If CREATECOMOBJECT returns 0, the COMERROR% global variable will contain a description of the error

See Also: FREECOMOBJECT

void FREECOMOBJECT([HANDLE Object...])

Free one or more COM objects from memory

Parameters

Object {in}
Handle to one or more COM objects

Example:

See CREATECOMOJBECT for an example

Notes:

- If FREECOMOBJECT fails, a description of the error will be stored in COMERROR%

See Also: CREATECOMOJBECT

QUERYINTERFACE (comhandle, clsid) -> comhandle

Query an interface using it's CLSID or PROGID (clsid) from a COM object (comhandle).

any INVOKE(HANDLE Object, string MethodName, [any Value...])

Call a method of a COM object

Parameters

Object {in}
Handle retrieved using CREATECOMOBJECT

MethodName {in}
Name of the function to call

Value {in}
One or more parameters to pass to *MethodName*

Return Value

INVOKE returns false if the function call did not succeed; if the function call was successful and *MethodName* returns no value, INVOKE returns true; if *MethodName* returns a value, INVOKE will return that value

Example:

For a comprehensive example of the COM functions, see the com.ppl example in the Demos folder of the install

```
//create a recordset object and open a table from ado.cdb
#ifdef _WIN32_WCE
  rec$ = CreateCOMObject("ADOCE.Recordset.3.1");
#else
  rec$ = CreateCOMObject("ADODB.Recordset");
#endif

Invoke(rec$, "Open", "MyTable", "\\ado.cdb", adOpenStatic, VT_NULL, adCmdTable);
```

GETPROPERTY (comhandle, propertyname) -> value

Return the value of a COM object (comhandle) property.

SETPROPERTY (comhandle, propertyname, value)

Set the property of a COM object (comhandle).

COMPROPERTIES (ComHandle, Var) -> Count

Return all the properties for a COM object (ComHandle) in a list (Var).

COMMETHODS (ComHandle, Var)

Return all the methods of a COM object (ComHandle) in a list (Var).

COMINFO (ComHandle, Name, DataInfo, ParamType)

Return information of a property or a method (Name) into structure (DataInfo) and a list of parameter types (ParamType). Make sure you free the DataInfo variable after you have used it.

Example:

```
  a$ = ActiveX(h$, "COMCTL.Slider.1", 10, 10, 400, 40, NULL);
  e$ = ActiveXEvents(a$);

  struct(data$, COMINFOSTRUCT);
  ComInfo(e$, "mousemove", data$, paramtypes$);
  ShowMessage(" ID:" + data.DispId$ + " Flag:" + data.Flag$ + " OutType:" +
data.OutType$ + " ParamCount:" + data.ParamCount$ + " [" + ListToStr
(paramtypes$, ",", "", "") + "]");
  free(data$);
```

T (value, type)

Convert a value to a VARIANT type. Use this function with COM object's functions. Possible types are:

```
          VT_EMPTY      = 0,
      VT_NULL      = 1,
      VT_I2            = 2,
      VT_I4            = 3,
      VT_R4            = 4,
      VT_R8            = 5,
      VT_CY            = 6,
      VT_DATE      = 7,
      VT_BSTR      = 8,
      VT_DISPATCH      = 9,
      VT_ERROR      = 10,
      VT_BOOL      = 11,
      VT_VARIANT      = 12,
      VT_UNKNOWN      = 13,
      VT_DECIMAL      = 14,
      VT_I1            = 16,
      VT_UI1      = 17,
      VT_UI2      = 18,
      VT_UI4      = 19,
      VT_I8            = 20,
      VT_UI8      = 21,
      VT_INT      = 22,
      VT_UINT      = 23,
      VT_VOID      = 24,
      VT_HRESULT      = 25,
      VT_PTR      = 26,
      VT_SAFEARRAY      = 27,
      VT_CARRAY      = 28,
      VT_USERDEFINED      = 29,
      VT_LPSTR      = 30,
      VT_LPWSTR      = 31,
      VT_RECORD      = 36,
      VT_FILETIME      = 64,
      VT_BLOB      = 65,
      VT_STREAM      = 66,
      VT_STORAGE      = 67,
      VT_STREAMED_OBJECT      = 68,
      VT_STORED_OBJECT      = 69,
      VT_BLOB_OBJECT      = 70,
      VT_CF            = 71,
      VT_CLSID      = 72,
      VT_VERSIONED_STREAM      = 73,
      VT_BSTR_BLOB      = 0xfff,
      VT_VECTOR      = 0x1000,
      VT_ARRAY      = 0x2000,
      VT_BYREF      = 0x4000,
      VT_RESERVED      = 0x8000,
      VT_ILLEGAL      = 0xffff,
      VT_ILLEGALMASKED      = 0xfff,
      VT_TYPEMASK      = 0xfff
```

COMOBJECTS (Var, Control) -> Count

Return a list of all the COM/ActiveX objects registered in the system into variable (Var). Each list element is a string with the ProgId and ServerLocation seperated with a comma. You can get either only the visible controls (Control = 1) or only COM objects (Control = 0) or both of them (Control = -1).

Example:

```
  COMObjects(objects$, 1);
  ShowMessage("\ActiveX Objects:\n\n" + ListToStr(objects$, #13#10, "", "")
+"\n");
  COMObjects(objects$, 0);
  ShowMessage("\nCOM Objects:\n\n" + ListToStr(objects$, #13#10, "", "")+"\n");
```

ACTIVEX (hWnd, Name, X, Y, Width, Height, Proc) -> ActiveXHandle

Create an ActiveX visual component (Name) on window (hWnd). The return value is the activex handle. To assign properties, get properties or invoke methods you will need to use the ComHandle returned by ActiveXObject().

Example:

```
// ActiveX controls
#include "windows.ppl"
#include "console.ppl"
#include "ole.ppl"

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

  case (Msg$)
    WM_CLOSE:
      FreeActiveX(a$, b$, c$);
  end;

  return (ok$);

end;

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

  write("Event #" + Msg$ + " hWnd:" + hWnd$ + " ActiveX Handle:" + Control$ + "
");

  case (Msg$)

    WM_MOUSEMOVE:
      Writeln("WM_MOUSEMOVE (" + LoWord(lParam$) + "," + HiWord(lParam$) + "," +
wParam$ + ")");
    WM_LBUTTONDOWN:
      Writeln("WM_LBUTTONDOWN (" + LoWord(lParam$) + "," + HiWord(lParam$) + ","
+ wParam$ + ")");
    WM_LBUTTONUP:
      Writeln("WM_LBUTTONUP (" + LoWord(lParam$) + "," + HiWord(lParam$) + "," +
wParam$ + ")");
    WM_RBUTTONDOWN:
      Writeln("WM_RBUTTONDOWN (" + LoWord(lParam$) + "," + HiWord(lParam$) + ","
+ wParam$ + ")");
    WM_RBUTTONUP:
      Writeln("WM_RBUTTONUP (" + LoWord(lParam$) + "," + HiWord(lParam$) + "," +
wParam$ + ")");
    WM_MBUTTONDOWN:
      Writeln("WM_MBUTTONDOWN (" + LoWord(lParam$) + "," + HiWord(lParam$) + ","
+ wParam$ + ")");
    WM_MBUTTONUP:
      Writeln("WM_MBUTTONUP (" + LoWord(lParam$) + "," + HiWord(lParam$) + "," +
wParam$ + ")");
    WM_KEYDOWN:
      Writeln("WM_KEYDOWN (" + wParam$ + "," + lParam$ + ")");
    WM_KEYUP:
      Writeln("WM_KEYUP (" + wParam$ + "," + lParam$ + ")");
    WM_KEYPRESS:
      dim(p$, lParam$);
      &p$ = wParam$;
      Writeln("WM_KEYPRESS (" + chr(p$[0]) + ")");
    WM_CLICK:
      Writeln("WM_CLICK");
    WM_DBLCLICK:
      Writeln("WM_DBLCLICK");

    default:
      if (lParam$ > 0)
        Write("(");
        dim(p$, lParam$);
        &p$ = wParam$;
        for (i$, 0, lParam$ - 1)
          Write(p$[i$]);
          if (i$ < lParam$ - 1)
            Write(",");
          end;
        end;
        Write(")");
      end;
      Writeln("");
  end;

  return (ok$);

end;

func winmain

  h$ = newform({OLE}, {OLEClass}, &wndproc);

  InitConsole;
  ConsoleUpdate$ = False;
  ShowConsole;

  COMObjects(objects$, 1);
  Writeln("\ActiveX Objects:\n\n" + ListToStr(objects$, #13#10, "", "")+"\n");
  COMObjects(objects$, 0);
  Writeln("\nCOM Objects:\n\n" + ListToStr(objects$, #13#10, "", "")+"\n");

  global(a$, b$, c$);
  a$ = ActiveX(h$, "COMCTL.Slider.1", 10, 10, 400, 40, &ActiveXProc);
  if (a$)
    o$ = ActiveXObject(a$);
    SetProperty(o$, "Max", 100);
    SetProperty(o$, "Value", 50);

    // Get list of properties
    ComProperties(o$, l$);
    Writeln("\nProperties:\n\n" + ListToStr(l$, #13#10, "", ""));

    // Get list of methods
    ComMethods(o$, l$);
    Writeln("\nMethods:\n");
    foreach (l$)
      struct(data$, COMINFOSTRUCT);
      ComInfo(o$, l$, data$, paramtypes$);
      Writeln(l$ + " ID:" + data.DispId$ + " Flag:" + data.Flag$ + " OutType:" +
data.OutType$ + " ParamCount:" + data.ParamCount$ + " [" + ListToStr
(paramtypes$, ",", "", "") + "]");
      free(data$);
    end;

    // Get list of event methods
    e$ = ActiveXEvents(a$);
    ComMethods(e$, l$);
    Writeln("\nEvents:\n");
    foreach (l$)
      struct(data$, COMINFOSTRUCT);
      ComInfo(e$, l$, data$, paramtypes$);
      Writeln(l$ + " ID:" + data.DispId$ + " Flag:" + data.Flag$ + " OutType:" +
data.OutType$ + " ParamCount:" + data.ParamCount$ + " [" + ListToStr
(paramtypes$, ",", "", "") + "]");
      free(data$);
    end;
  end;

  b$ = ActiveX(h$, "COMCTL.Slider.1", 10, 100, 150, 120, &ActiveXProc);

  c$ = ActiveX(h$, "COMCTL.ProgCtrl.1", 10, 200, 150, 220, &ActiveXProc);
  if (c$)
    o$ = ActiveXObject(c$);
    SetProperty(o$, "Max", 100);
    SetProperty(o$, "Value", 50);
  end;

  ShowWindow(h$, SW_SHOW);
  SetForeGroundWindow(h$);

  return (true);
end;
```

FREEACTIVEX (ActiveXHandle, [...])

Free an ActiveX object from memory.

SETACTIVEXPOS (ActiveXHandle, X, Y, Width, Height)

Move and resize an ActiveX object on the screen.

ACTIVEXOBJECT (ActiveXHandle) -> ComHandle

Return the ActiveX's COM handle.

ACTIVEXEVENTS (ActiveXHandle) -> ComHandle

Return the ActiveX's COM handle for events.

ACTIVEXWND (ActiveXHandle) -> hWnd

Return the ActiveX's object window handle.

MATCH (expression, string) -> match

Returns True if the string specified in the parameter string is an exact match of the expression, otherwise returns False.

Any error that occurs within the regular expressions functions is stored in the REXERROR% global variable.

PPL implements the following expressions:

\	Quote the next metacharacter
^	Match the beginning of the string
.	Match any character
$	Match the end of the string
|	Alternation
()	Grouping (creates a capture)
[]	Character class

==GREEDY CLOSURES==

*	Match 0 or more times
+	Match 1 or more times
?	Match 1 or 0 times
{n}	Match exactly n times
{n,}	Match at least n times
{n,m}	Match at least n but not more than m times

==ESCAPE CHARACTERS==

\t	tab	(HT, TAB)
\n	newline	(LF, NL)
\r	return	(CR)
\f	form feed	(FF)

==PREDEFINED CLASSES==

\l	lowercase next char
\u	uppercase next char
\a	letters
\A	non letters
\w	alphanimeric [0-9a-zA-Z]
\W	non alphanimeric
\s	space
\S	non space
\d	digits
\D	non nondigits
\x	exadecimal digits
\X	non exadecimal digits
\c	control charactrs
\C	non control charactrs
\p	punctation
\P	non punctation

SEARCH (expression, string, out_begin, out_end)

Searches the first match of the expressin in the string specified in the parameter string. If the match is found returns True and then sets out_begin to the beginning of the match and out_end at the end of the match; otherwise returns False.

SUBEXPCOUNT -> count

Returns the number of sub expressions matched by the last expression.

SUBEXP (string, index, out_begin, out_len)

Retrieves the beginning and the length of the sub expression indexed by index.

HANDLE OPENPACKAGE(string Filename, string Key)

Open a package

Parameters

Filename {in}
Name of package to open; if file doesn't exist, the handle will still be valid and the package will be created

Key {in}
String used to encrypt the package

Return Value
OPENPACKAGE returns a handle to the package

Example:

```
p$ = OpenPackage("MyPackage.pkg", "MyKey");
ClosePackage(p$);
```

See Also: CLOSEPACKAGE

void CLOSEPACKAGE(HANDLE Package)
Close a package and write it's contents to file if it was modified

Parameters

Package {in}
Handle of package to close; the handle is retrieved with a call to OPENPACKAGE

Example:

See OPENPACKAGE for an example

See Also: OPENPACKAGE

void ADDFILETOPACKAGE (HANDLE Package, string FileName)
Add a new file or replace an existing one with the contents of *FileName*

Parameters

Package {in}
Handle returned from a call to OPENPACKAGE

FileName {in}
Path and name of file to add to package

Example:

```
p$ = OpenPackage("MyPackage.pkg", "MyKey");
if (not IsNull(p$))
  AddFileToPackage(p$, "MyFile.txt");
  ClosePackage(p$);
end;
```

Notes:

- ADDFILETOPACKAGE uses the file's name as the name within the package file
- If you need a different name inside the package than the file name, use ADDFILETOPACKAGEEX instead
- If a file already exists in the package with this file's name, the file inside of the package will be replaced

See Also: ADDFILETOPACKAGEEX, DELETEFILEFROMPACKAGE

void DELETEFILEFROMPACKAGE(HANDLE Package, string Name)
Delete a file from the package

Parameters

Package {in}
Handle returned from a call to OPENPACKAGE

Name {in}
Name of the original file without the path

Example:

```
p$ = OpenPackage("MyPackage.pkg", "MyKey");
if (not IsNull(p$))
  AddFileToPackage(p$, "\\My Documents\\MyFile.txt");
  DeleteFileFromPackage(p$, "MyFile.txt");
  ClosePackage(p$);
end;
```

See Also: ADDFILETOPACKAGE

ptr LOADPACKAGEFILE (HANDLE Package, string FileName, long Size)

Loads a file from a package into memory

Parameters

Package {in}
Handle returned from a call to OPENPACKAGE

FileName {in}
Name of the file to retrieve from the package

Size {out}
Number of bytes returned by LOADPACKAGEFILE

Return Value

LOADPACKAGEFILE returns a pointer to the memory location where *FileName* was loaded

Example:

```
p$ = OpenPackage("MyPackage.pkg", "MyKey");
data$ = LoadPackageFile(p$, "intro.wav", &sz$);
intro$ = LoadSoundFromMem(data$, sz$, false);
ClosePackage(p$);
```

See Also: EXTRACTFILEFROMPACKAGE

string EXTRACTFILEFROMPACKAGE(HANDLE Package, string PackageName)

Retrieve a file from a package and write it to a temporary location

Parameters

Package {in}
Handle returned from a call to OPENPACKAGE

PackageName {in}
Name of file to retrieve from package

Return Value

EXTRACTFILEFROMPACKAGE returns the name of the temporary file used to store the contents of *PackageName*

Example:

```
p$ = OpenPackage("MyPackage.pkg", "MyKey");
if (not IsNull(p$))
  AddFileToPackage(p$, "\\My Documents\\MyFile.txt");
  fn$ = ExtractFileFromPackage(p$, "MyFile.txt");
  s$ = LoadStr(fn$, sz$);
  DeleteFile(fn$);
  ShowMessage(s$);
  ClosePackage(p$);
end;
```

Notes:

- The extracted file is not encrypted in any way, unless you encrypted it before adding it to the package
- After using the file, you should always delete it to conserve space
- If you want to specify the path and file name of the extracted file yourself, use EXTRACTFILEFROMPACKAGEEX instead

See Also: ADDFILETOPACKAGE, ADDFILETOPACKAGEEX, EXTRACTFILEFROMPACKAGEEX

boolean PACKAGEFILEEXISTS(HANDLE Package, string FileName)

Determine if *FileName* exists in *Package*

Parameters

Package {in}
  Handle retrieved from a call to OPENPACKAGE

FileName {in}
  Name of file to search for

Return Value

PACKAGEFILEEXISTS returns true if *FileName* is found, or false otherwise

Example:

```
p$ = OpenPackage("MyPackage.pkg", "MyKey");
if (not IsNull(p$))
  if (PackageFileExists("MyFile.txt"))
    ShowMessage("Exists!");
  end;
  ClosePackage(p$);
end;
```

Notes:

- The name parameter must be the name part of the filename without the original path

See Also: PACKAGEFILES

void PACKAGEFILES(list Files, HANDLE Package)

Returns a list of file names contained in *Package*

Parameters

Files {out}
  List containing the names of all of the files found in *Package*

Package {in}
  Handle retrieved from a call to OPENPACKAGE

Example:

```
p$ = OpenPackage("MyPackage.pkg", "MyKey");
PackageFiles(l$, p$);
ForEach(l$)
  ShowMessage(l$);
end;
ClosePackage(p$);
```

See Also: PACKAGEFILEEXISTS

void SAVEPACKAGE(HANDLE Package, string Filename, string Key)

Save a package, potentially changing the file name and key

Parameters

Package {in}
Handle returned from a call to OPENPACKAGE

Filename {in}
New name for the package file; if you don't want to change this, set *Filename* to NULL

Key {in}
New string to use for encryption; if you don't want to change this, set *Key* to NULL

Example:

```
if(PackageChanged(package$))
  SavePackage(package$, NULL, NULL);
end;
```

Notes:

- If you call SAVEPACKAGE without changing *Filename* it acts as a method of flushing the contents of *Package* to disk

See Also: OPENPACKAGE, PACKAGECHANGED

boolean PACKAGECHANGED(HANDLE Package)

Determines if a package has changed or not

Parameters

Package {in}
Handle returned from a call to OPENPACKAGE

Return Value

PACKAGECHANGED returns true if the contents of the package have changed since it was opened, or false otherwise

Example:

See SAVEPACKAGE for an example

See Also: SAVEPACKAGE

long PACKAGEFILESIZE(HANDLE Package, String PackageName)

Return the size of a file within a package

Parameters

Package {in}
Handle returned from a call to OPENPACKAGE

PackageName {in}
Name of file to determine size of

Return Value

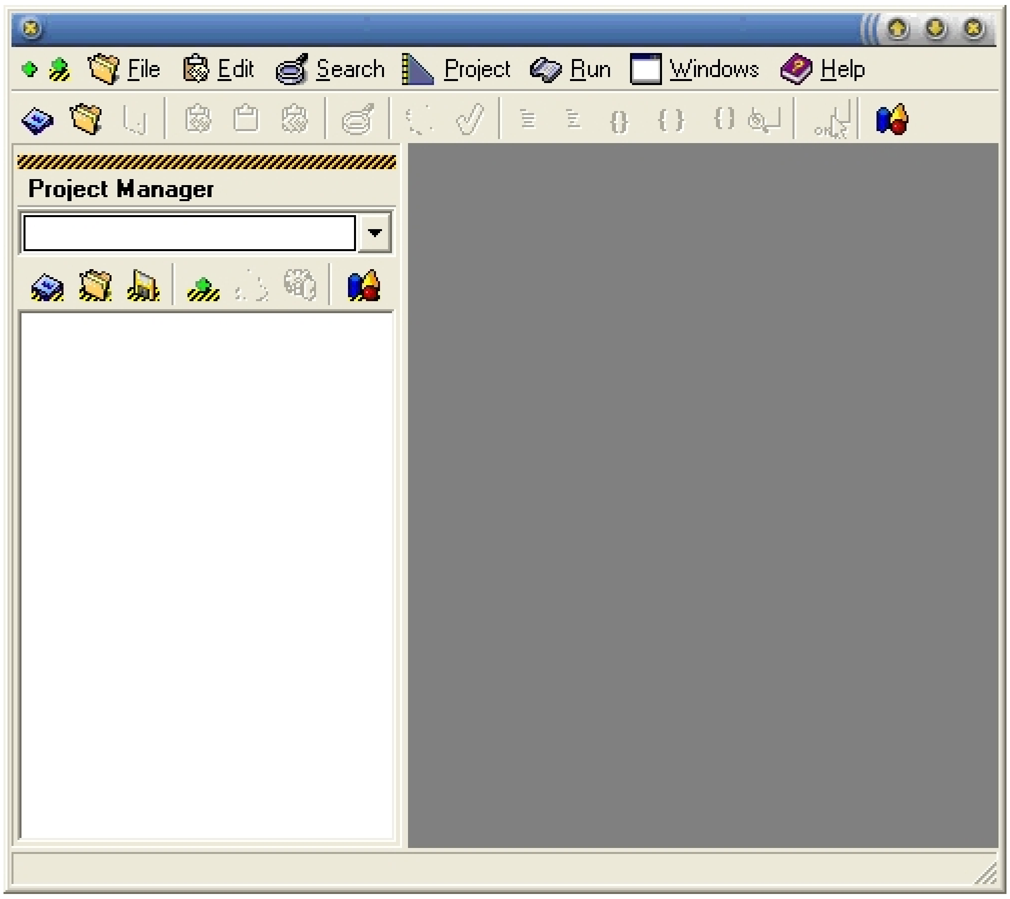
PACKAGEFILESIZE returns the size in bytes of the file when extracted from *Package*

Example:

```
if(PackageFileSize(package$, "MyFile.txt") > 2048)
  ShowMessage("File is too big to extract");
end;
```

See Also: EXTRACTFILEFROMPACKAGE, EXTRACTFILEFROMPACKAGEEX

What Is It?



PIDE stands for PPL Integrated Development Environment. This program allows you to develop PPL applications quickly and easily on your desktop PC. Unlike the PocketPC version of the IDE, PIDE provides a host of additional features, such as project management, profiling and so much more. The sections below go into more details about the various features of the PIDE.

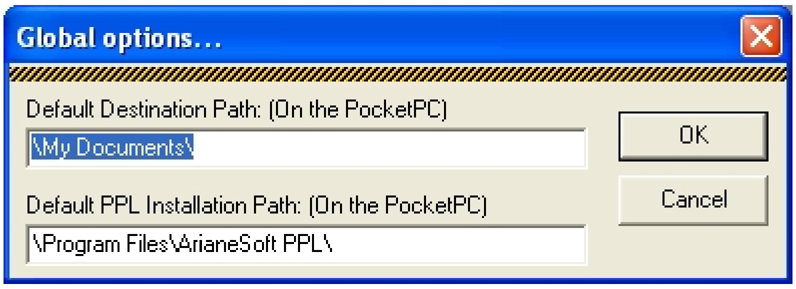
- Menus
- Toolbar
- Project Manager
- Visual Forms Editor

PIDE Menus

File

The file menu contains your standard set of operations for maintianing a PPL file: new, open, save, save as... and save all. In addition, you can configure your printer and print PPL files from this menu. Of course you can also Exit the application here.

The one non-standard selection in this menu is Options... Selecting this menu item brings up the following dialog:



Default Destination Path: (On the PocketPC)
\My Documents\
Default PPL Installation Path: (On the PocketPC)
\Program Files\ArianeSoft PPL\

The **Default Destination** is the location on your PocketPC or PC where the PPL file(s) you are working on will be copied to. **Default PPL Installation Path** is the location on your PocketPC or your PC that the PPL compiler is installed to. Entering values in this screen set the global defaults for the PIDE, but you can override these defaults on a project by project basis.

Edit

Within the edit menu you have standard Windows edit capabilities: **Undo** the last operation, **Redo** what you've just undone, **Cut, Copy,** and **Paste** sections of code.

Go To allows you to jump to a particular line of code within the currently active document.
Comment Code comment out or uncomment the selected piece of code.
RGB Color allows you to choose a color and the Red, Green and Blue color codes will be insert into the code.
Format Code will nicely format your code for you using indentation at appropriate places.
Remove comments will remove all the comments in your code selection.
Remove blank lines will remove all the blank lines in your code selection.

Search

The serach menu contains options to let you **Find...** certain text within the active document, as well as **Replace...** certain instances of text.

Find in files allow you to look for a string value in multiple files. Check here for more information.
Find Definition will find the definition of the current word under the cursor.
Open Selected File will try to find and open the file under the cursor.
Line Profile Result find the line in the profile report. You need to have ran the profiler first.

Tools

File Manager open up the file manager window. Check here for more information.
Visual Form Builder open up the Visual Form Builder window. Check here for more information.
Procedures List shows all the procs and funcs of the current code and let's you find specific names.

Windows

The **New** option is the equivalent of selecting **New** from the *File* menu. **Close** closes the currently selected PPL window, while **Close All** closes every PPL editor window. **Error Log**...

Help

Standard help functions: **Help** brings up this help file, and **About** gives you some information about the PIDE development tool.

The rest of the menus will be described in depth in their own sections.

Project

Run

Form

Controls

Project Menu

New starts a new PPL project

Open... opens a currently existing PPL project

Save saves the PPL project currently loaded in the editor

Save As... lets you save the currently loaded PPL project under a different name. Could especially be useful if you want to create a default project.

Add... allows you to add PPL files to your project

Remove removes PPL files from the current project

Edit this option pulls the currently selected file from the project manager into the editor window if it is not already open.

Select custom file destination... this option allows you to specify a path on the PocketPC for individual files within a project. This could be useful if you want some files to go to a subdirectory underneath your main program's directory, or even to a completely different path.

Synchronize file with PocketPC... this option compares the currently selected file in the project manager with the corresponding file on the PocketPC. Based on certain criteria the PIDE will prompt you whether you wish to copy the file from the PocketPC to your desktop or vice versa.

Transfer file to PocketPC - this option copies the currently selected file in the project manager over to the PocketPC. Keep in mind that the file will be copied regardless of whether it is older than the file currently on the PocketPC.

View Form Source
If you select a form in the project manager, you can generate and view it's PPL source code.

Options...
This item is the same as the **File | Options...** choice, but it sets the items only for the current project. This could be useful if you are working on several projects that you would like to go to separate folders on your PocketPC, or if you are working on projects that have different PocketPCs as their desitnation that might have different install locations for PPL.

Close
Closes the current project files.

Run Menu

(Note: with the exception of the Breakpoint menu options, all of these functions require that you be connected to your PocketPC. Also, any reference to "*current file*" in the topics below referes to either the file that is selected in the current project, or the the file which has the focus in the editor window if no project is open.)

Run
This option will compile the current PPL file and run it on your PocketPC.

Dedicated Run

Profile

Memory Analyzer

Compile
Compiles the current PPL file and displays the results in the error / debug window, which appears after the first time you compile an application. You can also open the log window manually by selecting the **Error Log** option from the *Windows* menu.

Compiler Options
Here you can set different compiler switches.

Warnings, turn on or off variables declaration warnings.
Explicit var check, each variable much be explicitely declared before being used.

Optimize, turn the optimizer on or off. The default is on.
Don't link, turn off or on the linker. Unused procs or funcs will be removed by the linker. You might want to leave them there if you are calling them using the Call internal function.
Forced transfers, force file transfers of compiled .ppl and .ppc files even if they exists in the target directory.

Temporary Options
Set the current code target compilation and running path.

Clear Temporary Options
Reset temporary options, next time you run or compile, they will be asked again.

Debug
This option will compile and launch a PPL file on your PocketPC, and then let you trace through the program at any breakpoints that you have set.

Step Over
Use this option to step through your code line by line. When you get to a function or procedure call, however, the debugger will call the routine without stepping through it.

Step Into
Similar to **Step Over**, but when the debugger gets to a function or procedure call it will actually step through the routine.

Run To Cursor
As you are stepping through your code, you can use this option to run the program to a certain point (whereever you have placed the cursor within in the code) without having to put a break point in.

Stop
Stops the execution of the PPL file that is currently running.

Toggle Breakpoint
Turns on / off a breakpoint at the location your cursor is at in the currently selected PPL file that is being edited.

Clear Breakpoints
Turns off all breakpoints that are currently active.

BreakPoints...
Bring up a list of all breakpoints assigned.

Watches
Bring up the trace window. Here you can create a list of variables you'd like the debugger to monitor.

Form Menu

When using the Visual Form Builder, the Form menu and Controls menu will be enabled.

AdjustForm
Re-adjust the forms boundaries to match current resolution.

Resolutions
Select the resolution of the screen you'd like to work on.

Initialization section code
Edit the initialization code section of the form. The initialization code section is generated just before any form creation code is done.

Form creation code
Edit the form creation code section. The form creation code is generated right after the form and all it's controls have been created but before the form is shown on screen.

Create Source
View the form generated PPL code.

Menu Editor
Access the current form's menu editor.

Form Options

Dialog Form, create a form that has creation code to be used as a dialog using the ShowModal() function.
Generate Library, will create a form that is used as a library without being executed automatically. The user can then include the form and create it when wanted.
Simplified Event Handling, PPL offers two types of syntaxes to code forms. The Simplified Event Handling allows you to code events for your forms or controls just like any RAD tools on the market today. The second one, allows to use the standard Windows API to code forms. It's much more complicated but you get more control.
Extended event code, when using Simplified Event Handling you can use extended code which is a one line code that is added at the top of each event code to simplify parameters handling.

Goto Map...
When using the Game Editor, you can go to a specific map. The global map (which is loaded at the beginning and always stay present) is number -1.

Information...
Enter user information about the form.

Preferences...
Allow you to specify the size of grid, turn it on or off, show or hide the PocketPC background image.

Controls Menu

Code
Edit the selected control's code.

Clear Code
Clear the selected control's code.

Bring Forward
Bring the selected control forward.

Send Backward
Send the selected control backward.

Bring To Front
Bring the selected control to front.

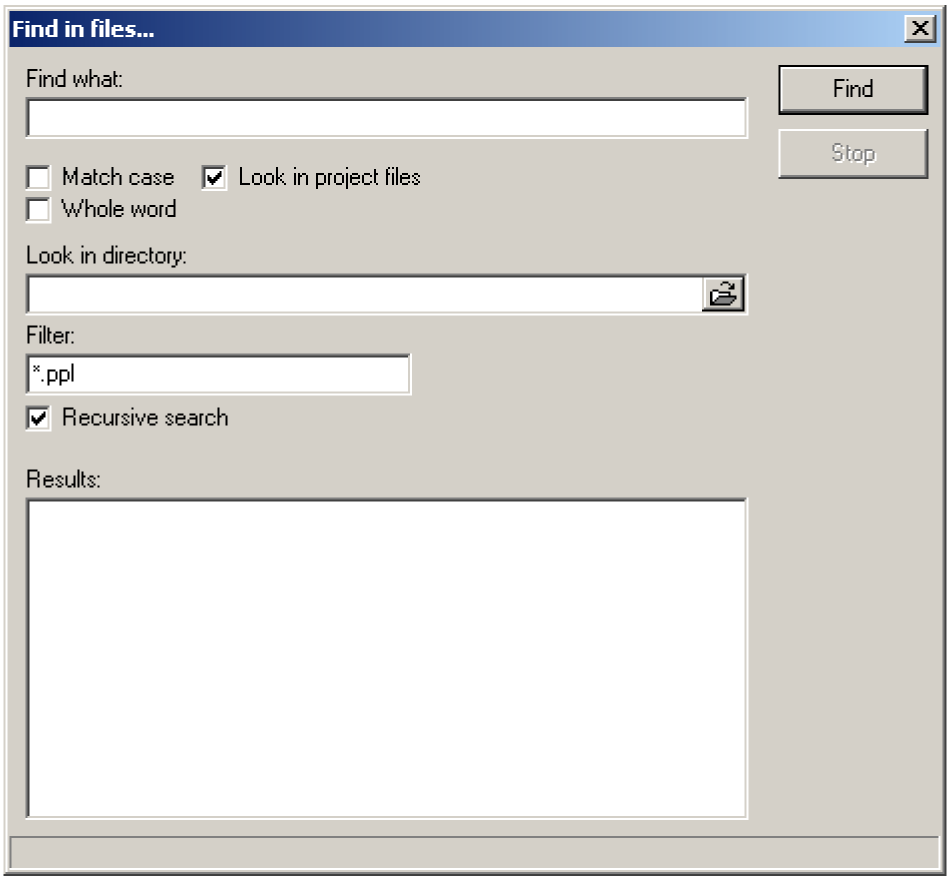
Send To Back
Send the selected control to the back.

Center control
Center the selected control on the form.

Insert ActiveX control...
Insert an activex control on the form.

Positions & Dimensions...
Change the positions and dimensions of the selected control or form.

Find in Files



Filter: *.ppl

☐ Match case ☑ Look in project files ☐ Whole word ☑ Recursive search

From here you can search for specific parts of text within a series of files. You can search from the files in the currently opened project or search in files from a specific folder.

Find What
Text you are looking for.

Match case
Match the case of the search text.

Whole word
Find only as a whole word.

Look in project files
Look in each file of the currently opened project.

Look in directory
Look in a specific directory.

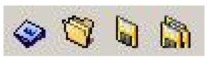
Filter
Search in only these file types.

Recursive search
When searching in directory, this will enabled recursive search through all directories.

Results
Double-click a result to bring up the editor.

Tool Bars

Below is a listing of what each of the icons in the toolbar represent. You can read the description of their associated menu item on the corresponding menu page.

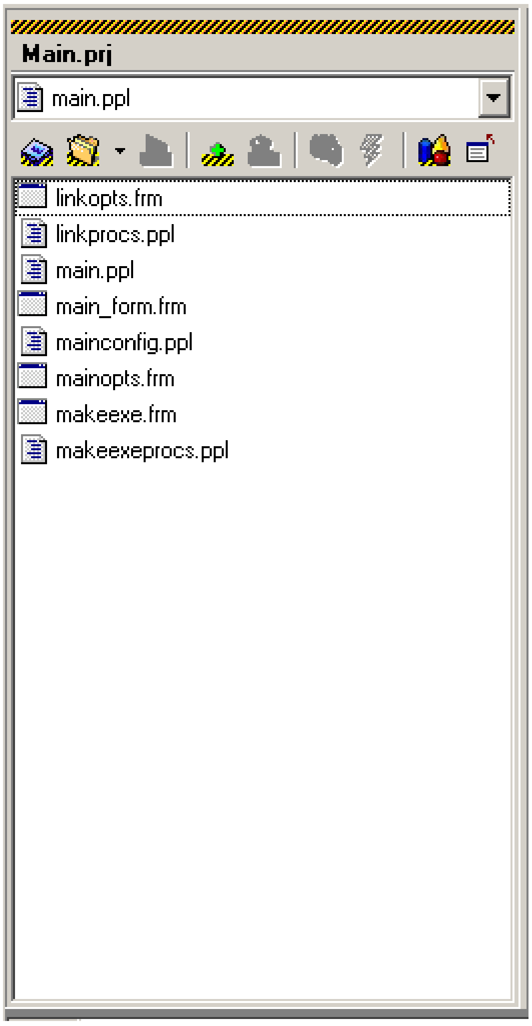

From left to right, these icons represent the following from the *File* menu: **New**, **Open...**, **Save**, and **Save All**

From left to right, these icons represent the following from the *Edit* menu: **Cut**, **Copy**, and **Paste**

From left to right, these icons represent the following: from the *Search* menu, **Find**; from the *Edit* menu, **Visual Form Builder, Menu Editor, Game Designer** and from the *File* menu, **Options...**

From left to right, these icons represent the following from the *Run* menu: **Compile, Run, Debug, Stop, Step Over, Step In To, Run To Cursor,** and **Trace**

Project Manager


Main.prj

main.ppl

linkopts.frm
linkprocs.ppl
main.ppl
main_form.frm
mainconfig.ppl
mainopts.frm
makeexe.frm
makeexeprocs.ppl

In the PIDE you can maintain projects. Projects allow you to organise your work and easily access code or forms in little time. You can add any type of files to a project. Each time you run or compile a project, all files are transfered to the destination folder, be it on the PC or PocketPC. You need to specify a main PPL file for each project. That is the file that will be ran everytime first. Each project can have it's own set of options (PPL installed folder and destination folder).

Debugging A Program

To simply run an application from the PIDE, select the **Run** option from the *Run* menu. This will compile the program that currently has focus in the edit window, or if you are working with a project, it will compile all the related PPL files. All files will then be transfered over to your PocketPC, where the PPL interpreter will be launched and your program executed. The only debug option at this point is **Stop**, which will halt the execution of the application. Note that even if you have set a break point, it will be ignored if you choose the **Run** option.

To actually step through an application, choose the **Debug** option from the *Run* menu. This will act similar to the **Run** option, but will allow you to actually trace through your application.

The first step to debugging is to set Breakpoints. You can do this by either selecting the **Toggle Breakpoint** option from the *Run* menu, or by placing the cursor on the line where you want to set a breakpoint and pressing the F9 key. Once you have your breakpoints set, select **Debug** to begin your application.

When the PPL interpreter gets to a point in your code where you've placed a breakpoint, execution of the application will be supsended and control will be returned back to the PIDE. At this point, you have a couple of different options.

The first is to **Step Over**. This is accomplished by selecting the appropriate menu or toolbar item, or by pressing the F10 key. Stepping over will execute each line of code in sequence, but if the line of code is a function or procedure that you have written, the code behind that function or procedure will be executed all at once, rather than line by line. Conditional loops (if, while, etc.) are treated like functions and procedures, so in order to trace through the code contained within a conditional you must use the **Step Into** option, which is described next.

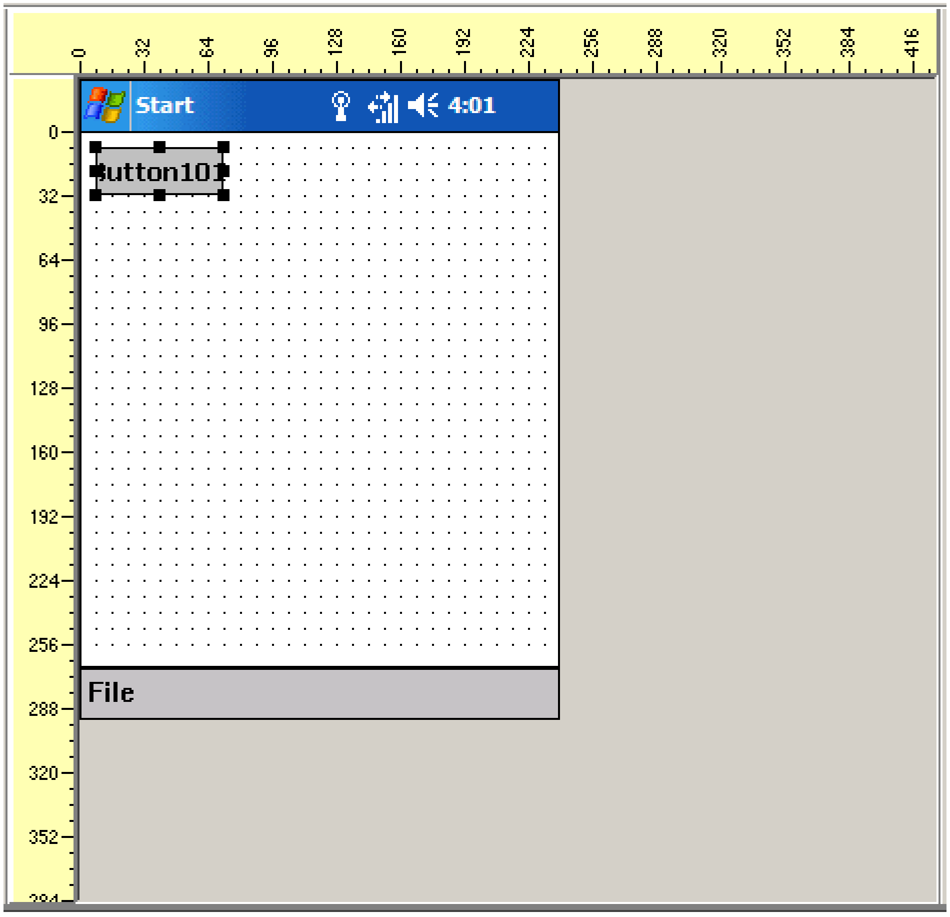
If you need to go through a particular procedure line by line, you must select the **Step Into** option. This is similar to **Step Over**, but when you reach a function or procedure, the PIDE will actually step through each line of the called function or procedure instead of running it all at once.

Another option is to **Run To Cursor**. After the program execution has been suspended and the PIDE is waiting for your input, you can place the cursor in the edit window on a particular line of code and select this option. The PIDE will then execute every line of code between where you had your breakpoint and where the cursor currently is sitting.

Finally, you can of course **Stop** the program at any time while it is running or while program execution has been suspended through the Debug process.

If you are going to be doing a lot of development using the PIDE (which is really the most efficent way to develop PPL programs), you will probably want some sort of tool that lets you control your PPC from the desktop. Microsoft has a free tool called Remote Display Control, which you can read more about here.

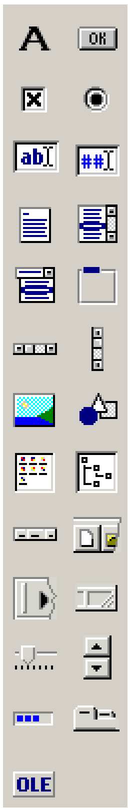
Visual Form Editor



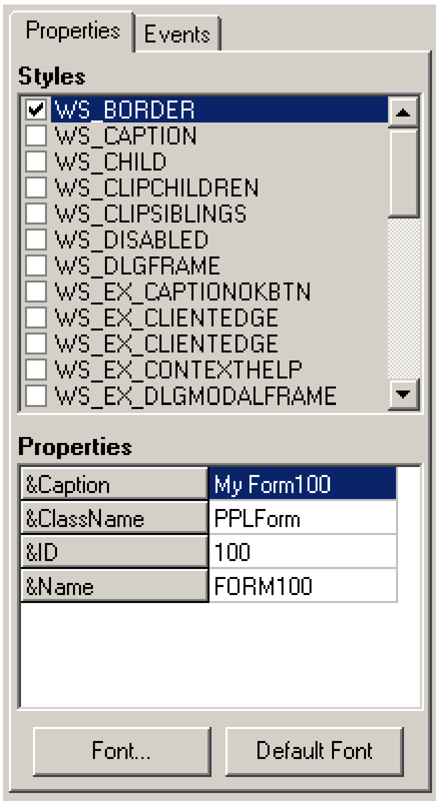
The visual form editor is a very powerful tool that lets you visually design a form with it's controls and set various properties and event code right from one interface.

To create a form that will be fullscreen on the PocketPC and readjust from the SIP on/off state or of default size on the PC, you should check the **WS_FORMDEFAULT** style in the form. The **WS_FORMDEFAULT** is checked by default when you create a new form. If you want your form to have the same size as the actual design size, uncheck **WS_FORMDEFAULT**.

You can also center the form on screen by default. The **WS_FORMCENTER** style is checked by default when you create a new form. If you don't want your form to be centered automatically, just uncheck WS_FORMCENTER. The form will be placed to the default position if **WS_FORMDEFAULT** is checked, else where the form is positioned in the actual design.



You can insert controls by clicking the left toolbar buttons.



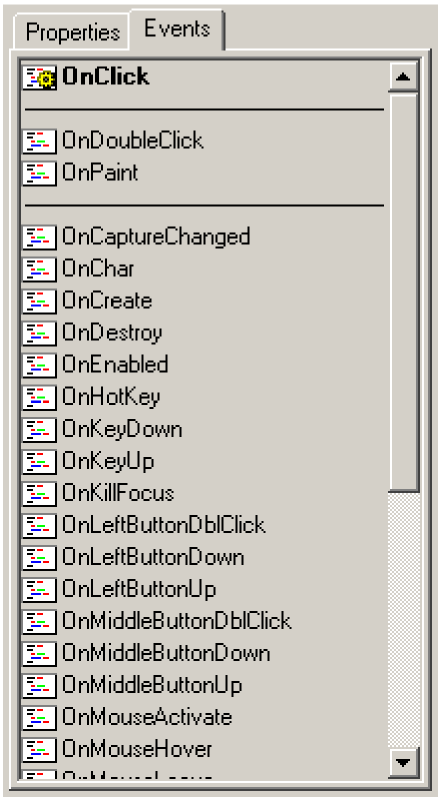
Styles

- [✓] WS_BORDER
- [ ] WS_CAPTION
- [ ] WS_CHILD
- [ ] WS_CLIPCHILDREN
- [ ] WS_CLIPSIBLINGS
- [ ] WS_DISABLED
- [ ] WS_DLGFRAME
- [ ] WS_EX_CAPTIONOKBTN
- [ ] WS_EX_CLIENTEDGE
- [ ] WS_EX_CLIENTEDGE
- [ ] WS_EX_CONTEXTHELP
- [ ] WS_EX_DLGMODALFRAME

Properties

&Caption	My Form100
&ClassName	PPLForm
&ID	100
&Name	FORM100

Each control has it's own set of styles and properties. You can check or uncheck control styles from the right panel and change some property's values.



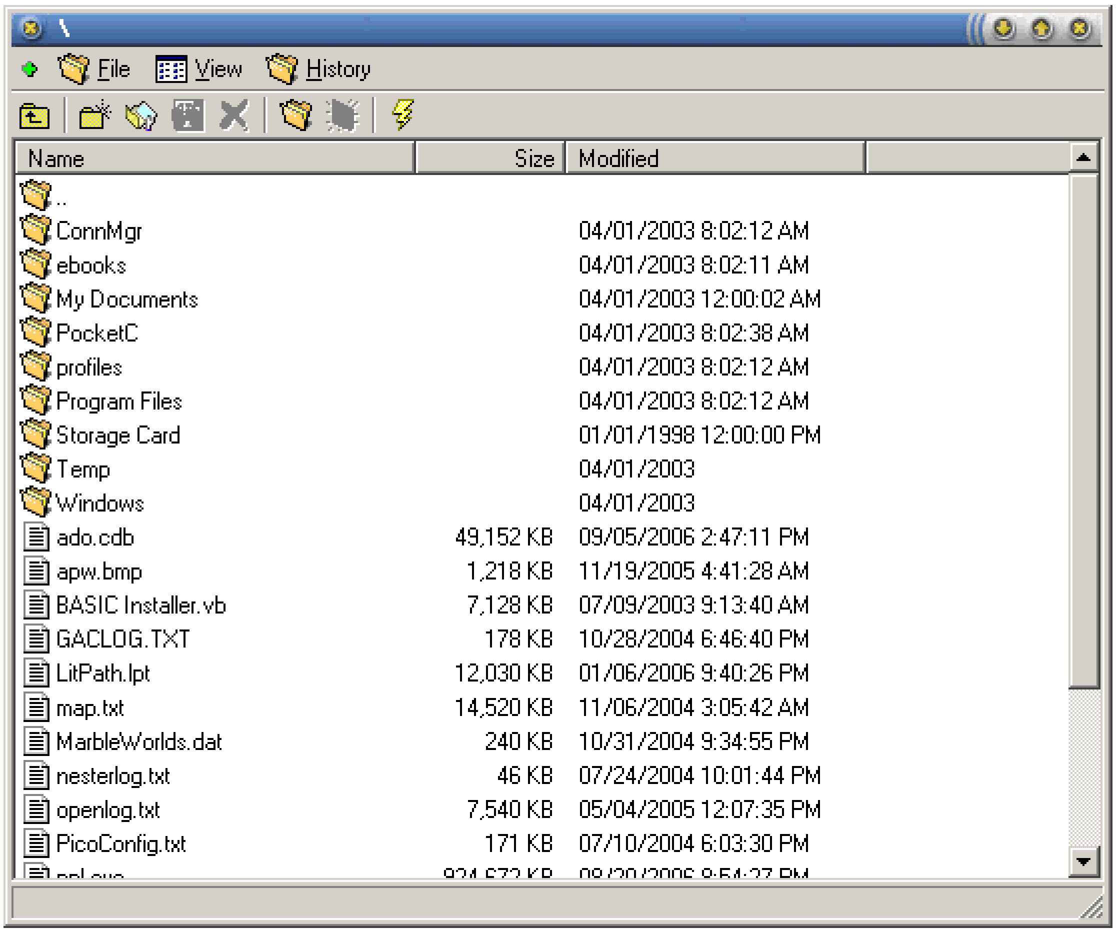
OnClick

OnDoubleClick
OnPaint

OnCaptureChanged
OnChar
OnCreate
OnDestroy
OnEnabled
OnHotKey
OnKeyDown
OnKeyUp
OnKillFocus
OnLeftButtonDblClick
OnLeftButtonDown
OnLeftButtonUp
OnMiddleButtonDblClick
OnMiddleButtonDown
OnMiddleButtonUp
OnMouseActivate
OnMouseHover

Each control has it's own set of events. You can edit it's code by simply double-clicking the desired event name. You will notice three sections. The first sections (in bold) is reserved for events that have code to them. The second is reserved for the control's root events and the third section is for inherited events shared by all controls.

File Manager



Name	Size	Modified
..		
ConnMgr		04/01/2003 8:02:12 AM
ebooks		04/01/2003 8:02:11 AM
My Documents		04/01/2003 12:00:02 AM
PocketC		04/01/2003 8:02:38 AM
profiles		04/01/2003 8:02:12 AM
Program Files		04/01/2003 8:02:12 AM
Storage Card		01/01/1998 12:00:00 PM
Temp		04/01/2003
Windows		04/01/2003
ado.cdb	49,152 KB	09/05/2006 2:47:11 PM
apw.bmp	1,218 KB	11/19/2005 4:41:28 AM
BASIC Installer.vb	7,128 KB	07/09/2003 9:13:40 AM
GACLOG.TXT	178 KB	10/28/2004 6:46:40 PM
LitPath.lpt	12,030 KB	01/06/2006 9:40:26 PM
map.txt	14,520 KB	11/06/2004 3:05:42 AM
MarbleWorlds.dat	240 KB	10/31/2004 9:34:55 PM
nesterlog.txt	46 KB	07/24/2004 10:01:44 PM
openlog.txt	7,540 KB	05/04/2005 12:07:35 PM
PicoConfig.txt	171 KB	07/10/2004 6:03:30 PM

The file manager allow you transfer files from and to your PocketPC.

Importing files will transfer from the PC to the PocketPC.

Exporting files will transfer files from the PocketPC to the PC.

Game Level Editor

The PIDE allow you to create sophisticated games visually in no time. It supports multiple levels we call Maps.

To add a sprite to your game level simply click on one of the icons in the left toolbar.

Sprite
This is a regular sprite.

World sprite
This is a sprite that will never be processed by the gameapi engine other than just being displayed. This type of sprite will speed up your game if you are using a lot of background sprites.

Physic sprite
This type of sprite will be processed by the gameapi physic engine. It has a mass and friction that can be modified.

You can design multiple maps within the same Game project. You will need to use the Goto Map... option from the Form Menu. The default map is -1. Map number -1 is the global map level. It will be loaded first, all sprites it contains will remain loaded even when you goto another map at runtime. You should use this map to store sprites like the main character.

At runtime, you will need to move from map to map. Each map has a loading and unloading code created for them. You can goto to a new map by using the following:

GotoMap (MapNumber);

You should review the GameAPI engine internals before venturing too far in the Game Level Editor. Some properties and events naming might sound too complicated for you at first.

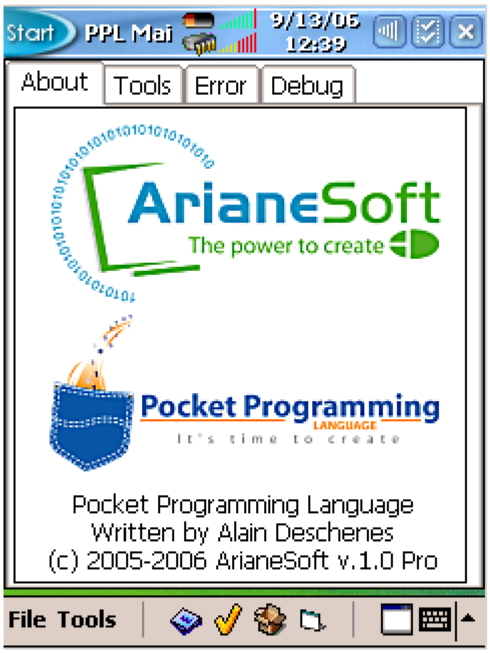
PockePC IDE

The PocketPC IDE is a set of programs written in PPL that allow you to use and create programs on your PocketPC device in an easy way.

The main user interface in PPL is strip down to a minimum requirement interface to allow for fast and easy access to .ppl and .ppc files. If you open a .ppc file from within the file explorer, the PPL interface will not show up and PPL will exit right after the execution of the .ppc file is finished.

The file menu allow you to bring up the PPL source code Editor, Visual Form Builder, Run (.ppc) or (.ppl) and compile a (.ppl) file.

Don't worry about compiling files with PPL; it is done transparently when a file is being run. If a .ppc file doesn't exist already, PPL will compile it first, if it already exists and its creation time is earlier than the .ppl creation time, PPL re-compiles it.



Pocket Programming Language
Written by Alain Deschenes
(c) 2005-2006 ArianeSoft v.1.0 Pro

File

Edit
Select a file to edit, then calls the editor.

Run
Select a file to run.

Compile
Select a file to compile.

Make EXE file
Create an .exe from a .ppl file. This will only work in the Pro version of PPL.

Console
Bring up the console.

Visual Form Builder
Bring up the visual form builder.

Program Manager
Bring up the program manager, which allows you see all running applications in PPL.

Package Manager
Bring up the package manager. This program only works in the Pro version of PPL.

Options
Edit the startup directory of the PPL applications to run.

Help
Bring up the help file.

Exit
Exit the main program.

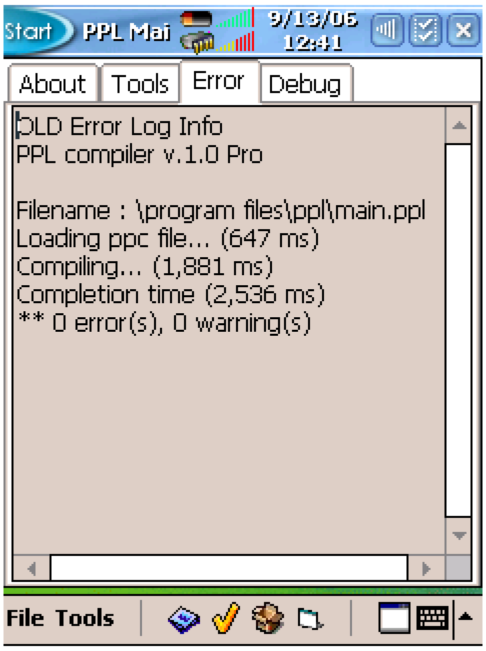
Tools

New Tool...
Create a new shortcut to a .ppl or .ppc application.

Edit Tool...
Edit the shortcut location.

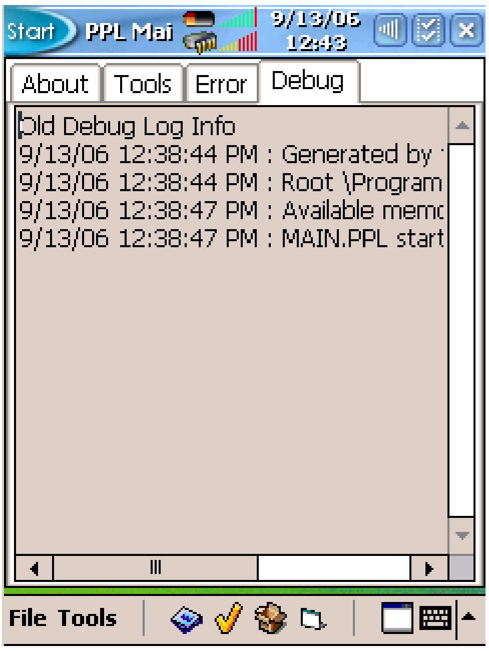
Change icon...
Change the shortcut icon.

Delete
Delete selected shortcut.



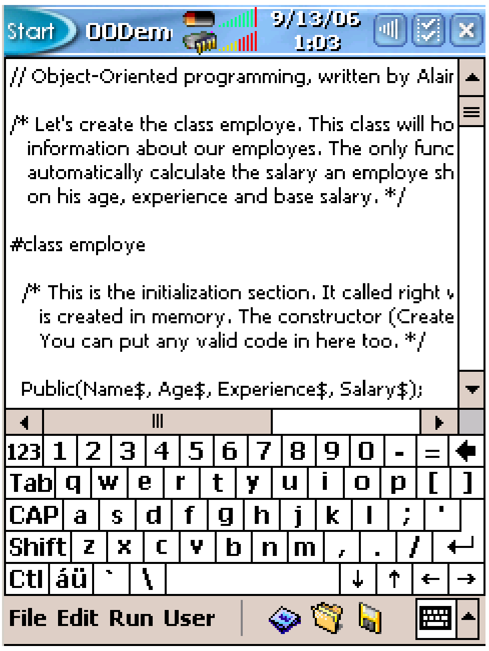
```
OLD Error Log Info
PPL compiler v.1.0 Pro

Filename : \program files\ppl\main.ppl
Loading ppc file... (647 ms)
Compiling... (1,881 ms)
Completion time (2,536 ms)
** 0 error(s), 0 warning(s)
```

The error log is produced after each compile and each run. It tells you specific information about the compilation, the time it took to compile etc...



```
Old Debug Log Info
9/13/06 12:38:44 PM : Generated by
9/13/06 12:38:44 PM : Root \Program
9/13/06 12:38:47 PM : Available memc
9/13/06 12:38:47 PM : MAIN.PPL start
```

The debug log file produced after each run. It tells the memory allocated in bytes, the time it took to execute the program etc...



```
// Object-Oriented programming, written by Alair

/* Let's create the class employe. This class will ho
   information about our employes. The only func
   automatically calculate the salary an employe sh
   on his age, experience and base salary. */

#class employe

  /* This is the initialization section. It called right v
    is created in memory. The constructor (Create
    You can put any valid code in here too. */

  Public(Name$, Age$, Experience$, Salary$);
```

The editor allow you to create and edit PPL source files. You can run or compile from it directly if you want.

File

New
Clear current text.

Open
Open a new text from file.

Save
Save current text.

Save as...
Save current text to another file.

Options...

Close
Close the editor.

Edit

Undo
Undo last change.

Cut
Cut selected text to the clipboard.

Copy
Copy selected text to the clipboard.

Paste
Paste clipboard text to the text.

Select All
Select all text.

Find...
Find a value in the text.

Find Next
Find the next occurence of a value in the text.

Replace...
Replace a value by another in the text.

Replace Next
Replace next occurence of a value.

Comment
Comment or uncomment a piece of code.

Proc List
Get a nice procedure list of current code.

Run

Run
Run current source code. You will need to save first.

Compile
Compile current source code. You will need to save first.

Error log...
Show the last errorlog.txt.

Debug log...
Show the last debuglog.txt.

User

User menus, check PPL\Users\ to modify this menu.