The PPL Development Kit

The PCBoard Programming Language
Reference Manual

Chapter 1 - Introduction to PPL 3
Chapter 2 - Installing PPLC 7
Chapter 3 - Developing PPL Applications 11
Step 1 - Creating Source Code 11
Step 2 - Compiling Source Code 11
Step 3 - Installing Your Application 12
PPE Files as Commands 12
PPE Files as Script Questionnaires 13
PPE Files Attached to PCBTEXT Display Prompts 13
PPE Files in Display Files 13
PPE Files as Display Menus 14
Step 4 - Testing Your Application 14
Chapter 4 - Using PPLC 17
Running PPLC 17
Specifying the Source Code File 17
Compiling Source Code 18
Compiler Warnings 19
Compiler Errors 20
Compiler Exit Codes 23
Chapter 5 - A PPL Tutorial 27
"Hello, World!" 27
Same Thing Done Differently 27
Fancy Variations 28
PPL Applications as Commands 29
Operator Page 30
Start 33
PPL Applications as Script Questionnaires 34

PPL Applications as PCBTEXT Display Prompts 36
Password Expiration Warning 36
Logon Language Prompt 37
PPL Applications in Display Files 39
Node Specific Display Files 39
Interactive Welcome Screens 39
PPL Applications as Display Menus 42
Chapter 6 - PPL Structure 47
Basics 47
Comments 47
Variable Declaration Statements 47
Code Statements 48
Expressions 49
Constants 49
Functions 50
Sub-Expressions 50
Operators 51
Chapter 7 - PPL Reference 55
Lists by Type 55
Constant List 55
Function List 55
Statement List 56
Type List 56
Variable List 56
Alphabetical List 57

1

Introduction to PPL

Introduction to PPL

Welcome to the PCBoard Programming Language (or PPL for short). A question that we regularly get is "Can we get source code for PCBoard?" Up until now the answer has been no. Well, the answer still is no (sorry, we aren't going to start selling our source code just yet!), but you can achieve almost the same level of control with PPL. You see, PPL applications can access files, control the local and remote displays, watch for input from the keyboard and modem, access user information, and more. And, all this can be accomplished without the need to go into a door or other external program because PPL support is built right in to PCBoard. PPE files (compiled PPL applications) are smaller than equivalent EXE files (and load and exit more quickly) because they take advantage of what PCBoard has already loaded into memory: a complete, robust operating environment for telecommunications!

Since PCBoard has the ability to run PPE files directly, you can write PPL applications to distribute to other SysOps as well. The PPLC package may not be distributed, but you may distribute your PPS files (PPL source code) and/or your PPE files in any way you see fit without any royalty payments or run-time licenses.

PPL is a new programming language. Although it is unique among other languages, it does bear many resemblences to BASIC and batch file programming. If you've never programmed before, don't worry; simple applications are a breeze to create. You need not understand everything that PPL has to offer and can learn as you go. If you are an experienced programmer in other languages, then PPL has a lot to offer you too.

Don't be fooled; although PPL is capable of many things, it is not designed to be all things to all people. You will still need (or want) other applications that you have used and trusted for a long time. Doors will not disappear because of PPL, nor should they. In fact, the *PCBoard DOOR Developer's Toolkit* makes a great companion to PPL for all of your board customization needs. If you are a C/C++ programmer, give us a call for more information about the *PCBoard DOOR Developer's Toolkit*.

However, what you can do with PPL (that you can't do with doors) is modify the action of your BBS. You can built new commands (or replace existing commands) by putting PPE files in the CMD.LST file. You can design intelligent script questionnaires by installing PPL applications in your SCR.LST files. You can attach PPE files to display prompts in PCBTEXT files. You can even launch a PPE from a display file or instead of a display menu! But before we can do any of this, we do have a few things to take care of first . . .

2

Installing PPLC

Installing PPLC

The PPLC development system comes on a single double density floppy disk. To install it do the following:

1. Insert the installation disk in an appropriate floppy drive (**A:** or **B:**).
2. Change to the drive (**A:** or **B:**) with the installation disk.
3. Type **INSTALL** to copy the files to your hard drive.

The **INSTALL** program will ask you for the drive to which the files should be installed. The default is **C:**. After selecting (or confirming) the drive letter you will be prompted to confirm default or enter new paths (**C:\PCB** and **C:\PCB\PPL**) to which to install the files. The following files will be installed on your system:

1. **PPLC.EXE** will be installed in your **C:\PCB** (or user selected) directory.
2. ***.PPS** files (files with a **PPS** extension) will be installed in your **C:\PCB\PPL** (or user selected) directory.

The following tutorial **PPS** files are included: **DOORS.PPS**, **HELLO1.PPS**, **HELLO2.PPS**, **HELLO3.PPS**, **HELLO4.PPS**, **HELLO5.PPS**, **HELLO6.PPS**, **HELLO7.PPS**, **LANGUAGE.PPS**, **NODEFILE.PPS**, **OPPAGE.PPS**, **ORDER.PPS**, **PWRDWARN.PPS**, **START.PPS**, and **WELFIRST.PPS**. Other examples may be included as disk space permits so be sure to check out the **C:\PCB\PPL** directory.

3

Developing PPL Applications

Developing PPL Applications

There are several steps involved in creating a PPL application. First, you have to write the source code. Second, you must compile the source code. Third, you must install the compiled application somewhere in PCBoard. Finally, you need to test the application. For every PPL program you write, you will likely repeat this process several times as you write, compile, test, modify, recompile, and test your modifications.

Step 1 - Creating Source Code

Before you can do anything you must write your source code. By default the compiler expects a file with a PPS extension for your source code, but you can make that extension anything you want except for PPE. You will need to use a text editor to write your source code. Any editor that will save files as unformatted ASCII text will suffice.

Step 2 - Compiling Source Code

After you've created your source code (as outlined above) you need to compile it. This is where you use PPLC.EXE. To run PPLC.EXE simply type PPLC at the DOS command prompt and hit ENTER:

```
C:\PCB\PPL>PPLC
```

Since we didn't specify a file name the following screen will be displayed:

```
C:\PCB\PPL>PPLC

PPLC Version 1.00 - PCBoard Programming Language Compiler

USAGE:  PPLC SRCNAME[.EXT]

C:\PCB\PPL>
```

The first line is to let you know what version of PPLC you are running. The second line is to remind you how to use the program. PPLC requires at least the base file name in order to execute. Additionally, you must specify the extension of the file if you didn't use the PPS default that PPLC automatically looks for. Finally, if the file you want to compile isn't in the current directory, you would need specify the path to the file. So, let's say we wanted to compile the source code we created in step 1. If the file name that we created was HELLO.PPS and it existed in the current directory then we would enter the following command:

```
C:\PCB\PPL>PPLC HELLO
```

We could have typed in HELLO.PPS instead of just HELLO. If the file was named HELLO.SCR we would have needed to type in the entire file name with extension. Finally, if the file wasn't in the current directory, we would have specified the path to file (for example,

SOURCE\HELLO or C:\PCB\HELLO.SCR). The following is a sample of the output that PPLC would generate after compiling HELLO.PPS without any problems:

```
C:\PCB\PPL>PPLC HELLO

PPLC Version 1.00 - PCBoard Programming Language Compiler

Pass 1 ...

Pass 2 ...

Source compilation complete...

C:\PCB\PPE>
```

After the compile is finished without errors, a new file named HELLO.PPE will exist in the same directory as the source code. For more information on using PPLC, see the Using PPLC section.

Step 3 - Installing Your Application

OK, now that we have a compiled application from step 2 we need to install it into PCBoard so that it may be used. Since there are several ways to install a PPE file, we will go over each of them at this time.

PPE Files as Commands

You can create new commands (or modify existing ones) with PPL. After compiling the application you install the PPE like this:

1. Run PCBSetup;
2. Hit B to select File Locations;
3. Hit B again to select Configuration Files;
4. Move down to "Name/Loc of Default CMD.LST File" and hit F2;
5. Add your PPE file to the list with an appropriate name and security level requirement.

In addition to the default CMD.LST file, each conference can have it's own specific CMD.LST file where you may want to install the PPE file:

1. Run PCBSetup;
2. Select the desired conference;
3. Hit the Page Down key to access screen two of the conference configuration;
4. Move down to "Conf-Specific CMD.LST File" and hit F2;
5. Add your PPE file to the list with an appropriate name and security level requirement.

If the PPE file can accept parameters you may enter them (on the same line) immediately following the PPE path and file name in the CMD.LST editor.

PPE Files as Script Questionnaires

You can create intelligent script questionnaires in place of (or in addition to) standard script questionnaires with PPL. After compiling the application you install the PPE like this:

1. Run PCBSetup;
2. Select the desired conference;
3. Move down to the Scripts Path/Name List File and hit F2;
4. Add your PPE file to the list in the desired position with an appropriate answer file.

In addition to the conference specific script listings, there are three other scripts that may utilize PPL: new user, logon and logoff questionnaires. They may be installed as follows:

1. Run PCBSetup;
2. Hit B to select File Locations;
3. Hit D to select New User/Logon/off Questionnaires;
4. Move down to the desired questionnaire and enter the path and file name of the PPE file to use.

If the answer path and file name is defined for a PPE based script questionnaire then a temporary file will be opened on channel 0 for questionnaire output. After the application has exited normally the temporary file will be appended to the answer file. If the application is exited abnormally the temporary file will not be appended to the answer file. In either case, the file will be deleted after PPE termination.

PPE Files Attached to PCBTEXT Display Prompts

You can attach PPE files to prompts in the PCBTEXT file that are displayed to the user. After compiling the application you install the PPE like this:

1. Run MKPCBTXT;
2. Select the desired display prompt;
3. Clear the existing prompt with the Ctrl-End key;
4. Enter an exclamation point (!) in the first column, followed immediately by the PPE path and file name.

If the PPE file can accept parameters you may enter them immediately following the PPE path and file name. Additionally, if it is a question prompt and you don't want PPL to display the standard question mark (?) and optional guides, add an underscore after the last character entered in the prompt field.

PPE Files in Display Files

You can include a PPE file in a display file. It will be run everytime the file is displayed. After compiling the application you install the PPE like this:

1. Load your text or graphics file editor;
2. Position the cursor on column one of a blank line;
3. Enter an exclamation point (!) in column one, followed immediately by the PPE path and file name.

If the PPE file can accept parameters you may enter them immediately following the PPE path and file name on the same line.

PPE Files as Display Menus

You can also completely replace a display menu (such as BRDM, BRDS, BLT, DOORS, etc.) with a PPE file. Simply give it the same base name as the display menu you want to replace (for example, you would replace BRDM with BRDM.PPE) and put it in the same directory as the original.

Step 4 - Testing Your Application

Alright, now that we've written our source code, compiled it to a PPE file, and installed it, all that is left is to run it. Before you make it available for others to use you should test it yourself and confirm that it works the way you intended. If you find a problem with the execution, go back and repeat steps 1, 2 and 4. (It is already installed so you probably don't need to re-install it!)

4

Using PPLC

Using PPLC

The core of the PPL development system is the PPL compiler, or PPLC.EXE (or just PPLC). PPLC.EXE is the program that reads source code files and generates compiled applications. A brief example of the usage of PPLC was given in the previous section. This section describes in detail how to use PPLC and what information it returns.

Running PPLC

Running PPLC is very simple. Type PPLC at the DOS prompt, just as you would with the name of any EXE, COM or BAT file. For example:

```
C:\PCB\PPL>PPLC
```

If PPLC.EXE in not in the current directory or any directory listed in the path, you would need to specify the entire path to the compiler. As an example:

```
C:\PCB\PPL>D:\PPL\PPLC
```

► NOTE: The remainder of the examples in this section will assume that PPLC is available in the current directory or the path.

Since we typed in PPLC without any arguments, it will come up with the help screen:

```
C:\PCB\PPL>PPLC

PPLC Version 1.00 - PCBoard Programming Language Compiler

USAGE:  PPLC SRCNAME[.EXT]

C:\PCB\PPL>
```

PPLC expects a single parameter to be passed: the path (optional), file name, and extension (optional) of the source code to compile into a PPL application, or PPE file. If the path is not specified then the current directory is assumed. If the extension is not specified, an extension of PPS is assumed. No matter what, never create a source code file with a PPE extension! The reasons will be explained shortly. In the meantime, just don't use that extension.

Specifying the Source Code File

The source code that is processed by PPLC to create a PPE file is created with a standard text editor as outlined in Developing PPL Applications. Let's say that you have created and saved a file named HELLO.PPS. Assuming it was saved in the current directory and that the current directory is C:\PCB\PPL, any of the following command lines could compile the source code:

```
C:\PCB\PPL>PPLC HELLO
C:\PCB\PPL>PPLC HELLO.PPS
C:\PCB\PPL>PPLC C:\PCB\PPL\HELLO
C:\PCB\PPL>PPLC C:\PCB\PPL\HELLO.PPS
```

Had we given the file an extension of something other than PPS we would have been required to specify it on the command line. So, to compile a file named HELLO.SCR, we would type:

```
C:\PCB\PPL>PPLC HELLO.SCR
```

If we created the file in a directory other than the default, we would need to specify the path with the file name. So, to compile a file named HELLO.PPS in the D:\SOURCE subdirectory, we would type:

```
C:\PCB\PPL>PPLC D:\SOURCE\HELLO
```

Of course, if the file contained a non-standard extension ***and*** was located in another directory, we would have to specify both the path and extension.

► NOTE: The remainder of this section assumes that source code files always have a PPS extension and are located in the current directory.

Compiling Source Code

OK, now that we know how to run PPLC and the command line parameters required to specify a file name, let's get down to the business of compiling a program. The most simple compile possible looks something like this:

```
C:\PCB\PPL>PPLC HELLO

PPLC Version 1.00 - PCBoard Programming Language Compiler

Pass 1 ...

Pass 2 ...

Source compilation complete...

C:\PCB\PPE>
```

We specify an existing file name and PPLC compiles it and generates the PPE file without warnings or errors. Of course, on our first attempt we will often make a mistake and get some sort of an error message:

```
C:\PCB\PPL>PPLC HELLO

PPLC Version 1.00 - PCBoard Programming Language Compiler

Pass 1 ... /

Warning in line number 2
Too many arguments passed (FRESHLINE:0:1)

Pass 2 ...

Source compilation complete...

C:\PCB\PPE>
```

In this case we encountered a warning. Warnings are not fatal; all they do is tell us that we may have made a mistake on a line of source code, but that the compiler can still generate a PPE file. The only warning message returned by PPLC is for too many arguments passed to a

statement, function or array variable. Since PPLC can safely ignore extra parameters it is not a fatal error. An example of error messages might look something like this:

```
C:\PCB\PPL>PPLC HELLO

PPLC Version 1.00 - PCBoard Programming Language Compiler

Pass 1 ... /

Error in line number 1
Variable not found (STR)

Error in line number 2
Not enough arguments passed (NEWLINES:1:0)

Error in line number 3
Variable not found (STR)

Error in line number 4
Variable not found (EMD)

Error(s) encountered, compile aborted...

C:\PCB\PPL>PPLC HELLO
```

As with the warnings, we are given the line number were the error was found. Unlike warnings, however, errors are fatal. The compiler cannot recover from an error and still generate a PPE file. If an error is found during the compile PPLC will finish the current pass so that you may see all errors that need to be fixed before the source code can be successfully compiled.

Compiler Warnings

Anytime a non-fatal syntax mistake is found during the compile a two line warning message is displayed. The first line will always have the following format:

```
Warning in line number #
```

The pound sign will be replaced with the line number where the warning was found. The second line will be the actual warning text. The following is the only warning message supported at this time:

```
Too many arguments passed (K:E:R)
```

The parentheses will be filled with specific information about the warning. The *K* will be replaced with the statement or function (or ARRAY if the problem is with an array) that generated the error. The *E* will be replaced with the expected number of arguments, and *R* with the number of arguments actually received.

Compiler Errors

Anytime a fatal error is encountered a two line error message is displayed. The first line will always have the following format:

Error in line number #

The pound sign will be replaced with the line number where the error was found. The second line will be the actual error text. The following is a list of error messages that may be returned and an explanation of what each means:

"Bad structure end statement (*BEG-END*)"

An attempt was made to close a block structure (IF block, WHILE loop, or FOR loop) with the wrong end statement. IF requires ENDIF, WHILE requires ENDWHILE, and FOR requires NEXT. *BEG* is replaced with the type of structure (IF, WHILE or FOR) and *END* is replaced with the statement that was used to attempt to close the block structure.

"Block start (IF/WHILE/FOR) must come before block end statement"

A block structure closing statement (ENDIF, ENDWHILE or NEXT) was used without a matching block begin statement (IF, WHILE or FOR).

"Closing parenthesis not found (*EXPR/ARGS/LINE*)"

An open parenthesis was found but a matching closing parenthesis was never found. The expression, argument list or line missing the parenthesis is displayed in place of *EXPR/ARGS/LINE*.

"Closing quote not found (*LINE*)"

An open quotation mark was found but a matching closing quote was never found. The line missing the quote is displayed in place of *LINE*.

"Error evaluating constant expression (*EXPR*)"

PPL requires a constant expression (an expression without variables) when defining the size of array dimensions. If there is anything wrong with a constant expression this message will be displayed. *EXPR* will be replaced by the constant expression with the error.

"Expression may not end with an operator (*EXPR*)"

The last term of any expression must be a variable, constant, function or subexpression. If an operator is found with nothing after it this error will be displayed. *EXPR* will be replaced by the expression with the error.

"IF/WHILE requires a conditional expression to evaluate"

IF and WHILE statements require an expression to evaluate to determine what course to take during program execution. If that expression doesn't exist, or if it is not surrounded by parentheses (which are required by PPL) then this error will be displayed.

"IF/WHILE requires a statement after the conditional expression"

IF and WHILE statements require a statement to execute if the condition is TRUE. If that statement doesn't exist then this error will be displayed.

"Illegal label (*LABEL*)"

A label must start with a letter (A-Z) and may contain letters, numbers (0-9) and the underscore character (_). Although they may be of any length, the compiler will only recognize the first 32 characters. If a label does not conform to these rules this error will be displayed.

"Illegal variable name (*VAR*)"

A variable name must start with a letter (A-Z) and may contain letters, numbers (0-9) and the underscore character (_). Although they may be of any length, the compiler will only recognize the first 32 characters. If a variable name does not conform to these rules this error will be displayed.

"Invalid character found in constant expression (*CHAR*)"

PPL requires a constant expression (an expression without variables) when defining the size of array dimensions. Constant expressions only allow +, -, *, /, %, (and) characters in addition to the numbers in the expression. All other characters are invalid in constant expressions. *CHAR* will be replaced by the invalid character from the constant expression.

"Invalid/Missing Operator in expression"

Variables, constants, functions and subexpressions need to be separated by operators. If an operator can't be found then this error message will be displayed.

"Invalid/Missing Variable/Constant in expression"

Variables, constants, functions and/or subexpressions are required on both sides of binary operators. If two operators are found back to back that is an error and will cause this message to be displayed.

"Label already used (*LABEL*)"

This error message will be displayed if you use a label name twice. *LABEL* will be replaced with the duplicate label.

"Label not found (*LABEL*)"

This error message will be displayed if you never define a label that is used in a GOTO or GOSUB statement. *LABEL* will be replaced with the missing label.

"Missing label"

This error message will be displayed if a label name doesn't follow a colon (used to start a label definition) on a line of source code.

"Missing variable name(s)"

This error message will be displayed if one or more variable names don't follow a type declaration keyword.

"No end found for block control statement (IF/WHILE/FOR)"

All block control structures must have matching beginning and ending statements. This message is displayed when an end statement doesn't exist for one or more IF, WHILE or FOR statements.

"No expression to evaluate"

This error message is displayed when an expected expression doesn't exist.

"Not enough arguments passed (*K:E:R*)"

All statements and functions expect a certain minimum number of arguments as input. If too few parameters are passed this message will be displayed. *K* will be replaced with the statement or function (or ARRAY if the problem is with an array) that generated the error. *E* will be replaced with the expected number of arguments, and *R* with the number of arguments actually received.

"Reserved constant name (*NAME*)"

PPL has a set of reserved constant names. They are like variables that never change value. If you try to create a variable with the same name as a reserved constant this error will be display. *NAME* will be replaced with the reserved constant that generated the conflict.

"Too many closing parenthesis (*EXPR/ARGS/LINE*)"

An open parenthesis was not found to match a closing parenthesis. The expression, argument list or line missing the parenthesis is displayed in place of *EXPR/ARGS/LINE*.

"Unable to allocate memory (*MSG*)."

This is a generic error message that may or may not be displayed with a line number. If it is displayed with a line number, the error occurred while performing a specific operation. *MSG* will be one of the following: CONVERTING EXPRESSION (translating it from the human readable source format into the PPE tokenized format), LABEL DEFINITION (adding the label to the label list maintained during the compile), or VARIALBE DECLARATION (adding the variable to the variable list maintained during the compile).

"Variable name already used (*VAR*)"

This error message will be displayed if you use a variable name twice. *VAR* will be replaced with the duplicate variable name.

"Variable not found (*VAR*)"

This error message will be displayed if you haven't yet defined a variable that is used in an expression or as a parameter to a statement. *VAR* will be replaced with the undeclared variable name.

Compiler Exit Codes

If you build projects via batch files, make utilities, or integrated text editors with compile options, the following exit codes (errorlevel in BAT file parlance) may prove useful:

0	This exit code is returned after a completely successful compile.
1	Indicates that one or more warnings occurred during a compile but that the PPE file was successfully created.
2	PPLC was started without a file name specified on the command line.
3	The file name specified could not be found.
4	One or more errors occurred and the compile was aborted without generating a PPE file.

5

A PPL Tutorial

A PPL Tutorial

Although very similar to BASIC and BAT file programming, PPL is not identical and will take a little practice to master. Skilled programmers with prior experience should have no problem making the transition. However, new or inexperienced programmers shouldn't feel left out since PPL is actually quite simple. This section will take you through writing several programs of varying complexity and explain why each is structured the way it is. Built in PPL constants, functions, statements, types and variables are in **bold** to help you identify those portions of the program that you can look up in the reference section for further explanation.

"Hello, World!"

It is traditional for the first program written in a new language to display "Hello, World!" So, being fond of tradition, we shall write that one first, and as simply as possible:

```
; HELLO1.PPS - "Hello, World!" #1
; Display (print) the string and terminate with a newline
PRINTLN "Hello, World!"
```

There is it, your first PPL program! Not much, huh? However, it does teach you one important concept, and that is that the PRINTLN statement is used to display (or print) information to the display (local and remote) and terminate it with a newline (carriage return and line feed).

Same Thing Done Differently

A valuable lesson to learn early on is that there are many ways to do even the simplest things. For example, each of the following code fragments does the exact same thing: displays "Hello, World!" and terminates it with a newline:

```
; HELLO2.PPS - "Hello, World!" #2
; This one displays it as two strings pasted together
PRINTLN "Hello, ","World!"

; HELLO3.PPS - "Hello, World!" #3
; Now we will display the string first without the newline,
; then print the newline with a separate command
PRINT "Hello, World!"
NEWLINE

; HELLO4.PPS - "Hello, World!" #4
; Finally we will display it to the local screen first, then
; to the remote screen, with two separate statements
SPRINTLN "Hello, World!"
MPRINTLN "Hello, World!"
```

As you can tell, each one does the same thing but in different ways. The moral of this story is: If at first you can't do something the way you thought it should be done, look for another way of doing it. Just about anything is possible if you look for a way to accomplish it and don't give up after your first attempt.

Fancy Variations

Now that we know how to display information to the screen, let's try to spice it up a little. The first thing we can try is adding color:

```
; HELLO5.PPS - "Hello, World!" #5
; Display the text in color (Bright white letters on a blue background)
PRINTLN "@X1FHello, World!"
```

Notice that we were able to use the PCBoard @X code in the string just like in display files to change the color. And, just like in display files, @X codes are automatically stripped out if the user doesn't support color displays. But even this is still kind of plain. What would be nice is to display the string in the middle of the display and wait for a key to be pressed:

```
; HELLO6.PPS - "Hello, World!" #6
; Center the text on the display
CLS             ; first clear the screen and position at the upper left
NEWLINES 11  ; Move down 11 lines to get to the center line
FORWARD 33   ; Move right 33 spaces to get to the correct column
PRINTLN "@X1FHello, World!"
STRING s
WHILE (s = "") LET s = INKEY() ; Loop until a key is pressed
```

Still not a lot of code, but it does things much differently than the original efforts. Now that we have this, let's try to change from displaying World to the users first name. As previously mentioned, there is more than one way of doing this. Here is an attempt that requires no user input:

```
; HELLO7.PPS - "Hello, World!" #7
; Center the text (with users name) on the display
STRING s
TOKENIZE U_NAME()            ; Separate the name into words (tokens)
LET s = GETTOKEN()           ; Grab the first name (word or token)
LET s = LEFT(s,1)+LOWER(RIGHT(s,LEN(s)-1)) ; Force it to mixed case
LET s = "@X1FHello, "+s+"!" ; Build the complete string to display
CLS
NEWLINES 11
FORWARD (80-(LEN(s)-4))/2   ; Move right to the correct column
PRINTLN s
LET s = ""
WHILE (s = "") LET s = INKEY()
```

OK, there are a couple of statements here are really deserve more explanation. The first is:

```
LET s = LEFT(s,1)+LOWER(RIGHT(s,LEN(s)-1)) ; Force it to mixed case
```

Let's assume that just before this line s had the value "SCOTT". Anyway, remembering that we always evaluate from the innermost parentheses to the outermost (after substituting variable values for variable names), here is how that expression would be evaluated:

```
The original statement
  LET s = LEFT(s,1)+LOWER(RIGHT(s,LEN(s)-1))

Step 1: Replace instances of s with "SCOTT" (the value of s)
  LET s = LEFT("SCOTT",1)+LOWER(RIGHT("SCOTT",LEN("SCOTT")-1))

Step 2: Evaluate LEN("SCOTT") (the innermost parentheses)
  LET s = LEFT("SCOTT",1)+LOWER(RIGHT("SCOTT",5-1))
```

```
Step 3: Evaluate 5-1
  LET s = LEFT("SCOTT",1)+LOWER(RIGHT("SCOTT",4))

Step 4: Evaluate RIGHT("SCOTT",4)
  LET s = LEFT("SCOTT",1)+LOWER("COTT")

Step 5: Evaluate LEFT("SCOTT",1)
  LET s = "S"+LOWER("COTT")

Step 6: Evaluate LOWER("COTT")
  LET s = "S"+"cott"

Step 7: Finally, evaluate "S"+"cott")
  LET s = "Scott"

Step 8: Now that we have a final result, assign it to s
  LET s = "Scott"
```

Notice how we always work from the innermost levels and evaluate them as we work our way out? This sort of debugging can be very useful on paper when trying to find out why something isn't working quite right.

The second statement that may not be immediately clear is:

```
FORWARD (80-(LEN(s)-4))/2
```

We've already seen the FORWARD statement used to move the cursor foward a specified number of columns on the current line. The part that could get confusing is the expression used to tell PPL how many columns to move the cursor. Let's break it down and see why we wrote it this way:

```
The original equation
  (80-(LEN(s)-4))/2

Step 1: Find the length of s ("@X1FHello, Scott!" from our example)
  (80-(17-4))/2

Step 2: Subtract 4 from the length (to adjust for the @X1F color code)
  (80-13)/2

Step 3: Subtract 13 from the screen width (count of total spaces)
  67/2

Step 4: Divide total spaces by 2 (half on one side, half on the other)
  33
```

(Don't forget that 67/2 is integer arithmetic, hence the answer of 33 instead of 33.5 as we would get with floating point arithmetic.) So, now we know how many columns to the right to move the cursor. Not very hard at all, huh?

PPL Applications as Commands

It is possible to replace existing commands as well as create new commands with PPL. Following are a couple of tutorials on how to implement commands. Remember, new commands are installed in the CMD.LST file in PCBSetup; just enter the command letter or

keyword, the minimum security level necessary to access the command, and the path and file name of the PPE file. Optional parameters may be specified after the path and file name as space permits. These optional parameters are accessed as tokens with the **GETTOKEN** function and statement. Note that user specified parameters may also be accessed via the **GETTOKEN** statement and function, but only if the SysOp doesn't specify any parameters in the CMD.LST file (in other words, SysOp specified parameters override user specified parameters).

Operator Page

One easy thing to replace is the operator page command. With PPL you can completely control how long the page will last, how long each beep will be, and the time between beeps. Lct's take a look at a sample operator page module:

```
;*********************************************************************

; OPPAGE.PPS - An O command (operator page) replacement

;*********************************************************************

; Variable Declarations

TIME    pTime     ' The time at which the user requested the page
TIME    sTime     ' The start time at which paging is allowed for all
TIME    eTime     ' The end time at which paging is allowed for all

INTEGER x         ' Temporary storage for cursor x position
INTEGER y         ' Temporary storage for cursor y position
INTEGER i         ' Index variable for page loop
INTEGER maxTries  ' The maximum tries allowed to page the SysOp

STRING  msg       ' A variable to hold the message to be displayed
                  ' to the SysOp
STRING  ynAns     ' A generic variable to hold a yes/no response

STRING  BEEP      ' An ASCII beep
STRING  CR        ' An ASCII carriage return
STRING  ANSI      ' ANSI escape sequence header
STRING  HOME      ' ANSI home sequence
STRING  CLREOL    ' ANSI clear to end of line sequence

;*********************************************************************

; Initializations

LET pTime    = TIME()                    ' Start time of the page
LET sTime    = READLINE(PCBDAT(),189) ' Read these two from the
LET eTime    = READLINE(PCBDAT(),190) ' PCBOARD.DAT file

LET maxTries = 5

LET BEEP     = CHR(7)
LET CR       = CHR(13)
LET ANSI     = CHR(27)+"["
LET HOME     = ANSI+"0;0H"
LET CLREOL   = ANSI+"K"

;*********************************************************************
```

```
; Main Program

' If pagins is allowed right now or if the user has SysOp level access
IF (((pTime>=sTime) & (pTime<=eTime)) | (CURSEC()>=SYSOPSEC())) THEN

  ' If SysOp level access or caller hasn't already paged
  IF ((CURSEC() >= SYSOPSEC()) | !PAGESTAT()) THEN

    ' The user may page (either a valid time or high security level)

    DISPTEXT 579,LFBEFORE ' Display the paging SysOp message
    DISPTEXT 97,LFBEFORE  ' Display the time and abort information

    ' Tell SysOp what to do
    LET msg = SPACE(15)+"Press (Space) to acknowledge Page, "
    LET msg = msg+"(Esc) when done."
    GOSUB topLineMsg

    FOR i = 1 TO maxTries

      ' Display a walking dot and beep at remote caller and SysOp
      PRINT "."
      MPRINT BEEP
      GOSUB localBeep

      ' If SysOp hits the space bar . . .
      IF (KINKEY() = " ") THEN
        LET msg = ""      ' Clear the SysOp message
        GOSUB topLineMsg
        CHAT              ' Start SysOp chat
        PAGEOFF           ' Since we've chatted, turn off page indicator
        END               ' Exit
      ENDIF

      ' If user aborted page, set up to exit loop
      IF (ABORT()) LET i = maxTries+1

    NEXT

    ' Clear the SysOp message
    LET msg = ""
    GOSUB topLineMsg

    ' If user aborted page . . .
    IF (ABORT()) THEN
      RESETDISP ' Reset the display so more info may be displayed
      NEWLINE   ' Send a newline
      END       ' Exit
    ELSE
      NEWLINE   ' Otherwise a newline is sufficient
    ENDIF

  ENDIF

ENDIF

' The user shouldn't be allowed to page (or page not successful), so
PAGEON                                          ' Turn on paged indicator

DISPTEXT 128,LFBEFORE+NEWLINE                   ' SysOp not available

LET ynAns = NOCHAR()                            ' Default to no
PROMPTSTR 571,ynAns,1,"",YESNO+NEWLINE+LFAFTER+FIELDLEN+UPCASE
```

```
IF (ynAns = YESCHAR()) KBDSTUFF "C"+CR+"Y"+CR  ' If yes do a comment

END

;*************************************************************************

:topLineMsg ' Clear the top line of the BBS screen and display a message

LET x = GETX()-1    ' Save the cursor position
LET y = GETY()-1

SPRINT HOME,CLREOL ' Pos in upper left of display and clear the line

SPRINT msg          ' Display message to the SysOp

SPRINT ANSI+STRING(y)+";"+STRING(x)+"H" ' Restore original position

RETURN              ' Return to the calling routine

;*************************************************************************

:localBeep ' Routine to alert the SysOp (not the caller)

SOUND 110   ' Sound a 110 hertz tone locally
DELAY 2     ' Pause for a couple of clock ticks
SOUND 220   ' Sound a 220 hertz tone locally
DELAY 2     ' Pause for a couple of clock ticks
SOUND 440   ' Sound a 440 hertz tone locally
DELAY 2     ' Pause for a couple of clock ticks
SOUND 880   ' Sound a 880 hertz tone locally
DELAY 2     ' Pause for a couple of clock ticks
SOUND 0     ' Turn off the speaker
DELAY 10    ' Pause for the remainder of the clock ticks

RETURN      ' Return to the calling routine

;*************************************************************************
```

This PPL application functions almost identically to the built in operator page command. It really only does three major things differently. The first is the length of the page. PCBoard's built in O command waits for thirty seconds (fifteen beeps at two seconds between beeps) for the SysOp to respond. This PPE implements a variable length page which is initialized to five tries by default (the length of each try depends on the **localBeep** subroutine). It can easily be changed by just changing the value **maxTries** is initialized to. For example:

```
LET maxTries = 15
```

This line, used instead of the default in the listing above, will change the operator page PPE to try to page the SysOp fifteen times (just like the default O command). The second major difference is way it pages the SysOp. PCBoard uses a standard beep every two seconds when paging the SysOp. This PPL program will sound a custom alarm sound once per page attempt. Since the default **localBeep** subroutine takes about a second, our PPE will attempt paging the SysOp for about 5 seconds. Again, you could change that rewriting the **localBeep** routine to make different sounds and/or to use different delays. As an example:

```
:localBeep  ' Routine to alert the SysOp (not the caller)

SPRINT BEEP ' Beep at the SysOp
DELAY 36    ' Wait for approximately two seconds
```

```
RETURN      ' Return to the calling routine
```

Again, with this change we are more like the built in O command in that we will beep at the SysOp once every two seconds. Finally, the third major difference is who is allowed to page the SysOp. Normally the SysOp may be paged during a certain set of hours, but only if the page bell is on. This PPE file ignores the page bell on/off status (since it is easily forgotten or accidentally changed when setting things up) and allows all to page the SysOp during the page window. Also, normally the O command is an all or none proposition; that is, everyone can page during the defined times and no one can page any other time. This PPE allows everyone to page during the defined times, **and** it allows users with SysOp level access (as defined in PCBSetup) to page **anytime**! Nifty, huh? Analysis of the code should explain just about everything else that is going on (of course, you will probably want to refer to the reference section if you find a statement or function that you don't understand).

Start

PPL is not limited to replacing existing commands. You can also create new commands with it. The following is the source code to the START command (used on the Salt Air BBS to give SysOps information necessary to start BETA testing software):

```
;*************************************************************************

; START.PPS - A new command used to start BETA testing PCBoard

;*************************************************************************

; Variables

INTEGER minSec ' The minimum security required to BETA test
STRING  CR     ' A carriage return

;*************************************************************************

; Initializations

LET minSec = 20
LET CR     = CHR(13)

;*************************************************************************

; Main Program

' If the user doesn't have current support
IF (CURSEC() < minSec) THEN
  DISPFILE PPEPATH()+"SBAD",GRAPH+SEC+LANG ' Display information file
  END                                      ' Exit PPE
ENDIF

DISPFILE PPEPATH()+"SOK",GRAPH+SEC+LANG    ' Display information file

' If the user isn't in conference 6 force them to join
IF (CURCONF() <> 6) KBDSTUFF "J 6 NS"+CR

' Force them to read messages pertaining to the start of the BETA test
KBDSTUFF "R O 61977+"+CR

;*************************************************************************
```

This is a very simple (but useful) PPL application. To sum things up: first the users security level is tested. If it is less than our minimum required security level to BETA test, we display a file to the user (that could have graphics, security, and/or language specific variants) and exit. Otherwise we display a file with information to the caller on starting the BETA test. If they are not in conference 6 (the BETA conference on Salt Air) we stuff a command to join 6 into the keyboard buffer, and then we force them to start reading messages that have some additional information. It is as simple as that!

PPL Applications as Script Questionnaires

Very powerful script questionnaires can be written with PPL. To install a PPE file as a script questionnaire, just enter the name of the PPE file (including the extension) in the SCRIPT field of the SCRIPT.LST file. Here is an example of what a PPL based script might look like:

```
;************************************************************************

; ORDER.PPS - A script questionnaire to order a product

;************************************************************************

; Variable Declarations

STRING Question ' The question to ask the user
STRING Answer   ' The users answer

;************************************************************************

; Main Program

' Display the script header
NEWLINE
PRINTLN "@X0F-------------------------------------------------------"
PRINTLN "We have several items available for sale.  From"
PRINTLN "Hardware to software, we have products that fit your"
PRINTLN "needs and wants.  If we don't have it, just ask!"
PRINTLN "-------------------------------------------------------"
NEWLINE

' Confirm that the user wants to answer the questionnaire
LET Answer = NOCHAR()
PROMPTSTR 84,Answer,1,"",YESNO+UPCASE+FIELDLEN
NEWLINE

' If user answers other than affirmative then stop script
IF (Answer <> YESCHAR()) STOP

NEWLINE ' Display a blank line for spacing

' List products available to the user
PRINTLN "@X0FWe have the following products available for sale:"
NEWLINE
PRINTLN " 1.  Complete 80486 system with SVGA video system ($1000 US)"
PRINTLN " 2.  Whiz-bang hard-drive ($500 Australian)"
PRINTLN " 3.  Plain paper bag software ($5 Monopoly)"

' Ask the user which product
LET Question = "Which item would you like to order?"
GOSUB ask
```

```
' List shipping options available
PRINTLN "@X0FYou may choose from the available shipping options:"
NEWLINE
PRINTLN " 1.  U.S. Mail"
PRINTLN " 2.  United Parcel Service"
PRINTLN " 3.  Federal Express"

' Ask the user how to ship
LET Question = "How would you like it shipped?"
GOSUB ask

' List payment options available
PRINTLN "@X0FYou have the following payment options:"
NEWLINE
PRINTLN " 1.  Visa"
PRINTLN " 2.  MasterCard"
PRINTLN " 3.  American Express"
PRINTLN " 4.  Discover"
PRINTLN " 5.  COD"

' Ask the user how he wants to pay
LET Question = "How would you like to pay?"
GOSUB ask

' Confirm that the user wants to save his answers
LET Answer = NOCHAR()
INPUTYN "Do you want to save your answers (ENTER=no)",Answer,@X0E
NEWLINE

' If user answers other than affirmative then stop script
IF (Answer <> YESCHAR()) STOP

NEWLINE ' Display a blank line for spacing
END     ' Exit script

;*****************************************************************

:ask ' Subroutine to ask questions and store answers

NEWLINE                          ' A blank line for spacing
PRINTLN "@X0E",Question          ' Display the question
LET Answer = ""                  ' Initialize answer to empty
INPUT "",Answer                  ' Get answer with no prompt on line
NEWLINES 2                       ' A couple of newlines for formatting

LET Question = STRIPATX(Question) ' Remove @X codes from question
FPUTLN 0,"Q: ",Question          ' Write the question to the file
FPUTLN 0,"A: ",Answer            ' Write the answer to the file

RETURN                           ' Return to the caller

;*****************************************************************
```

There are several items of interest in this program. The first is the use of file channel 0 in the **ask** subroutine without ever opening that channel. When a PPE file is installed as a script questionnaire, PPL automatically opens channel 0 in write mode for append access to the answer file. This way your application need not know the name of the answer file; all it needs to know is that channel 0 is where it's output should go. The second item of interest is the **STOP** statement. Normally the **END** statement would be used to exit a program (and commit

all information to the answer file in the case of a script questionnaire). **STOP** may be used to end script questionnaire processing and abort writing information to the answer file. Finally, the **ask** subroutine itself. This routine is used to ask all questions that should be written to the answer file and to perform the actual writing of information. Because we've used the single routine we can ask all questions and log them to the answer file in a consistent manner. Additionally, by not having to write the routine three (or more) times (once per question) we've saved ourselves sixteen lines of code and avoided the possibility of introducing bugs by not making changes to all three copies (when necessary) at the same time or in the same way.

PPL Applications as PCBTEXT Display Prompts

Compiled PPL applications may be attached to PCBTEXT display prompts. This may be done to change the way in which a question is asked (or whether it is asked at all) or to provide extra information that would not normally be available. To install a prompt replacement PPE, use MKPCBTXT to edit your PCBTEXT file. Select the prompt to replace and enter an exclamation mark in column one, followed immediately by the path and file name (including extension) of the PPE file. Optional parameters may be specified after the path and file name as space permits. These optional parameters are accessed as tokens with the **GETTOKEN** function and statement.

Password Expiration Warning

PCBoard 15.0 has enhanced password support. One of the features of the new password system is the ability to set an expiration date for a users password. When this is done the user will be warned a certain number of days before the password expires that they will need to change their password soon. This is done to give them an opportunity to change it before it becomes mandatory. Prompt 711 in the PCBTEXT file is displayed during the warning period to them. Unfortunately, if a user is calling in via a script and isn't there to see the one line prompt on screen, he will never know about the impending password expiration until it has already expired. This PPE file attempts to remedy that by sending the user a message in addition to the prompt:

```
;*********************************************************************

; PWRDWARN.PPS - A replacement for prompt 711 in the PCBTEXT file
;                to warn the user about impending password expiration
;                failure both on screen and via a message

;*********************************************************************

; Variable Declarations

INTEGER conf ' The conference in which to post the message
STRING  to   ' The user to send the message to
STRING  from ' The user the message is from
STRING  subj ' The subject of the message
STRING  msec ' The security of the message
DATE    pack ' The pack out date of the message
BOOLEAN rr   ' Return receipt flag
BOOLEAN echo ' Network echo flag
```

```
STRING  file ' The file with the message text

;*********************************************************************

; Initializations

LET conf = CURCONF()                        ' Post message in the current conf
LET to   = ""                               ' Default to the user online
LET from = "SYSTEM DAEMON"                  ' Any 'user' may leave the message
LET subj = "Password Expiration"            ' The subject of the message
LET msec = "R"                              ' Receiver only message
LET pack = DATE()+3                         ' Pack it out in 3 days
LET rr   = FALSE                            ' No return receipt requested
LET echo = FALSE                            ' No need to echo this message
LET file = PPEPATH()+PPENAME()+".MSG" ' Path and file name of message

;*********************************************************************

; Main Program

' First we need to tell the caller on screen
PRINT "Your password will expire in @OPTEXT@ days.
PRINTLN "Use the (W) command to change it."

' Now let's leave the caller a message.  We do this in case he is
' calling in an automated fashion (via script) and won't see the on
' screen warning.  This way the user will still be notified (if he
' downloads and reads mail) that his password will soon expire.
MESSAGE conf,to,from,subj,msec,pack,rr,echo,file

;*********************************************************************
```

This is a very simple application. Because we've replaced the prompt with the PPE, we go ahead and display the original default prompt so that we remain compatible. Then we generate a message for the user with basically the same information. Hopefully he will see this message just in case he doesn't see the on screen prompt.

Logon Language Prompt

One of the biggest concerns of adding any new feature is that it breaks compatibility. Of course, breaking compatibility can't be the only reason to decide not to add something, but it should be weighed carefully against the benefits. On the *Salt Air BBS* we had never used the multi-lingual capabilities for the support board. (Don't worry, we've tested them extensively on our in-house test systems!) With all of the new abilities of PPL, we wanted to add some highly customized prompts to assist us (and our SysOps) in tech support. At the same time, we didn't want to break everyones scripts. So we created two languages, one with the custom prompts and one with the standard prompts. Now people can select the one they want. The only remaining problem was that the language selection prompt might break peoples scripts (since we weren't using it before). So what we did was add a PPE to the language selection prompt that would time out after 20 seconds. This way we can have multiple languages and keep everybody happy without breaking scripts. (A lot of explanation for such a simple PPE, huh?)

```
;*********************************************************************

; LANGUAGE.PPS - A replacement for prompt 387 in the PCBTEXT file
;                to prompt for the desired language with a 20 second
;                timeout if the user doesn't respond (just in case the
;                user is automated we don't want to break their script)

;*********************************************************************

; Variable Declarations

STRING prompt     ' A variable to hold the language question
STRING lang_ansr  ' A variable to hold the users response

STRING CR         ' A carriage return character

;*********************************************************************

; Initializations

LET prompt = "Enter Language # to use (enter)=no change"

LET CR     = CHR(13)

;*********************************************************************

; Main Program

' Ask the user what language they want to use
INPUTSTR prompt,lang_ansr,@X07,2,MASK_NUM(),LFAFTER+AUTO

' If the user didn't answer the question (empty response) . . .
IF (LANG_ANSR = "") THEN

   ' We need to stuff the keyboard buffer with a CR so that
   ' PCBoard won't ask a second question (without a prompt)
   KBDSTUFF CR

ELSE

   ' Otherwise we just need to stuff the answer so that PCBoard
   ' knows that the PPE asked the question and got the answer
   KBDSTUFF lang_ansr

ENDIF

END

;*********************************************************************
```

An important point must be made here. If you want your PPE to get the input for the prompt and pass it to PCBoard, you must stuff the response into the keyboard buffer. If no response is required or desired then you should simply stuff a carriage return (**CHR(13)**) as we did above. If you want PCBoard to go ahead and ask the question, then you will need to print a prompt before exiting the PPE so that the user will know that a response is expected (and what the response should be).

PPL Applications in Display Files

For those times that you need really precise control of what the user is seeing you can embed a PPE within a display file. To do this simply include a line with an exclamation mark in column one, followed immediately by the path and file name (and extension) of the PPE file. Optional parameters may be specified after the file name. However, nothing else should be on the line after the file name or parameters. Here are a couple of examples of what you might use a PPE within a display file for.

Node Specific Display Files

We sometimes have a need to display a file to callers on a particular node (for example, our extended support nodes). However, we still want to display the standard news to them as well, and we don't want to have to maintain a separate NEWS file for each node (all we want to do is maintain the differences). With this PPL application we can force the NEWS file (or any display file) to display an additional, node specific file at a particular point:

```
;*********************************************************************

; NODEFILE.PPS - A PPE to be used from any display file.
;                By default display files can have security, graphics,
;                and language specific variants.  This PPE allows the
;                addition of node specific variants while continuing to
;                allow the other variations that are shared among all
;                nodes.

;*********************************************************************

; Variable Declarations

STRING file

;*********************************************************************

; Main Program

' If no parameter was passed then exit
IF (TOKCOUNT() = 0) END

' The file we are looking for is tokenized from the command line and has
' a ### extension (### is the node number)
LET file = GETTOKEN()+"."+RIGHT("00"+STRING(PCBNODE()),3)

' If the node specific news file exists, display it
' (security/graphics/languages variants aren't allowed because we use
' the file extension to indicate the node the file should be used for)
IF (EXIST(file)) DISPFILE file,DEFS

END ' Exit

;*********************************************************************
```

Interactive Welcome Screens

Though ANSI animated displays can go a long way to improving the look and feel of your BBS, they come with a cost: they are large. In fact, many people don't use them just because of the time involved in the transmission (especially to 2400 bps callers). However, there is a

way around this with PPL. We can write a program that will display a file to the caller in 'pieces' while prompting for input from the caller. After each 'piece' it will check to see if the user has hit a key and act on it as needed. If no key was pressed, it will display the next piece and check for keys again.

```
;*************************************************************************

; WELFIRST.PPS - A PPE to be used in a WELCOMEG file.
;                Because of the time required to display ANSI animation
;                to low speed callers it is usually avoided.  However,
;                this PPL allows you to send a long ANSI animation file
;                in pieces while waiting for the user to enter his
;                first name.  WARNING!  Do not use in a WELCOME file,
;                only WELCOMEG or WELCOMER, as it assumes ANSI graphics
;                are available.

;*************************************************************************

; Variable Declarations

BOOLEAN exitflag ' Flag to determine when we should exit

INTEGER x        ' Last column position of cursor
INTEGER y        ' Last row position of cursor
INTEGER c        ' Last color used

STRING  fn       ' The first name of the user
STRING  s        ' A miscellaneous string variable

STRING  file     ' The ANSI animation file to display
STRING  line     ' The ANSI animation line to display

STRING  BS       ' An ASCII backspace character
STRING  CR       ' An ASCII carriage return character

;*************************************************************************

; Initializations

LET BS = CHR(8)  ' Backspace
LET CR = CHR(13) ' Carriage return

;*************************************************************************

; Main Program

IF (TOKCOUNT() = 0) END ' If a file wasn't specified, exit

LET file = GETTOKEN()   ' Get the path and file name to display
IF (!EXIST(file)) END   ' If the file doesn't exist, exit

FOPEN 1,file,O_RD,S_DN  ' Open channel 1 for read/deny none access

ANSIPOS 1,23            ' Position on the bottom line of the display
PRINT "@X0EWhat is your first name? " ' and display the prompt

' While the user hasn't exited and no file errors have occurred . . .
WHILE (!exitflag & !FERR(1)) DO

  FGET 1,line         ' Get a line to display
  PRINT line          ' Display it
```

```
  LET x = GETX()      ' Save the cursor position
  LET y = GETY()
  LET c = CURCOLOR() ' Save the current color

  ANSIPOS 1,23        ' Position at the bottom of the display
  PRINT "@X0EWhat is your first name? " ' and display the prompt

  DEFCOLOR            ' Change to the system default color

  LET s = INKEY()   ' Get a keypress from the user

  IF ((s >= " ") & (s <= "~") & (LEN(fn) < 50)) THEN

    ' If it's ASCII append it
    LET fn = fn + s

  ELSEIF ((s == BS) & (LEN(fn) > 0)) THEN

    ' If it's a backspace remove the last character
    LET fn = LEFT(fn,LEN(fn)-1)

  ELSEIF (s == CR) THEN

    ' If it's a carriage return append it and prepare to exit
    LET fn = fn + s
    LET exitflag = TRUE

  ENDIF

  PRINT fn," ",BS ' Display the first name

  ANSIPOS x,y      ' Restore the last cursor position
  COLOR c          ' Restore the last color

ENDWHILE

FCLOSE 1 ' Close the file

' If we exited due to a file error and not a carriage return
IF (!exitflag) THEN

  ANSIPOS 1,23 ' Position at the bottom of the display
  DEFCOLOR     ' Change to the system default color
  CLREOL       ' Clear to the end of the line

  KBDSTUFF fn  ' Stuff the name into the keyboard buffer for INPUTSTR

  LET s  = "What is your first name" ' Initialize the prompt
  LET fn = ""                        ' Clear out the first name

  INPUTSTR s,fn,@X0E,50,MASK_ASCII(),DEFS ' Finish getting the name
  LET fn = fn + CR                        ' Append a CR to the end

ENDIF

CLS          ' Clear the screen
KBDSTUFF fn ' Stuff the first name into the keyboard buffer

END ' Exit the PPE file

;*************************************************************************
```

The main **WHILE** loop is actually quite simple once you understand what it is doing. First, read a line from the ANSI display file and print it to the screen. Second, save the last cursor position and color and display the first name prompt. Third, get a keystroke (if available) and process it (either add it to the string, remove the last character from the string if a backspace, or add it to the string and set the exit flag if a carriage return). Fourth, display the name with any changes since the last display. Finally, loop back up to the top and keep doing these four steps until the user hits enter or we reach the end of the file. The file that is being displayed should not have any single line longer than 256 characters. Most ANSI drawing and animation programs allow you to specify the maximum line length to save. A short line length (such as 32) will slow down display of the animation but will check the keyboard and serial port more often. A longer line length will speed up the display but will give the user fewer opportunities to enter his name.

PPL Applications as Display Menus

Finally, you can replace menu files (such as BRDM, DOORS, DIR, etc.) with PPE files. To do this simply create a PPE file with the same name as the menu to replace and store it in the same directory as the main menu file. PCBoard will automatically find it and use it. Here is a sample PPL based menu for a DOORS listing:

```
;*************************************************************************

; DOORS.PPS - A PPE to be used in place of the DOORS menu file.
;             This PPL application is designed to provide a hot key
;             interface to door selection.

;*************************************************************************

; Variable Declarations

BOOLEAN exitflag ' Flag to indicate when to exit the PPE

INTEGER i        ' A miscellaneous index variable
INTEGER x        ' The cursor column to display the whirly-gig
INTEGER y        ' The cursor row   to display the whirly-gig
INTEGER off      ' The offset of the whirly-gig in char array

STRING  name(25) ' A list of door names
STRING  char(3)  ' An array of whirly-gig animation
STRING  key      ' The users keypress
STRING  CR       ' An ASCII carriage return character

;*************************************************************************

; Initializations

LET exitflag = FALSE

FOR i = 0 TO 4                      ' These should be initialized to the
  LET name(i) = "DOOR"+STRING(i)    ' actual door names or nothing ("")
NEXT                                ' for that letter to abort the PPE
FOR i = 5 TO 25                     ' menu
  LET name(i) = ""                  ' NOTE that A = 0, B = 1, etc, Z = 25
NEXT
```

```
LET char(0)  = "/"
LET char(1)  = "-"
LET char(2)  = "\"
LET char(3)  = "|"

LET CR       = CHR(13)

;*********************************************************************

; Main Program

DEFCOLOR ' Change to the system default and clear the screen
CLS

' Display the PPE display file (base name + alternates)
DISPFILE PPEPATH()+PPENAME(),SEC+GRAPH+LANG

' Display the prompt
FRESELINE
PRINT "@X09Hit key to select door:
LET x = GETX()
LET y = GETY()

WHILE (!exitflag) DO

  ANSIPOS x,y          ' Position the cursor in the whirly-gig spot
  DELAY 2              ' Wait for a couple of clock ticks
  PRINT char(off%4) ' Display the current stage of whirly-gig animation
  INC off              ' Update off for the next stage of whirly-gig

  LET key = UPPER(INKEY()) ' Get the users keypress

  ' If the user pressed a hot key . . .
  IF ((key >= "A") & (key <= "Z")) THEN
    LET exitflag = TRUE                    ' Get ready to exit
    KBDSTUFF name(ASC(key)-ASC("A"))+CR ' Stuff the door name and a CR
  ELSEIF (key <> "")
    LET exitflag = TRUE                    ' Get ready to exit
    KBDSTUFF CR                            ' Stuff CR to abort door prompt
  ENDIF

ENDWHILE

;*********************************************************************
```

This PPL application is set up to provide a hot key interface for selecting a door. Note that we must initialize the list of door commands available up above (the **STRING** array name). Then if we hit a key (A-Z) that is assigned to a door, that door will be instantly selected. Otherwise, the PPE will simply exit (automatically hitting enter for the door name prompt to get past it).

6

PPL Structure

PPL Structure

Basics

A PPL program is created by a programmer with a standard text editor. Each line consists of standard ASCII text (up to 2048 characters long) terminated with a carriage return/line feed pair. Character case is not significant except in literal text strings. Three types of lines are recognized by the compiler: comment lines, variable declaration statements and code statements.

Comments

Comments are used by the PPL programmer to make notes in the source code about what the code is supposed to do and generally clarify things so that code maintenance is easier. They are completely ignored by the PPL compiler so they may contain any text desired. A comment may be on a line all by itself or at the end of a line after a valid statement. A blank line is considered a comment. Any text following a quote character (') or semi-colon (;) is also a comment. The following are all valid comments:

```
; This is a comment line
STRING buf, str, ssNum ' This is a comment too

' The blank line above this (as well as these
' lines) are all comments
CLS                    ; Yet *ANOTHER* comment!
```

Variable Declaration Statements

Variable declaration statements must start with a keyword denoting the variable type. Valid type keywords are **BOOLEAN**, **DATE**, **INTEGER**, **MONEY**, **STRING** and **TIME**. The keyword must be followed by one or more valid variable names (or array declarations) which should be separated by commas (,). A valid variable name must start with a letter (A-Z) and may contain letters, numbers (0-9) and the underscore character (_). Any number of characters may be used but only the first 32 will be recognized by PPL. If the variable is an array then the name should be followed by an open parenthesis [(], one, two or three constant subscript expressions (separated by commas), and finally a closing parenthesis [)]. Here are some examples:

```
BOOLEAN adultFlag
DATE    this_IS_a_VARIABLE_to_HOLD_todays_DATE
; Only this_IS_a_VARIALBE_to_HOLD_today is significant
INTEGER age
MONEY   prices(2,5)
STRING  buf, labels(10), ssNum
TIME    start, stop
```

Code Statements

Code declaration statements must start with a keyword indicating the operation or process to be performed. There is one exception to this rule, however, and that is the **LET** statement. If no keyword is found at the beginning of a line, a **LET** statement is implied and the rest of the line should follow the format:

```
VAR = EXPRESSION
```

There are many statements defined in PPL and it is beyond the scope of this part of the manual to cover the precise syntax for each and every one of them. Simply put, a statement takes zero, one or more expressions (see Expressions later in this section) and/or variable names (see Variable Declaration Statements) as arguments (separated by commas), does something, using any passed expressions and/or variables, and assigning new values, as needed, to passed variables. Here are a few sample statements:

```
' This statement clears the screen and takes no arguments
CLS

' Evaluates the single expression and assigns the result to ans
LET ans = 5+4*3/2-1

; Evaluates all three (could be more, could be less) expressions (two of
; which have only one term) and prints them in order, following them
; with a carriage return
PRINTLN "The answer "+"is ",STRING(ans),"."

; Evaluate the expression on the left, display it, then get a string
; from the user and assign it to the variable name on the left
INPUT "What is "+"your age",current_Age
```

Here are the valid statements accepted in PPL source code:

ADJTIME	DISPSTR	GOODBYE	MPRINT	RESTSCRN
ANSIPOS	DISPTEXT	GOSUB	MPRINTLN	RETURN
BACKUP	DOINTR	GOTO	NEWLINE	SAVESCRN
BLT	DTROFF	HANGUP	NEWLINES	SENDMODEM
BROADCAST	DTRON	IF	NEWPWD	SHELL
BYE	ELSE	INC	NEXT	SHOWOFF
CALL	ELSEIF	INPUT	OPENCAP	SHOWON
CDCHKOFF	END	INPUTCC	OPTEXT	SOUND
CDCHKON	ENDIF	INPUTDATE	PAGEOFF	SPRINT
CHAT	ENDWHILE	INPUTINT	PAGEON	SPRINTLN
CLOSECAP	FAPPEND	INPUTMONEY	POKEB	STARTDISP
CLREOL	FCLOSE	INPUTSTR	POKEDW	STOP
CLS	FCREATE	INPUTTEXT	POKEW	TOKENIZE
COLOR	FGET	INPUTTIME	POP	VARADDR
CONFFLAG	FOPEN	INPUTYN	PRINT	VAROFF
CONFUNFLAG	FOR	JOIN	PRINTLN	VARSEG
DBGLEVEL	FORWARD	KBDCHKOFF	PROMPTSTR	WAIT
DEC	FPUT	KBDCHKON	PUSH	WAITFOR
DEFCOLOR	FPUTLN	KBDFILE	PUTUSER	WHILE
DELAY	FPUTPAD	KBDSTUFF	QUEST	WRUNET
DELETE	FRESHLINE	LET	RDUNET	WRUSYS
DELUSER	FREWIND	LOG	RDUSYS	
DIR	GETTOKEN	MESSAGE	RENAME	
DISPFILE	GETUSER	MORE	RESETDISP	

Expressions

An expression in PPL can take just about any form imaginable. It consists of one or more constants, variables (see Variable Declaration Statements), functions (which take zero, one or more arguments), or sub-expressions, all of which are separated by PPL operators. Although most statements and functions in PPL expect expressions of a specific type as arguments, you need not pass it an expression of the correct type; PPL will automatically convert from one type to another when it needs to. Here are a few sample expressions:

```
' Define a few variables to hold expression results
INTEGER i, j, k
STRING s, t, u

' Single term expressions
' (All expressions here are to the right of the =)
LET i = 2
LET j = 3
LET k = 4
LET s = "STRING"

' Complex expressions
LET i = i*j*k+2*i+3*j+k/2-5
LET j = i*j*(k+2)*(i+3)*(j+k)/(2-5)
LET k = (RANDOM(5)+1)*5+ABS(j)
LET t = CHR(i%256)
LET u = s+" "+t
```

Constants

PPL supports both user defined constants and pre-defined constants. User defined constants may be any of the following:

$#.##	A **MONEY** constant (dollar sign followed by optional dollars followed by decimal point followed by cents; # = 0-9)
##h	An **INTEGER** hexadecimal constant (a decimal digit followed by zero, one or more hexadecimal digits followed by an H; # = 0-9 & A-F)
##d	An **INTEGER** decimal constant (one or more decimal digits followed by a D; # = 0-9)
##o	An **INTEGER** octal constant (one or more octal digits followed by an O; # = 0-7)
##b	An **INTEGER** binary constant (one or more binary digits followed by a B; # = 0-1)
+/-##	An **INTEGER** constant (an optional plus or minus sign followed by one or more decimal digits; # = 0-9)
"X"	A **STRING** constant (a double quote followed by displayable text followed by another double quote; X = any displayable text)
@X##	An **INTEGER** @X constant (a commercial at sign followed by an X followed by two hexadecimal digits; # = 0-9 & A-F)

The following predefined constant labels are also available. Their values and uses will be defined in the **PPL Reference** section.

AUTO	FNS	HIGHASCII	NOCLEAR	S_DR
BELL	F_EXP	LANG	O_RD	S_DW
DEFS	F_MW	LFAFTER	O_RW	TRUE
ECHODOTS	F_REG	LFBEFORE	O_WR	UPCASE
ERASELINE	F_SEL	LOGIT	SEC	WORDWRAP
FALSE	F_SYS	LOGITLEFT	STACKED	YESNO
FCL	GRAPH	NC	S_DB	
FIELDLEN	GUIDE	NEWLINE	S_DN	

Functions

PPL supports many functions which may be used by the programmer in expressions. Here is a list of valid PPL functions. As with the predefined constants, their return values and uses will be documented in the **PPL Reference** section.

ABORT	HELPPATH	MONTH	REGDS	U_BDL
ABS	HOUR	NOCHAR	REGDX	U_BDLDAY
AND	I2S	NOT	REGES	U_BUL
ANSION	INKEY	ONLOCAL	REGF	U_FDL
ASC	INSTR	OR	REGSI	U_FUL
B2W	KINKEY	PAGESTAT	REPLACE	U_INCONF
CALLID	LANGEXT	PCBDAT	RIGHT	U_LDATE
CALLNUM	LEFT	PCBNODE•	RTRIM	U_LDIR
CARRIER	LEN	PEEKB	S2I	U_LOGONS
CCTYPE	LOGGEDON	PEEKDW	SCRTEXT	U_LTIME
CDON	LOWER	PEEKW	SEC	U_MSGRD
CHR	LTRIM	PPENAME	SHOWSTAT	U_MSGWR
CURCOLOR	MASK_ALNUM	PPEPATH	SLPATH	U_NAME
CURCONF	MASK_ALPHA	PSA	SPACE	U_PWDHIST
CURSEC	MASK_ASCII	RANDOM	STRING	U_PWDLC
DATE	MASK_FILE	READLINE	STRIP	U_PWDTC
DAY	MASK_NUM	REGAH	STRIPATX	U_RECNUM
DBGLEVEL	MASK_PATH	REGAL	SYSOPSEC	U_STAT
DEFCOLOR	MASK_PWD	REGAX	TEMPPATH	U_TIMEON
DOW	MAXNODE	REGBH	TIME	VALCC
EXIST	MGETBYTE	REGBL	TIMEAP	VALDATE
FERR	MID	REGBX	TOKCOUNT	VALTIME
FILEINF	MIN	REGCF	TOKENSTR	VER
FMTCC	MINKEY	REGCH	TRIM	XOR
GETENV	MINLEFT	REGCL	UPPER	YEAR
GETTOKEN	MINON	REGCX	UN_CITY	YESCHAR
GETX	MKADDR	REGDH	UN_NAME	
GETY	MKDATE	REGDI	UN_OPER	
GRAFMODE	MODEM	REGDL	UN_STAT	

Sub-Expressions

A sub-expression is simply any valid PPL expression surrounded by parentheses. For example, this is an expression:

```
7+6-5*4/3%2
```

To make it into a sub-expression, surround it with parentheses like this:

```
(7+6-5*4/3%2)
```

This sub-expression could be used in yet another expression:

```
PRINTLN 2*(7+6-5*4/3%2)*RANDOM(4)
```

Operators

PPL supports a full set of operators in addition to the functions listed previously. They are:

Operator	Function
(	Starts a sub-expression; requires a) to terminate Example: `3*(2+1)` (result is 9, not 7)
)	Ends a sub-expression Example: `3*(2+1)` (result is 9, not 7)
^	Returns the result of raising a number to a specified power Expects and returns type **INTEGER** Example: `3^2` (result is 9)
*	Returns the product of two numbers Expects and returns type **INTEGER** Example: `3*2` (result is 6)
/	Returns the quotient of two numbers Expects and returns type **INTEGER** Example: `9/4` (result is 2)
%	Returns the remainder of two numbers Expects and returns type **INTEGER** Example: `9%4` (result is 1)
+	Returns the sum of two numbers or a string concatenated to another Expects and returns type **INTEGER** or **STRING** Example: `1+2` (result is 3) Example: `"String plus "+"String"` (result is "String plus String")
-	Returns the difference between two numbers Expects and returns type **INTEGER** Example: `3-2` (result is 1)
=	Returns **TRUE** if two values are equal Expects any type; returns type **BOOLEAN** Example: `3 = 3` (result is **TRUE**) Example: `"String" = "STRING"` (result is **FALSE**)
<>	Returns **TRUE** if two values are not equal Expects any type; returns type **BOOLEAN** Example: `3 <> 3` (result is **FALSE**) Example: `"String" <> "STRING"` (result is **TRUE**)

<	Returns **TRUE** if a value is less than another Expects any type; returns type **BOOLEAN** Example: `2 < 3` (result is **TRUE**) Example: `"STRING" < "STRING"` (result is **FALSE**)
<=	Returns **TRUE** if a value is less than or equal to another Expects any type; returns type **BOOLEAN** Example: `2 <= 3` (result is **TRUE**) Example: `"STRING" <= "STRING"` (result is **TRUE**)
>	Returns **TRUE** if a value is greater than another Expects any type; returns type **BOOLEAN** Example: `2 > 3` (result is **FALSE**) Example: `"STRING" > "STRING"` (result is **FALSE**)
>=	Returns **TRUE** if a value is greater than or equal to another Expects any type; returns type **BOOLEAN** Example: `2 >= 3` (result is **FALSE**) Example: `"STRING" >= "STRING"` (result is **TRUE**)
!	Returns the logical not of a **BOOLEAN** value Expects and returns type **BOOLEAN** Example: `!TRUE` (result is **FALSE**)
&	Returns the logical and of two **BOOLEAN** values Expects and returns type **BOOLEAN** Example: `TRUE & FALSE` (result is **FALSE**)
|	Returns the logical or of two **BOOLEAN** values Expects and returns type **BOOLEAN** Example: `TRUE | FALSE` (result is **TRUE**)

PPL operators have a precedence between one and six that determines which operators get processed first. A precedence of one gets processed first, six gets processed last.

Precedence	Operators
1	`( )`
2	`^`
3	`* / %`
4	`+ -`
5	`= <> < <= > >=`
6	`! & |`

Binary operators expect both the left and right operands to be of the same type. If they are not then appropriate type conversions will be performed automatically.

7

PPL Reference

PPL Reference

Lists by Type

PPL is composed of basically five different token types. They are constants, functions, statements, types, and variables.

Constant List

AUTO	FNS	HIGHASCII	NOCLEAR	S_DR
BELL	F_EXP	LANG	O_RD	S_DW
DEFS	F_MW	LFAFTER	O_RW	TRUE
ECHODOTS	F_REG	LFBEFORE	O_WR	UPCASE
ERASELINE	F_SEL	LOGIT	SEC	WORDWRAP
FALSE	F_SYS	LOGITLEFT	STACKED	YESNO
FCL	GRAPH	NC	S_DB	
FIELDLEN	GUIDE	NEWLINE	S_DN	

Function List

ABORT	HELPPATH	MONTH	REGDS	U_BDL
ABS	HOUR	NOCHAR	REGDX	U_BDLDAY
AND	I2S	NOT	REGES	U_BUL
ANSION	INKEY	ONLOCAL	REGF	U_FDL
ASC	INSTR	OR	REGSI	U_FUL
BZW	KINKEY	PAGESTAT	REPLACE	U_INCONF
CALLID	LANGEXT	PCBDAT	RIGHT	U_LDATE
CALLNUM	LEFT	PCBNODE	RTRIM	U_LDIR
CARRIER	LEN	PEEKB	S2I	U_LOGONS
CCTYPE	LOGGEDON	PEEKDW	SCRTEXT	U_LTIME
CDON	LOWER	PEEKW	SEC	U_MSGRD
CHR	LTRIM	PPENAME	SHOWSTAT	U_MSGWR
CURCOLOR	MASK_ALNUM	PPEPATH	SLPATH	U_NAME
CURCONF	MASK_ALPHA	PSA	SPACE	U_PWDHIST
CURSEC	MASK_ASCII	RANDOM	STRING	U_PWDLC
DATE	MASK_FILE	READLINE	STRIP	U_PWDTC
DAY	MASK_NUM	REGAH	STRIPATX	U_RECNUM
DBGLEVEL	MASK_PATH	REGAL	SYSOPSEC	U_STAT
DEFCOLOR	MASK_PWD	REGAX	TEMPPATH	U_TIMEON
DOW	MAXNODE	REGBH	TIME	VALCC
EXIST	MGETBYTE	REGBL	TIMEAP	VALDATE
FERR	MID	REGBX	TOKCOUNT	VALTIME
FILEINF	MIN	REGCF	TOKENSTR	VER
FMTCC	MINKEY	REGCH	TRIM	XOR
GETENV	MINLEFT	REGCL	UPPER	YEAR
GETTOKEN	MINON	REGCX	UN_CITY	YESCHAR
GETX	MKADDR	REGDH	UN_NAME	
GETY	MKDATE	REGDI	UN_OPER	
GRAFMODE	MODEM	REGDL	UN_STAT	

Statement List

ADJTIME	DISPSTR	GOODBYE	MPRINT	RESTSCRN
ANSIPOS	DISPTEXT	GOSUB	MPRINTLN	RETURN
BACKUP	DOINTR	GOTO	NEWLINE	SAVESCRN
BLT	DTROFF	HANGUP	NEWLINES	SENDMODEM
BROADCAST	DTRON	IF	NEWPWD	SHELL
BYE	ELSE	INC	NEXT	SHOWOFF
CALL	ELSEIF	INPUT	OPENCAP	SHOWON
CDCHKOFF	END	INPUTCC	OPTEXT	SOUND
CDCHKON	ENDIF	INPUTDATE	PAGEOFF	SPRINT
CHAT	ENDWHILE	INPUTINT	PAGEON	SPRINTLN
CLOSECAP	FAPPEND	INPUTMONEY	POKEB	STARTDISP
CLREOL	FCLOSE	INPUTSTR	POKEDW	STOP
CLS	FCREATE	INPUTTEXT	POKEW	TOKENIZE
COLOR	FGET	INPUTTIME	POP	VARADDR
CONFFLAG	FOPEN	INPUTYN	PRINT	VAROFF
CONFUNFLAG	FOR	JOIN	PRINTLN	VARSEG
DBGLEVEL	FORWARD	KBDCHKOFF	PROMPTSTR	WAIT
DEC	FPUT	KBDCHKON	PUSH	WAITFOR
DEFCOLOR	FPUTLN	KBDFILE	PUTUSER	WHILE
DELAY	FPUTPAD	KBDSTUFF	QUEST	WRUNET
DELETE	FRESHLINE	LET	RDUNET	WRUSYS
DELUSER	FREWIND	LOG	RDUSYS	
DIR	GETTOKEN	MESSAGE	RENAME	
DISPFILE	GETUSER	MORE	RESETDISP	

Type List

BOOLEAN	INTEGER	STRING
DATE	MONEY	TIME

Variable List

U_CLS	U_LONGHDR	U_PAGELEN	U_CITY	U_PWD
U_DEF79	U_SCROLL	U_SEC	U_CMNT1	U_TRANS
U_EXPERT	U_EXPDATE	U_ADDR(5)	U_CMNT2	U_VER
U_FSE	U_PWDEXP	U_ALIAS	U_HVPHONE	
U_FSEP	U_EXPSEC	U_BDPHONE	U_NOTES(4)	

ABORT() *Function*

Function

Returns a flag indicating whether or not the user has aborted the display of information.

Syntax

ABORT()

No arguments are required

Return Type & Value

BOOLEAN If the user has aborted the display of information by answering no to a MORE? prompt or by hitting ^K or ^X display, this function returns **TRUE**. Otherwise **FALSE** is returned.

Remarks

Unless specifically disabled, the user can abort any display at any time by hitting ^K or ^X or by answering no to a MORE? prompt. If the user does this, PCBoard will not display any further information until the display is reset via the **RESETDISP** statement. This function should be checked occasionally during long displays of information to determine if the user wants to abort. If the function returns **TRUE**, you should stop printing information and continue with the next part of the program after using **RESETDISP**.

Examples

```
INTEGER I
STARTDISP FCL
' While the user has not aborted, continue
WHILE (!ABORT()) DO
  PRINTLN "I is equal to ",I
  INC I
ENDWHILE
RESETDISP
```

See Also

RESETDISP *Statement*, **STARTDISP** *Statement*

ABS() *Function*

Function

Returns the absolute value of an integer expression.

Syntax

ABS(iexp)

iexp	Any integer expression.

Return Type & Value

INTEGER	If **iexp** is greater than or equal to 0, this function returns **iexp**. Otherwise this function returns **-iexp**.

Remarks

The most significant use of the absolute value function is to determine the difference between two values. For example, you may need to know in a program the difference between 8 and 13. Normal subtraction would yield a result of -5 (8-13). You don't need the mathematical difference though, you need the logical difference between the two integers. The absolute value function will return that. In other words, while 8-13 is -5, **ABS(8-13)** is 5, which may be a more desirable result in many cases. Also, it is easier to code and understand than this:

```
INTEGER D
LET D = 8-13
IF (D < 0) LET D = -D
```

Examples

```
INTEGER num
' Loop while num is < 6 or num > 10
' ... ABS(4-8)=4 ABS( 5-8)=3 ABS( 6-8)=2 ABS( 7-8)=1
'     ABS(9-8)=1 ABS(10-8)=2 ABS(11-8)=3 ABS(12-8)=4 ...
WHILE (ABS(num-8) > 2) DO
  PRINTLN "Enter a number from 6 to 10:"
  INPUT "Number",num
ENDWHILE

INTEGER i, r
' Generate 10 random numbers from -5 to 5
' Print each number and it's absolute value
FOR i = 1 TO 10
  LET r = RANDOM(10)-5
  PRINTLN "The absolute value of ",r," is ",ABS(r)
NEXT
```

See Also

RANDOM() ***Function***

ADJTIME *Statement*

Function

Adjust the users time up or down.

Syntax

```
ADJTIME minutes
```

`minutes` An integer expression containing the number of minutes that the users time left should be adjusted by. A value greater than 0 will add time; a value less than 0 will deduct time.

Remarks

Use this statement to reward (or penalize) the user with more (or less) time based on any condition or event you wish. However, the added/deducted time is only applied to the current call. It will not be remembered after the caller hangs up, except that it will be reflected in the time online today. For example, if a caller has a normal daily limit of 30 minutes and you add 15 minutes, they can stay online for up to 45 minutes. If they only stay online for 15 minutes and hangup, they will only have 15 minutes left at the beginning of the next call, not 30; the added time isn't saved. If they stay online for 40 minutes though, it will have given them their entire normal allotment of time plus 10 of the 15 extra minutes. If they try to call back to use their last 5 minutes they will not be able to because PCBoard will see that they've used their entire daily time limit plus 10 minutes. The last 5 minutes wasn't saved. Note that time may only be added if the users time has not been adjusted for an event. Time may always be subtracted.

Examples

```
STRING yn
INPUTYN "Do you wish to gamble 5 minutes for 10",yn,@X0E
IF (yn = YESCHAR()) THEN
  IF (RANDOM(1) = 1) THEN
    PRINTLN "You *WON*!  10 extra minutes awarded . . ."
    ADJTIME 10
  ELSE
    PRINTLN "You lost.  Sorry, but I have to take 5 minutes now .
    ADJTIME -5
  ENDIF
ELSE
  PRINTLN "Chicken!  :)"
ENDIF
```

See Also

MINLEFT() ***Function*****, MINON()** ***Function*****, U_TIMEON()** ***Function***

AND() *Function*

Function

Calculate the bitwise AND of two integer arguments.

Syntax

AND(iexp1,iexp2)

iexp1	Any integer expression.
iexp2	Any integer expression.

Return Type & Value

INTEGER	Returns the bitwise AND of **iexp1** and **iexp2**.

Remarks

This function may be used to clear selected bits in an integer expression by ANDing the expression with a mask that has the bits to clear set to 0 and the bits to ignore set to 1. Another use is to calculate the remainder of a division operation by a power of two by ANDing the dividend with the power of two minus one.

Examples

```
' Clear the high word, keeping only the low word
PRINTLN "07FFFFFFFh AND 0FFFFh = ",AND(07FFFFFFFh,0FFFFh)
' In this case 123%16 = AND(123,15) (15 = 1111b)
PRINTLN "The remainder of 123 divided by 16 is ",AND(123,1111b)
```

See Also

NOT() *Function*, **OR**() *Function*, **XOR**() *Function*

ANSION() *Function*

Function

Report the status of ANSI availability with the current caller.

Syntax

```
ANSION()
```

No arguments are required

Return Type and Value

BOOLEAN If the caller can support ANSI then **TRUE** is returned, otherwise **FALSE** is returned.

Remarks

This function will return **TRUE** if the caller has ANSI capabilities. This could have been determined one of two ways. If the user answered yes to the `Do you want graphics?` prompt this function will return **TRUE**. If the user answered no, there is still a chance that the user has ANSI capabilities; PCBoard will interrogate the remote computer to find out if ANSI is available. If it is, this function will return **TRUE**. Finally, if the user answered no and PCBoard was unable to detect ANSI at login this function will return **FALSE**. There is still a chance that the user could support ANSI but the only safe approach at this point is to assume that there is no ANSI available.

Examples

```
IF (ANSION()) PRINTLN "You have ANSI support available!"
```

See Also

ANSIPOS *Statement*, **BACKUP** *Statement*, **FORWARD** *Statement*, **GETX()** *Function*, **GETY()** *Function*, **GRAFMODE()** *Function*

ANSIPOS *Statement*

Function

Position the cursor anywhere on the screen using an ANSI positioning escape sequence.

Syntax

```
ANSIPOS xpos, ypos
```

xpos	An integer expression with the screen column (x position) in which to place the cursor. Valid columns are 1 through 80.
ypos	An integer expression with the screen row (y position) on which to place the cursor. Valid rows are 1 through 23.

Remarks

This statement will position the cursor to the specified (X,Y) coordinate on the screen but only if the current caller has ANSI support. If you are writing a program that will require ANSI positioning, check the value of the **ANSION()** function. If ANSI is not available, this statement will be ignored.

Examples

```
CLS
IF (ANSION()) THEN
  ANSIPOS 1,1
  PRINTLN "This starts at (1,1)"
  ANSIPOS 3,3
  PRINTLN "This starts at (3,3)"
  ANSIPOS 2,2
  PRINTLN "And *THIS* starts at (2,2)"
ENDIF
```

See Also

ANSION() *Function*, **BACKUP** *Statement*, **FORWARD** *Statement*, **GETX()** *Function*, **GETY()** *Function*, **GRAFMODE()** *Function*

ASC() *Function*

Function

Converts a character to it's ASCII code.

Syntax

ASC(sexp)

sexp	Any string expression.

Return Type & Value

INTEGER	Returns the ASCII code of the first character of **sexp** (1-255) or 0 if sexp is an empty string.

Remarks

In other languages (such as BASIC) you can have any of the 256 possible ASCII codes (0-255) in a string. In PPL you are limited to 255 codes (1-255) because ASCII 0 is used to terminate strings and can't appear in the middle of a string. So, if you ever get a 0 returned from this function, it is because you passed it an empty string.

Examples

```
PRINTLN "The ASCII code for S is ",ASC("S")
' Convert a lowercase s to uppercase
STRING s
LET s = CHR(ASC("s")-ASC("a")+ASC("A"))
```

See Also

CHR() ***Function***

AUTO *Constant*

Function

Set the auto answer flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

8192 = 10000000000000b = 20000o = 2000h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to automatically answer themselves if left alone for 20 seconds. It can be especially useful if you are writing a program that should work with automated systems; use the **AUTO** constant and the question will automatically be answered after 20 seconds just in case the automation system doesn't know what to do with it.

Examples

```
STRING ans
LET ans = NOCHAR()
INPUTSTR "Run program now",ans,@X0E,1,"",AUTO+YESNO
IF (ans = NOCHAR()) END
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

B2W() *Function*

Function

Convert two byte-sized arguments into a single word-sized argument.

Syntax

```
B2W(iexp1,iexp2)
```

iexp1	Any integer expression with a value between 000h and 0FFh.
iexp2	Any integer expression with a value between 000h and 0FFh.

Return Type & Value

INTEGER Returns a word-sized value between 00000h and 0FFFFh. The return value is computed by the following expression: **iexp1+iexp2***0100h.

Remarks

It is sometimes necessary to combine two bytes together to form a word. This function simplifies that process, and speeds it up a little as well by doing it internally instead of requiring you to perform the arithmetic yourself. It can be especially useful when used with the **DOINTR** statement.

Examples

```
' This line will display 25 *'s at the current screen position
' NOTES:  10h is the Video BIOS interrupt
'         B2W(ASC("*"),09h) is the char to print and the service number
'         0007h is video page 0, attribute 7
'         25 is the number of characters to print
'         All others are 0 and not needed for this function
DOINTR 10h,B2W(09h,ASC("*")),0007h,25,0,0,0,0,0
```

See Also

DOINTR *Statement*

BACKUP *Statement*

Function

Move the cursor backward a specified number of columns.

Syntax

```
BACKUP numcols
```

`numcols` An integer expression of the number of columns to move backward. Valid values are 1 through 79.

Remarks

This statement will move the cursor backward, nondestructively, a specified number of columns. It will work with or without ANSI. If ANSI is available (as reported by the **ANSION()** function) then it will use an ANSI positioning command; otherwise it will use the specified number of backspace characters. ANSI is usually faster, but backspace characters will get the job done. Note that you cannot use this function to move beyond column 1; to do so would require ANSI. So, if the cursor is already in column 1 this statement will have no effect. And if the cursor is in column 80 the maximum you could move backward would be 79 (column 80 - 79 columns = column 1).

Examples

```
PRINT "Rolling dice -- "
FOR i = 1 TO 10
  LET d1 = RANDOM(5)+1
  LET d2 = RANDOM(5)+1
  PRINT d1,"-",d2
  BACKUP 3
NEXT
NEWLINE
```

See Also

ANSION() *Function*, **ANSIPOS** *Statement*, **FORWARD** *Statement*, **GETX()** *Function*, **GETY()** *Function*, **GRAFMODE()** *Function*

BELL *Constant*

Function

Set the bell flag in a **DISPTEXT** statement.

Value

2048 = 100000000000b = 4000o = 800h

Remarks

The **DISPTEXT** statement has the ability to sound a bell before displaying the actual text of a prompt. This is useful when you want to get the users attention when displaying information. It sends a ^G (ASCII 7) character to the remote caller and sounds the alarm on the local computer running PCBoard (unless the alarm has been toggled off). It is the responsibility of the users terminal software to support the ^G.

Examples

```
' Get the users attention and display the closed board prompt
DISPTEXT 11,BELL+LFAFTER+LFBEFORE
```

See Also

DISPTEXT *Statement*

BLT *Statement*

Function

Display a specified bulletin number to the user.

Syntax

```
BLT bltnum
```

`bltnum` The number of the bulletin to display to the user. Valid values are 1 through the number of bulletins available.

Remarks

This statement will display a specified bulletin number to the user. The BLT.LST file for the current conference will be searched for the bulletin. If the bulletin number is invalid (less than 1 or greater than the highest bulletin number defined) then nothing will be displayed.

Examples

```
INTEGER num
INPUT "Bulletin to view",num
BLT num
```

See Also

DIR *Statement*, **JOIN** *Statement*, **QUEST** *Statement*

BOOLEAN *Type*

Function

Declare one or more variables of type boolean.

Syntax

```
BOOLEAN var|arr(s[,s[,s]])[,var|arr(s[,s[,s]])]
BOOLEAN var|arr(s[,s[,s]])[,var|arr(s[,s[,s]])]
```

`var`	The name of a variable to declare. Must start with a letter [A-Z] which may be followed by letters, digits [0-9] or the underscore [_]. May be of any length but only the first 32 characters are used.
`arr`	The name of an array variable to declare. The same naming conventions as **var** are used.
`s`	The size (0-based) of an array variable dimension. Any constant integer expression is allowed.

Remarks

BOOLEAN variables can hold two values: 1 or 0 (**TRUE** or **FALSE**). It is stored internally as a one byte unsigned character. If a **BOOLEAN** is assigned to or from an **INTEGER** type then the value 1 or 0 is assigned. If a **BOOLEAN** is assigned to a **STRING** type then it is automatically converted to a string (either "1" or "0"). If a **STRING** is assigned to a **BOOLEAN** then the value of the string will be used; a 0 value will be taken as is, another other value will be converted to 1. All other types, when assigned to or from a **BOOLEAN**, will be converted to an **INTEGER** first before being assigned to or from the **BOOLEAN** type.

Examples

```
BOOLEAN flag, bit, isPrime(100), leapYears(2079-1900)
```

See Also

DATE *Type*, **INTEGER** *Type*, **MONEY** *Type*, **STRING** *Type*, **TIME** *Type*

BROADCAST *Statement*

Function

Broadcast a single line message to a range of nodes.

Syntax

BROADCAST lonode,hinode,message

lonode	An integer expression containing the low node number to which the message should be broadcast.
hinode	An integer expression containing the high node number to which the message should be broadcast.
message	A string expression containing the message text which should be broadcast to the specified nodes.

Remarks

This PPL statement functions the same as the PCBoard BROADCAST command, which is normally reserved for SysOp security level. This statement allows you to programatically broadcast a message to a range of nodes without giving users the ability to manually broadcast at any time they choose.

Examples

```
' Broadcast a message to a specific node
BROADCAST 5,5,"This broadcast from "+STRING(PCBNODE())
' Broadcast to a range of nodes
BROADCAST 4,8,"Stand-by for log off in 10 seconds"
' Broadcast to all nodes
BROADCAST 1,65535,"Hello all!"
```

See Also

RDUNET *Statement*, **UN_...()** *Functions*, **WRUNET** *Statement*

BYE *Statement*

Function

Log the user off as though they had typed the BYE command.

Syntax

BYE

No arguments are required

Remarks

There are multiple ways for the user to log off. One is by typing G at the command prompt. That will warn them if they have files flagged for download and (optionally) confirm their selection (incase they accidentally hit G and ENTER). Another is the BYE command. PCBoard assumes that, if the user typed BYE instead of G, that they really want to log off. didn't type it in accidentally, and want to leave **now**. The **BYE** statement does just that. It is intended to provide you PPL with the same functionality as many PCBoard prompts where G or BYE can be entered at any point.

Examples

```
STRING s
INPUT "What do you want to do",s
IF     (s = "G")   THEN GOODBYE
ELSEIF (s = "BYE") THEN BYE
ELSE                    KBDSTUFF s
ENDIF
```

See Also

DTROFF *Statement*, **DTRON** *Statement*, **GOODBYE** *Statement*, **HANGUP** *Statement*

CALL *Statement*

Function

Call (execute) another PPE file from the currently executing PPE.

Syntax

```
CALL filename
```

`filename` A string expression containing the complete path and filename of a PPE file to load and execute.

Remarks

It is sometimes convienient to load and run complete programs from other programs, similar to how you process subroutines with **GOSUB** and **RETURN**. PPL supports running both external EXE and COM files via the **SHELL** statement and other PPE files via the **CALL** statement. **CALL** allows you to load and run another PPE file, after which control returns to the first PPE at the statement after the **CALL**. The second PPE is completely separate from the first. You may pass values to the PPE by tokenizing a string with the **TOKENIZE** statement. If you need to pass values back to the first PPE, you will need to create some sort of parameter passing convention yourself. For example, you may have the second PPE create a file that has the needed information for the first PPE.

Examples

```
STRING s
INPUT "What PPE file do you wish to run",s
CALL "C:\PCB\PPE\"+s+".PPE"
```

See Also

SHELL *Statement*, **TOKENIZE** *Statement*

CALLID() *Function*

Function

Access caller ID information returned from caller ID compatible modems.

Syntax

```
CALLID()
```

No arguments are required

Return Type & Value

STRING Returns a string with caller ID information captured from a caller ID compatible modem.

Remarks

Some areas of the country have an optional service available which will send, to your modem (other other telephone device), the phone number and/or name of the person calling you. This service is known as 'Caller ID'. Some modems are starting to support it directly by capturing the information and sending it to you between the first and second rings. It can be very helpful in determining who is calling (or abusing) your BBS or for statistical purposes. This function will return the information if your modem supports it.

Examples

```
FAPPEND 1,"CID.LOG",O_WR,S_DW
FPUTLN 1,LEFT(U_NAME(),30)+CALLID()
FCLOSE 1
```

See Also

CARRIER() *Function*, **MODEM()** *Function*

CALLNUM() *Function*

Function

Returns the current caller number.

Syntax

CALLNUM()

No arguments are required

Return Type & Value

INTEGER Returns the caller number of the user online.

Remarks

Everytime a user logs on to the system the system caller number is incremented. This function will return the caller number for use in your PPL applications. It is kept in the main conference MSGS file. Note that the number is not incremented until after the user has completely logged on to the system so you should generally wait until **LOGGEDON()** reports **TRUE** before using this function.

Examples

```
IF (LOGGEDON() & (CALLNUM() = 1000000)) THEN
  PRINTLN "@BEEP@CONGRATULATIONS!!!"
  PRINTLN "@BEEP@YOU ARE THE 1,000,000th CALLER!!!"
  PRINTLN "Upgrading security . . ."
  GETUSER
  LET U_SEC = 99
  PUTUSER
ENDIF
```

See Also

LOGGEDON() *Function*, **ONLOCAL()** *Function*, **U_LOGONS()** *Function*

CARRIER() *Function*

Function

Determine what speed the current caller is connected at.

Syntax

CARRIER()

No arguments are required

Return Type & Value

INTEGER Returns an integer with the connect speed of the current caller.

Remarks

Should the need arise for you to know what speed the caller is connected to the BBS at, this function will return that information. You should note that this information is not guaranteed accurate. It is the responsibility of the modem to tell PCBoard the actual connect speed, especially in locked port environments. For example, if your serial port is locked at 38400 bps, the modem can usually be configured to report either the actual connect speed (9600 bps, for example) or the locked port rate (38400 bps). PCBoard has to trust the modem; if the modem tells it 38400, it will have to live with that, as will your PPL applications.

Examples

```
IF (CARRIER() < 9600) THEN
  PRINTLN "Sorry, downloads are not permitted at speeds below 9600 bps"
  END
ENDIF
```

See Also

CALLID() *Function*, **MODEM()** *Function*

CCTYPE() *Function*

Function

Determine what the type of a credit card is based on the credit card number.

Syntax

```
CCTYPE(ccnum)
```

ccnum A string expression with the credit card number that is to be checked.

Return Type & Value

STRING Returns a string with the name of the card.

Remarks

PPL can be used to perform some simple credit card validation. This function returns the issuer of a credit card based on the credit card number. For example, a valid credit card number that starts with a "4" is a Visa card, so the string "VISA" will be returned. If a credit card is invalid (**VALCC**() = **FALSE**) or not recognized, then "UNKNOWN" will be returned. Other valid credit card with known types will return the appropriate string. The following card types are recognized by PPL: "DISCOVER", "CARTE BLANCHE", "DINERS CLUB", "OPTIMA", "AMERICAN EXPRESS", "VISA", and "MASTERCARD".

Examples

```
STRING s
INPUT "Credit card number",s
IF (VALCC(s)) PRINTLN LEFT(CCTYPE(s),20)," - ",FMTCC(s)
```

See Also

FMTCC() *Function*, **VALCC**() *Function*

CDCHKOFF *Statement*

Function

Turn off carrier detect checking.

Syntax

CDCHKOFF

No arguments are required

Remarks

PCBoard has built in automatic carrier detecting. What this means is that if someone should hangup unexpectedly, PCBoard will detect it, log it to the callers log, and recycle back to the call waiting screen. Some applications require the ability to turn this off; for example, a callback verification PPE needs to hangup on the caller and then do more processing. Normally, PCBoard would just recycle at that point. So, just before you start a section of code that should continue regardless of the existence of a caller online, you should issue a **CDCHKOFF** statement. It will turn off the automatic carrier checking. When you've finished the block where carrier checking has been disabled, issue the **CDCHKON** statement to turn it back on.

Examples

```
CDCHKOFF
DTROFF
DELAY 18
DTRON
SENDMODEM "ATDT1800DATAFON" ' Please don't call this number! :)
WAITFOR "CONNECT",60
CDCHKON
```

See Also

CDCHKON *Statement*, **CDON()** *Function*, **KBDCHKOFF** *Statement*, **KBDCHKON** *Statement*

CDCHKON *Statement*

Function

Turn on carrier detect checking.

Syntax

CDCHKON

No arguments are required

Remarks

PCBoard has built in automatic carrier detecting. What this means is that if someone should hangup unexpectedly, PCBoard will detect it, log it to the callers log, and recycle back to the call waiting screen. Some applications require the ability to turn this off; for example, a callback verification PPE needs to hangup on the caller and then do more processing. Normally, PCBoard would just recycle at that point. So, just before you start a section of code that should continue regardless of the existence of a caller online, you should issue a **CDCHKOFF** statement. It will turn off the automatic carrier checking. When you've finished the block where carrier checking has been disabled, issue the **CDCHKON** statement to turn it back on.

Examples

```
CDCHKOFF
DTROFF
DELAY 18
DTRON
SENDMODEM "ATDT1800DATAFON"+CHR(13) ' Please don't call this number! :)
WAITFOR "CONNECT",60
CDCHKON
```

See Also

CDCHKOFF *Statement*, **CDON()** *Function*, **KBDCHKOFF** *Statement*, **KBDCHKON** *Statement*

CDON() *Function*

Function

Determine if carrier detect is on or not.

Syntax

CDON ( )

No arguments are required

Return Type & Value

BOOLEAN Returns a boolean **TRUE** if carrier detect is on, **FALSE** otherwise.

Remarks

If you have used **CDCHKOFF** to turn off automatic carrier detect checking PCBoard will not automatically detect and act on a carrier loss. If necessary, this function can be used to detect a carrier loss condition and act appropriately.

Examples

```
IF (!CDON()) THEN
  LOG "Carrier lost in PPE "+PPENAME(),FALSE
  HANGUP
ENDIF
```

See Also

CDCHKOFF *Statement*, **CDCHKON** *Statement*

CHAT *Statement*

Function

Enter SysOp chat mode.

Syntax

CHAT

No arguments are required

Remarks

One of the features of PCBoard where change is often requested is the operator page facility. Some people want to be able to configure multiple ranges of availability per day, some want a different sounding page bell, longer or shorter page attempts, etc, etc. This statement, along with the **PAGEON** and **PAGEOFF** statements and the **PAGESTAT()** function, allow you to implement an operator page in any way desired. Of course, the SysOp may still start a chat with the F10 key or by responding to the default O (operator page) command, and the **CHAT** statement may be used at anytime (although you'll generally want to avoid starting it unless you've confirmed that the SysOp is available since the user has no way to exit it himself).

Examples

```
PAGEON
FOR i = 1 TO 10
  PRINT "@BEEP@"
  DELAY 18
  IF (KINKEY() = " ") THEN
    CHAT
    GOTO exit
  ENDIF
NEXT
:exit
```

See Also

PAGEOFF *Statement*, **PAGEOFF** *Statement*, **PAGESTAT()** *Function*

CHR() *Function*

Function

Converts an ASCII code to a character.

Syntax

CHR(iexp)

iexp	Any integer expression between 0 and 255.

Return Type & Value

STRING	Returns a one character long string for ASCII codes from 1 to 255 or an empty string for ASCII code 0.

Remarks

In other languages (such as BASIC) you can have any of the 256 possible ASCII codes (0-255) in a string. In PPL you are limited to 255 codes (1-255) because ASCII 0 is used to terminate strings and can't appear in the middle of a string. So, if you ever get an empty string from this function, it is because you passed it a 0. Any other value will return a valid string with a single character.

Examples

```
PRINTLN "The ASCII code for S is ",ASC("S")
' Convert a lowercase s to uppercase
STRING s
LET s = CHR(ASC("s")-ASC("a")+ASC("A"))
```

See Also

ASC() *Function*

CLOSECAP *Statement*

Function

Close the screen capture file.

Syntax

CLOSECAP

No arguments are required

Remarks

PCBoard has the ability to capture screen output to a file for later reference. PPL allows that same ability via the **OPENCAP** and **CLOSECAP** statements. This could be useful in a program that executes a series of commands in non-stop mode. The process could open a capture file first, execute the commands, close the capture file, then allow the user to view or download the capture file. **CLOSECAP** closes the capture file and turns off screen capturing. Also, the **SHOWON** and **SHOWOFF** statements can be used to turn on and off showing information to the screen while allowing that same information (even if not displayed or transmitted via modem) to be captured to a file. The **SHOWSTAT()** function can be used to check the current status of the **SHOWON** and **SHOWOFF** statements.

Examples

```
BOOLEAN ss
LET ss = SHOWSTAT()
SHOWOFF
OPENCAP "CAP"+STRING(PCBNODE()),ocFlag
IF (ocFlag) THEN
  DIR "U:NS"
  CLOSECAP
  KBDSTUFF "FLAG CAP"+STRING(PCBNODE())+CHR(13)
ENDIF
IF (ss) THEN
  SHOWON
ELSE
  SHOWOFF
ENDIF
```

See Also

OPENCAP *Statement*, **SHOWOFF** *Statement*, **SHOWON** *Statement*, **SHOWSTAT()** *Function*

CLREOL *Statement*

Function

Clear the current line from the cursor to the end of the line using the current color.

Syntax

CLREOL

No arguments are required

Remarks

This statement will work one of two ways depending on the mode the caller is in. If the caller is in graphics mode (or non-graphics ANSI-positioning) then PCBoard will issue the ANSI sequence to clear to the end of the line using the current color. ANSI emulators, when written properly, will echo the color all the way to column 80 of the current line when they receive this ANSI sequence. If the user is in non-graphics non-ANSI mode, PCBoard will write sufficient spaces to the display to move to column 80 and then backspace to the original position. Note that this will not clear the 80th column; the reason for this is to always keep the cursor on the current line. If the cursor wrote a space to column 80 and moved to the beginning of the next line it wouldn't be able to move back up to the previous line without ANSI (which we already know we don't have). This should be adequate for most applications.

Examples

```
COLOR @X47
CLS
PRINT "This is some sample text.  (This will disappear.)"
WHILE (INKEY() = "") DELAY 1
BACKUP 22
COLOR @X1F
CLREOL
PRINT "This goes to the end of the line."
```

See Also

CLS *Statement*

CLS *Statement*

Function

Clear the screen using the current color.

Syntax

CLS

No arguments are required

Remarks

This statement will work one of two ways depending on the mode the caller is in. If the caller is in graphics mode (or non-graphics ANSI-positioning) then PCBoard will issue the ANSI sequence to clear to the screen using the current color. If the user is in non-graphics non-ANSI mode, PCBoard will write send an ASCII 12 (form feed) character to the remote terminal in a last ditch effort to clear the remote callers screen. Many terminal programs do support this, but not all, so be aware that callers may see the ASCII 12 instead of a clear screen.

Examples

```
COLOR @X47
CLS
PRINT "This is some sample text.  (This will disappear.)"
WHILE (INKEY() = "") DELAY 1
BACKUP 22
COLOR @X1F
CLREOL
PRINT "This goes to the end of the line."
```

See Also

CLREOL *Statement*

COLOR *Statement*

Function

Change the current active color.

Syntax

```
COLOR newcolor
```

`newcolor` An integer expression containing the new color to be used by PCBoard and the remote terminal software.

Remarks

This statement will change the color in use by PCBoard and send the appropriate ANSI sequence to change color to the remote terminal software. Note that this statement will only affect a color change if the user is in graphics mode. If the user is in non-graphics mode this statement will be ignored.

Examples

```
COLOR @X47
CLS
PRINT "This is some sample text.  (This will disappear.)"
WHILE (INKEY() = "") DELAY 1
BACKUP 22
COLOR @X1F
CLREOL
PRINT "This goes to the end of the line."
```

See Also

CURCOLOR() *Function*, **DEFCOLOR** *Statement*, **DEFCOLOR()** *Function*

CONFFLAG *Statement*

Function

Set specified flags in the current conference for the current user.

Syntax

```
CONFFLAG confnum,flags
```

`confnum`	An integer expression containing the conference number to affect.
`flags`	An integer expression containing the flags to set.

Remarks

Each user on the BBS has a set of five flags for each conference that control various settings. These flags control the users registration in a conference, their expired status in a conference, whether or not they have a conference selected, whether or not they have mail waiting in a conference, and whether or not they have SysOp priviledges in a conference. Any or all of these flags may be set at once. To assist you in using this statement, five predefined constants are available to specify each flag: **F_REG**, **F_EXP**, **F_SEL**, **F_MW**, and **F_SYS**. To use these constants simply add the ones you need together.

Examples

```
' Automatically register them in selected conferences
INTEGER i
FOR i = 1 TO 10
  CONFFLAG i,F_REG+F_EXP+F_SEL
NEXT
FOR i = 11 TO 20
  CONFFLAG i,F_REG+F_SEL
NEXT
```

See Also

CONFUNFLAG *Statement*

CONFUNFLAG *Statement*

Function

Clear specified flags in the current conference for the current user.

Syntax

CONFUNFLAG confnum,flags

confnum	An integer expression containing the conference number to affect.
flags	An integer expression containing the flags to clear.

Remarks

Each user on the BBS has a set of five flags for each conference that control various settings. These flags control the users registration in a conference, their expired status in a conference, whether or not they have a conference selected, whether or not they have mail waiting in a conference, and whether or not they have SysOp priviledges in a conference. Any or all of these flags may be cleared at once. To assist you in using this statement, five predefined constants are available to specify each flag: **F_REG**, **F_EXP**, **F_SEL**, **F_MW**, and **F_SYS**. To use these constants simply add the ones you need together.

Examples

```
' Automatically deregister them from selected conferences
INTEGER i
FOR i = 1 TO 10
  CONFUNFLAG i,F_REG+F_EXP+F_SEL
NEXT
FOR i = 11 TO 20
  CONFUNFLAG i,F_REG+F_SEL
NEXT
```

See Also

CONFFLAG *Statement*

CURCOLOR() *Function*

Function

Returns the color in use by the ANSI driver.

Syntax

CURCOLOR()

No arguments are required

Return Type & Value

INTEGER Returns the color code most recently issued to the ANSI driver.

Remarks

The @X code processor within PCBoard has the ability to save and restore color codes built in. PCBoard accomplishes this by saving the current color whenever it encounters an @X00 and reissuing the color change when it encounters an @XFF. Unfortunately, PCBoard will only remember one color at a time. With this function you can save and restore as many colors as your application needs.

Examples

```
INTEGER cc,x,y
COLOR @X0F
ANSIPOS 26,23
PRINT "Hit the SPACE BAR to continue"
WHILE (KINKEY() <> " ") DO
  CLS
  LET x = 1+RANDOM(57)
  LET y = 1+RANDOM(21)
  PUSH 1+RANDOM(14)
  GOSUB sub
  LET cc = CURCOLOR()
  PUSH @X0F
  GOSUB sub
  PUSH cc
  GOSUB sub
  ANSIPOS 1,y
  CLREOL
ENDWHILE

:sub
INTEGER c
POP c
COLOR c
ANSIPOS x,y
PRINT "PCBoard 15.0 with PPL!"
DELAY 18
RETURN
```

See Also

COLOR *Statement*, **DEFCOLOR** *Statement*, **DEFCOLOR()** *Function*

CURCONF() *Function*

Function

Get the current conference number.

Syntax

CURCONF ( )

No arguments are required

Return Type & Value

INTEGER Returns an integer with the current conference number.

Remarks

This function can be useful in configuring a PPL program to work in different ways in different conferences. As a quick example, we have a PPE file on Salt Air that interfaces with the enter message command. If a user is in certain conferences we prompt them for additional information that we will likely need, otherwise we skip to the normal enter message process. Of course, that's just one example; you are sure to have other uses for it.

Examples

```
IF (CURCONF() = 6) THEN ' The Salt Air beta conference is 6
  PRINTLN "You are leaving a message in the beta conference."
  PRINTLN "Be sure to leave your file date and time"
  PRINTLN "and a complete description of the problem."
ENDIF
KBDSTUFF TOKENSTR()
```

See Also

MESSAGE *Statement*, **U_NAME()** *Function*

CURSEC() *Function*

Function

Get the users current security level.

Syntax

CURSEC ( )

No arguments are required

Return Type & Value

INTEGER Returns an integer with the current security level of the user.

Remarks

Although the users primary security level may be accessed via the **U_SEC** variable after using the **GETUSER** statement, it is often necessary to know the users security level right now after taking into account whether or not they have expired access, additional security from joining a specific conference, or additional security from the keyboard. This function will take all variables into account and return the current 'logical' security level.

Examples

```
IF (CURSEC() < 100) PRINTLN "Insufficient security!"
```

See Also

U_EXPSEC *Variable*, **U_SEC** *Variable*

DATE *Type*

Function

Declare one or more variables of type date.

Syntax

```
DATE var|arr(s[,s[,s]])[,var|arr(s[,s[,s]])]
```

`var`	The name of a variable to declare. Must start with a letter [A-Z] which may be followed by letters, digits [0-9] or the underscore [_]. May be of any length but only the first 32 characters are used.
`arr`	The name of an array variable to declare. The same naming conventions as **var** are used.
`s`	The size (0-based) of an array variable dimension. Any constant integer expression is allowed.

Remarks

DATE variables are stored as julian dates. Valid dates are 0 (a special case to represent an invalid date) and 1 (1 JAN 1900) through 36524 (31 DEC 1999) through 65535 (5 JUN 2079). It is stored internally as a two byte unsigned integer. If a **DATE** is assigned to or from an **INTEGER** type then the julian date (0-65535) is assigned. If a **DATE** is assigned to a **STRING** type then it is automatically converted to the following format: "MM/DD/YY", where MM is the two digit month (01-12), DD is the two digit day of the month (01-31), and YY is the two digit year (00-99). If a foreign language is in use that uses a different date format (for example, "DD/MM/YY" or "YY.MM.DD") then that will be taken into account. If a **STRING** is assigned to a **DATE** then PPL will do it's best to convert the string back to the appropriate julian date. However, dates before 1980 will not be handled correctly because only a two digit year is used in strings. All other types, when assigned to or from a **DATE**, will be converted to an **INTEGER** first before being assigned to or from the **DATE** type.

Examples

```
DATE dob, today, range(2), leapYears(50)
```

See Also

BOOLEAN *Type*, **INTEGER** *Type*, **MONEY** *Type*, **STRING** *Type*, **TIME** *Type*

DATE() *Function*

Function

Get today's date.

Syntax

DATE ( )

No arguments are required

Return Type & Value

DATE Returns a date for today.

Remarks

The date returned is represented internally in a julian format (the number of days since January 1, 1900). It may be used as is (for display, storage or as an argument to another function or statement) or assigned to an integer for arithmetic purposes.

Examples

```
PRINTLN "Today is ",DATE()
```

See Also

DAY() *Function*, **DOW**() *Function*, **MKDATE**() *Function*, **MONTH**() *Function*, **TIME**() *Function*, **YEAR**() *Function*

DAY() *Function*

Function

Extracts the day of the month from a date.

Syntax

DAY(dexp)

dexp	Any date expression.

Return Type & Value

INTEGER	Returns the day of the month from the specified date expression (**dexp**). Valid return values are from 1 to 31.

Remarks

This function allows you to extract a particular piece of information about a **DATE** value, in this case the day of the month of the date.

Examples

```
PRINTLN "Today is:  ",DAY(DATE())
```

See Also

DATE() *Function*, **DOW()** *Function*, **MONTH()** *Function*, **YEAR()** *Function*

DBGLEVEL *Statement*

Function

Set a new debug level for PCBoard.

Syntax

```
DBGLEVEL level
```

`level` An integer expression with the new debug level.

Remarks

PCBoard supports an internal variable that allows debug information to be written to the callers log. Level 0 specified no debug information. Levels 1 through 3 specify different (increasing) levels of debug information. It can also be useful for debugging your PPL programs. This statement allows you to change the PCBoard debug level on the fly without the need to have the SysOp exit and change it in the BOARD.BAT file.

Examples

```
INTEGER newlvl
INPUT "New level",newlvl
NEWLINE
DBGLEVEL newlvl
```

See Also

DBGLEVEL() ***Function*****, LOG** ***Statement***

DBGLEVEL() *Function*

Function

Returns the debug level in effect.

Syntax

DBGLEVEL()

No arguments are required

Return Type & Value

INTEGER Returns the current debug level.

Remarks

PCBoard supports an internal variable that allows debug information to be written to the callers log. Level 0 specified no debug information. Levels 1 through 3 specify different (increasing) levels of debug information. It can also be useful for debugging your PPL programs. Using this function you can tie your debug information to a specified debug level of your choosing.

Examples

```
IF (DBGLEVEL() = 1) LOG "Writing DEBUG info for "+PPENAME(),0
```

See Also

DBGLEVEL *Statement*, **LOG** *Statement*

DEC *Statement*

Function

Decrement the value of a variable.

Syntax

```
DEC var
```

var	The variable with the value to decrement.

Remarks

Many programs require extensive addition and subtraction, and most often, a value is increased or decreased by 1. This statement allows for a shorter, more efficient method of decreasing (decrementing) a value by 1 (**DEC i**) than subtracting 1 from a variable and assigning the result to the same variable (**LET i = i - 1**).

Examples

```
INTEGER i
PRINTLN "Countdown:"
LET i = 10
WHILE (i >= 0) DO
  PRINTLN "T minus ",i
  DEC i
ENDWHILE
```

See Also

INC *Statement*

DEFCOLOR *Statement*

Function

Change the current color to the system default color.

Syntax

DEFCOLOR

No arguments are required

Remarks

This statement will change the color in use by PCBoard to the system default and send the appropriate ANSI sequence to change color to the remote terminal software. This statement is equivalent to **COLOR DEFCOLOR()**. Note that this statement will only affect a color change if the user is in graphics mode. If the user is in non-graphics mode this statement will be ignored.

Examples

```
COLOR @X47
CLS
PRINT "This is some sample text.  (This will disappear.)"
WHILE (INKEY() = "") DELAY 1
BACKUP 22
DEFCOLOR
CLREOL
PRINT "This goes to the end of the line."
```

See Also

COLOR *Statement*, **CURCOLOR()** *Function*, **DEFCOLOR()** *Function*

DEFCOLOR() *Function*

Function

Return the system default color.

Syntax

DEFCOLOR()

No arguments are required

Return Type & Value

INTEGER Returns the system default color as defined in PCBSetup.

Remarks

This function is useful in cases where you must pass a color to a statement but you want to honor the SysOp's choice of default color for the system. In that case you cannot use the **DEFCOLOR** statement because it does not return a value that you can pass to another statement.

Examples

```
STRING yn
DEFCOLOR
CLS
LET yn = YESCHAR()
INPUTYN "Continue",yn,DEFCOLOR()
IF (yn = NOCHAR()) END
```

See Also

COLOR *Statement*, **CURCOLOR()** *Function*, **DEFCOLOR** *Statement*

DEFS *Constant*

Function

Used when no special statement parameters or flags are needed and defaults are sufficient.

Value

0 = 0b = 0o = 0h

Remarks

There are many statements that take special values as parameters or flags as an indication to do some special processing. This constant is meant to be used by itself when you do not need any other special constant value.

Examples

```
STRING ans
LET ans = NOCHAR()
INPUTSTR "Run program now",ans,@X0E,1,"YyNn",DEFS
IF (UPPER(ans) = NOCHAR()) END
```

See Also

FALSE *Constant*, **TRUE** *Constant*

DELAY *Statement*

Function

Pause execution for a specified period of clock ticks.

Syntax

```
DELAY ticks
```

`ticks` An integer expression with the number of clock ticks to pause.

Remarks

It is often desireable to wait for a precise time interval for various purposes. This function will allow you to specify an interval to delay in clock ticks. One clock tick is approximately 1/18.2 of a second. So to delay for approximately one second, you should use **DELAY 18**. The basic formula to use is (seconds to delay*18.2) and then round off to the nearest whole number. Note however that PPL doesn't support floating point arithmetic, so if you want to calculate the delay interval at run time you should use something like (seconds to delay*182)/10.

Examples

```
INTEGER i
PRINTLN "Countdown:"
LET i = 10
WHILE (i >= 0) DO
  PRINTLN "T minus ",i
  DEC i
  DELAY 18
ENDWHILE
```

See Also

SOUND *Statement*

DELETE *Statement*

Function

Delete a specified file from the disk it resides on.

Syntax

DELETE file

file A string expression with the drive, path and file name to delete.

Remarks

It is always a good idea to leave things as you found them (as much as possible). This statement allows you to delete temporary files created by your PPE with the **FCREATE/FOPEN/FAPPEND** statements.

Examples

```
INTEGER retcode
STRING  s
FCREATE 1,"TMP.LST",O_WR,S_DB
LET s = "START"
WHILE (LEN(s) > 0) DO
  LET s = ""
  PRINTLN "Enter a name or ENTER alone to quit:"
  INPUT "Name",s
  IF (LEN(s) > 0) FPUTLN 1,s
ENDWHILE
FCLOSE 1
SHELL 1,retcode,"SORT","< TMP.LST > TMP.SRT"
NEWLINE
PRINTLN "Unsorted List:"
PRINTLN "--------------"
DISPFILE "TMP.LST",DEFS
NEWLINE
PRINTLN "Sorted List:"
PRINTLN "--------------"
DISPFILE "TMP.SRT",DEFS
DELETE "TMP.LST"
DELETE "TMP.SRT"
```

See Also

EXIST() *Function*, **FILEINF()** *Function*, **READLINE()** *Function*, **RENAME** *Statement*

DELUSER *Statement*

Function

Flag the user online on the current node for deletion.

Syntax

DELUSER

No arguments are required

Remarks

This statement will set the delete user record flag to **TRUE**. This will merely flag PCBSystemManager to pack out the user during the next pack operation. If you want to make sure the user doesn't log back in before being packed out, use **GETUSER**, set his **U_SEC** and **U_EXPSEC** variables to 0, and use the **PUTUSER** statement to write the changes to the user record.

Examples

```
GETUSER
IF (U_CMNT2 = "BAD USER") THEN
  PRINTLN "Just a friendly note to say."
  PRINTLN "I hope you have a rotten day!"
  PRINTLN "Proceeding with automatic user record deletion..."
  DELUSER
  LET U_SEC = 0
  LET U_EXPSEC = 0
  PUTUSER
ENDIF
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_EXPSEC** *Variable*, **U_SEC** *Variable*

DIR *Statement*

Function

Execute the file directories command with desired sub-commands.

Syntax

DIR cmds

cmds — A string expression with any desired sub-commands for the file directory command.

Remarks

This statement will allow you to access file directories (the F command from the main menu). and any file directory sub-commands, under PPE control. Note that this statement will destroy any previously tokenized string expression. If you have string tokens pending at the time of the **DIR** statment you should save them first and then retokenize after the **DIR** statement is complete.

Examples

```
INTEGER retcode
SHOWOFF
OPENCAP "NEWFILES.LST",retcode
KBDSTUFF CHR(13)
DIR "N;S;A;NS"
CLOSECAP
SHOWON
SHELL TRUE,retcode,"PKZIP","-mex NEWFILES NEWFILES.LST"
KBDSTUFF "FLAG NEWFILES.ZIP"
```

See Also

BLT ***Statement*****, JOIN** ***Statement*****, QUEST** ***Statement***

DISPFILE *Statement*

Function

Display a specified (or alternate) file.

Syntax

```
DISPFILE file,flags
```

`file`	A string expression with the file name (or base file name) to display.
`flags`	An integer expression with alternate file flags.

Remarks

This statement will allow you to display a file to the user, and optionally to have PCBoard look for alternate security, graphics, and/or language specific files. The **flags** parameter should be 0 for no alternate searching, **GRAPH** (1) for graphics specific searching, **SEC** (2) for security specific searching, **LANG** (4) for language specific searching, or any combination thereof for multiple alternate searches simultaneously.

Examples

```
STRING s
DISPFILE "MNUA",SEC+GRAPH+LANG
INPUT "Option",s
```

See Also

DISPSTR *Statement*, **DISPTEXT** *Statement*, **OPTEXT** *Statement*

DISPSTR *Statement*

Function

Display a string of text.

Syntax

```
DISPSTR str
```

`str` A string expression to display (or %filename or !PPEfile to execute).

Remarks

This statement in intended to allow you to easily display a string to the user and provide some of the functionality of **DISPTEXT**. If the string to display begins with a percent sign and is followed by a valid file name, then the file will be displayed to the caller instead of the string. Alternately, the string could begin with an exclamation mark (and be followed by a legal file name) to run a PPE file.

Examples

```
STRING s
INPUT "String",s
DISPSTR s
LET s = "Regular string"
DISPSTR s
DISPSTR "%C:\PCB\GEN\BRDM"
DISPSTR "!"+PPEPATH()+"SUBSCR.PPE"
```

See Also

DISPFILE *Statement*, **DISPTEXT** *Statement*, **OPTEXT** *Statement*

DISPTEXT *Statement*

Function

Display a specified prompt from the PCBTEXT file.

Syntax

```
DISPTEXT rec,flags
```

`rec`	An integer expression with the PCBTEXT record number to display.
`flags`	An integer expression with display flags.

Remarks

This statement will allow you to display any prompt from the PCBTEXT file to the user according to a set of display flags. Valid display **flags** are **BELL**, **DEFS**, **LFAFTER**, **LFBEFORE**, **LOGIT**, **LOGITLEFT**, and **NEWLINE**.

Examples

```
DISPTEXT 192,BELL+NEWLINE+LOGIT
HANGUP
```

See Also

DISPFILE *Statement*, **DISPSTR** *Statement*, **OPTEXT** *Statement*

DOINTR *Statement*

Function

Generate a system interrupt.

Syntax

```
DOINTR int,ax,bx,cx,dx,si,di,flags,ds,es
```

int	An integer expression with the interrupt number to call (0 through 255).
others	Integer expressions with 16-bit register values to pass to the interrupt (**ax**, **bx**, **cx**, **dx**, **si**, and **di** are general purpose registers; **ds** and **es** are segment registers; **flags** is the 80x86 processor status register).

Remarks

This statement allows practically unlimited flexibility in PPL. It allows you to access any system service available via the BIOS (video, disk, time, etc), DOS or other third party interface (DESQview, NETBIOS, IPX/SPX, Btrieve, etc). The possibilities are limited only by your imagination. Values that are returned via register may be accessed via the **REG...()** functions. The values to pass to specific interrupts will vary by the interrupt and function desired. **WARNING!!!** The **DOINTR** function can be a very valuable tool when used wisely; it can also be extremely destructive when used improperly (either accidentally or intentionally). Use it at your own risk!

Examples

```
' Create subdirectory - DOS function 39h
INTEGER addr
STRING  path
LET path = "C:\$TMPDIR$"
VARADDR path,addr
DOINTR 21h,39h,0,0,addr%00010000h,0,0,0,addr/00010000h,0
IF (REGCF() & (REGAX() = 3)) THEN
  PRINTLN "Error:  Path not found"
ELSE IF (REGCF() & (REGAX() = 5)) THEN
  PRINTLN "Error:  Access Denied"
ELSE IF (REGCF()) THEN
  PRINTLN "Error:  Unknown Error"
ELSE
  PRINTLN "Directory successfully created..."
ENDIF
```

See Also

B2W() *Function*, **REG...()** *Functions*

DOW() *Function*

Function

Determine the day of the week of a particular date.

Syntax

DOW(dexp)

dexp	Any date expression.

Return Type & Value

INTEGER	Returns the day of the week from the specified date expression (**dexp**). Valid return values are from 0 (Sunday) to 6 (Saturday).

Remarks

This function allows you to extract a particular piece of information about a **DATE** value, in this case the day of the week of the date. The specified date can be any valid **DATE** expression.

Examples

```
PRINTLN "Today is:  ",DOW(DATE())
```

See Also

DATE() *Function*, **DAY**() *Function*, **MONTH**() *Function*, **YEAR**() *Function*

DTROFF *Statement*

Function

Turn off the serial port DTR signal.

Syntax

DTROFF

No arguments are required

Remarks

This statement turns off the serial port DTR signal. Most modems take this condition to mean that they should hang up on a caller, and this is how PCBoard uses it. This statement can be used when you need to hangup on a caller but don't want PCBoard to perform it's logoff processing. Simply turn off CD checking and keyboard timeout checking and issue the **DTROFF** statement. Do whatever processing you want, then turn DTR, keyboard timeout testing, and CD loss testing back on to allow PCBoard to recycle normally. Note that DTR should remain off for a period of time to ensure that the modem has time to react to it. Consider 1/2 second (about 9 clock ticks) a reasonable delay.

Examples

```
BOOLEAN flag
KBDCHKOFF
CDCHKOFF
DTROFF
DELAY 18
DTRON
SENDMODEM "ATDT5551212" ' Please don't really dial this number!
WAITFOR "CONNECT",flag,60
IF (!flag) SPRINLN "No connect found in 60 seconds"
CDCHKON
KBDCHKON
```

See Also

BYE *Statement*, **DTRON** *Statement*, **GOODBYE** *Statement*, **HANGUP** *Statement*

DTRON *Statement*

Function

Turn on the serial port DTR signal.

Syntax

DTRON

No arguments are required

Remarks

This statement turns on the serial port DTR signal. This statement should be used after you've used the **DTROFF** statement to hangup the modem when you need to hangup on a caller but don't want PCBoard to perform it's logoff processing. Note that DTR should remain off for a period of time, to ensure that the modem has time to react to it, before turning it back on. Consider 1/2 second (about 9 clock ticks) a reasonable delay.

Examples

```
BOOLEAN flag
KBDCHKOFF
CDCHKOFF
DTROFF
DELAY 18
DTRON
SENDMODEM "ATDT5551212" ' Please don't really dial this number!
WAITFOR "CONNECT",flag,60
IF (!flag) SPRINLN "No connect found in 60 seconds"
CDCHKON
KBDCHKON
```

See Also

BYE *Statement*, **DTROFF** *Statement*, **GOODBYE** *Statement*, **HANGUP** *Statement*

ECHODOTS *Constant*

Function

Set the echo dots flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

1 = 1b = 1o = 1h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to disable echoing of user input and instead echo dots in place of the user's input. This is useful in situations where the information being entered is confidential and shouldn't be revealed to any other party. A good example of this is the user's password.

Examples

```
STRING pwd
PROMPTSTR 148,pwd,12,MASK_PWD(),ECHODOTS+UPCASE
GETUSER
IF (pwd <> U_PWD) HANGUP
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

END *Statement*

Function

Terminate PPE execution.

Syntax

END

No arguments are required

Remarks

This statement may be used to normally terminate PPE execution at any point. If you do not have one in your program one is automatically inserted at the end of your source for you at compile time. Additionally, if your PPL application is being used as a script questionnaire, this statement will save any responses written to channel 0 to the script answer file.

Examples

```
DATE    d
INTEGER i
STRING  s
LET s = "01-20-93"
LET d = s
IF (DATE() < d) THEN
  PRINTLN "Your calendar is off!"
  END
ENDIF
LET i = d
PRINTLN "The seige continues:  Day ",DATE()-i+1
END
```

See Also

RETURN *Statement*, **STOP** *Statement*

ERASELINE *Constant*

Function

Set the erase line flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

32 = 100000b = 40o = 20h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to erase the current line after the user presses ENTER. This is the technique used by the **MORE** and **WAIT** statements to clean up after themselves.

Examples

```
STRING s
INPUTSTR "Press ENTER to continue",s,@X0E,0,"",ERASELINE
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

EXIST() *Function*

Function

Determine whether or not a file exists.

Syntax

```
EXIST(file)
```

`file`	A string expression with the drive, path and file name to check.

Return Type & Value

BOOLEAN	Returns **TRUE** if the file exists on the specified drive and path, **FALSE** otherwise.

Remarks

It is often necessary to check for the existence of a file. For example, you wouldn't want to display or otherwise process a file that doesn't exist. This function will report whether or not a specified file exists on a particular drive and path. The drive will default to the current drive and the path will default to the current directory if not specified.

Examples

```
STRING file
LET file = "NEWS."+STRING(CURNODE())
IF (EXIST(file)) DISPFILE file,0
```

See Also

DELETE *Statement*, **FILEINF()** *Function*, **READLINE()** *Function*, **RENAME** *Statement*

FALSE *Constant*

Function

To provide a named constant for the boolean false value in boolean expressions.

Value

0 = 0b = 0o = 0h

Remarks

BOOLEAN logic is based on two values: **TRUE** (1) and **FALSE** (0). The literal numeric constants 0 and 1 may be used in expressions, or you may use the predefined named constants **TRUE** and **FALSE**. They make for more readable, maintainable code and have no more overhead than any other constant value at run time.

Examples

```
BOOLEAN flag
LET flag = TRUE
WHILE (!flag) DO
  INPUTSTR "Text",s,0X0E,60,"ABCDEFGHIJKLMNOPQRSTUVWXYZ ",UPCASE
  PRINTLN s
  IF (s = "QUIT") LET flag = FALSE
ENDWHILE
```

See Also

DEFS *Constant*, **TRUE** *Constant*

FAPPEND *Statement*

Function

Open a file for append access.

Syntax

```
FAPPEND chan,file,am,sm
```

`chan`	An integer expression with the channel to use for the file (0 through 7).
`file`	A string expression with the file specification to open.
`am`	An integer expression with the desired access mode for the file.
`sm`	An integer expression with the desired share mode flags for the file.

Remarks

This statement allows a PPL application to open a file for append access. Often you need to add information to an existing file without destroying the existing information in the file. **FCREATE** completely destroys the file being opened if it already exists, and **FOPEN** will simply position you at the beginning of the file where you would overwrite data. This statement will allow you to add the necessary information to the end of a file without destroying the file or any existing information in the file. The **chan** parameter must be 0 through 7; 0 is reserved for the answer file when a PPL script questionnaire is in use but is available for all other applications. However, it is recommended you avoid channel 0 unless you really need to open 8 files at once. The **am** parameter should be one of the following constant values: **O_RD** (for read access), **O_WR** (for write access), or **O_RW** (for read/write access). Note that the **FAPPEND** statement actually requires **O_RW** access; whatever you specify doesn't really matter as it will be overridden by PPL, but you must specify it to maintain compatibility with the **FCREATE** and **FOPEN** statements. Finally, the **sm** parameter should be one of the following constants: **S_DN** (for deny none sharing), **S_DR** (for deny read sharing), **S_DW** (for deny write sharing), or **S_DB** (for deny both sharing). Also, if the file specified doesn't exist, it will automatically be created.

Examples

```
FAPPEND 1,"C:\PCB\MAIN\PPE.LOG",O_RW,S_DB
FPUTLN 1,"Ran "+PPENAME()+" on "+STRING(DATE())+" at "+STRING(TIME())
FCLOSE 1
```

See Also

FCLOSE *Statement*, **FCREATE** *Statement*, **FOPEN** *Statement*, **FREWIND** *Statement*

FCL *Constant*

Function

Forces PCBoard to count lines and provide prompts after every screen full of information.

Value

2 = 10b = 2o = 2h

Remarks

The **STARTDISP** statement takes a single argument to start displaying information in a certain format. **FCL** tells PCBoard to count lines and pause as needed during the display of information. **FNS** tells PCBoard to not stop during the display of information. **NC** instructs PCBoard to start over with the last specified mode (**FCL** or **FNS**).

Examples

```
INTEGER i
STARTDISP FCL
FOR i = 1 to 100
  PRINTLN "This is line ",i
NEXT
```

See Also

FNS *Constant*, **NC** *Constant*

FCLOSE *Statement*

Function

Close an open file.

Syntax

```
FCLOSE chan
```

`chan` An integer expression with the open channel to close (0 through 7).

Remarks

This statement should be used to close a file channel after it has been created/opened with an **FCREATE**, **FOPEN**, or **FAPPEND** statement. If you should forget to close your files by the end of your PPL application, PPL will automatically close them for you. However, if you need to process many files, it will usually be required that you open a few at a time and close them before going on to the next set of files.

Examples

```
FOPEN 1,"C:\PCB\MAIN\PPE.LOG",O_RD,S_DW
FGET 1,hdr
FCLOSE 1
IF (hdr <> "Creating PPE.LOG file . . .") THEN
  PRINTLN "Error: PPE.LOG invalid"
  END
ENDIF
```

See Also

FAPPEND *Statement*, **FCREATE** *Statement*, **FOPEN** *Statement*, **FREWIND** *Statement*

FCREATE *Statement*

Function

Create and open a file.

Syntax

FCREATE chan,file,am,sm

chan	An integer expression with the channel to use for the file (0 through 7).
file	A string expression with the file specification to create and open.
am	An integer expression with the desired access mode for the file.
sm	An integer expression with the desired share mode flags for the file.

Remarks

This statement allows a PPL application to force the creation and opening of a file, even if it already exists. Creation means that any information previously in the file (if it already exists) will be lost and you will be starting over with an empty file. The **chan** parameter must be 0 through 7; 0 is reserved for the answer file when a PPL script questionnaire is in use but is available for all other applications. However, it is recommended you avoid channel 0 unless you really need to open 8 files at once. The **am** parameter should be one of the following constant values: **O_RD** (for read access), **O_WR** (for write access), or **O_RW** (for read/write access). Note that the **FCREATE** statement forces the creation of an empt file so it doesn't make much sense to use **O_RD**, as there is nothing to read, unless you only want to create the file. Finally, the **sm** parameter should be one of the following constants: **S_DN** (for deny none sharing), **S_DR** (for deny read sharing), **S_DW** (for deny write sharing), or **S_DB** (for deny both sharing).

Examples

```
FCREATE 1,"C:\PCB\MAIN\PPE.LOG",O_WR,S_DN
FPUTLN 1,"Creating PPE.LOG file . . ."
FCLOSE 1
```

See Also

FAPPEND *Statement*, **FCLOSE** *Statement*, **FOPEN** *Statement*, **FREWIND** *Statement*

FERR() *Function*

Function

Determine whether or not an error has occurred on a channel since last checked.

Syntax

FERR(chan)

chan	An integer expression with the channel to use for the file (0 through 7).

Return Type & Value

BOOLEAN	Returns **TRUE** if an error has occurred on the specified channel since last checked, **FALSE** otherwise.

Remarks

There are many reasons why errors can occur during file processing. The drive, path or file may not exist, the end of the file may have been reached, the drive may be full, there could be errors with the hardware. and so on. For maximum reliability, you should use the function to check for errors after every file channel statement. PCBoard will automatically handle alerting the user of the error in most cases. All you need is to know that an error occurred so that you may continue processing else where or clean up and exit.

Examples

```
INTEGER i
STRING  s
FOPEN 1,"FILE.DAT",O_RD,S_DW
IF (FERR(1)) THEN
  PRINTLN "Error, exiting..."
  END
ENDIF
FGET 1,s
WHILE (!FERR(1)) DO
  INC i
  PRINTLN "Line ",RIGHT(i,3),":  ",s
  FGET 1,s
ENDWHILE
FCLOSE 1
```

See Also

FAPPEND *Statement*, **FCLOSE** *Statement*, **FCREATE** *Statement*, **FGET** *Statement*, **FOPEN** *Statement*, **FPUT** *Statement*, **FPUTLN** *Statement*, **FPUTPAD** *Statement*, **FREWIND** *Statement*

FGET *Statement*

Function

Get (read) a line from an open file.

Syntax

FGET chan,var

chan	An integer expression with the channel to read from (0 through 7).
var	The variable into which to read the next line from chan.

Remarks

This statement is to be used for reading information, a line at a time, from a file that was previously opened with read access. If there are multiple fields of information on the line then you must parse them out manually.

Examples

```
INTEGER i
STRING  s
FOPEN 1,"FILE.DAT",O_RD,S_DW
IF (FERR(1)) THEN
  PRINTLN "Error, exiting..."
  END
ENDIF
FGET 1,s
WHILE (!FERR(1)) DO
  INC i
  PRINTLN "Line ",RIGHT(i,3),":  ",s
  FGET 1,s
ENDWHILE
FCLOSE 1
```

See Also

FPUT/FPUTLN ***Statements*****, FPUTPAD** ***Statement***

FIELDLEN *Constant*

Function

Set the display field length flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

2 = 10b = 2o = 2h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to display the length of an input field using "( )" if the user has ANSI available. If you want to ensure that the user knows how wide the input area is regardless of ANSI support being available, also use the **GUIDE** constant.

Examples

```
STRING pwd
INPUTSTR "Enter id number",pwd,@X0E,4,"0123456789",FIELDLEN+GUIDE
IF (pwd <> "1234") PRINTLN "Bad id number"
```

See Also

GUIDE *Constant*, **INPUTSTR** *Statement*, **PROMPTSTR** *Statement*

FILEINF() *Function*

Function

Access a piece of information about a file.

Syntax

```
FILEINF(file,item)
```

`file`	A string expression with the path and file name to access information about.
`item`	An integer expression with the desired piece of information (1 through 9) to retrieve about the specified file.

Return Type & Value

BOOLEAN	Returns **TRUE** if the file exists or **FALSE** if file doesn't exist if **item** is 1.
DATE	Returns the date stamp of the file if item is 2.
INTEGER	Returns one of the following for the specified values of **item**:
4	The size of the file in bytes;
5	The attribute bits of the file.
STRING	Returns one of the following for the specified values of **item**:
6	The drive of the file;
7	The path of the file;
8	The base name of the file;
9	The extension of the file.
TIME	Returns the time stamp of the file if **item** is 3.

Remarks

This function is designed to return information about a file. The file date, time, size and attributes are accessible from DOS. In addition, this function can parse out the drive, path, base name and extension if needed from the complete file specification. Finally, the **EXIST**() function is duplicated in **FILEINF**().

Examples

```
STRING file
WHILE (FILEINF(file,1)) INPUT "File",file
PRINTLN " Date: ",FILEINF(file,2)
PRINTLN " Time: ",FILEINF(file,3)
PRINTLN " Size: ",FILEINF(file,4)
PRINTLN " Attr: ",FILEINF(file,5)
PRINTLN "Drive: ",FILEINF(file,6)
PRINTLN " Path: ",FILEINF(file,7)
PRINTLN " Name: ",FILEINF(file,8)
PRINTLN "  Ext: ",FILEINF(file,9)
```

See Also

DELETE *Statement*, **EXIST()** *Function*, **READLINE()** *Function*, **RENAME** *Statement*

FMTCC() *Function*

Function

Formats a credit card number for display purposes.

Syntax

FMTCC(sexp)

sexp	Any string expression.

Return Type & Value

STRING	Returns **sexp** formatted in a credit card style for display purposes.

Remarks

This function will do one of the following: one, take a 13 character string and format it as "XXXX XXX XXX XXX"; two, take a 15 character string and format it as "XXXX XXXXXX XXXXX"; three, take a 16 character string and format it as "XXXX XXXX XXXX XXXX"; or four, return the string unmodified if it is not 13, 15 or 16 characters long.

Examples

```
STRING s
WHILE (!VALCC(s)) DO
  INPUT "CC #",s
  NEWLINES 2
ENDWHILE
PRINTLN CCTYPE(s)," - ",FMTCC(s)
```

See Also

CCTYPE() ***Function*****, VALCC()** ***Function***

FNS *Constant*

Function

Forces PCBoard to not stop to provide prompts while displaying information.

Value

1 = 1b = 1o = 1h

Remarks

The **STARTDISP** statement takes a single argument to start displaying information in a certain format. **FCL** tells PCBoard to count lines and pause as needed during the display of information. **FNS** tells PCBoard to not stop during the display of information. **NC** instructs PCBoard to start over with the last specified mode (**FCL** or **FNS**).

Examples

```
INTEGER i
STARTDISP FNS
FOR i = 1 to 100
  PRINTLN "This is line ",i
NEXT
```

See Also

FCL *Constant*, **NC** *Constant*

FOPEN *Statement*

Function

Open a file.

Syntax

FOPEN chan,file,am,sm

chan	An integer expression with the channel to use for the file (0 through 7).
file	A string expression with the file specification to open.
am	An integer expression with the desired access mode for the file.
sm	An integer expression with the desired share mode flags for the file.

Remarks

This statement allows a PPL application to open a file for read and/or write access and to specify the method of sharing desired. The **chan** parameter must be 0 through 7; 0 is reserved for the answer file when a PPL script questionnaire is in use but is available for all other applications. However, it is recommended you avoid channel 0 unless you really need to open 8 files at once. The **am** parameter should be one of the following constant values: **O_RD** (for read access), **O_WR** (for write access), or **O_RW** (for read/write access). Note that the **O_RD** constant expects the file to already exist; the other open constants will create the file if it already doesn't exist. Finally, the **sm** parameter should be one of the following constants: **S_DN** (for deny none sharing), **S_DR** (for deny read sharing), **S_DW** (for deny write sharing), or **S_DB** (for deny both sharing).

Examples

```
STRING hdr
FOPEN 1,"C:\PCB\MAIN\PPE.LOG",O_RD,S_DW
FGET 1,hdr
FCLOSE 1
IF (hdr <> "Creating PPE.LOG file . . .") THEN
  PRINTLN "Error: PPE.LOG invalid"
  END
ENDIF
```

See Also

FAPPEND *Statement*, **FCLOSE** *Statement*, **FCREATE** *Statement*, **FREWIND** *Statement*

FOR/NEXT *Statement*

Function

Execute a block of statments for a range of values.

Syntax

```
FOR var = start TO end [STEP inc]
  statement(s)
NEXT
```

var	The index variable for the loop that will be set to each value.
start	Any valid PPL expression.
end	Any valid PPL expression.
inc	Any valid PPL expression. (1 if not specified).

Remarks

A **FOR** loop can consist of one or more statements. At the beginning of the loop the specified variable (**var**) is initialized to the **start** expression. It is then checked against the **end** expression. If **start** is greater than **end** (for positive values of **inc**) or less than **end** (for negative values of **inc**) then the loop terminates. Otherwise, all the statements in the loop are executed in order. At the **NEXT** statement the **inc** value (1 if not explicitly defined) is added to **var** and the loop value is retested as described above.

Examples

```
BOOLEAN p(100)
INTEGER i
FOR i = 1 TO 100 ' Initialize all to TRUE
  LET p(i) = TRUE
NEXT
LET p(1) = FALSE
FOR i = 4 TO 100 STEP 2 ' Initialize every other one to FALSE
  LET p(i) = FALSE
NEXT
```

See Also

GOSUB *Statement*, **GOTO** *Statement*, **IF/ELSEIF/ELSE/ENDIF** *Statement*, **WHILE/ENDWHILE** *Statement*, **RETURN** *Statement*

FORWARD *Statement*

Function

Move the cursor forward a specified number of columns.

Syntax

```
FORWARD numcols
```

`numcols` An integer expression of the number of columns to move forward. Valid values are 1 through 79.

Remarks

This statement will move the cursor forward, nondestructively, a specified number of columns. It will work with or without ANSI. If ANSI is available (as reported by the **ANSION()** function) then it will use an ANSI positioning command; otherwise it will re-display the specified number of characters that are already on screen. ANSI is usually faster, but re-displaying the existing information will get the job done. Note that you cannot use this function to move beyond column 80; to do so would require ANSI to move back up if necessary. So, if the cursor is already in column 80 this statement will have no effect. And if the cursor is in column 1 the maximum you could move forward would be 79 (column 1 + 79 columns = column 80).

Examples

```
PRINT "PIRNT is wrong"
DELAY 5*182/10
BACKUP 13
PRINT "RI"
FORWARD 6
PRINT "RIGHT"
DELAY 5*182/10
NEWLINE
WAIT
```

See Also

ANSION() *Function*, **ANSIPOS** *Statement*, **BACKUP** *Statement*, **GETX()** *Function*, **GETY()** *Function*, **GRAFMODE()** *Function*

FPUT/FPUTLN *Statements*

Function

Put (write) a line to an open file (with an optional carriage return/line feed appended).

Syntax

```
FPUT chan,exp[,exp]
-or-
FPUT chan[,exp[,exp]]
```

chan	An integer expression with the channel to write to (0 through 7).
exp	An expression of any type to evaluate and write to chan.

Remarks

These statements will evalutate zero, one or more expressions of any type and write the results to the specified channel number. The **FPUTLN** statement will append a carriage return/line feed sequence to the end of the expressions; **FPUT** will not. Note that at least one expression must be specified for **FPUT**, unlike the **FPUTLN** statement which need not have any arguments passed to it other than the channel number.

Examples

```
FAPPEND 1,"FILE.DAT",O_WR,S_DB
FPUT 1,U_NAME(),": ",DATE()
FPUTLN 1," ",TIME()," ",CURSEC()
FPUT 1,"Logged!"
FPUTLN 1
FPUTLN 1,"Have a nice"+" day!"
FCLOSE 1
```

See Also

FGET *Statement*, **FPUTPAD** *Statement*

FPUTPAD *Statement*

Function

Put (write) a line of a specified width to an open file.

Syntax

FPUT chan,exp,width

chan	An integer expression with the channel to write to (0 through 7).
exp	An expression of any type to evaluate and write to chan.
width	An integer expression with the width to use to write exp. Valid values are -256 through 256

Remarks

This statement will evalutate an expressions of any type and write the result to the specified channel number. This statement will append a carriage return/line feed sequence to the end of the expression after padding it to the specified width with spaces. If width is positive, then exp will be written right justified (left padded) to the file. If width is negative, then exp will be written left justified (right padded) to the file.

Examples

```
FAPPEND 1,"FILE.DAT",O_WR.S_DB
FPUTPAD 1,U_NAME(),40
FPUTPAD 1,U_DATE(),20
FPUTPAD 1,U_TIME(),20
FCLOSE 1
```

See Also

FGET *Statement*, FPUT/FPUTLN *Statements*

FRESHLINE *Statement*

Function

Move the cursor to a fresh line for output.

Syntax

FRESHLINE

No arguments are required

Remarks

Often while displaying information to the screen you will print a certain amount then want to make sure you are on a clean line before continuing. This statement checks to see if you are in column 1 of the current line. If you are, it assumes you are on a clean line and does nothing. Otherwise, it calls the **NEWLINE** statement for you automatically.

Examples

```
INTEGER i, end
LET end = RANDOM(20)
FOR i = 1 TO end
  PRINT RIGHT(RANDOM(10000),8)
NEXT
FRESHLINE
PRINTLN "Now we continue . . ."
```

See Also

NEWLINE *Statement*, **NEWLINES** *Statement*

FREWIND *Statement*

Function

Rewind an open file.

Syntax

```
FREWIND chan
```

`chan` An integer expression with the open channel to rewind (0 through 7).

Remarks

This statement should be used when you need to rewind a file channel after it has been created/opened with an **FCREATE**, **FOPEN**, or **FAPPEND** statement. Rewinding a file channel will flush file buffers, commit the file to disk, and reposition the file pointer to the beginning of the file. This is useful when you need to start over processing a file that may have changed and don't want to close and re-open the file.

Examples

```
STRING s
FAPPEND 1,"C:\PCB\MAIN\PPE.LOG",O_RW,S_DN
FPUTLN 1,U_NAME()
FREWIND 1
WHILE (!FERR(1)) DO
  FGET 1,s
  PRINTLN s
ENDWHILE
FCLOSE 1
```

See Also

FAPPEND *Statement*, **FCLOSE** *Statement*, **FCREATE** *Statement*, **FOPEN** *Statement*

F_EXP *Constant*

Function

Set the conference expired access flag in a **CONFFLAG** or **CONFUNFLAG** statement.

Value

2 = 10b = 2o = 2h

Remarks

There are five flags per conference maintained for each user. This flag is used to indicate whether or not a user is registered in a specified conference after their subscription expiration date.

Examples

```
CONFUNFLAG 5,F_REG+F_EXP+F_SEL ' Clear reg, exp & sel flags from conf 5
CONFFLAG 9,F_REG+F_EXP+F_SEL   ' Set reg, exp & sel flags for conf 9
```

See Also

F_MW *Constant*, **F_REG***Constant*, **F_SEL** *Constant*, **F_SYS** *Constant*

F_MW *Constant*

Function

Set the conference mail waiting flag in a **CONFFLAG** or **CONFUNFLAG** statement.

Value

10 = 10000b = 20o = 10h

Remarks

There are five flags per conference maintained for each user. This flag is used to indicate whether or not a user has mail waiting in a specified conference.

Examples

```
CONFUNFLAG 5,F_MW ' Clear mail waiting flag from conf 5
CONFFLAG 9,F_MW   ' Set mail waiting flag for conf 9
```

See Also

F_EXP *Constant*, **F_REG***Constant*, **F_SEL** *Constant*, **F_SYS** *Constant*

F_REG *Constant*

Function

Set the conference registration flag in a **CONFFLAG** or **CONFUNFLAG** statement.

Value

1 = 1b = 1o = 1h

Remarks

There are five flags per conference maintained for each user. This flag is used to indicate whether or not a user is registered in a specified conference.

Examples

```
CONFUNFLAG 5,F_REG+F_EXP+F_SEL ' Clear reg, exp & sel flags from conf 5
CONFFLAG 9,F_REG+F_EXP+F_SEL   ' Set reg, exp & sel flags for conf 9
```

See Also

F_EXP *Constant*, **F_MW** *Constant*, **F_SEL** *Constant*, **F_SYS** *Constant*

F_SEL *Constant*

Function

Set the conference selected flag in a **CONFFLAG** or **CONFUNFLAG** statement.

Value

4 = 100b = 4o = 4h

Remarks

There are five flags per conference maintained for each user. This flag is used to indicate whether or not a user has a specified conference selected for message scans.

Examples

```
CONFUNFLAG 5,F_REG+F_EXP+F_SEL ' Clear reg, exp & sel flags from conf 5
CONFFLAG 9,F_REG+F_EXP+F_SEL   ' Set reg, exp & sel flags for conf 9
```

See Also

F_EXP *Constant*, **F_MW** *Constant*, **F_REG** *Constant*, **F_SYS** *Constant*

F_SYS *Constant*

Function

Set the conference SysOp access flag in a **CONFFLAG** or **CONFUNFLAG** statement.

Value

8 = 1000b = 10o = 8h

Remarks

There are five flags per conference maintained for each user. This flag is used to indicate whether or not a user has conference SysOp access in a specified conference.

Examples

```
CONFUNFLAG 5,F_SYS ' Remove (unflag) conf sysop access from conf 5
CONFFLAG 9,F_SYS   ' Grant (flag) conf sysop access for conf 9
```

See Also

F_EXP *Constant*, **F_MW** *Constant*, **F_REG** *Constant*, **F_SEL** *Constant*

GETENV() *Function*

Function

Access the value of an environment variable.

Syntax

GETENV(name)

name A string expression with the name of the environment variable to access.

Return Type & Value

STRING Returns the value of the environment variable specified by name.

Remarks

This function allows you to access the value of any environment variable set at the time that PCBoard was started. So, for example, the PATH environment variable could be used to access data files somewhere on the path.

Examples

```
STRING path
LET path = GETENV()
TOKENIZE path
LET path = "DATAFILE.TXT"
WHILE (!EXIST(path) & (TOKCOUNT() > 0)) DO
  LET PATH = GETTOKEN()+"DATAFILE.TXT"
ENDWHILE
IF (EXIST(path)) PRINTLN "Found ",path."!"
```

See Also

PCBDAT() *Function*

GETTOKEN *Statement*

Function

Retrieve a token from a previous **TOKENIZE** statement.

Syntax

```
GETTOKEN var
```

`var` Variable to store the retrieved token in.

Remarks

One of the strongest features of PCBoard is it's ability to take a series of stacked parameters from a command line and use them all at once instead of requiring the user to navigate a series of menus and select one option at each step of the way. The **TOKENIZE** statement is the PPL equivalent of what PCBoard uses to break a command line into individual commands (tokens). The number of tokens available may be accessed via the **TOKCOUNT()** function, and each token may be accessed, one at a time, by the **GETTOKEN** statement and/or the **GETTOKEN()** function.

Examples

```
STRING  cmdline
INPUT "Command",cmdline
TOKENIZE cmdline
PRINTLN "You entered ",TOKCOUNT()," tokens"
WHILE (TOKCOUNT() > 0) DO
  GETTOKEN cmdline
  PRINTLN "Token:  ",CHR(34),cmdline,CHR(34)
ENDWHILE
```

See Also

GETTOKEN() *Function*, **TOKCOUNT()** *Function*, **TOKENIZE** *Statement*, **TOKENSTR()** *Function*

GETTOKEN() *Function*

Function

Retrieve a token from a previous **TOKENIZE** statement.

Syntax

GETTOKEN()

No arguments are required

Return Type & Value

STRING Returns the next available token from the most recent **TOKENIZE** statement.

Remarks

One of the strongest features of PCBoard is it's ability to take a series of stacked parameters from a command line and use them all at once instead of requiring the user to navigate a series of menus and select one option at each step of the way. The **TOKENIZE** statement is the PPL equivalent of what PCBoard uses to break a command line into individual commands (tokens). The number of tokens available may be accessed via the **TOKCOUNT()** function, and each token may be accessed, one at a time, by the **GETTOKEN** statement and/or the **GETTOKEN()** function.

Examples

```
STRING  cmdline
INPUT "Command",cmdline
TOKENIZE cmdline
PRINTLN "You entered ",TOKCOUNT(),' tokens"
WHILE (TOKCOUNT() > 0) DO
  LET cmdline = GETTOKEN()
  PRINTLN "Token:  ",CHR(34),cmdline,CHR(34)
ENDWHILE
```

See Also

GETTOKEN *Statement*, **TOKCOUNT()** *Function*, **TOKENIZE** *Statement*, **TOKENSTR()** *Function*

GETUSER *Statement*

Function

Fill predeclared variables with values from user record.

Syntax

GETUSER

No arguments are required

Remarks

There are many predeclared variables which may be used to access and change user information. However, their values are undefined until you use the **GETUSER** statement, and any changes you make don't take hold until you use the **PUTUSER** statement.

Examples

```
IF (PSA(3)) THEN
  GETUSER
  INPUT "Addr 1",U_ADDR(0)
  INPUT "Addr 2",U_ADDR(1)
  INPUT "City  ",U_ADDR(2)
  INPUT "State ",U_ADDR(3)
  INPUT "ZIP   ",U_ADDR(4)
  INPUT "Cntry ",U_ADDR(5)
  PUTUSER
ENDIF
```

See Also

PUTUSER *Statement*

GETX() *Function*

Function

Report the X coordinate (column) of the cursor on screen.

Syntax

GETX()

No arguments are required

Return Type and Value

INTEGER Returns the column (1-80) of the cursor on screen.

Remarks

This function is used to query the ANSI emulator in PCBoard the current X position of the cursor. It may be used for saving the cursor position for future use or for saving the horizontal cursor position while changing the vertical position with the **ANSIPOS** statement.

Examples

```
INTEGER x,y
STRING s

WHILE (UPPER(s) <> "QUIT") DO

  INPUT "Text",s
  PRINTLN " - ",s

  LET x = GETX()
  LET y = GETY()
  IF (y = 23) THEN
    CLS
    LET x = GETX()
    LET y = GETY()
  ENDIF

  ANSIPOS 40,23
  PRINT "@X8Fs=",s
  ANSIPOS x,y

ENDWHILE
```

See Also

ANSIPOS *Statement*, **ANSION()** *Function*, **BACKUP** *Statement*, **FORWARD** *Statement*, **GETY()** *Function*, **GRAFMODE()** *Function*

GETY() *Function*

Function

Report the Y coordinate (row) of the cursor on screen.

Syntax

GETY()

No arguments are required

Return Type and Value

INTEGER Returns the row (1-23) of the cursor on screen.

Remarks

This function is used to query the ANSI emulator in PCBoard the current Y position of the cursor. It may be used for saving the cursor position for future use or for saving the verticle cursor position while changing the horizontal position with the **ANSIPOS** statement.

Examples

```
INTEGER x,y
STRING s

WHILE (UPPER(s) <> "QUIT") DO

  INPUT "Text",s
  PRINTLN " - ",s

  LET x = GETX()
  LET y = GETY()
  IF (y = 23) THEN
    CLS
    LET x = GETX()
    LET y = GETY()
  ENDIF

  ANSIPOS 40,23
  PRINT "@X8Fs=",s
  ANSIPOS x,y

ENDWHILE
```

See Also

ANSIPOS *Statement*, **ANSION()** *Function*, **BACKUP** *Statement*, **FORWARD** *Statement*, **GETX()** *Function*, **GRAFMODE()** *Function*

GOODBYE *Statement*

Function

Log the user off as though they had typed the G (goodbye) command.

Syntax

GOODBYE

No arguments are required

Remarks

There are multiple ways for the user to log off. One is by typing G at the command prompt. That will warn them if they have files flagged for download and (optionally) confirm their selection (incase they accidentally hit G and ENTER). Another is the BYE command. PCBoard assumes that, if the user typed BYE instead of G, that they really want to log off, didn't type it in accidentally, and want to leave **now**. The **GOODBYE** statement performs the same processing as the PCBoard G command.

Examples

```
STRING s
INPUT "What do you want to do",s
IF      (s = "G")   THEN GOODBYE
ELSEIF (s = "BYE") THEN BYE
ELSE                    KBDSTUFF s
ENDIF
```

See Also

BYE *Statement*, **DTROFF** *Statement*, **DTRON** *Statement*, **HANGUP** *Statement*

GOSUB *Statement*

Function

Transfer program control and save the return information.

Syntax

```
GOSUB label
```

`label` The label to which control should be transferred.

Remarks

It is often necessary to perform an indentical set of instructions several times in a program. This leaves you with two choices. One, rewrite the code several times (and hope you do it right each time), or two, write it once as a subroutine, then use **GOSUB** to run it. This statement will save the address of the next line so that a **RETURN** statement at the end of the subroutine can instruct PPL to resume execution with the line following the **GOSUB**.

Examples

```
STRING Question, Answer
LET Question = "What is your street address ..."
GOSUB ask
LET Question = "What is your city, state and zip ..."
GOSUB ask
END

:ask ' Sub to ask a question, get an answer, and log them to a file
LET Answer = ""
PRINTLN "@X0E",Question
INPUT "",Answer
NEWLINES 2
FPUTLN 0,"Q: ",STRIPATX(Question)
FPUTLN 0,"A: ",Answer
RETURN
```

See Also

GOTO *Statement*, **FOR/NEXT** *Statement*, **IF/ELSEIF/ELSE/ENDIF** *Statement*, **WHILE/ENDWHILE** *Statement*, **RETURN** *Statement*

GOTO *Statement*

Function

Transfer program control.

Syntax

```
GOTO label
```

`label` The label to which control should be transferred.

Remarks

GOTO is an essential part of just about every programming language, and it is also an overused part of every one of those languages. When you need to make a decision and alter program flow based on some condition it is a necessary evil. For example, it is very useful in getting out of deeply nested loops when a critical error of some sort occurs. For the most part, avoid it if at all possible. Look for other options to write your program, such as block **IF**, **WHILE**, and **FOR** statements. They are much easier to understand and maintain than code with **GOTO** statements sprinkled liberally throughout.

Examples

```
INTEGER i
STRING  s

FOPEN 1,"FILE.DAT",O_RD,S_DW

WHILE (UPPER(s) <> "QUIT") DO
  FGET 1,s
  IF (FERR(1)) THEN
    PRINTLN "Error, aborting..."
    GOTO exit
  ENDIF
  INC i
  PRINTLN "Line ",i,": ",s
ENDWHILE

:exit
FCLOSE 1
```

See Also

GOSUB *Statement*, **FOR/NEXT** *Statement*, **IF/ELSEIF/ELSE/ENDIF** *Statement*, **WHILE/ENDWHILE** *Statement*, **RETURN** *Statement*

GRAFMODE() *Function*

Function

Report the graphics mode in use.

Syntax

GRAFMODE ( )

No arguments are required

Return Type and Value

STRING Returns a letter indicating the current graphics supported.

Remarks

This function will return one of four possible responses. "N" will be returned if no graphics support is currently available. "A" will be returned for non-graphics users that do have ANSI support available for positioning. "G" will be returned for users who support full ANSI graphics. Finally, "R" will be returned for users who support RIPscrip.

Examples

```
IF      (GRAFMODE() = "R") THEN PRINT "RIPscrip"
ELSE IF (GRAFMODE() = "G") THEN PRINT "Full ANSI"
ELSE IF (GRAFMODE() = "A") THEN PRINT "ANSI positioning"
ELSE IF (GRAFMODE() = "N") THEN PRINT "No"
ELSE                            PRINT "Unknown"
ENDIF
PRINTLN " Graphics Supported"
```

See Also

ANSIPOS *Statement*, **ANSION()** *Function*, **BACKUP** *Statement*, **FORWARD** *Statement*, **GETX()** *Function*, **GETY()** *Function*

GRAPH *Constant*

Function

Set the graphics specific file search flag in a **DISPFILE** statement.

Value

1 = 1b = 1o = 1h

Remarks

The **DISPFILE** statement will allow you to display a file to the user, and optionally to have PCBoard look for alternate security, graphics, and/or language specific files. This flag instructs PCBoard to search for alternate graphics files (ANSI or RIPscrip) via the G and R suffix. The current graphics mode may be obtained with the **GRAFMODE()** function.

Examples

```
STRING s
DISPFILE "MNUA",SEC+GRAPH+LANG
INPUT "Option",s
```

See Also

DISPFILE *Statement*, **GRAFMODE()** *Function*, **LANG** *Constant*, **SEC** *Constant*

GUIDE *Constant*

Function

Set the display input guide flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

4 = 100b = 4o = 4h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to display the length of an input field, regardless of ANSI availability, if you use this constant with the **FIELDLEN** constant. If ANSI is not available and this constant is used, the user will see the input field width marked using "(---)" above the input field.

Examples

```
STRING pwd
INPUTSTR "Enter id number",pwd,@X0E,4,"0123456789",FIELDLEN+GUIDE
IF (pwd <> "1234") PRINTLN "Bad id number"
```

See Also

FIELDLEN *Constant*, **INPUTSTR** *Statement*, **PROMPTSTR** *Statement*

HANGUP *Statement*

Function

Hangup on the user and perform abnormal logoff processing.

Syntax

HANGUP

No arguments are required

Remarks

This statement is useful in situations where you need to get the caller off immediately without any delay or notice. It will hangup on the caller, do all logoff processing, and log an abnormal logoff to the callers log.

Examples

```
STRING s
INPUT "What do you want to do",s
IF      (s = "G")     THEN GOODBYE
ELSEIF (s = "BYE")   THEN BYE
ELSEIF (s = "HANG") THEN HANGUP
ELSE                        KBDSTUFF s
ENDIF
```

See Also

BYE *Statement*, **DTROFF** *Statement*, **DTRON** *Statement*, **GOODBYE** *Statement*

HELPPATH() *Function*

Function

Return the path of help files as defined in PCBSetup.

Syntax

HELPPATH()

No arguments are required

Return Type & Value

STRING Returns the path of the PCBoard help files.

Remarks

This function will return the path where help files are located as defined in PCBSetup. This can be useful when you want to add system help capabilities to your PPE application.

Examples

```
PRINTLN "HELP FOR THE R (READ) COMMAND:"
PRINTLN "-----------------------------"
NEWLINE
DISPFILE HELPPATH()+"HLPR",GRAPH+LANG+SEC
```

See Also

PPEPATH() ***Function*****, SLPATH()** ***Function*****, TEMPPATH()** ***Function***

HIGHASCII *Constant*

Function

Set the allow high ASCII flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

4096 = 1000000000000b = 10000o = 1000h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to allow high ASCII characters to be input regardless of the valid character string specified, but only if the SysOp has disabled the high ASCII filter in PCBSetup.

Examples

```
STRING pwd
INPUTSTR "Enter password",pwd,@X0E,4,MASK_ASCII(),HIGHASCII
GETUSER
IF (pwd <> U_PWD) HANGUP
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

HOUR() *Function*

Function

Extract the hour from a specified time of day.

Syntax

```
HOUR(texp)
```

texp	Any time expression.

Return Type & Value

INTEGER	Returns the hour from the specified time expression (**texp**). Valid return values are from 0 to 23.

Remarks

This function allows you to extract a particular piece of information about a **TIME** value, in this case the hour of the time of day expression.

Examples

```
PRINTLN "The hour is ",HOUR(TIME())
```

See Also

MIN() *Function*, SEC() *Function*, TIME() *Function*

I2S() *Function*

Function

Convert an integer to a string in a specified number base.

Syntax

`I2S(int,base)`

`int`	Any integer expression to convert to string format.
`base`	An integer expression with the number base (2 through 36) to convert to.

Return Type & Value

STRING	Returns **int** converted to a string in the specified number base.

Remarks

People work with decimal (base 10) numbers, whereas computers work with binary (base 2) numbers. However, it is often more convienient to display numbers to the user in a format other than decimal for clarity, compactness, or other reasons. This function will convert a number to string format in any number base from 2 to 36. So, **I2S(10,2)** would return a string of "1010"; **I2S(35,36)** would return "Z".

Examples

```
INTEGER i,num
INPUTINT "Enter a number (decimal)",num,@X0E
FOR i = 2 TO 36
  PRINTLN num," base 10 = ",I2S(num,i)," base ",i
NEXT
```

See Also

S2I() *Function*

IF/ELSEIF/ELSE/ENDIF *Statement*

Function

Execute one or more statments if a condition is true.

Syntax

```
IF (bexp) statement
-or-
IF (bexp) THEN
  statement(s)
[ELSEIF (bexp) THEN] ' optional in a block IF
[  statement(s)]
[ELSEIF (bexp) THEN] ' you may have multiple ELSEIF statement(s)
[  statement(s)]
[ELSE]               ' optional in a block IF
[  statement(s)]
ENDIF
```

`bexp`	Any boolean expression.
`statement`	Any valid PPL statement.

Remarks

The **IF** statement supports two types of structures: logical and block. A logical **IF** statement is a single statement; if a condition is **TRUE**, execute a single statement. A block **IF** can be one or more statements with multiple conditions to test for. The start of a block **IF** loop is differentiated from a logical **IF** loop by the **THEN** keyword immediately after the condition. In a block **IF** statement the first condition to evaluate to **TRUE** will be executed, after which control will be transferred to the statement following the **ENDIF**. If none of the conditions are **TRUE** by the time an **ELSE** statement is reached then the statements between the **ELSE** and **ENDIF** will be processed. If none of the conditions are **TRUE** and an **ELSE** statement is never found then none of the conditions will be executed; instead, control will be transferred to the statement after the **ENDIF**.

Examples

```
IF (CURSEC() < 10) END ' Insufficient security, terminate execution
IF (CURSEC() < 20) THEN
  PRINTLN "Your security is less than 20"
ELSEIF (CURSEC() > 30) THEN
  PRINTLN "Your security is greater than 30"
ELSEIF (CURSEC() = 25) THEN
  PRINTLN "Your security is 25"
ELSE
  PRINTLN "Your security is level",CURSEC()
ENDIF
```

See Also

GOSUB *Statement*, **GOTO** *Statement*, **FOR/NEXT** *Statement*, **WHILE/ENDWHILE** *Statement*, **RETURN** *Statement*

INC *Statement*

Function

Increment the value of a variable.

Syntax

```
INC var
```

`var`	The variable with the value to increment

Remarks

Many programs require extensive addition and subtraction, and often a value is only increased or decreased by 1. This statement allows for a shorter, more efficient method of increasing (incrementing) a value by 1 (**INC i**) than adding 1 to a variable and assigning the result to the same variable (**LET i = i + 1**).

Examples

```
INTEGER i
PRINTLN "Countdown:"
LET i = 0
WHILE (i <= 10) DO
  PRINTLN "T plus ",i
  INC i
ENDWHILE
```

See Also

DEC *Statement*

INKEY() *Function*

Function

Get the next key input.

Syntax

INKEY()

No arguments are required

Return Type and Value

STRING Returns a single character for displayable characters or a string for cursor movement and function keys.

Remarks

This function will return a single character long string for most key presses. Additionally, it will return key names for function keys and cursor movement keys if it finds an ANSI sequence or DOORWAY mode sequence. It will return keystrokes from both the remote caller as well as the local BBS node. However, realize that many function keys are reserved by PCBoard for BBS related uses and may not be available for your applications that require SysOp input.

Examples

```
STRING key
WHILE (key <> CHR(27)) DO
  LET key = INKEY()
  IF (LEFT(key,5) = "SHIFT") THEN
    PRINTLN "It was a shifted key"
  ELSEIF (LEFT(key,4) = "CTRL") THEN
    PRINTLN "It was a control key"
  ELSEIF (LEFT(key,3) = "ALT") THEN
    PRINTLN "It was an alternate key"
  ENDIF
  PRINTLN "The key was ",key
ENDWHILE
```

See Also

KINKEY() *Function*, **MGETBYTE**() *Function*, **MINKEY**() *Function*

INPUT *Statement*

Function

Prompt the user for a string of text.

Syntax

```
INPUT prompt,var
```

`prompt`	A string expression with the prompt to display to the user.
`var`	The variable in which to store the user's input.

Remarks

This statement will accept any string of input from the user, up to 60 characters maximum length. In addition to displaying the prompt, it will display parenthesis around the input field if the user is in ANSI mode. Because of this, you should generally limit your prompts to 15 characters or less.

Examples

```
BOOLEAN b
DATE    d
INTEGER i
MONEY   m
STRING  s
TIME    t
INPUT "Enter BOOLEAN",b
INPUT "Enter DATE",d
INPUT "Enter INTEGER",i
INPUT "Enter MONEY",m
INPUT "Enter STRING",s
INPUT "Enter TIME",t
PRINTLN b," ",d," ",i
PRINTLN m," ",s," ",t
```

See Also

INPUT... ***Statements*****, INPUTSTR** ***Statement*****, INPUTTEXT** ***Statement*****, LET** ***Statement*****, PROMPTSTR** ***Statement***

INPUT... *Statements*

Function

Prompt the user for a string of text of a specific length and with type dependent valid characters.

Syntax

```
INPUT... prompt,var,color
```

prompt	A string expression with the prompt to display to the user.
var	The variable in which to store the user's input.
color	An integer expression with the color to display the prompt in.

INPUT should be followed by one of the following types (without spaces between the INPUT and type): CC, DATE, INT, MONEY, TIME, or YN.

Remarks

This statement will accept a string of input from the user, with a set of valid characters and up to a maximum length (MAXLEN) determined by the statement in use. In addition to displaying the prompt, it will display parenthesis around the input field if the user is in ANSI mode. Because of this, you should generally limit your prompts to a length determined by the following formula: (80-MAXLEN-4). Here are the valid character masks and maximum length values for each of the input statements:

INPUT	CC	DATE	INT	MONEY	TIME	YN
Val Chars	"0123456789"	"0123456789-/"	"0123456789+-"	"0123456789+-$."	"0123456789:"	*
Max Length	16	8	11	13	8	1

* The **INPUTYN** statement valid characters are dependent on the users language selection. Usually they will be "YN" for english language systems. Other letters may be defined for different languages in PCBML.DAT.

Examples

```
DATE    d
INTEGER i
MONEY   m
STRING  cc, yn
TIME    t
INPUTCC "Enter Credit Card Number",cc
INPUTDATE "Enter DATE",d
INPUTINT "Enter INTEGER",i
INPUTMONEY "Enter MONEY",m
INPUTTIME "Enter TIME",t
INPUTYN "Enter Yes/No Response",s
PRINTLN cc," ",d," ",i
PRINTLN m," ",t," ",yn
```

See Also

INPUT *Statement*, **INPUTSTR** *Statement*, **INPUTTEXT** *Statement*, **LET** *Statement*, **PROMPTSTR** *Statement*

INPUTSTR *Statement*

Function

Prompt the user for a string of text in a specific format.

Syntax

```
INPUTSTR prompt,var,color,len,valid,flags
```

`prompt`	A string expression with the prompt to display to the user.
`var`	The variable in which to store the user's input.
`color`	An integer expression with the color to display the prompt in.
`len`	An integer expression with maximum length of text to input.
`valid`	A string expression with the valid characters that the user may enter.
`flags`	An integer expression with flags to modify how the statement works.

Remarks

This statement will accept a string of input from the user, up to the length defined. The **prompt** parameter will be displayed to the user in the specified **color** before accepting input. Only characters found in the **valid** parameter will be accepted. However, the **flags** parameter may affect how prompt is displayed and the valid characters that are accepted. Individual **flags** may be added together as needed. Several functions exist to easily specify commonly used valid character masks. They are **MASK_ALNUM()**, **MASK_ALPHA()**, **MASK_ASCII()**, **MASK_FILE()**, **MASK_NUM()**, **MASK_PATH()**, and **MASK_PWD()**. Defined flag values are **AUTO**, **DEFS**, **ECHODOTS**, **ERASELINE**, **FIELDLEN**, **GUIDE**, **HIGHASCII**, **LFAFTER**, **LFBEFORE**, **NEWLINE**, **NOCLEAR**, **STACKED**, **UPCASE**, **WORDWRAP**, and **YESNO**.

Examples

```
BOOLEAN b
DATE    d
INTEGER i
MONEY   m
STRING  s
TIME    t
INPUTSTR "Enter BOOLEAN",b,@X0E,1,"10",LFBEFORE+NEWLINE
INPUTSTR "Enter DATE",d,@X0F,8,"0123456789-",NEWLINE+NOCLEAR
INPUTSTR "Enter INTEGER",i,@X07,20,MASK_NUM(),NEWLINE
INPUTSTR "Enter MONEY",m,@X08,9,MASK_NUM()+".",NEWLINE+DEFS+FIELDLEN
INPUTSTR "Enter STRING",s,@X09,63,MASK_ASCII(),NEWLINE+FIELDLEN+GUIDE
INPUTSTR "Enter TIME",t,@X0A,5,"0123456789"+":",NEWLINE+LFAFTER
PRINTLN b," ",d," ",i
PRINTLN m," ",s," ",t
```

See Also

INPUT *Statement*, **INPUT...** *Statements*, **INPUTTEXT** *Statement*, **LET** *Statement*, **PROMPTSTR** *Statement*

INPUTTEXT *Statement*

Function

Prompt the user in a specified color for a string of text of specified length.

Syntax

```
INPUTTEXT prompt,var,color,len
```

`prompt`	A string expression with the prompt to display to the user.
`var`	The variable in which to store the user's input.
`color`	An integer expression with the color to display the prompt in.
`len`	An integer expression with maximum length of text to input.

Remarks

This statement will accept any string of input from the user, up to the length defined. In addition to displaying the prompt, it will display parenthesis around the input field if the user is in ANSI mode. Because of this, you should generally limit your prompts to (80-**len**-4) characters or less.

Examples

```
BOOLEAN b
DATE    d
INTEGER i
MONEY   m
STRING  s
TIME    t
INPUTTEXT "Enter BOOLEAN",b,@X0E,1
INPUTTEXT "Enter DATE",d,@X0F,8
INPUTTEXT "Enter INTEGER",i,@X07,20
INPUTTEXT "Enter MONEY",m,@X08,9
INPUTTEXT "Enter STRING",s,@X09,63
INPUTTEXT "Enter TIME",t,@X0A,5
PRINTLN b," ",d," ",i
PRINTLN m," ",s," ",t
```

See Also

INPUT *Statement*, **INPUT...** *Statements*, **INPUTSTR** *Statement*, **LET** *Statement*, **PROMPTSTR** *Statement*

INSTR() *Function*

Function

Find the position of one string within another string.

Syntax

```
INSTR(str,sub)
```

`str`	A string expression to look for sub in.
`sub`	A string expression to search for.

Return Type & Value

INTEGER	Returns the 1-based position of **sub** within **str** or 0 if **sub** is not found within **str**.

Remarks

This function is useful for determining if a particular word or phrase exists in a string. The return value is the position of the sub string within the longer string. The first character of **str** is position 1, the second is position 2, and so on. If **sub** is not found in **str**, 0 is returned.

Examples

```
STRING s
WHILE (INSTR(UPPER(s),"QUIT") = 0) DO
  INPUTTEXT "Enter string",s,@X0E,40
  NEWLINE
  PRINTLN s
ENDWHILE
```

See Also

LEN() *Function*, **SCRTEXT**() *Function*, **SPACE**() *Function*, **STRING**() *Function*

INTEGER *Type*

Function

Declare one or more variables of type integer.

Syntax

```
INTEGER var|arr(s[,s[,s]])[,var|arr(s[,s[,s]])]
```

`var`	The name of a variable to declare. Must start with a letter [A-Z] which may be followed by letters, digits [0-9] or the underscore [_]. May be of any length but only the first 32 characters are used.
`arr`	The name of an array variable to declare. The same naming conventions as **var** are used.
`s`	The size (0-based) of an array variable dimension. Any constant integer expression is allowed.

Remarks

INTEGER variables are stored as four byte signed long integers. The range of an **INTEGER** is -2,147,483,648 - +2,147,483,647. An **INTEGER** assignment to a **STRING** will result in a string with the representation of the number (similar to BASIC's STR$ function and C's ltoa function). A **STRING** to **INTEGER** assignment will convert the string back to the four byte binary integer value (similar to BASIC's VAL function and C's atol function). If an **INTEGER** is assigned to or from any other type, an appropriate conversion is performed automatically by PPL.

Examples

```
INTEGER i, year, cardDeck(4*13), matrix(2,2), matrices(3,4,5)
```

See Also

BOOLEAN *Type*, **DATE** *Type*, **MONEY** *Type*, **STRING** *Type*, **TIME** *Type*

JOIN *Statement*

Function

Execute the join conference command with desired sub-commands.

Syntax

```
JOIN cmds
```

`cmds`	A string expression with any desired sub-commands for the join conference command.

Remarks

This statement will allow you to access the join conference command (the J command from the main menu), and any join conference sub-commands, under PPE control. Note that this statement will destroy any previously tokenized string expression. If you have string tokens pending at the time of the **JOIN** statment you should save them first and then retokenize after the **JOIN** statement is complete.

Examples

```
STRING yn
INPUTYN "Join SysOp conference",yn,@X0E
IF (yn = YESCHAR()) JOIN 4
```

See Also

BLT *Statement*, **DIR** *Statement*, **QUEST** *Statement*

KBDCHKOFF *Statement*

Function

Turn off keyboard timeout checking.

Syntax

KBDCHKOFF

No arguments are required

Remarks

PCBoard has built in automatic keyboard timeout detecting. What this means is that if someone should remain online for a SysOp defined period of time without typing anything for PCBoard to process, PCBoard will detect it, log it to the callers log, and recycle back to the call waiting screen. Some applications require the ability to turn this off; for example, a process that will take a while without interacting with the caller should turn off keyboard timeout testing to keep PCBoard from thinking that the user has stopped entering information. Normally, PCBoard would just recycle at that point. So, just before you start a section of code that should continue for a while without user input, you should issue a **KBDCHKOFF** statement. It will turn off the automatic keyboard timeout checking. When you've finished the block where keyboard timeout checking has been disabled, issue the **KBDCHKON** statement to turn it back on.

Examples

```
KBDCHKOFF
WHILE (RANDOM(10000) <> 0) PRINT "." ' Something to take a long time!
KBDCHKON
```

See Also

CDCHKOFF *Statement*, **CDCHKON** *Statement*, **KBDCHKON** *Statement*

KBDCHKON *Statement*

Function

Turn on keyboard timeout checking.

Syntax

KBDCHKON

No arguments are required

Remarks

PCBoard has built in automatic keyboard timeout detecting. What this means is that if someone should remain online for a SysOp defined period of time without typing anything for PCBoard to process, PCBoard will detect it, log it to the callers log, and recycle back to the call waiting screen. Some applications require the ability to turn this off; for example, a process that will take a while without interacting with the caller should turn off keyboard timeout testing to keep PCBoard from thinking that the user has stopped entering information. Normally, PCBoard would just recycle at that point. So, just before you start a section of code that should continue for a while without user input, you should issue a **KBDCHKOFF** statement. It will turn off the automatic keyboard timeout checking. When you've finished the block where keyboard timeout checking has been disabled, issue the **KBDCHKON** statement to turn it back on.

Examples

```
KBDCHKOFF
WHILE (RANDOM(10000) <> 0) PRINT "." ' Something to take a long time!
KBDCHKON
```

See Also

CDCHKOFF *Statement*, **CDCHKON** *Statement*, **KBDCHKOFF** *Statement*

KBDFILE *Statement*

Function

Stuff the contents of a text file into the keyboard buffer for later processing.

Syntax

```
KBDFILE file
```

`file` A string expression with the file name whose contents should be stuffed into the keyboard buffer.

Remarks

This statement allows you to feed a series of keystrokes to PCBoard as though they were typed in by the user. This is useful when you need to feed a series of commands to PCBoard one right after another and they would add up to more than 256 characters (the maximum buffer size for the **KBDSTUFF** statement).

Examples

```
INTEGER retcode
SHOWOFF
OPENCAP "NEWFILES.LST",retcode
KBDSTUFF CHR(13)
DIR "N;S;A;NS"
CLOSECAP
SHOWON
SHELL "TRUE,retcode,"PKZIP","-mex NEWFILES NEWFILES.LST"
KBDFILE "FLAGFILE.CMD"
```

See Also

KBDSTUFF *Statement*

KBDSTUFF *Statement*

Function

Stuff a string into the keyboard buffer for later processing.

Syntax

```
KBDSTUFF str
```

str	A string expression to stuff into the keyboard buffer for later processing.

Remarks

This statement allows you to feed a series of keystrokes to PCBoard as though they were typed in by the user. This can be especially useful when you are replacing an existing command; add your PPE file to the CMD.LST file so that it takes the place of the built in command, then have your PPE stuff the original (or modified) command back to the keyboard buffer. PCBoard will then process it as soon as you exit your PPE application. It can also be used when building new commands that should perform several built in operations automatically. A maximum of 256 characters at a time can be stuffed into the keyboard buffer. If you need more than this, you should use the **KBDFILE** statement. Note that this statement may not be used to access commands defined in the CMD.LST file.

Examples

```
INTEGER retcode
SHOWOFF
OPENCAP "NEWFILES.LST",retcode
KBDSTUFF CHR(13)
DIR "N;S;A;NS"
CLOSECAP
SHOWON
SHELL TRUE,retcode,"PKZIP","-mex NEWFILES NEWFILES.LST"
KBDSTUFF "FLAG NEWFILES.ZIP"
```

See Also

KBDFILE *Statement*

KINKEY() *Function*

Function

Get the next key input from the local keyboard only.

Syntax

```
KINKEY()
```

No arguments are required

Return Type and Value

STRING Returns a single character for displayable characters or a string for cursor movement and function keys.

Remarks

This function will return a single character long string for most key presses. Additionally, it will return key names for function keys and cursor movement keys. It will only return keystrokes from the local BBS node's keyboard. However, realize that many function keys are reserved by PCBoard for BBS related uses and may not be available for your applications that require SysOp input.

Examples

```
STRING key
WHILE (key <> CHR(27)) DO
  LET key = KINKEY()
  IF (LEFT(key,5) = "SHIFT") THEN
    PRINTLN "It was a shifted key"
  ELSEIF (LEFT(key,4) = "CTRL") THEN
    PRINTLN "It was a control key"
  ELSEIF (LEFT(key,3) = "ALT") THEN
    PRINTLN "It was an alternate key"
  ENDIF
  PRINTLN "The key was ",key
ENDWHILE
```

See Also

INKEY() *Function*, **MGETBYTE()** *Function*, **MINKEY()** *Function*

LANG *Constant*

Function

Set the language specific file search flag in a **DISPFILE** statement.

Value

4 = 100b = 4o = 4h

Remarks

The **DISPFILE** statement will allow you to display a file to the user, and optionally to have PCBoard look for alternate security, graphics, and/or language specific files. This flag instructs PCBoard to search for alternate language files via the language extension. The current language extension may be obtained with the **LANGEXT()** function.

Examples

```
STRING s
DISPFILE "MNUA",SEC+GRAPH+LANG
INPUT "Option",s
```

See Also

DISPFILE *Statement*, **GRAPH** *Constant*, **LANGEXT()** *Function*, **SEC** *Constant*

LANGEXT() *Function*

Function

Get the file extension for the current language.

Syntax

LANGEXT ( )

No arguments are required

Return Type and Value

STRING Returns a ".XXX" formatted string where XXX is the extension text (could be 1, 2 or 3 characters long depending on the configuration and language in use).

Remarks

This function allows you to access the file extension used by SysOp definable and system language specific files. You may use it to create your own filenames that are language specific.

Examples

```
PRINTLN "Brief user profile"
NEWLINE
PRINTLN "      Security:  ",CURSEC()
PRINTLN "Graphics Mode:  ",GRAFMODE()
PRINTLN "      Language:  ",LANGEXT()
```

See Also

LANG *Constant*

LEFT() *Function*

Function

Access the left most characters from a string.

Syntax

```
LEFT(str,chars)
```

`str`	A string expression to take the left most characters of.
`chars`	An integer expression with the number of characters to take from the left end of str.

Return Type & Value

STRING	Returns a string with the left most chars characters of str.

Remarks

This function will return a sub string with the left most **chars** characters of a specified string. This can be useful in data processing as well as text formatting. If **chars** is less than or equal to 0 then the returned string will be empty. If **chars** is greater than the length of **str** then the returned string will have spaces added to the left to pad it out to the full length specified.

Examples

```
WHILE (RANDOM(250) <> 0) PRINT LEFT(RANDOM(250),4)," "

STRING s
FOPEN 1,"DATA.TXT",O_RD,S_DN
WHILE (!FERR(1)) DO
  FGET 1,s
  PRINT RTRIM(LEFT(s,25)," "),"  - "
  PRINTLN RIGHT(s,LEN(s)-25)
ENDWHILE
FCLOSE 1
```

See Also

MID() ***Function***, **RIGHT()** ***Function***

LEN() *Function*

Function

Access the length of a string.

Syntax

LEN(str)

str	Any string expression.

Return Type & Value

INTEGER	Returns the length of a string.

Remarks

This function will return the length of a string. The value returned will always be between 0 (an empty string) and 256 (the maximum length of a string).

Examples

```
STRING s
FOPEN 1,"DATA.TXT",O_RD,S_DN
WHILE (!FERR(1)) DO
  FGET 1,s
  PRINTLN "The length of the current string is ",LEN(s)
ENDWHILE
FCLOSE 1
```

See Also

INSTR() ***Function*****, SCRTEXT()** ***Function*****, SPACE()** ***Function*****, STRING()** ***Function***

LET *Statement*

Function

Evaluate an expression and assign the result to a variable.

Syntax

```
LET var = expr
-or-
var = expr
```

`var`	Variable to which the result of **expr** should be assigned.
`expr`	Any valid PPL expression.

Remarks

The **LET** statement supports modes of operation: explicit and implicit. An explicit **LET** statement always includes all of the parts in the first example above (the **LET** keyword, the variable, the equal sign, and the expression). An implicit **LET** statement does not need the **LET** keyword; the format (**var = expr**) is sufficient. However, the implicit form will not always work. For example, if you had a variable named **PRINT** (which is also a statement name) you could not use **PRINT = expr**; PPL expects the first word on a line to be a statement name, and if it isn't, it is an implicit **LET** statement. Since PPL would find the **PRINT** keyword first it would try to process the rest of the line as a **PRINT** statement. This is easily avoided by using the **LET** keyword and making it an explicit **LET** statement (**LET PRINT = expr**).

Examples

```
INTEGER i
STRING  s
GETUSER
LET U_PWD = "NEWPWD"
LET s = "This is a string"
LET i = 7*9+9*7
PUTUSER
```

See Also

INPUT *Statement*, **INPUT...** *Statements*, **INPUTSTR** *Statement*, **INPUTTEXT** *Statement*, **PROMPTSTR** *Statement*

LFAFTER *Constant*

Function

Set the extra line feed after prompt flag in a **DISPTEXT**, **INPUTSTR**, or **PROMPTSTR** statement.

Value

256 = 100000000b = 400o = 100h

Remarks

The **INPUTSTR**, **PROMPTSTR**, and **DISPTEXT** statements have the ability to send an extra carriage return/line feed after a prompt is displayed automatically and without the need to make a separate call to the **NEWLINE** statement.

Examples

```
STRING pwd
INPUTSTR "Enter id",pwd,0X0E,4,"0123456789",LFBEFORE+NEWLINE+LFAFTER
IF (pwd <> "1234") PRINTLN "Bad id number"
```

See Also

DISPTEXT *Statement*, **INPUTSTR** *Statement*, **LFBEFORE** *Constant*, **NEWLINE** *Constant*, **PROMPTSTR** *Statement*

LFBEFORE *Constant*

Function

Set the line feed before prompt flag in a **DISPTEXT**, **INPUTSTR**, or **PROMPTSTR** statement.

Value

128 = 10000000b = 200o = 80h

Remarks

The **INPUTSTR**, **PROMPTSTR**, and **DISPTEXT** statements have the ability to send a carriage return/line feed before a prompt is displayed automatically and without the need to make a separate call to the **NEWLINE** statement.

Examples

```
STRING pwd
INPUTSTR "Enter id",pwd,0X0E,4,"0123456789",LFBEFORE+NEWLINE+LFAFTER
IF (pwd <> "1234") PRINTLN "Bad id"
```

See Also

DISPTEXT *Statement*, **INPUTSTR** *Statement*, **LFAFTER** *Constant*, **NEWLINE** *Constant*, **PROMPTSTR** *Statement*

LOG *Statement*

Function

Log a message to the callers log.

Syntax

```
LOG msg,left
```

`msg`	A string expression to write to the callers log.
`left`	A boolean expression with value **TRUE** if msg should be left justified. **FALSE** if msg should be indented six spaces.

Remarks

There are two primary uses for this statement. First and foremost, it allows you to keep the SysOp informed of what the user does while using your PPL application. Secondly, it can allow you to track information within your PPE while debugging.

Examples

```
BOOLEAN flag
PRINT "Type QUIT to exit..."
WAITFOR "QUIT",flag,60
IF (!flag) LOG "User did not type QUIT",FALSE
LOG "****EXITING PPE****",TRUE
```

See Also

DBGLEVEL *Statement*, **DBGLEVEL()** *Function*

LOGGEDON() *Function*

Function

Determine if a user has completely logged on to the BBS.

Syntax

LOGGEDON()

No arguments are required

Return Type and Value

BOOLEAN Returns **TRUE** if the user has completed logging in, **FALSE** otherwise.

Remarks

There are some features of PPL that are not available until the user has completely logged in, such as the user variables and functions and the **CALLNUM()** function. This function will allow you to detect whether or not a user has completely logged in and if selected PPL features are available.

Examples

```
IF (!LOGGEDON()) LOG "USER NOT LOGGED ON",0
```

See Also

CALLNUM() *Function*, **ONLOCAL()** *Function*, **U_LOGONS()** *Function*

LOGIT *Constant*

Function

Set the write prompt to callers log flag in a **DISPTEXT** statement.

Value

32768 = 1000000000000000b = 100000o = 8000h

Remarks

The **DISPTEXT** statement has the ability to write a specified prompt to the callers log automatically without the need to use the **LOG** statement. This flag will indent the prompt six spaces in the callers log.

Examples

```
DISPTEXT 4,LFBEFORE+LFAFTER+BELL+LOGIT
```

See Also

DISPTEXT *Statement*, **LOGITLEFT** *Constant*

LOGITLEFT *Constant*

Function

Set the write prompt to callers log left justified flag in a **DISPTEXT** statement.

Value

65536 = 1000000000000000b = 200000o = 10000h

Remarks

The **DISPTEXT** statement has the ability to write a specified prompt to the callers log automatically without the need to use the **LOG** statement. This flag will not indent the prompt in the callers log.

Examples

```
DISPTEXT 4,LFBEFORE+LFAFTER+BELL+LOGITLEFT
```

See Also

DISPTEXT *Statement*, **LOGIT** *Constant*

LOWER() *Function*

Function

Converts uppercase characters in a string to lowercase.

Syntax

LOWER(sexp)

sexp	Any string expression.

Return Type & Value

STRING	Returns **sexp** with all uppercase characters converted to lowercase.

Remarks

Although "STRING" is technically different from "string" (ie, the computer docsn't recognize them as being the same because one is uppercase and the other is lowercase), it is often necessary to save, display or compare information in a case insensitive format. This function will return a string with all uppercase characters converted to lowercase. So, using the above example, **LOWER("STRING")** would return "string".

Examples

```
STRING s
WHILE (UPPER(s) <> "QUIT") DO
  INPUT "Text",s
  PRINTLN LOWER(s)
ENDWHILE
```

See Also

UPPER() ***Function***

LTRIM() *Function*

Function

Trim a specified character from the left end of a string.

Syntax

```
LTRIM(str,ch)
```

str	Any string expression.
ch	A string with the character to strip from the left end of **str**.

Return Type & Value

STRING	Returns the trimmed **str**.

Remarks

A common need in programming is to strip leading and/or trailing spaces (or other characters). This function will strip a specified character from the left end of a string and return the trimmed string.

Examples

```
STRING s
LET s = "   TEST   "
PRINTLN LTRIM(s," ") ' Will print "TEST   "
PRINTLN LTRIM(".....DA"+"TA.....",".") ' Will print "DATA....."
PRINTLN LTRIM(".....DA"+"TA....."," ") ' Will print ".....DATA....."
```

See Also

RTRIM() *Function*, **TRIM()** *Function*

MASK_...() *Functions*

Function

Return a string for use as a valid character mask.

Syntax

MASK_...()

No arguments are required

MASK_ should be followed by one of the following mask types: ALNUM, ALPHA, ASCII, FILE, NUM, PATH, or PWD.

Return Type and Value

STRING Returns a string with a set of characters to use as valid input for an **INPUTSTR** or **PROMPTSTR** statement.

Remarks

There are many situations in which you will need to use an **INPUTSTR** or **PROMPTSTR** statement to access the input field length of flags. However, all you need to use a 'standard' set of input characters. These functions provide you with some of the most common valid character masks. They are: **MASK_ALNUM()** which returns A-Z, a-z, and 0-9; **MASK_ALPHA()** which returns A-Z and a-z; **MASK_ASCII()** which returns all characters from space (ASCII 32) to tilde (ASCII 126); **MASK_FILE()** which returns all legal file name characters; **MASK_NUM()** which returns 0-9; **MASK_PATH()** which returns all legal path name characters; and, finally, **MASK_PWD()** which returns a set of valid characters for use in passwords.

Examples

```
INTEGER i
STRING  s
INPUTSTR "Enter a number from 0 to 1000",i,@X0E,4,MASK_NUM(),DEFS
PROMPTSTR 148,s,12,MASK_PWD(),ECHODOTS
INPUTSTR "Enter your comment",s,@X0E,60,MASK_ASCII(),DEFS
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

MAXNODE() *Function*

Function

Determine how many nodes a system may have.

Syntax

MAXNODE ( )

No arguments are required

Return Type and Value

INTEGER Returns the node limit available to the system running the PPE file.

Remarks

Every package of PCBoard purchased comes with a license agreement that limits it to a maximum number of nodes. This node limit restricts various features of PCBoard, such as the WHO display and CHAT functions. This limit is available to your PPL applications via this function.

Examples

```
INTEGER i
FOR i = 1 TO MAXNODE()
  RDUNET i
  IF ((UN_STAT() = "A") | (UN_STAT() = "U")) THEN
    BROADCAST i,i,"Hello, how are you?"
    IF (PCBNODE() = i) PRINLN "Quit talking to yourself"
  ENDIF
NEXT
```

See Also

PCBNODE() ***Function***

MESSAGE *Statement*

Function

Enter a message under PPL control.

Syntax

```
MESSAGE conf,to,from,sub,sec,pack,rr,echo,file
```

conf	An integer expression with the conference in which to post the message.
to	A string expression with the user name to which the message should be sent.
from	A string expression with the user name that the message should be sent from.
sub	A string expression with the subject of the message.
sec	A string expression with the desired security for the message ("N"for none or "R" for receiver only).
pack	A date expression with the packout date for the message (or 0 for no packout date).
rr	A boolean expression with the return receipt requested flag (**TRUE** to request a return receipt, **FALSE** otherwise).
echo	A boolean expression with the echo flag (**TRUE** to echo the message, **FALSE** otherwise).
file	A string expression with the path and file name of the text file to use as the message text.

Remarks

This statement will allow you to leave a message from any user (or any 'name' you wish to use) to any user on your system. This can be useful if you want to notify a user of information that they should download in a QWK packet or that they might miss too easily as a quick one liner on screen from the PPL.

Examples

```
IF (CURSEC() < 20) THEN
  MESSAGE 0,U_NAME(),"SYSOP","REGISTER","R",DATE(),TRUE,FALSE,"REG.TXT"
ENDIF
```

See Also

CURCONF() *Function*, U_NAME() *Function*

MGETBYTE() *Function*

Function

Get the next byte input from the modem.

Syntax

MGETBYTE()

No arguments are required

Return Type and Value

INTEGER Returns the value (0-255) of the next byte from the modem input buffer or -1 if no bytes are pending.

Remarks

Any character may be received from the users modem. Normally PCBoard will filter and convert strings (ESC sequences and DOORWAY codes) automatically. However, sometimes this isn't desired and you need to access the incoming bytes directly. This function will look directly for incoming characters from the modem and return them as a value from 0 to 255. These numbers may be converted to characters with the **CHR()** function if necessary.

Examples

```
INTEGER byte
WHILE (byte <> 27) DO
  LET byte = MGETBYTE()
  PRINTLN "The byte value is ",byte
ENDWHILE
```

See Also

INKEY() ***Function*****, KINKEY()** ***Function*****, MINKEY()** ***Function***

MID() *Function*

Function

Access any sub string of a string.

Syntax

MID(str,pos,chars)

str	A string expression to take the left most characters of.
pos	An integer expression with the position within **str** to start taking the sub string from.
chars	An integer expression with the number of characters to take from **str**.

Return Type & Value

STRING	Returns a string with the specified number of characters from the specified position of **str**.

Remarks

This function will return a sub string with the specified number of characters and from the specified position of **str**. This can be useful in data processing as well as text formatting. The **pos** parameter may be less than 1 (the beginning of **str**) and greater than the length of **str**; if it is spaces will be added to the beginning and/or ending as needed. If **chars** is less than or equal to 0 then the returned string will be empty. If **chars** is greater than the available length of **str** then the returned string will have spaces added to the end(s) to pad it out to the full length specified.

Examples

```
WHILE (RANDOM(250) <> 0) PRINT MID(RANDOM(250),0,4)," "

STRING s
FOPEN 1,"DATA.TXT",O_RD,S_DN
WHILE (!FERR(1)) DO
  FGET 1,s
  PRINT LEFT(s,5),RTRIM(MID(s,5,20)," "),"  - "
  PRINTLN RTRIM(MID(s,LEN(s)-25,60)," ")
ENDWHILE
FCLOSE 1
```

See Also

LEFT() ***Function*****, RIGHT()** ***Function***

MIN() *Function*

Function

Extract the minute of the hour from a specified time of day.

Syntax

```
MIN(texp)
```

texp	Any time expression.

Return Type & Value

INTEGER	Returns the minute of the hour from the specified time expression (**texp**). Valid return values are from 0 to 59.

Remarks

This function allows you to extract a particular piece of information about a **TIME** value, in this case the minute of the hour of the time of day expression.

Examples

```
PRINTLN "The minute is ",MIN(TIME())
```

See Also

HOUR() ***Function***, **SEC()** ***Function***, **TIME()** ***Function***

MINKEY() *Function*

Function

Get the next key input from the modem only.

Syntax

MINKEY()

No arguments are required

Return Type and Value

STRING Returns a single character for displayable characters or a string for cursor movement and function keys.

Remarks

This function will return a single character long string for most key presses. Additionally, it will return key names for function keys and cursor movement keys if it encounters ESC sequences or DOORWAY codes. It will only return keystrokes from the remote users modem.

Examples

```
STRING key
WHILE (key <> CHR(27)) DO
  LET key = MINKEY()
  IF (LEFT(key,5) = "SHIFT") THEN
    PRINTLN "It was a shifted key"
  ELSEIF (LEFT(key,4) = "CTRL") THEN
    PRINTLN "It was a control key"
  ELSEIF (LEFT(key,3) = "ALT") THEN
    PRINTLN "It was an alternate key"
  ENDIF
  PRINTLN "The key was ",key
ENDWHILE
```

See Also

INKEY() *Function*, **KINKEY()** *Function*, **MGETBYTE()** *Function*

MINLEFT() *Function*

Function

Return the users minutes left.

Syntax

```
MINLEFT()
```

No arguments are required

Return Type and Value

INTEGER Returns the number of minutes the user online has left to use.

Remarks

This function will allow you to access how much time the user has remaining. You could use it to disable certain features at a certain point in their session. Note that this number can be either the minutes left today or this session if the SysOp does not enforce daily time limits.

Examples

```
IF (MINLEFT() > 10) THEN
  KBDSTUFF "D"+CHR(13)
ELSE
  PRINTLN "Sorry, not enough time left to download"
ENDIF
```

See Also

ADJTIME() *Function*, **MINON**() *Function*, **U_TIMEON**() *Function*

MINON() *Function*

Function

Return the users minutes online.

Syntax

MINON ( )

No arguments are required

Return Type and Value

INTEGER Returns the number of minutes the user online has used this session.

Remarks

This function will allow you to access how much time the user has used this session. You could use it to allow or disallow certain features before a certain point in their session. Note that this number will always be the minutes used this session regardless of whether or not the SysOp enforces daily time limits.

Examples

```
IF (MINON() >= 10) THEN
  KBDSTUFF "D"+CHR(13)
ELSE
  PRINTLN "Sorry, you haven't been on long enough yet to download"
ENDIF
```

See Also

ADJTIME() ***Function*****, MINLEFT()** ***Function*****, U_TIMEON()** ***Function***

MODEM() *Function*

Function

Access the connect string as reported by the modem.

Syntax

MODEM()

No arguments are required

Return Type & Value

STRING Returns the modem connect string.

Remarks

PCBoard expects and requires certain information to be reported by the modem anytime a user connects to the BBS. The minimum requirement is a string with the word CONNECT; other information may be included, such as the connect speed, error correction, data compression, etc. Should your PPL application have need of this information as well, it may be accessed with this function.

Examples

```
FAPPEND 1,"MODEM.LOG",O_WR,S_DW
FPUTLN 1,LEFT(U_NAME(),30)+MODEM()
FCLOSE 1
```

See Also

CALLID() ***Function*****, CARRIER()** ***Function***

MONEY *Type*

Function

Declare one or more variables of type money.

Syntax

```
MONEY var|arr(s[,s[,s]])[,var|arr(s[,s[,s]])]
```

`var`	The name of a variable to declare. Must start with a letter [A-Z] which may be followed by letters, digits [0-9] or the underscore [_]. May be of any length but only the first 32 characters are used.
`arr`	The name of an array variable to declare. The same naming conventions as **var** are used.
`s`	The size (0-based) of an array variable dimension. Any constant integer expression is allowed.

Remarks

MONEY variables are stored as positive or negative cents. The range of **MONEY** is $-21,474,836.48 through $+21,474,836.47. It is stored internally as a four byte signed long integer. If **MONEY** is assigned to or from an **INTEGER** type then the cents (-2,147,483,648 - +2,147,483,647) are assigned. If **MONEY** is assigned to a **STRING** type then it is automatically converted to the following format: "$sD.CC", where s is the sign (- for negative amounts, nothing for positive amounts), D is the dollar amount (one or more digits as needed) and CC is the cents amount (00-99). If a **STRING** is assigned to **MONEY** then PPL will do it's best to convert the string back to the appropriate amount of money. All other types, when assigned to or from **MONEY**, will be converted to an **INTEGER** first before being assigned to or from the **MONEY** type.

Examples

```
MONEY itemAmt, subTot, total, priceList(2,17)
```

See Also

BOOLEAN *Type*, **DATE** *Type*, **INTEGER** *Type*, **STRING** *Type*, **TIME** *Type*

MONTH() *Function*

Function

Extracts the month of the year from a specified date.

Syntax

MONTH(dexp)

dexp	Any date expression.

Return Type & Value

INTEGER	Returns the month from the specified date expression (**dexp**). Valid return values are from 1 to 12.

Remarks

This function allows you to extract a particular piece of information about a **DATE** value, in this case the month of the date.

Examples

```
PRINTLN "This month is:  ",MONTH(DATE())
```

See Also

DATE() *Function*, **DAY**() *Function*, **DOW**() *Function*, **YEAR**() *Function*

MORE *Statement*

Function

Pause the display and ask the user how to continue.

Syntax

MORE

No arguments are required

Remarks

It is often necessary to pause in the display of information and wait for the user to catch up. This statement allows you prompt the user on how to continue. The acceptable responses are Y (or whatever letter is appropriate for the users language selection) to continue, N (or, again, whatever letter is appropriate) to abort, or NS to continue in non-stop mode. It displays prompt number 196 from the PCBTEXT file for the current language to let the user know what is expected.

Examples

```
PRINTLN "Your account has expired!"
PRINTLN "You are about to be logged off"
MORE
PRINTLN "Call me voice to renew your subscription"
```

See Also

ABORT() *Function*, **DISPTEXT** *Statement*, **INKEY()** *Function*, **PROMPTSTR** *Statement*, **WAIT** *Statement*

MPRINT/MPRINTLN *Statements*

Function

Print (write) a line to the caller's screen (modem) only (with an optional newline appended).

Syntax

```
MPRINT exp[,exp]
-or-
MPRINTLN [exp[,exp]]
```

`exp` An expression of any type to evaluate and write to the caller's screen.

Remarks

These statements will evalutate zero, one or more expressions of any type and write the results to the modem for the caller's display. The **MPRINTLN** statement will append a newline to the end of the expressions; **MPRINT** will not. Note that at least one expression must be specified for **MPRINT**, unlike the **MPRINTLN** statement which need not have any arguments passed to it. These statements only send information to the modem and do not interpret @ codes; if the remote caller has ANSI then ANSI will be interpreted.

Examples

```
MPRINT "The name of the currently running PPE file is "
MPRINTLN PPENAME(),"."
MPRINT "The path where it is located is "
MPRINTLN PPEPATH(),"."
MPRINT "The date is ",DATE()," and the time is ",TIME(),"."
MPRINTLN
```

See Also

PRINT/PRINTLN *Statements*, **SPRINT/SPRINTLN** *Statements*

NC *Constant*

Function

To re-start the display of information according to the current mode.

Value

0 = 0b = 0o = 0h

Remarks

The **STARTDISP** statement takes a single argument to start displaying information in a certain format. **FCL** tells PCBoard to count lines and pause as needed during the display of information. **FNS** tells PCBoard to not stop during the display of information. **NC** instructs PCBoard to start over with the last specified mode (**FCL** or **FNS**).

Examples

```
INTEGER i,j
STARTDISP FCL
FOR i = 1 TO 5
  STARTDISP NC
  FOR j = 1 to 50
    PRINTLN "This is line ",j
  NEXT
NEXT
```

See Also

FCL *Constant*, **FNS** *Constant*

NEWLINE *Constant*

Function

Set the new line after prompt flag in an **INPUTSTR**, **PROMPTSTR**, or **DISPTEXT** statement.

Value

64 = 1000000b = 100o = 40h

Remarks

The **INPUTSTR**, **PROMPTSTR**, and **DISPTEXT** statements have the ability to send a carriage return/line feed after a prompt is displayed automatically and without the need to make a separate call to the **NEWLINE** statement.

Examples

```
STRING pwd
INPUTSTR "Enter id",pwd,0X0E,4,"0123456789",LFBEFORE+NEWLINE+LFAFTER
IF (pwd <> "1234") PRINTLN "Bad id"
```

See Also

DISPTEXT *Statement*, **INPUTSTR** *Statement*, **LFAFTER** *Constant*, **LFBEFORE** *Constant*, **PROMPTSTR** *Statement*

NEWLINE *Statement*

Function

Move the cursor to the beginning of the next line.

Syntax

NEWLINE

No arguments are required

Remarks

This statement should be used for moving to the beginning of the next line on screen, scrolling if necessary. It will do so regardless of the current cursor position, unlike the **FRESHLINE** statement.

Examples

```
INTEGER i, end
LET end = RANDOM(20)
FOR i = 1 TO end
   PRINT RIGHT(RANDOM(10000),8)
NEXT
FRESHLINE
NEWLINE
PRINTLN "Now we continue with a blank line between"
```

See Also

FRESHLINE *Statement*, **NEWLINES** *Statement*

NEWLINES *Statement*

Function

Execute a specified number of **NEWLINE** statements.

Syntax

```
NEWLINES count
```

`count` An integer expression with the number of times to execute **NEWLINE**.

Remarks

This statement is convienient when executing multiple and/or variable **NEWLINE** statements for screen formatting. It takes a single integer expression argument and automatically executes that many **NEWLINE** statements for you without the need to set up a loop or to write multiple **NEWLINE** lines in your source code.

Examples

```
INTEGER i, end
LET end = RANDOM(20)
FOR i = 1 TO end
  PRINT RIGHT(RANDOM(10000),8)
NEXT
FRESHLINE
NEWLINE 5
PRINTLN "Now we continue with a 5 blank lines between"
```

See Also

FRESHLINE *Statement*, **NEWLINE** *Statement*

NEWPWD *Statement*

Function

Change the users password and maintain the password PSA if installed.

Syntax

NEWPWD pwd,var

pwd	A string expression with the new password for the user.
var	A variable to hold the password change status: **TRUE** if the password was changed or **FALSE** otherwise.

Remarks

There are two ways to change the users password under PPL control. The first is to simply use the **GETUSER** statement, assign the new password to the **U_PWD** variable, then issue the **PUTUSER** statement. However, this isn't adequate if the SysOp has installed the password PSA. This statement will take care of validating the password, checking it against the password history, updating the password history, setting a new expiration date if necessary and incrementing the times changed counter. If the password fails a validity test then this statement will set the **var** parameter to **FALSE** to let you know that the password wasn't changed. If the password PSA isn't installed or if the password conforms to the PSA requirements, then **var** will be set to **TRUE**.

Examples

```
BOOLEAN changed
STRING  pwd
INPUTSTR "Enter a new password",pwd,@X0E,12,MASK_PWD(),ECHODOTS
NEWLINE
NEWPWD pwd,changed
IF (!changed) PRINTLN "Password not changed"
```

See Also

MASK_PWD() ***Function***, **U_PWD** ***Variable***, **U_PWDEXP** ***Variable***, **U_PWDHIST()** ***Function***, **U_PWDLC()** ***Function***, **U_PWDTC()** ***Function***

NOCHAR() *Function*

Function

Get the no response character for the current language.

Syntax

NOCHAR ( )

No arguments are required

Return Type & Value

STRING Returns the no character for the current language.

Remarks

Support for foreign language yes/no responses can be easily added by using this function to determine what a negative response should be instead of hardcoding the english "N" character.

Examples

```
STRING ans
LET ans = YESCHAR()
INPUTSTR "Run program now",ans,@X0E,1,"",AUTO+YESNO
IF (ans = NOCHAR()) END
```

See Also

YESCHAR() *Function*, **YESNO** *Constant*

NOCLEAR *Constant*

Function

Set the no clear input field flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

1024 = 10000000000b = 2000o = 400h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to automatically clear the default value from the input field when the users presses his first key if ANSI is available. This is the default mode of operations. If you don't want this to happen, you may use this flag to disable this feature.

Examples

```
STRING cmds
LET cmds = "QUIT"
INPUTSTR "Commands",cmds,@X0E,60,MASK_ASCII(),STACKED+NOCLEAR
TOKENIZE cmds
LET cmds = GETTOKEN()
IF (cmds = "QUIT") END
KBDSTUFF cmds+TOKENSTR()
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

NOT() *Function*

Function

Calculate the bitwise NOT of an integer argument.

Syntax

```
NOT(iexp)
```

iexp	Any integer expression.

Return Type & Value

INTEGER	Returns the bitwise NOT of **iexp**.

Remarks

This function may be used to toggle all bits in an integer expression. Wherever a bit had been set it will be clear after this function call, and vice versa.

Examples

```
' Toggle the bits
PRINTLN NOT(1248h)
' Toggle all flag
INTEGER flag
LET flag = NOT(flag)
```

See Also

AND() *Function*, **OR()** *Function*, **XOR()** *Function*

ONLOCAL() *Function*

Function

Determine whether or not a caller is on locally.

Syntax

ONLOCAL()

No arguments are required

Return Type & Value

BOOLEAN Returns **TRUE** if the caller is on locally, **FALSE** otherwise.

Remarks

There are some features that work differently for local and remote callers, such as file transfers and modem communications. This function will report to you whether or not a user is logged on locally and allow you to handle local processing differently than remote processing.

Examples

```
IF (ONLOCAL()) THEN
  PRINTLN "Call back verification cannot be performed for"
  PRINTLN "users logged in locally!"
  END
ENDIF
CALL "CALLBACK.PPE"
```

See Also

CALLNUM() *Function*, **LOGGEDON()** *Function*, **U_LOGONS()** *Function*

OPENCAP *Statement*

Function

Open the screen capture file.

Syntax

```
OPENCAP file,stat
```

`file`	A string expression with the file name to open.
`stat`	A variable to hold the return status (**TRUE** if error opening file, **FALSE** otherwise).

Remarks

PCBoard has the ability to capture screen output to a file for later reference. PPL allows that same ability via the **OPENCAP** and **CLOSECAP** statements. This could be useful in a program that executes a series of commands in non-stop mode. The process could open a capture file first, execute the commands, close the capture file, then allow the user to view or download the capture file. **CLOSECAP** closes the capture file and turns off screen capturing. Also, the **SHOWON** and **SHOWOFF** statements can be used to turn on and off showing information to the screen while allowing that same information (even if not displayed or transmitted via modem) to be captured to a file. The **SHOWSTAT**() function can be used to check the current status of the **SHOWON** and **SHOWOFF** statements.

Examples

```
BOOLEAN ss
LET ss = SHOWSTAT()
SHOWOFF
OPENCAP "CAP"+STRING(PCBNODE()),ocFlag
IF (ocFlag) THEN
  DIR "U;NS"
  CLOSECAP
  KBDSTUFF "FLAG CAP"+STRING(PCBNODE())+CHR(13)
ENDIF
IF (ss) THEN
  SHOWON
ELSE
  SHOWOFF
ENDIF
```

See Also

CLOSECAP *Statement*, **SHOWOFF** *Statement*, **SHOWON** *Statement*, **SHOWSTAT**() *Function*

OPTEXT *Statement*

Function

Set the text to be used by the @OPTEXT@ macro.

Syntax

```
OPTEXT str
```

str	Any string expression.

Remarks

The @OPTEXT@ macro is used to include operation specific text in prompts and display files. Normally PCBoard automatically fills it in with the appropriate value. However, you can use it for your own purposes by issuing this statement to set the text and immediately displaying the information that should use it (by either printing a line or displaying a file).

Examples

```
OPTEXT STRING(DATE())+" & "+STRING(TIME())
PRINTLN "The date and time are @OPTEXT@"
DISPFILE "FILE",GRAPH+SEC+LANG
```

See Also

DISPFILE *Statement*, **DISPSTR** *Statement*, **DISPTEXT** *Statement*, **PRINT/PRINTLN** *Statements*

OR() *Function*

Function

Calculate the bitwise OR of two integer arguments.

Syntax

```
OR(iexp1,iexp2)
```

iexp1	Any integer expression.
iexp2	Any integer expression.

Return Type & Value

INTEGER	Returns the bitwise OR of **iexp1** and **iexp2**.

Remarks

This function may be used to set selected bits in an integer expression by ORing the expression with a mask that has the bits to set set to 1 and the bits to ignore set to 0.

Examples

```
' Set the bits in the low byte
PRINTLN OR(1248h,00FFh)
' Randomly set a flag the hard way
INTEGER flag
LET flag = OR(RANDOM(1),RANDOM(1))
```

See Also

AND() ***Function*****, NOT()** ***Function*****, XOR()** ***Function***

O_RD *Constant*

Function

Set the open for read access flag in a **FCREATE/FOPEN/FAPPEND** statement.

Value

0 = 0b = 0o = 0h

Remarks

Files may be opened for read, write or combined read/write access. You should only use the access you need to allow other processes to open files at the same time in multitasking and networked environments. This constant will allow your PPE to read from a file without writing any information out to it.

Examples

```
FOPEN 1,"FILE.DAT",O_RD,S_DN ' Open for read access
FOR i = 1 TO 10
  FGET 1,s
  PRINTLN s
NEXT
FCLOSE 1
```

See Also

O_RW ***Constant*****, O_WR** ***Constant***

O_RW *Constant*

Function

Set the open for read and write access flag in a **FCREATE/FOPEN/FAPPEND** statement.

Value

2 = 10b = 2o = 2h

Remarks

Files may be opened for read, write or combined read/write access. You should only use the access you need to allow other processes to open files at the same time in multitasking and networked environments. This constant will allow your PPE to both read from and write to a file without the need to close and reopen it between accesses.

Examples

```
FOPEN 1,"FILE.DAT",O_RW,S_DN ' Open for read and write access
FOR i = 1 TO 10
  FPUT 1,"X"
  FGET 1,s
  PRINTLN s
NEXT
FCLOSE 1
```

See Also

O_RD *Constant*, **O_WR** *Constant*

O_WR *Constant*

Function

Set the open for write access flag in a **FCREATE/FOPEN/FAPPEND** statement.

Value

1 = 1b = 1o = 1h

Remarks

Files may be opened for read, write or combined read/write access. You should only use the access you need to allow other processes to open files at the same time in multitasking and networked environments. This constant will allow your PPE to write to a file but will restrict read access.

Examples

```
FOPEN 1,"FILE.DAT",O_WR,S_DN ' Open for write access
FOR i = 1 TO 10
  FPUTLN 1,"Line ",i
NEXT
FCLOSE 1
```

See Also

O_RD *Constant*, **O_RW** *Constant*

PAGEOFF *Statement*

Function

Turn off the SysOp paged indicator.

Syntax

PAGEOFF

No arguments are required

Remarks

One of the features of PCBoard where change is often requested is the operator page facility. Some people want to be able to configure multiple ranges of availability per day, some want a different sounding page bell, longer or shorter page attempts, etc, etc. This statement, along with the **CHAT** and **PAGEON** statements and the **PAGESTAT()** function, allow you to implement an operator page in any way desired.

Examples

```
PAGEON
FOR i = 1 TO 10
  PRINT "@BEEP@"
  DELAY 18
  IF (KINKEY() = " ") THEN
    PAGEOFF
    SHELL TRUE,i,"SUPERCHT",""
    GOTO exit
  ENDIF
NEXT
:exit
```

See Also

CHAT *Statement*, **PAGEON** *Statement*, **PAGESTAT()** *Function*

PAGEON *Statement*

Function

Turn on the SysOp paged indicator and update user statistics.

Syntax

PAGEON

No arguments are required

Remarks

One of the features of PCBoard where change is often requested is the operator page facility. Some people want to be able to configure multiple ranges of availability per day, some want a different sounding page bell, longer or shorter page attempts, etc. etc. This statement, along with the **CHAT** and **PAGEOFF** statements and the **PAGESTAT()** function, allow you to implement an operator page in any way desired. Note that this statement will also update the current callers statistics PSA if it is installed.

Examples

```
PAGEON
FOR i = 1 TO 10
  PRINT "@BEEP@"
  DELAY 18
  IF (KINKEY() = " ") THEN
    CHAT
    GOTO exit
  ENDIF
NEXT
:exit
```

See Also

CHAT *Statement*, **PAGEOFF** *Statement*, **PAGESTAT()** *Function*

PAGESTAT() *Function*

Function

Determine if the current user has paged the SysOp.

Syntax

PAGESTAT()

No arguments are required

Return Type and Value

BOOLEAN Returns **TRUE** if the user has paged the SysOp, **FALSE** otherwise.

Remarks

One of the features of PCBoard where change is often requested is the operator page facility. Some people want to be able to configure multiple ranges of availability per day, some want a different sounding page bell, longer or shorter page attempts, etc, etc. This function, along with the **CHAT**, **PAGEON** and **PAGEOFF** statements, allow you to implement an operator page in any way desired.

Examples

```
IF (PAGESTAT()) THEN
  PRINTLN "You have already paged the SysOp,"
  PRINTLN "please be patient."
ELSE
  PAGEON
  PRINTLN "The SysOp has been paged, continue"
ENDIF
```

See Also

CHAT *Statement*, **PAGEOFF** *Statement*, **PAGEON** *Statement*

PCBDAT() *Function*

Function

Return the path and file name of the PCBOARD.DAT file.

Syntax

PCBDAT()

No arguments are required

Return Type & Value

STRING Returns the path and file name of the PCBOARD.DAT file for the currnet node.

Remarks

The PCBOARD.DAT file is the master confiuration file for each node running PCBoard. As such, there are many useful pieces of information that can be obtained from it. It is a standard text file with one piece of information per line. You may use the **READLINE()** function to read individual pieces of information from it.

Examples

```
STRING s
LET s = READLINE(PCBDAT(),1)
PRINTLN "PCBOARD.DAT version info - ",s
```

See Also

GETENV() ***Function*****, READLINE()** ***Function***

PCBNODE() *Function*

Function

Return the current node number.

Syntax

```
PCBNODE()
```

No arguments are required

Return Type & Value

INTEGER Returns the node number for the current node.

Remarks

You may have need to know what node is in use for certain applications (for example, to create temporary files with unique names or to restrict features to a particular node or nodes). This function will return a number from 1 to the maximum number of nodes allowed with a given copy of PCBoard. Note that the node number may not be what is defined in PCBOARD.DAT if the /FLOAT or /NODE switches are used.

Examples

```
STRING file
LET file = "TMP"+STRING(PCBNODE())+".$$$"
DELETE file
```

See Also

MAXNODE() ***Function***

PEEKB() *Function*

Function

Return the value of a byte at a specified memory address.

Syntax

```
PEEKB(addr)
```

addr	An integer expression with the address of the byte to peek.

Return Type & Value

INTEGER	Returns the value of the byte at **addr**.

Remarks

It is sometimes necessary to read values from memory directly (for example, from the system BIOS data segment). This function will return a byte quantity (0-255) from a specified memory address.

Examples

```
PRINTLN "The current video mode is ",PEEKB(MKADDR(40h,49h))
```

See Also

MKADDR() *Function*, **PEEKDW()** *Function*, **PEEKW()** *Function*, **POKEB()** *Function*, **POKEDW()** *Function*, **POKEW()** *Function*, **VARADDR** *Statement*, **VAROFF** *Statement*, **VARSEG** *Statement*

PEEKDW() *Function*

Function

Return the value of a double word at a specified memory address.

Syntax

```
PEEKDW(addr)
```

addr	An integer expression with the address of the double word to peek.

Return Type & Value

INTEGER	Returns the value of the double word at **addr**.

Remarks

It is sometimes necessary to read values from memory directly (for example, from the system BIOS data segment). This function will return a double word quantity as a signed integer (-2,147,483,648 - +2,147,483,647) from a specified memory address.

Examples

```
PRINTLN "Timer ticks since midnight = ",PEEKDW(MKADDR(40h,6Ch))
```

See Also

MKADDR() *Function*, **PEEKB()** *Function*, **PEEKW()** *Function*, **POKEB()** *Function*, **POKEDW()** *Function*, **POKEW()** *Function*, **VARADDR** *Statement*, **VAROFF** *Statement*, **VARSEG** *Statement*

PEEKW() *Function*

Function

Return the value of a word at a specified memory address.

Syntax

```
PEEKW(addr)
```

addr	An integer expression with the address of the word to peek.

Return Type & Value

INTEGER Returns the value of the word at **addr**.

Remarks

It is sometimes necessary to read values from memory directly (for example, from the system BIOS data segment). This function will return a word quantity (0-65,535) from a specified memory address.

Examples

```
PRINTLN "The usable memory size is ",PEEKW(MKADDR(40h,13h))
```

See Also

MKADDR() *Function*, **PEEKB()** *Function*, **PEEKDW()** *Function*, **POKEB()** *Function*, **POKEDW()** *Function*, **POKEW()** *Function*, **VARADDR** *Statement*, **VAROFF** *Statement*, **VARSEG** *Statement*

POKEB *Statement*

Function

Write a byte to a specified memory address.

Syntax

```
POKEB addr,value
```

`addr`	An integer expression with the address to write to.
`value`	An integer expression with the value to write to **addr**.

Remarks

You may have need to write directly to memory from time to time. This statement complements the **PEEKB**() function and allows you to write a byte value (0-255) to a specific memory location.

Examples

```
BOOLEAN flag
INTEGER addr
VARADDR flag,addr
POKEB addr,TRUE ' Set the flag to TRUE the hard way
```

See Also

MKADDR() *Function*, **PEEKB**() *Function*, **PEEKDW**() *Function*, **PEEKW**() *Function*, **POKEDW**() *Function*, **POKEW**() *Function*, **VARADDR** *Statement*, **VAROFF** *Statement*, **VARSEG** *Statement*

POKEDW *Statement*

Function

Write a double word to a specified memory address.

Syntax

```
POKEDW addr,value
```

`addr`	An integer expression with the address to write to.
`value`	An integer expression with the value to write to **addr**.

Remarks

You may have need to write directly to memory from time to time. This statement complements the **PEEKDW()** function and allows you to write a double word value (-2,147,483,648 - +2,147,483,647) to a specific memory location.

Examples

```
MONEY   amt
INTEGER addr
VARADDR amt,addr
POKEDW addr,123456 ' Set amt to $1234.56 the hard way
```

See Also

MKADDR() *Function*, **PEEKB()** *Function*, **PEEKDW()** *Function*, **PEEKW()** *Function*, **POKEB()** *Function*, **POKEW()** *Function*, **VARADDR** *Statement*, **VAROFF** *Statement*, **VARSEG** *Statement*

POKEW *Statement*

Function

Write a word to a specified memory address.

Syntax

```
POKEW addr,value
```

`addr`	An integer expression with the address to write to.
`value`	An integer expression with the value to write to **addr**.

Remarks

You may have need to write directly to memory from time to time. This statement complements the **PEEKW()** function and allows you to write a word value (0-65,535) to a specific memory location.

Examples

```
DATE    dob
INTEGER addr
VARADDR dob,addr
POKEW addr,MKDATE(1967,10,31) ' Set dob the hard way
```

See Also

MKADDR() *Function*, **PEEKB()** *Function*, **PEEKDW()** *Function*, **PEEKW()** *Function*, **POKEB()** *Function*, **POKEDW()** *Function*, **VARADDR** *Statement*, **VAROFF** *Statement*, **VARSEG** *Statement*

POP *Statement*

Function

Pop the results of one or more expressions from a stack.

Syntax

```
POP var[,var]
```

`var` A variable of any type in which to retrieve previously pushed expression.

Remarks

This statement will retrieve the results of one or more expressions of any type from a stack into a list of variables. The values should have been previously pushed with the **PUSH** statement. Together **PUSH** and **POP** can be used for parameter passing, to create 'local' variables, or to reverse the order of arguments.

Examples

```
INTEGER i, tc
STRING  s
LET tc = TOKCOUNT()
WHILE (TOKCOUNT() > 0) PUSH GETTOKEN() ' push them in order
FOR i = 1 TO tc
  POP s                                ' pop them in reverse
  PRINTLN s
NEXT

INTEGER i
FOR i = 1 TO 10
  PRINT i," - "
  GOSUB sub
NEXT
END
:sub
PUSH i                                 ' temporarily save i
LET i = i*i
PRINTLN i
POP i                                  ' restore saved i
RETURN

INTEGER v
PRINT "A cube with dimensions 2X3X4"
PUSH 2,3,4                             ' pass pushed parameters
GOSUB vol
POP v                                  ' pop result
PRINTLN "has volume ",v
END
:vol
INTEGER w,h,d
POP d,h,w                              ' pop passed parameter
PUSH w*h*d                             ' push result
RETURN
```

See Also

PUSH *Statement*

PPENAME() *Function*

Function

Return the base name of an executing PPE file.

Syntax

PPENAME ( )

No arguments are required

Return Type & Value

STRING Returns the base file name (without path or extension) of the currently executing PPE.

Remarks

This function will return the name of the PPE file that is running. This can be useful when writing PPL applications that will use data files that you would like to keep named the same as the parent application regardless of what the PPE name may change to.

Examples

```
STRING s
FOPEN 1,PPEPATH()+PPENAME()+".CFG",O_RD,S_DN
FGET 1,s
FCLOSE 1
```

See Also

PPEPATH() *Function*

PPEPATH() *Function*

Function

Return the path of an executing PPE file.

Syntax

```
PPEPATH()
```

No arguments are required

Return Type & Value

STRING Returns the path (without file name or extension) of the currently executing PPE.

Remarks

This function will return the path of the PPE file that is running. This can be useful when writing PPL applications that will use files that you would like to keep in the same location as the parent application regardless of where the PPE may be installed.

Examples

```
STRING s
FOPEN 1,PPEPATH()+PPENAME()+".CFG",O_RD,S_DN
FGET 1,s
FCLOSE 1
```

See Also

HELPPATH() *Function*, PPENAME() *Function*, SLPATH() *Function*, TEMPPATH() *Function*

PRINT/PRINTLN *Statements*

Function

Print (write) a line to the screen (with an optional newline appended).

Syntax

```
PRINT exp[,exp]
-or-
PRINTLN [exp[,exp]]
```

exp An expression of any type to evaluate and write to the screen.

Remarks

These statements will evalutate zero, one or more expressions of any type and write the results to the display. The **PRINTLN** statement will append a newline to the end of the expressions; **PRINT** will not. Note that at least one expression must be specified for **PRINT**, unlike the **PRINTLN** statement which need not have any arguments passed to it. Finally, both statements will process all @ codes and display them as expected.

Examples

```
PRINT "The name of the currently running PPE file is "
PRINTLN PPENAME(),"."
PRINT "The path where it is located is "
PRINTLN PPEPATH(),"."
PRINT "The date is ",DATE()," and the time is ",TIME(),"."
PRINTLN
PRINT "@X1FThis is bright white on blue..."
PRINTLN "how do you like it @FIRST@"
```

See Also

MPRINT/MPRINTLN *Statements*, **OPTEXT** *Statement*, **SPRINT/SPRINTLN** *Statements*

PROMPTSTR *Statement*

Function

Prompt the user for a string of text in a specific format.

Syntax

```
PROMPTSTR prompt,var,len,valid,flags
```

`prompt`	An integer expression with the prompt number from PCBTEXT to display to the user.
`var`	The variable in which to store the user's input.
`len`	An integer expression with maximum length of text to input.
`valid`	A string expression with the valid characters that the user may enter.
`flags`	An integer expression with flags to modify how the statement works.

Remarks

This statement will accept a string of input from the user, up to the length defined. The prompt parameter will be used to find the prompt from PCBTEXT (which includes the prompt color) to display to the user. Only characters found in the valid parameter will be accepted. However, the flags parameter may affect how prompt is displayed and the valid characters that are accepted. Individual flags may be added together as needed. Several functions exist to easily specify commonly used valid character masks. They are **MASK_ALNUM()**, **MASK_ALPHA()**, **MASK_ASCII()**, **MASK_FILE()**, **MASK_NUM()**, **MASK_PATH()**, and **MASK_PWD()**. Defined flag values are **AUTO**, **DEFS**, **ECHODOTS**, **ERASELINE**, **FIELDLEN**, **GUIDE**, **HIGHASCII**, **LFAFTER**, **LFBEFORE**, **NEWLINE**, **NOCLEAR**, **STACKED**, **UPCASE**, **WORDWRAP**, and **YESNO**.

Examples

```
BOOLEAN b
DATE    d
INTEGER i
MONEY   m
STRING  s
TIME    t
' NOTE:  prompt 706 is used here for all statements;
'        you may use any prompt you wish
PROMPTSTR 706,b,1,"10",LFBEFORE+NEWLINE
PROMPTSTR 706,d,8,"0123456789-",NEWLINE+NOCLEAR
PROMPTSTR 706,i,20,MASK_NUM(),NEWLINE
PROMPTSTR 706,m,9,MASK_NUM()+".",NEWLINE+DEFS+FIELDLEN
PROMPTSTR 706,s,63,MASK_ASCII(),NEWLINE+FIELDLEN+GUIDE
PROMPTSTR 706,t,5,"0123456789"+":",NEWLINE+LFAFTER
PRINTLN b," ",d," ",i
PRINTLN m," ",s," ",t
```

See Also

INPUT *Statement*, **INPUT...** *Statements*, **INPUTSTR** *Statement*, **INPUTTEXT** *Statement*

PSA() *Function*

Function

Determine whether or not a given PSA is installed.

Syntax

```
PSA(num)
```

num	An integer expression with the number of the PSA to check for the existence of.

Return Type & Value

BOOLEAN	Returns **TRUE** if the specified PSA exists or **FALSE** if it doesn't exist for the following values of **num**:
1	The Alias PSA;
2	The Verification PSA.
3	The Address PSA;
4	The Password PSA;
5	The Statistics PSA.
6	The Notes PSA.

Remarks

This function allows you to determine whether or not a given PCBoard Supported Allocation (PSA) is installed. For each of the six PSAs it will return **TRUE** if installed or **FALSE** if not installed. It is useful when you want to write a generic PPL application that will access one or more PSAs that may or may not be installed.

Examples

```
STRING ynStr(1)
LET ynStr(0) = "NO"
LET ynStr(1) = "YES"
PRINTLN "       Alias Support Enabled?  ",ynStr(PSA(1))
PRINTLN "Verification Support Enabled?  ",ynStr(PSA(2))
PRINTLN "     Address Support Enabled?  ",ynStr(PSA(3))
PRINTLN "    Password Support Enabled?  ",ynStr(PSA(4))
PRINTLN "  Statistics Support Enabled?  ",ynStr(PSA(5))
PRINTLN "       Notes Support Enabled?  ",ynStr(PSA(6))
```

See Also

VER() ***Function***

PUSH *Statement*

Function

Push (save) the results of one or more expressions on a stack.

Syntax

```
PUSH exp[,exp]
```

`exp` An expression of any type to evaluate and push.

Remarks

This statement will evalutate one or more expressions of any type and push the results onto a stack for temporary storage. The results of those expressions may be retrieved via the **POP** statement. Together **PUSH** and **POP** can be used for parameter passing, to create 'local' variables, or to reverse the order of arguments.

Examples

```
INTEGER i, tc
STRING  s
LET tc = TOKCOUNT()
WHILE (TOKCOUNT() > 0) PUSH GETTOKEN() ' push them in order
FOR i = 1 TO tc
  POP s                                ' pop them in reverse
  PRINTLN s
NEXT

INTEGER i
FOR i = 1 TO 10
  PRINT i," - "
  GOSUB sub
NEXT
END
:sub
PUSH i                                 ' temporarily save i
LET i = i*i
PRINTLN i
POP i                                  ' restore saved i
RETURN

INTEGER v
PRINT "A cube with dimensions 2X3X4"
PUSH 2,3,4                             ' pass pushed parameters
GOSUB vol
POP v                                  ' pop result
PRINTLN "has volume ",v
END
:vol
INTEGER w,h,d
POP d,h,w                              ' pop passed parameter
PUSH w*h*d                             ' push result
RETURN
```

See Also

POP *Statement*

PUTUSER *Statement*

Function

Copy values from predeclared variables to user record.

Syntax

PUTUSER

No arguments are required

Remarks

There are many predeclared variables which may be used to access and change user information. However, their values are undefined until you use the **GETUSER** statement, and any changes you make don't take hold until you use the **PUTUSER** statement.

Examples

```
IF (PSA(3)) THEN
  GETUSER
  INPUT "Addr 1",U_ADDR(0)
  INPUT "Addr 2",U_ADDR(1)
  INPUT "City  ",U_ADDR(2)
  INPUT "State ",U_ADDR(3)
  INPUT "ZIP   ",U_ADDR(4)
  INPUT "Cntry ",U_ADDR(5)
  PUTUSER
ENDIF
```

See Also

GETUSER *Statement*

QUEST *Statement*

Function

Allow the user to answer a specified script questionnaire.

Syntax

```
QUEST scrnum
```

`scrnum` The number of the script for the user to answer. Valid values are 1 through the number of script questionnaires available.

Remarks

This statement will present the user a specified script questionnaire number to answer. The SCR.LST file for the current conference will be searched for the script. If the questionnaire number is invalid (less than 1 or greater than the highest script number defined) then nothing will be displayed.

Examples

```
INTEGER num
INPUT "Script to answer",num
QUEST num
```

See Also

BLT *Statement*, **DIR** *Statement*, **JOIN** *Statement*

RANDOM() *Function*

Function

Return a random value between 0 and a specified limit.

Syntax

RANDOM(limit)

limit	An integer expression with the maximum random value desired.

Return Type & Value

INTEGER	Returns the random number in the range 0 to **limit**.

Remarks

Random numbers have many applications from statistics to video games. This function allows you to generate pseudo-random numbers in the range 0 to **limit** inclusive.

Examples

```
INTEGER x,y
WHILE (KINKEY() <> " ") DO
  CLS
  LET x = 1+RANDOM(50)
  LET y = 1+RANDOM(22)
  COLOR 1+RANDOM(14)
  ANSIPOS x,y
  PRINT "Hit the SPACE BAR to continue"
  DELAY 18
  ANSIPOS x,y
  CLREOL
ENDWHILE
```

See Also

ABS() ***Function***

RDUNET *Statement*

Function

Read information from the USERNET file for a specific node.

Syntax

```
RDUNET node
```

`node` An integer expression with the node to read.

Remarks

To facilitate internode communications, a file named USERNET.XXX is maintained with an entry for each node on the system. This file is used by the BROADCAST command of PCBoard and to prevent multiple simultaneous logins, among other things. This statement may be used to read information for any node.

Examples

```
RDUNET PCBNODE()
WRUNET PCBNODE(),UN_STAT(),UN_NAME(),UN_CITY(),"Running "+PPENAME(),""
RDUNET 1
WRUNET 1,UN_STAT(),UN_NAME(),UN_CITY(),UN_OPER(),"Hello there node 1"
```

See Also

BROADCAST *Statement*, **UN_...()** *Functions*, **RDUNET** *Statement*

RDUSYS *Statement*

Function

Read a USERS.SYS file in from disk.

Syntax

RDUSYS

No arguments are required

Remarks

Some DOOR applications require a USERS.SYS file to access information about the caller. This statement allows you to read the USERS.SYS file back into memory in case any changes were made by the DOOR during the **SHELL** statement. This statement should only be used after a **SHELL** statement that was preceeded by a **WRUSYS** statement.

Examples

```
INTEGER ret
WRUSYS
SHELL FALSE,ret,"MYAPP.EXE",""
RDUSYS
```

See Also

SHELL *Statement*, **WRUSYS** *Statement*

READLINE() *Function*

Function

Read a specific line number from a text file.

Syntax

```
READLINE(file,line)
```

`file`	A string expression with the file name to read from.
`line`	An integer expression with the line number to read.

Return Type & Value

STRING	Returns the specified **line** number from **file**.

Remarks

It is often convienient to read a specified line number from a file without going to all the overhead of opening, reading and closing. This function will open the file in read mode for share deny none access and quickly read up to the line number you specify. If the line you want doesn't exist an empty string will be returned. Additionally, this function will remember the last file and line read so that it may quickly continue where it left off if you try to read a number of lines sequentially from the same file. Finally, the last file specified will remain open until the PPE exits and returns control to PCBoard.

Examples

```
PRINTLN "This system is running on IRQ ",READLINE(PCBDAT(),158)
PRINTLN "with a base IO address of ",READLINE(PCBDAT(),159)
```

See Also

DELETE *Statement*, **EXIST()** *Function*, **FILEINF()** *Function*, **RENAME** *Statement*

REG...() *Functions*

Function

Get the value of a register.

Syntax

REG . . . ( )

No arguments are required

REG should be followed by one of the following register names: AH, AL, AX, BH, BL, BX, CF, CH, CL, CX, DH, DI, DL, DS, DX, ES, F or SI.

Return Type & Value

BOOLEAN (**REGCF**() only) Returns **TRUE** if the carry flag is set, **FALSE** otherwise.

INTEGER (All others) Returns the value of the specified register.

Remarks

There are actually 18 different functions that return the values of registers. AL, AH, BL, BH, CL, CH, DL, and DH will always return byte sized values (0-255). AX, BX, CX, DX, DI, SI, DS, and ES will always return word sized values (0-65535). F (flags) returns the settings for the various 80x86 processor flags. CF is a subset of F in that it only returns the status of the carry flag. It exists because the carry flag is often used to report success or failure in assembly language. The **REGF**() function returns the settings for the following flags: **C**arry, **P**arity, **A**uxilliary, **Z**ero, Sign, **T**rap, **I**nterrupt, **D**irection, and **O**verflow. Their bit values are as follows:

O	D	I	T	S	Z	-	A	-	P	-	C
800h	400h	200h	100h	080h	040h	020h	010h	008h	004h	002h	001h

Examples

```
' Create subdirectory - DOS function 39h
INTEGER addr
STRING  path
LET path = "C:\$TMPDIR$"
VARADDR path,addr
DOINTR 21h,39h,0,0,addr%00010000h,0,0,0,addr/00010000h,0
IF (REGCF() & (REGAX() = 3)) THEN
  PRINTLN "Error:  Path not found"
ELSE IF (REGCF() & (REGAX() = 5)) THEN
  PRINTLN "Error:  Access Denied"
ELSE IF (REGCF()) THEN
  PRINTLN "Error:  Unknown Error"
ELSE
  PRINTLN "Directory successfully created..."
ENDIF
```

See Also

DOINTR *Statement*

RENAME *Statement*

Function

Rename (or move) a file.

Syntax

```
RENAME old,new
```

old	A string expression with the old path and/or file name.
new	A string expression with the new path and/or file name.

Remarks

Similar to how the RENAME command works from the DOS prompt, this statement will take a file and give it a new name. Unlike the RENAME command, the **RENAME** statement will not accept wildcards in the **old** or **new** parameters. Also, it doesn't require that the old path and the new path be the same (the drive letters must match, but the paths need not), so it may be used to move files from one location to another on a single drive. So, you could use it to move a file from C:\PCB\NODE1 to C:\PCB\NODE2 (renaming it at the same time if you wish), but you couldn't use it to move a file from C:\PCB\NODE1 to D:\WORK\NODE1.

Examples

```
' Swap the PCBOARD DAT & NXT files
RENAME "PCBOARD.DAT","PCBOARD.TMP"
RENAME "PCBOARD.NXT","PCBOARD.DAT"
RENAME "PCBOARD.TMP","PCBOARD.NXT"

' Move the file to the backup directory
RENAME "PPE.LOG","LOGBAK\"+I2S(DATE()*86400+TIME(),36)
```

See Also

DELETE *Statement*, **EXIST()** *Function*, **FILEINF()** *Function*, **READLINE()** *Function*

REPLACE() *Function*

Function

Change all occurences of a given character to another character in a string.

Syntax

```
REPLACE(str,old,new)
```

`str`	Any string expression.
`old`	A string expression with the old character to be replaced.
`new`	A string expression with the new character to replace with.

Return Type & Value

STRING	Returns **str** with all occurences of **old** changed to **new**.

Remarks

This function will search a string for a given character and replace all instances of that character with another character. This can be useful in many scenarios, especially when formatting text for display purposes.

Examples

```
PRINTLN "Your internet address on this system is:"
PRINTLN REPLACE(LOWER(U_NAME())," ",".")+"@clarkdev.com"
```

See Also

STRIP() *Function*, **STRIPATX**() *Function*

RESETDISP *Statement*

Function

Reset the display to allow more information after an abort.

Syntax

RESETDISP

No arguments are required

Remarks

PCBoard normally automatically counts lines and, if enabled, pauses the display after every screenful. The user may (unless disabled) abort the display at any MORE? prompt or with the ^K/^X keys. If this happens no further information will be displayed until you use the **RESETDISP** statement. You can check to see if **RESETDISP** is necessary (ie, has the user aborted the display) with the **ABORT()** function.

Examples

```
INTEGER I
STARTDISP FCL
' While the user has not aborted, continue
WHILE (!ABORT()) DO
  PRINTLN "I is equal to ",I
  INC I
ENDWHILE
RESETDISP
```

See Also

ABORT() *Function*, **STARTDISP** *Statement*

RESTSCRN *Statement*

Function

Restore the screen from a previously saved buffer.

Syntax

RESTSCRN

No arguments are required

Remark

PCBoard will save and restore the screen before and after certain functions, such as SysOp chat. This allows the user to continue right where he left off without having to remember what was on the screen before being interrupted. You can add that same functionality with the **SAVESCRN** and **RESTSCRN** statements. The **SAVESCRN** statement allocates memory for a buffer in which to save the screen. If the **SAVESCRN** statement isn't followed by a **RESTSCRN** statement then that memory will never be deallocated. Finally, this statement will work regardless of ANSI availability; the screen is only saved up to the position of the cursor and this statement assumes that it can safely restore the screen using standard teletype conventions to just scroll the data onto the display.

Examples

```
SAVESCRN
CLS
PRINTLN "We interrupt your regular BBS session"
PRINTLN "with this important message:"
NEWLINE
PRINTLN "A subscription to this system only costs $5!"
PRINTLN "Subscribe today!"
NEWLINES 2
WAIT
RESTSCRN
```

See Also

SAVESCRN *Statement*

RETURN *Statement*

Function

Transfer program control back to a previously saved address.

Syntax

RETURN

No arguments are required

Remarks

It is often necessary to perform an indentical set of instructions several times in a program. This leaves you with two choices. One, rewrite the code several times (and hope you do it right each time), or two, write it once as a subroutine, then use **GOSUB** to run it. This statement will save the address of the next line so that a **RETURN** statement at the end of the subroutine can instruct PPL to resume execution with the line following the **GOSUB**.

Examples

```
STRING Question, Answer
LET Question = "What is your street address ..."
GOSUB ask
LET Question = "What is your city, state and zip ..."
GOSUB ask
END

:ask ' Sub to ask a question, get an answer, and log them to a file
LET Answer = ""
PRINTLN "@X0E",Question
INPUT "",Answer
NEWLINES 2
FPUTLN 0,"Q: ",STRIPATX(Question)
FPUTLN 0,"A: ",Answer
RETURN
```

See Also

END *Statement*, **GOSUB** *Statement*, **GOTO** *Statement*, **FOR/NEXT** *Statement*, **IF/ELSEIF/ELSE/ENDIF** *Statement*, **STOP** *Statement*, **WHILE/ENDWHILE** *Statement*

RIGHT() *Function*

Function

Access the right most characters from a string.

Syntax

```
RIGHT(str,chars)
```

`str`	A string expression to take the right most characters of.
`chars`	An integer expression with the number of characters to take from the right end of str.

Return Type & Value

STRING	Returns a string with the right most **chars** characters of **str**.

Remarks

This function will return a sub string with the right most **chars** characters of a specified string. This can be useful in data processing as well as text formatting. If **chars** is less than or equal to 0 then the returned string will be empty. If **chars** is greater than the length of **str** then the returned string will have spaces added to the right to pad it out to the full length specified.

Examples

```
WHILE (RANDOM(250) <> 0) PRINT RIGHT(RANDOM(250),4)," "

STRING s
FOPEN 1,"DATA.TXT",O_RD,S_DN
WHILE (!FERR(1)) DO
  FGET 1,s
  PRINT RTRIM(LEFT(s,25)," "),"  - "
  PRINTLN RIGHT(s,LEN(s)-25)
ENDWHILE
FCLOSE 1
```

See Also

LEFT() *Function*, **MID()** *Function*

RTRIM() *Function*

Function

Trim a specified character from the right end of a string.

Syntax

RTRIM(str,ch)

str	Any string expression.
ch	A string with the character to strip from the right end of **str**.

Return Type & Value

STRING	Returns the trimmed **str**.

Remarks

A common need in programming is to strip leading and/or trailing spaces (or other characters). This function will strip a specified character from the right end of a string and return the trimmed string.

Examples

```
STRING s
LET s = "   TEST   "
PRINTLN RTRIM(s," ") ' Will print "   TEST"
PRINTLN RTRIM(".....DA"+"TA.....",".") ' Will print ".....DATA"
PRINTLN RTRIM(".....DA"+"TA....."," ") ' Will print ".....DATA....."
```

See Also

LTRIM() *Function*, **TRIM()** *Function*

S2I() *Function*

Function

Convert a string in a specified number base to an integer.

Syntax

S2I(str,base)

str	Any string expression to convert to integer format.
base	An integer expression with the number base (2 through 36) to convert from.

Return Type & Value

INTEGER	Returns **str** converted from the specified number base to an integer.

Remarks

People work with decimal (base 10) numbers, whereas computers work with binary (base 2) numbers. However, it is often more convienient to store or input numbers in a format other than decimal for clarity, compactness, or other reasons. This function will convert a string in any number base from 2 to 36 to a number. So, **S2I("1010",2)** would return a 10; **S2I("Z",36)** would return 35.

Examples

```
INTEGER i
STRING  s
INPUTTEXT "Enter a string (any base)",s,@X0E,40
FOR i = 2 TO 36
  PRINTLN s," = ",S2I(s,i)," base ",i
NEXT
```

See Also

I2S() *Function*

SAVESCRN *Statement*

Function

Save the screen to a buffer for later restoration.

Syntax

SAVESCRN

No arguments are required

Remark

PCBoard will save and restore the screen before and after certain functions, such as SysOp chat. This allows the user to continue right where he left off without having to remember what was on the screen before being interrupted. You can add that same functionality with the **SAVESCRN** and **RESTSCRN** statements. The **SAVESCRN** statement allocates memory for a buffer in which to save the screen. If the **SAVESCRN** statement isn't followed by a **RESTSCRN** statement then that memory will never be deallocated. Finally, this statement will work regardless of ANSI availability; this statement will only save the screen up to the position of the cursor. It is assumed that the screen can be safely restored using standard teletype conventions to just scroll the data onto the display.

Examples

```
SAVESCRN
CLS
PRINTLN "We interrupt your regular BBS session"
PRINTLN "with this important message:"
NEWLINE
PRINTLN "A subscription to this system only costs $5!"
PRINTLN "Subscribe today!"
NEWLINES 2
WAIT
RESTSCRN
```

See Also

RESTSCRN *Statement*

SCRTEXT() *Function*

Function

Access text and attribute information directly from BBS screen memory.

Syntax

```
SCRTEXT(x,y,len,color)
```

`x`	An integer expression with the x coordinate (column) from which to read screen memory.
`y`	An integer expression with the y coordinate (row) from which to read screen memory.
`len`	An integer expression with the length, in columns, of the string to read from screen memory.
`color`	A boolean expression with **TRUE** if color information should be included, **FALSE** otherwise.

Return Type & Value

STRING	Returns the specified region of screen memory.

Remarks

This function is useful for temporarily saving a portion of screen memory, with or without color information. If the color parameter is set to **TRUE** color information will be included in the form of @X codes embedded in the text. Note that the maximum length of a string is 256 characters; however, a row of 80 characters could be as long as 400 characters (4 bytes for the @X code and 1 byte for the character itself). You should generally limit yourself to a length of 51 characters or less if you want to include color information unless you are certain that attribute changes will not exceed the 256 character string limit.

Examples

```
' scroll the screen to the left 5 columns and down 3 rows
INTEGER r
STRING  s
FOR r = 20 TO 1 STEP -1
  LET s = SCRTEXT(6,r,75,TRUE)
  ANSIPOS 1,r+3
  CLREOL
  PRINT s
NEXT
FOR r = 1 TO 3
  ANSIPOS 1,r
  CLREOL
NEXT
```

See Also

INSTR() *Function*, LEN() *Function*, SPACE() *Function*, STRING() *Function*

SEC *Constant*

Function

Set the security level specific file search flag in a **DISPFILE** statement.

Value

2 = 10b = 2o = 2h

Remarks

The **DISPFILE** statement will allow you to display a file to the user, and optionally to have PCBoard look for alternate security, graphics, and/or language specific files. This flag instructs PCBoard to search for alternate security level files via the security level suffix. The current security level may be obtained with the **CURSEC()** function.

Examples

```
STRING s
DISPFILE "MNUA",SEC+GRAPH+LANG
INPUT "Option",s
```

See Also

CURSEC() *Function*, **DISPFILE** *Statement*, **GRAPH** *Constant*, **LANG** *Constant*

SEC() *Function*

Function

Extract the second of the minute from a specified time of day.

Syntax

```
SEC(texp)
```

texp	Any time expression.

Return Type & Value

INTEGER	Returns the second of the minute from the specified time expression (**texp**). Valid return values are from 0 to 59.

Remarks

This function allows you to extract a particular piece of information about a **TIME** value. in this case the second of the minute of the time of day expression.

Examples

```
PRINTLN "The minute is ",SEC(TIME())
```

See Also

HOUR() ***Function*****, MIN()** ***Function*****, TIME()** ***Function***

SENDMODEM *Statement*

Function

Send a string to the modem.

Syntax

```
SENDMODEM str
```

`str` A string expression to send to the modem.

Remarks

The primary use of this statement is to send commands to a modem when no one is online. For example, you would use this to send a dial command to the modem in a call back PPL application. However, it is not restricted to sending commands. Note that modem commands must be terminated by a carriage return and that this statement will not automatically do it for you. This allows you to send a command to the modem is several stages and only terminate the final stage with a carriage return.

Examples

```
BOOLEAN flag
CDCHKOFF
KBDCHKOFF
DTROFF
DELAY 18
DTRON
SENDMODEM "ATDT"
SENDMODEM "5551212"
SENDMODEM CHR(13)
WAITFOR "CONNECT",flag,60
IF (!flag) LOG "No CONNECT after 60 seconds",FALSE
KBDCHKON
CDCHKON
```

See Also

WAITFOR *Statement*

SHELL *Statement*

Function

Shell out to a program or batch file.

Syntax

```
SHELL viacc,retcode,prog,cmds
```

viacc	A boolean expression with value **TRUE** if the shell should be made via COMMAND.COM; **FALSE** if it should be shelled to directly.
retcode	A variable in which to store the return code.
prog	A string expression with the file name to shell to.
cmds	A string expression with any arguments to pass to **prog**.

Remarks

You may have have need to run a COM, EXE or BAT file from your PPE. You may need to do this to simulate running a DOOR or to access some service not normally available from PCBoard or PPL. This function will allow you to do that. If the **viacc** parameter is **TRUE** (you want COMMAND.COM to load the specified file) your PATH environment variable will be searched for **prog** if it isn't in the current directory or isn't fully qualified (path and extension), just as it would be if entered on the command line. If **viacc** is **FALSE** then you must specify the path and extension of the program to run. Additionally, the **retcode** variable will only be meaningful if **viacc** is **FALSE**.

Examples

```
INTEGER rc
SHELL TRUE,rc,"DOOR",""

INTEGER rc
STRING  p,c
LET p = "DOORWAY.EXE"
LET c = "com2 /v:d^O /m:600 /g:on /o: /k:v0 /x: /c:dos"
SHELL FALSE,rc,p,c
```

See Also

CALL *Statement*, **RDUSYS** *Statement*, **WRUSYS** *Statement*

SHOWOFF *Statement*

Function

Turn off showing information to the display.

Syntax

SHOWOFF

No arguments are required

Remark

This statement allows your PPL application to turn off writing information to the local and remote displays. Used in conjunction with the **SHOWSTAT()** function and the **OPENCAP**, **CLOSECAP**, and **SHOWON** statements it allows you to temporarily turn off the display while capturing output to the screen. This can be useful anytime you want to automate a feature for the user and allow them to download the resulting capture file instead of spending lots of time online.

Examples

```
BOOLEAN ss
LET ss = SHOWSTAT()
SHOWOFF
OPENCAP "CAP"+STRING(PCBNODE()),ocFlag
IF (ocFlag) THEN
  DIR "U:NS"
  CLOSECAP
  KBDSTUFF "FLAG CAP"+STRING(PCBNODE())+CHR(13)
ENDIF
IF (ss) THEN
  SHOWON
ELSE
  SHOWOFF
ENDIF
```

See Also

CLOSECAP *Statement*, **OPENCAP** *Statement*, **SHOWON** *Statement*, **SHOWSTAT()** *Function*

SHOWON *Statement*

Function

Turn on showing information to the display.

Syntax

SHOWON

No arguments are required

Remark

This statement allows your PPL application to turn on writing information to the local and remote displays. Used in conjunction with the **SHOWSTAT()** function and the **OPENCAP**, **CLOSECAP**, and **SHOWOFF** statements it allows you to temporarily turn off the display while capturing output to the screen. This can be useful anytime you want to automate a feature for the user and allow them to download the resulting capture file instead of spending lots of time online.

Examples

```
BOOLEAN ss
LET ss = SHOWSTAT()
SHOWOFF
OPENCAP "CAP"+STRING(PCBNODE()),ocFlag
IF (ocFlag) THEN
  DIR "U;NS"
  CLOSECAP
  KBDSTUFF "FLAG CAP"+STRING(PCBNODE())+CHR(13)
ENDIF
IF (ss) THEN
  SHOWON
ELSE
  SHOWOFF
ENDIF
```

See Also

CLOSECAP *Statement*, **OPENCAP** *Statement*, **SHOWOFF** *Statement*, **SHOWSTAT()** *Function*

SHOWSTAT() *Function*

Function

Determine if data is being shown on the display.

Syntax

SHOWSTAT()

No arguments are required

Return Type and Value

BOOLEAN Returns **TRUE** if data is being shown on the display, **FALSE** otherwise.

Remarks

This function allows your PPL application to determine the status of writing information to the local and remote displays. Used in conjunction with the **OPENCAP**, **CLOSECAP**, **SHOWON**, and **SHOWOFF** statements it allows you to temporarily turn off the display while capturing output to the screen. This can be useful anytime you want to automate a feature for the user and allow them to download the resulting capture file instead of spending lots of time online.

Examples

```
BOOLEAN ss
LET ss = SHOWSTAT()
SHOWOFF
OPENCAP "CAP"+STRING(PCBNODE()),ocFlag
IF (ocFlag) THEN
  DIR "U:NS"
  CLOSECAP
  KBDSTUFF "FLAG CAP"+STRING(PCBNODE())+CHR(13)
ENDIF
IF (ss) THEN
  SHOWON
ELSE
  SHOWOFF
ENDIF
```

See Also

CLOSECAP *Statement*, **OPENCAP** *Statement*, **SHOWOFF** *Statement*, **SHOWON** *Statement*

SLPATH() *Function*

Function

Return the path of login security files as defined in PCBSetup.

Syntax

SLPATH()

No arguments are required

Return Type & Value

STRING Returns the path of the PCBoard login security files.

Remarks

This function will return the path where login security files are located as defined in PCBSetup. It can be used to create and change them on the fly.

Examples

```
FAPPEND 1,SLPATH()+STRING(CURSEC),O_WR,S_DB
FPUTLN 1,U_NAME()
FCLOSE 1
```

See Also

HELPPATH() *Function*, **PPEPATH()** *Function*, **TEMPPATH()** *Function*

SOUND *Statement*

Function

Turn on the speaker on the local computer at a specific frequency.

Syntax

```
SOUND freq
```

`freq`	An integer expression with the frequency (in hertz) at which to turn on the speaker or 0 to turn off the speaker.

Remarks

This statement can be used to generate just about any tone desired on the speaker on the local PC. It has no effect on the remote computer and will only work with the built in speaker (in other words, it has no way of communicating with advanced sound cards). You specify the frequency of the tone you wish to generate in hertz and pass it to the statement, or pass 0 to turn off the speaker.

Examples

```
PAGEON
FOR i = 1 TO 10
  MPRINT CHR(7)
  SOUND 440
  DELAY 9
  SOUND 0
  DELAY 9
  IF (KINKEY() = " ") THEN
    CHAT
    GOTO exit
  ENDIF
NEXT
:exit
```

See Also

DELAY *Statement*

SPACE() *Function*

Function

Create a string with a specified of spaces.

Syntax

```
SPACE(len)
```

len	An integer expression with the number of spaces for the new string.

Return Type & Value

STRING	Returns a string of **len** spaces.

Remarks

This function is useful when formatting screen displays without ANSI and when writing formatted information out to a file. It will create a string of the length specified with nothing but spaces. The returned string may have anywhere from 0 to 256 spaces.

Examples

```
PRINT RANDOM(9),SPACE(5),RANDOM(9),SPACE(5),RANDOM(9)

FCREATE 1,"NEWFILE.DAT",O_WR,S_DB
FPUTLN 1,"NAME",SPACE(24),"CITY",SPACE(23),"PHONE"
FCLOSE 1
```

See Also

INSTR() ***Function*****, LEN()** ***Function*****, SCRTEXT()** ***Function*****, STRING()** ***Function***

SPRINT/SPRINTLN *Statements*

Function

Print (write) a line to the local screen (BBS) only (with an optional newline appended).

Syntax

```
SPRINT exp[,exp]
-or-
SPRINTLN [exp[,exp]]
```

exp An expression of any type to evaluate and write to the caller's screen.

Remarks

These statements will evalutate zero, one or more expressions of any type and write the results to the BBS for the SysOp's display. The **SPRINTLN** statement will append a newline to the end of the expressions; **SPRINT** will not. Note that at least one expression must be specified for **SPRINT**, unlike the **SPRINTLN** statement which need not have any arguments passed to it. These statements only send information to the local display and do not interpret @ codes; however, complete ANSI sequences will be interpreted.

Examples

```
SPRINT "The name of the currently running PPE file is "
SPRINTLN PPENAME(),"."
SPRINT "The path where it is located is "
SPRINTLN PPEPATH(),"."
SPRINT "The date is ",DATE()," and the time is ",TIME(),"."
SPRINTLN
```

See Also

MPRINT/MPRINTLN *Statements*, **PRINT/PRINTLN** *Statements*

STACKED *Constant*

Function

Set the allow stacked commands flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

16 = 10000b = 20o = 10h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to allow space and semi-colon characters to be input independent of the valid character string specified. This facilitates entering stacked commands (commands separated by space or semi-colon delimiters) by only requiring a single value be set in the input statement instead of having to add " ;" to every valid character mask.

Examples

```
STRING cmds
INPUTSTR "Commands",cmds,0X0E,60,MASK_ASCII(),STACKED
TOKENIZE cmds
LET cmds = GETTOKEN()
IF (cmds = "QUIT") END
KBDSTUFF cmds+TOKENSTR()
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

STARTDISP *Statement*

Function

Start PCBoard's display routines in a specified mode.

Syntax

```
STARTDISP mode
```

mode An integer expression with the mode for display.

Remarks

PCBoard has two modes for displaying information: non stop and line count. Non stop mode (initiated by passing **FNS**, for Force Non Stop, as the mode parameter) displays information without regard to how fast the display is or whether or not the user can read it all. Line count mode (initiated by passing **FCL**, for Force Count Lines, as the mode parameter) displays information while counting lines and pausing after every screenful to wait for user input. Finally, **NC** may be specified to reinitialize the internal display counters without changing the current mode.

Examples

```
STARTDISP FCL
FOR i = 1 TO 100
  PRINTLN "Line ",i
NEXT
STARTDISP FNS
FOR i = 1 TO 100
  PRINTLN "Line ",i
NEXT
STARTDISP NC
FOR i = 1 TO 100
  PRINTLN "Line ",i
NEXT
```

See Also

ABORT() *Function*, **RESETDISP** *Statement*

STOP *Statement*

Function

Abort PPE execution.

Syntax

STOP

No arguments are required

Remarks

This statement may be used to abnormally terminate PPE execution at any point. The only real difference between this statement and **END** is whether or not information written to channel 0 is saved when the . **END** will save the output to the script answer file; **STOP** will not.

Examples

```
STRING Question, Answer
LET Question = "What is your street address ..."
GOSUB ask
INPUTYN "Save address",Answer,@X0E
IF (Answer = NOCHAR()) STOP
END

:ask ' Sub to ask a question, get an answer, and log them to a file
LET Answer = ""
PRINTLN "@X0E",Question
INPUT "",Answer
NEWLINES 2
FPUTLN 0,"Q: ",STRIPATX(Question)
FPUTLN 0,"A: ",Answer
RETURN
```

See Also

END *Statement*, **RETURN** *Statement*

STRING *Type*

Function

Declare one or more variables of type string.

Syntax

```
STRING var|arr(s[,s[,s]])[,var|arr(s[,s[,s]])]
```

`var`	The name of a variable to declare. Must start with a letter [A-Z] which may be followed by letters, digits [0-9] or the underscore [_]. May be of any length but only the first 32 characters are used.
`arr`	The name of an array variable to declare. The same naming conventions as **var** are used.
`s`	The size (0-based) of an array variable dimension. Any constant integer expression is allowed.

Remarks

STRING variables are stored as pointers arrays of characters from 0 to 257 bytes in size. If the array has 0 or 1 characters in it, it is a 0 length string. Arrays with 2 to 257 characters have a length of the array size minus one. Valid string characters are ASCII 1 through ASCII 255. ASCII 0 is reserved for terminating the string and may not appear in the middle of the string. A **STRING** assignment to an **INTEGER** will convert the string to the four byte binary integer value (similar to BASIC's VAL function and C's atol function). An **INTEGER** to **STRING** assignment will result in a string with the representation of the number (similar to BASIC's STR$ function and C's ltoa function). If a **STRING** is assigned to or from any other type, an appropriate conversion is performed automatically by PPL.

Examples

```
STRING char, str, tmp, labels(10), names(20,3)
```

See Also

BOOLEAN *Type*, **DATE** *Type*, **INTEGER** *Type*, **MONEY** *Type*, **TIME** *Type*

STRING() *Function*

Function

Convert any expression to a string.

Syntax

STRING(exp)

exp Any expression.

Return Type & Value

STRING Returns **exp** formatted as a string.

Remarks

This function is immensely useful anytime you need to convert any expression to string format. For example, to append an integer value to the end of a string without this function, you would need to assign the integer to a string and then append the temporary string to actual string. This is because PPL's normal course of action when performing arithmetic with incompatible types is to convert everything to integer first. With this function, you can accomplish the same function in one line of code with one expression because you are forcing addition of compatible types (strings). Note that PPL does automatically convert incompatible types whenever possible, making this function unnecessary in many cases. This function should only be necessary when trying to append the text representation of a non-string type to a string via the + operator.

Examples

```
INTEGER i
STRING  s(5)
FOR i = 1 to 5
  LET s(i) = "This is string "+STRING(i)
NEXT

STRING s
LET s = STRING(ABORT())+" "+STRING(DATE())+" "+STRING(10)+" "
LET s = s+STRING($10.00)+" "+STRING(TIME())
PRINTLN s ' will print "0 10-31-67 10 $10.00 03:27:00" (or similar)
```

See Also

INSTR() *Function*, **LEN()** *Function*, **SCRTEXT()** *Function*, **SPACE()** *Function*

STRIP() *Function*

Function

Remove all occurrences of a character from a string.

Syntax

STRIP(str,ch)

str	Any string expression.
ch	String with character to remove from **str**.

Return Type & Value

STRING	Returns **str** without occurrences of **ch** that may have been present previously.

Remarks

This function is used to strip a selected character from a string. This can be useful when you need to remove known formatting characters from a string, such as slashes and hyphens from a date string.

Examples

```
STRING s
WHILE (LEN(s) < 6) DO
  INPUTSTR "Enter date (MM-DD-YY)",s,@X0E,8,"0123456789-",DEFS
  LET s = STRIP(s,"-")
ENDWHILE
PRINTLN "Date (MMDDYY):  ",s
```

See Also

REPLACE() ***Function***, **STRIPATX()** ***Function***

STRIPATX() *Function*

Function

Remove @X codes from a string.

Syntax

```
STRIPATX(sexp)
```

sexp	Any string expression.

Return Type & Value

STRING	Returns **sexp** without any @X codes that may have been present previously.

Remarks

This function is used to strip PCBoard @X color codes from a string or string expression. This is useful when you want to log information to a file without the @X codes used in the screen display.

Examples

```
STRING Question, Answer
LET Question = "What is your street address ..."
GOSUB ask
END
:ask ' Sub to ask a question, get an answer, and log them to a file
LET Answer = ""
PRINTLN "@X0E",Question
INPUT "",Answer
NEWLINES 2
FPUTLN 0,"Q: ",STRIPATX(Question)
FPUTLN 0,"A: ",Answer
RETURN
```

See Also

REPLACE() ***Function***, **STRIP()** ***Function***

SYSOPSEC() *Function*

Function

Get the security level as the SysOp security level.

Syntax

SYSOPSEC()

No arguments are required

Return Type & Value

INTEGER Returns the SysOp security level as defined in PCBSetup.

Remarks

This function is useful for those occasions when you need to limit functionality in your PPL applications to users having a security level greater than or equal to the defined SysOp security level in PCBSetup.

Examples

```
INTEGER min
IF (CURSEC() >= SYSOPSEC()) THEN
  LET min = 60
ELSE
  LET min = 5
ENDIF
ADJTIME min
PRINTLN "Your time available has been increased by ",min," minutes"
```

See Also

CURSEC() ***Function***

S_DB *Constant*

Function

Set the share deny both (read and write) flag in a **FCREATE/FOPEN/FAPPEND** statement.

Value

3 = 11b = 3o = 3h

Remarks

DOS 3.1 or later (which is what is required by PCBoard) allows processes to decide what mode of file sharing should be allowed. This constant allows you to specify that other processes may not open the same file for either read or write access from the time you open the file to the time you close the file. This is useful when you need exclusive access to a file for any reason and need to restrict other processes access to the same file.

Examples

```
FOPEN 1,"FILE.DAT",O_RD,S_DB ' Deny other processes all access
FOR i = 1 TO 10
  FGET 1,s
  PRINTLN s
NEXT
FCLOSE 1 ' Close the file and allow others to open it in any mode
```

See Also

S_DN *Constant*, **S_DR** *Constant*, **S_DW** *Constant*

S_DN *Constant*

Function

Set the share deny none flag in a **FCREATE/FOPEN/FAPPEND** statement.

Value

0 = 0b = 0o = 0h

Remarks

DOS 3.1 or later (which is what is required by PCBoard) allows processes to decide what mode of file sharing should be allowed. This constant allows you to specify that other processes may open the same file for read or write access from the time you open the file to the time you close the file. This is useful when you don't need exclusive access to a file for any reason and need not restrict other processes.

Examples

```
FOPEN 1,"FILE.DAT",O_RD,S_DN ' Do not deny other processes any access
FOR i = 1 TO 10
  FGET 1,s
  PRINTLN s
NEXT
FCLOSE 1 ' Close the file and allow others to open it in any mode
```

See Also

S_DB *Constant*, **S_DR** *Constant*, **S_DW** *Constant*

S_DR *Constant*

Function

Set the share deny read flag in a **FCREATE/FOPEN/FAPPEND** statement.

Value

1 = 1b = 1o = 1h

Remarks

DOS 3.1 or later (which is what is required by PCBoard) allows processes to decide what mode of file sharing should be allowed. This constant allows you to specify that other processes may open the same file, but that they may not open it for read access, from the time you open the file to the time you close the file.

Examples

```
FOPEN 1,"FILE.DAT",O_RD,S_DR ' Deny other processes read access
FOR i = 1 TO 10
  FGET 1,s
  PRINTLN s
NEXT
FCLOSE 1 ' Close the file and allow others to open it in any mode
```

See Also

S_DB *Constant*, **S_DN** *Constant*, **S_DW** *Constant*

S_DW *Constant*

Function

Set the share deny write flag in a **FCREATE/FOPEN/FAPPEND** statement.

Value

2 = 10b = 2o = 2h

Remarks

DOS 3.1 or later (which is what is required by PCBoard) allows processes to decide what mode of file sharing should be allowed. This constant allows you to specify that other processes may open the same file, but that they may not open it for write access, from the time you open the file to the time you close the file. This is useful when you want to ensure that data will not change while you are reading it.

Examples

```
FOPEN 1,"FILE.DAT",O_RD,S_DW ' Deny other processes write access
FOR i = 1 TO 10
  FGET 1,s
  PRINTLN s
NEXT
FCLOSE 1 ' Close the file and allow others to open it in any mode
```

See Also

S_DB *Constant*, **S_DN** *Constant*, **S_DR** *Constant*

TEMPPATH() *Function*

Function

Return the path to the temporary work directory as defined in PCBSetup.

Syntax

TEMPPATH()

No arguments are required

Return Type & Value

STRING Returns the path of the node temporary work files area.

Remarks

This function will return the path where temporary work files should be created as defined in PCBSetup. This path is a good place for small temporary files that need not be kept permanently since it often points to a RAM drive or other fast local storage.

Examples

```
INTEGER rc
SHELL TRUE,rc,"DIR",">"+TEMPPATH()+"TMPDIR"
DISPFILE TEMPPATH()+"TMPDIR",DEFS
DELETE TEMPPATH()+"TMPDIR"
```

See Also

HELPPATH() *Function*, PPEPATH() *Function*, SLPATH() *Function*

TIME *Type*

Function

Declare one or more variables of type time.

Syntax

```
TIME var|arr(s[,s[,s]])[,var|arr(s[,s[,s]])]
```

`var`	The name of a variable to declare. Must start with a letter [A-Z] which may be followed by letters, digits [0-9] or the underscore [_]. May be of any length but only the first 32 characters are used.
`arr`	The name of an array variable to declare. The same naming conventions as **var** are used.
`s`	The size (0-based) of an array variable dimension. Any constant integer expression is allowed.

Remarks

TIME variables are stored as seconds elapsed since midnight. Valid times are 0 (00:00:00) through 86399 (23:59:59). It is stored internally as a four byte unsigned long integer. If a **TIME** is assigned to or from an **INTEGER** type then the seconds since midnight (0-86399) is assigned. If a **TIME** is assigned to a **STRING** type then it is automatically converted to the following format: "HH:MM:SS", where HH is the two digit hour (00-23), MM is the two digit minute (00-59), and SS is the two digit second (00-59). If a foreign language is in use that uses a different time format (for example, "HH.MM.SS") then that will be taken into account. If a **STRING** is assigned to a **TIME** then PPL will do it's best to convert the string back to the appropriate time. All other types, when assigned to or from a **TIME**, will be converted to an **INTEGER** first before being assigned to or from the **TIME** type.

Examples

```
TIME tob, now, pageHours(2), hourList(24)
```

See Also

BOOLEAN *Type*, **DATE** *Type*, **INTEGER** *Type*, **MONEY** *Type*, **STRING** *Type*

TIME() *Function*

Function

Get the current time.

Syntax

TIME()

No arguments are required

Return Type & Value

TIME Returns the current time.

Remarks

The time returned is represented internally as the number of seconds elapsed since midnight. It may be used as is (for display, storage or as an argument to another function or statement) or assigned to an integer for arithmetic purposes. 00:00:00 (midnight) has a value of 0, 00:00:01 a value of 1, 00:01:00 a value of 60, 01:00:00 a value of 3600, etc, until 23:59:59 which has a value of 86399.

Examples

```
PRINTLN "The time is ",TIME()
```

See Also

DATE() *Function*, **HOUR()** *Function*, **MIN()** *Function*, **MKDATE()** *Function*, **SEC()** *Function*, **TIMEAP()** *Function*

TIMEAP() *Function*

Function

Converts a time value to a 12-hour AM/PM formatted string.

Syntax

```
TIMEAP(texp)
```

`texp`	Any time expression.

Return Type & Value

STRING	Returns a string formatted with the time specified by **texp** in a 12-hour AM/PM format.

Remarks

TIME values are, by default, formatted for military time ("HH:MM:SS") when displayed or assigned to a string variable. You may wish to format them in a 12-hour AM/PM format in some circumstances, however. This function perform the conversion and format the time in "HH:MM:SS XM" format (HH = hour, MM = minute, SS = second, X = A or P).

Examples

```
PRINTLN "The current time is ",TIMEAP(TIME())
```

See Also

TIME() *Function*

TOKCOUNT() *Function*

Function

Access the number of tokens pending.

Syntax

TOKCOUNT()

No arguments are required

Return Type & Value

INTEGER Returns the number of tokens available.

Remarks

Parameter passing between PCBoard and PPL applications (and between PPL applications) and command line parsing is accomplished via tokens. This function will return the number of tokens available via the **GETTOKEN** statement and the **GETTOKEN()** function. The value returned by this will be decremented after each token is retrieved until it reaches 0 (no more tokens available). The **TOKENIZE** function will overwrite any pending tokens with new tokens and reinitialize this function to the new number. Finally, the **TOKENSTR()** function will clear this function to 0 and return all tokens in a string with semi-colons separating individual tokens.

Examples

```
PRINTLN "There are ",TOKCOUNT()," tokens"
WHILE (TOKCOUNT() > 0) PRINTLN GETTOKEN()
```

See Also

GETTOKEN *Statement*, **GETTOKEN()** *Function*, **TOKENIZE** *Statement*, **TOKENSTR()** *Function*

TOKENIZE *Statement*

Function

Split up a string into tokens separated by semi-colons or spaces.

Syntax

```
TOKENIZE sexp
```

`sexp` Any string expression.

Remarks

One of the strongest features of PCBoard is it's ability to take a series of stacked parameters from a command line and use them all at once instead of requiring the user to navigate a series of menus and select one option at each step of the way. The **TOKENIZE** statement is the PPL equivalent of what PCBoard uses to break a command line into individual commands (tokens). The number of tokens available may be accessed via the **TOKCOUNT()** function, and each token may be accessed, one at a time, by the **GETTOKEN** statement and/or the **GETTOKEN()** function.

Examples

```
STRING  cmdline
INPUT "Command",cmdline
TOKENIZE cmdline
PRINTLN "You entered ",TOKCOUNT()," tokens"
WHILE (TOKCOUNT() > 0) PRINTLN "Token:  ",CHR(34),GETTOKEN(),CHR(34)
```

See Also

GETTOKEN *Statement*, **GETTOKEN()** *Function*, **TOKCOUNT()** *Function*, **TOKENSTR()** *Function*

TOKENSTR() *Function*

Function

Rebuild and return a previously tokenized string.

Syntax

TOKENSTR()

No arguments are required

Return Type & Value

STRING Returns the rebuilt string that was previously tokenized.

Remarks

One of the strongest features of PCBoard is it's ability to take a series of stacked parameters from a command line and use them all at once instead of requiring the user to navigate a series of menus and select one option at each step of the way. The **TOKENIZE** statement is the PPL equivalent of what PCBoard uses to break a command line into individual commands (tokens). This function will take all pending tokens and build a string with appropriate token separators. For example, the string "R A S" would be broken into three separate tokens; "R", "A" and "S". **TOKENSTR()** would take those tokens and return the following string: "R;A;S". Note that, regardless of the separator used in the original string, the semi-colon character will be used in the rebuilt string.

Examples

```
STRING  cmdline
INPUT "Command",cmdline
TOKENIZE cmdline
PRINTLN "You entered ",TOKCOUNT()," tokens"
PRINTLN "Original string:  ",cmdline
PRINTLN "      TOKENSTR():  ",TOKENSTR()
```

See Also

GETTOKEN *Statement*, **GETTOKEN()** *Function*, **TOKCOUNT()** *Function*, **TOKENIZE** *Statement*

TRIM() *Function*

Function

Trim a specified character from both ends of a string.

Syntax

TRIM(str,ch)

str	Any string expression.
ch	A string with the character to strip from both ends of **str**.

Return Type & Value

STRING	Returns the trimmed **str**.

Remarks

A common need in programming is to strip leading and/or trailing spaces (or other characters). This function will strip a specified character from both ends of a string and return the trimmed string.

Examples

```
STRING s
LET s = "   TEST   "
PRINTLN TRIM(s," ") ' Will print "TEST"
PRINTLN TRIM(".....DA"+"TA.....",".") ' Will print "DATA"
PRINTLN TRIM(".....DA"+"TA.....","") ' Will print ".....DATA....."
```

See Also

LTRIM() ***Function***, **RTRIM()** ***Function***

TRUE *Constant*

Function

To provide a named constant for the boolean true value in boolean expressions.

Value

1 = 1b = 1o = 1h

Remarks

BOOLEAN logic is based on two values: **TRUE** (1) and **FALSE** (0). The literal numeric constants 0 and 1 may be used in expressions, or you may use the predefined named constants **TRUE** and **FALSE**. They make for more readable, maintainable code and have no more overhead than any other constant value at run time.

Examples

```
BOOLEAN flag
LET flag = TRUE
WHILE (!flag) DO
  INPUTSTR "Text",s,0X0E,60,"ABCDEFGHIJKLMNOPQRSTUVWXYZ ",UPCASE
  PRINTLN s
  IF (s = "QUIT") LET flag = FALSE
ENDWHILE
```

See Also

DEFS *Constant*, **FALSE** *Constant*

UN_...() *Functions*

Function

Get a piece of information about a node.

Syntax

UN_. . . ()

No arguments are required

UN_ should be followed by one of the following: CITY, NAME, OPER, or STAT.

Return Type & Value

STRING Returns a string with the desired piece of information.

Remarks

There are actually four different functions that return information from the USERNET.XXX file. **UN_CITY**() will return the city field, **UN_NAME**() will return the user name field, **UN_OPER**() will return the operation text field, and **UN_STAT**() will return the status field. The information returned by these functions is only meaningful after executing the **RDUNET** statement for a specific node.

Examples

```
RDUNET PCBNODE()
WRUNET PCBNODE(),UN_STAT(),UN_NAME(),UN_CITY(),"Running "+PPENAME(),""
RDUNET 1
WRUNET 1,UN_STAT(),UN_NAME(),UN_CITY(),UN_OPER(),"Hello there node 1"
```

See Also

BROADCAST *Statement*, **RDUNET** *Statement*, **WRUNET** *Statement*

UPCASE *Constant*

Function

Set the force uppercase flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

8 = 1000b = 10o = 8h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to force all input characters to uppercase. This is useful in getting case insensitive replies from the user. If this flag is used, you need not pass lowercase valid characters as they will be automatically converted at runtime. If this flag is not used and you need to input alphabetic characters, you should pass both lowercase and uppercase characters in the valid character string.

Examples

```
STRING s
WHILE (s <> "QUIT") DO
  INPUTSTR "Text",s,@X0E,60,"ABCDEFGHIJKLMNOPQRSTUVWXYZ ",UPCASE
  PRINTLN s
ENDWHILE
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

UPPER() *Function*

Function

Converts lowercase characters in a string to uppercase.

Syntax

UPPER(sexp)

sexp	Any string expression.

Return Type & Value

STRING	Returns **sexp** with all lowercase characters converted to uppercase.

Remarks

Although "STRING" is technically different from "string" (ie, the computer doesn't recognize them as being the same because one is uppercase and the other is lowercase), it is often necessary to save, display or compare information in a case insensitive format. This function will return a string with all lowercase characters converted to uppercase. So, using the above example, **UPPER("string")** would return "STRING".

Examples

```
STRING s
WHILE (UPPER(s) <> "QUIT") DO
  INPUT "Text",s
  PRINTLN LOWER(s)
ENDWHILE
```

See Also

LOWER() *Function*

U_ADDR() *VARIABLE ARRAY*

Function

Allow reading and writing of the current users address information.

Type & Value

STRING

Subscript 0	Address Line 1 (50 characters max).
Subscript 1	Address Line 2 (50 characters max).
Subscript 2	City (25 characters max).
Subscript 3	State (10 characters max).
Subscript 4	ZIP Code (10 characters max).
Subscript 5	Country (15 characters max).

Remarks

This array is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that the array is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Additionally, the array will only have meaningful information if the address PSA is installed. The existence of the address PSA may be checked with the **PSA()** function.

Examples

```
IF (PSA(3)) THEN
  GETUSER
  INPUT "Addr 1",U_ADDR(0)
  INPUT "Addr 2",U_ADDR(1)
  INPUT "City  ",U_ADDR(2)
  INPUT "State ",U_ADDR(3)
  INPUT "ZIP   ",U_ADDR(4)
  INPUT "Cntry ",U_ADDR(5)
  PUTUSER
ENDIF
```

See Also

GETUSER *Statement*, **PSA()** *Function*, **PUTUSER** *Statement*

U_ALIAS *VARIABLE*

Function

Allow reading and writing of the current users alias.

Type & Value

STRING The current users alias (25 characters max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Additionally, it will only have meaningful information if the alias PSA is installed. The existence of the alias PSA may be checked with the **PSA()** function.

Examples

```
IF (PSA(1)) THEN
  GETUSER
  PRINTLN "Your alias is ",U_ALIAS
ELSE
  PRINTLN "Your name is ",U_NAME()
ENDIF
```

See Also

GETUSER *Statement*, **PSA()** *Function*, **PUTUSER** *Statement*

U_BDL() *Function*

Function

Access the total number of bytes downloaded by the current user.

Syntax

U_BDL()

No arguments are required

Return Type & Value

INTEGER Returns the current users total bytes downloaded.

Remarks

This function will return information that can be useful in modifying PCBoard's built in ratio management system and the view user information command. Of course, it is not limited to that; anywhere you need to know how many bytes the current user has downloaded, this function will provide that information. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You have ULed ",U_BUL()," bytes and DLed ",U_BDL()," bytes."
```

See Also

U_BDLDAY() *Function*, **U_BUL**() *Function*, **U_FDL**() *Function*, **U_FUL**() *Function*

U_BDLDAY() *Function*

Function

Access the number of bytes downloaded by the current user today.

Syntax

```
U_BDLDAY()
```

No arguments are required

Return Type & Value

INTEGER Returns the current users bytes downloaded today.

Remarks

This function will return information that can be useful in modifying PCBoard's built in ratio management system and the view user information command. Of course, it is not limited to that; anywhere you need to know how many bytes the current user has downloaded today, this function will provide that information. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You have downloaded ",U_BDLDAY(),"bytes today."
```

See Also

U_BDL() ***Function***, **U_BUL()** ***Function***, **U_FDL()** ***Function***, **U_FUL()** ***Function***

U_BDPHONE *VARIABLE*

Function

Allow reading and writing of the current users business/data phone number.

Type & Value

STRING The current users business/data phone number (13 characters max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "   Home/Voice Phone Number:  ",U_HVPHONE
PRINTLN "Business/Data Phone Number:  ",U_BDPHONE
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_HVPHONE** *Variable*

U_BUL() *Function*

Function

Access the total number of bytes uploaded by the current user.

Syntax

```
U_BUL()
```

No arguments are required

Return Type & Value

INTEGER Returns the current users total bytes uploaded.

Remarks

This function will return information that can be useful in modifying PCBoard's built in ratio management system and the view user information command. Of course, it is not limited to that; anywhere you need to know how many bytes the current user has uploaded, this function will provide that information. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You have ULed ",U_BUL()," bytes and DLed ",U_BDL()," bytes."
```

See Also

U_BDL() *Function*, **U_BDLDAY()** *Function*, **U_FDL()** *Function*, **U_FUL()** *Function*

U_CITY *VARIABLE*

Function

Allow reading and writing of the current users city information.

Type & Value

STRING The current users city information (24 characters max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Note that this information is separate from the address PSA and does not require that the address PSA be installed.

Examples

```
GETUSER
LET U_CITY = "Timbuktu"
PRINTLN "You are now from Timbuktu!  :)"
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*

U_CLS *VARIABLE*

Function

Allow reading and writing of the current users message clear screen flag.

Type & Value

BOOLEAN The current users clear screen flag status (**TRUE** or **FALSE**).

Remarks

This **BOOLEAN** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Turning on the message clear screen flag..."
LET U_CLS = TRUE
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_LONGHDR** *Variable*, **U_SCROLL** *Variable*

U_CMNT1 *VARIABLE*

Function

Allow reading and writing of the current users comment field.

Type & Value

STRING The current users comment field (30 characters max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Note that this information is separate from the notes PSA (though similar) and does not require that the notes PSA be installed.

Examples

```
GETUSER
PRINTLN " User Comment:   ",U_CMNT1
PRINTLN "SysOp Comment:   ",U_CMNT2
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_CMNT2** *Variable*

U_CMNT2 *VARIABLE*

Function

Allow reading and writing of the current users SysOp comment field.

Type & Value

STRING The current users SysOp comment field (30 characters max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Note that this information is separate from the notes PSA (though similar) and does not require that the notes PSA be installed.

Examples

```
GETUSER
PRINTLN " User Comment:   ",U_CMNT1
PRINTLN "SysOp Comment:   ",U_CMNT2
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_CMNT1** *Variable*

U_DEF79 *VARIABLE*

Function

Allow reading and writing of the current users message editor default width flag.

Type & Value

BOOLEAN The current users default editor width flag status (**TRUE** or **FALSE**).

Remarks

This **BOOLEAN** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Turning on the wide message editor flag..."
LET U_DEF79 = TRUE
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_FSE** *Variable*, **U_FSEP** *Variable*

U_EXPDATE *VARIABLE*

Function

Allow reading and writing of the current users subscription expiration date.

Type & Value

DATE The current users subscription expiration date.

Remarks

This **DATE** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Your subscription will expire on ",U_EXPDATE
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_EXPSEC** *Variable*

U_EXPERT *VARIABLE*

Function

Allow reading and writing of the current users expert status flag.

Type & Value

BOOLEAN The current users expert flag status (**TRUE** or **FALSE**).

Remarks

This **BOOLEAN** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Turning off expert mode..."
LET U_EXPERT = FALSE
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*

U_EXPSEC *VARIABLE*

Function

Allow reading and writing of the current users expired security level.

Type & Value

INTEGER The current users security level (0 - 255).

Remarks

This **INTEGER** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Your security after subscription expiration will be ",U_SEC
```

See Also

CURSEC() ***Function***, **GETUSER** ***Statement***, **PUTUSER** ***Statement***, **U_EXPDATE** ***Variable***, **U_SEC** ***Variable***

U_FDL() *Function*

Function

Access the total number of files downloaded by the current user.

Syntax

U_FDL()

No arguments are required

Return Type & Value

INTEGER Returns the current users total files downloaded.

Remarks

This function will return information that can be useful in modifying PCBoard's built in ratio management system and the view user information command. Of course, it is not limited to that; anywhere you need to know how many files the current user has downloaded. this function will provide that information. Unlike the predefined **U_...** user variables. this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You have ULed ",U_FUL()," bytes and DLed ",U_FDL()," files."
```

See Also

U_BDL() *Function*, **U_BDLDAY()** *Function*, **U_BUL()** *Function*, **U_FUL()** *Function*

U_FSE *VARIABLE*

Function

Allow reading and writing of the current users full screen editor default flag.

Type & Value

BOOLEAN The current users full screen editor default flag status (**TRUE** or **FALSE**).

Remarks

This **BOOLEAN** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Turning on full screen editor as default..."
LET U_FSE = TRUE
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_DEF79** *Variable*, **U_FSEP** *Variable*

U_FSEP *VARIABLE*

Function

Allow reading and writing of the current users full screen editor prompt flag.

Type & Value

BOOLEAN The current users full screen editor prompt flag status (**TRUE** or **FALSE**).

Remarks

This **BOOLEAN** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Turning off full screen editor prompting..."
LET U_FSEP = FALSE
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_DEF79** *Variable*, **U_FSE** *Variable*

U_FUL() *Function*

Function

Access the total number of files uploaded by the current user.

Syntax

```
U_FUL()
```

No arguments are required

Return Type & Value

INTEGER Returns the current users total files uploaded.

Remarks

This function will return information that can be useful in modifying PCBoard's built in ratio management system and the view user information command. Of course, it is not limited to that; anywhere you need to know how many files the current user has uploaded, this function will provide that information. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You have ULed ",U_FUL()," bytes and DLed ",U_FDL()," files."
```

See Also

U_BDL() ***Function*****, U_BDLDAY()** ***Function*****, U_BUL()** ***Function*****, U_FDL()** ***Function***

U_HVPHONE *VARIABLE*

Function

Allow reading and writing of the current users home/voice phone number.

Type & Value

STRING The current users home/voice phone number (13 characters max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "   Home/Voice Phone Number:  ",U_HVPHONE
PRINTLN "Business/Data Phone Number:  ",U_BDPHONE
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_BDPHONE** *Variable*

U_INCONF() *Function*

Function

Determine if a user is registered in a conference.

Syntax

U_INCONF(rec,conf)

rec	An integer expression with the record number of the user to check.
conf	An integer expression with the conference number to check.

Return Type & Value

BOOLEAN	Returns **TRUE** if the user is registered in the specified conference, **FALSE** otherwise.

Remarks

It is sometimes necessary to know if a user is registered in a conference (for example, when entering a message to a particular user). This function will return **TRUE** if the user is registered in the conference specified. Before calling this function you need to find the users record number from the USERS file with the **U_RECNUM()** function.

Examples

```
INTEGER i,rec
STRING  un,ynStr(1)
LET ynStr(0) = "NO"
LET ynStr(1) = "YES"
INPUT "User name",un
NEWLINE
LET rec = U_RECNUM(un)
FOR i = 1 TO 10
  PRINTLN un," in conf ",i,":  ",ynStr(U_INCONF(i,rec))
NEXT
```

See Also

U_RECNUM() ***Function***

U_LDATE() *Function*

Function

Access the last log on date of a user.

Syntax

```
U_LDATE()
```

No arguments are required

Return Type & Value

DATE Returns the current users last log on date.

Remarks

PCBoard tracks the last log on date for each user. This function will return that date for the user currently online. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You last logged on ",U_LDATE(),"."
```

See Also

U_LDIR() *Function*, U_LTIME() *Function*

U_LDIR() *Function*

Function

Access the latest file date found in a file scan by a user.

Syntax

```
U_LDIR()
```

No arguments are required

Return Type & Value

DATE Returns the latest file date found by the current user.

Remarks

PCBoard tracks the latest file found by each user. This function will return that date for the user currently online. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "Latest file found was dated ",U_LDIR(),"."
```

See Also

U_LDATE() ***Function***, **U_LTIME()** ***Function***

U_LOGONS() *Function*

Function

Access the total number of system logons by the current user.

Syntax

U_LOGONS ( )

No arguments are required

Return Type & Value

INTEGER Returns the current users total system logons.

Remarks

PCBoard tracks the total number of logons for each user. This function will return that number for the user currently online. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You have logged on to @BOARDNAME@ ",U_LOGONS(),' times.'
```

See Also

CALLNUM() *Function*, **LOGGEDON()** *Function*, **ONLOCAL()** *Function*

U_LONGHDR *VARIABLE*

Function

Allow reading and writing of the current users long message header flag.

Type & Value

BOOLEAN The current users long message header flag status (**TRUE** or **FALSE**).

Remarks

This **BOOLEAN** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Turning on long message headers..."
LET U_LONGHDR = TRUE
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, **U_CLS** *Variable*, **U_SCROLL** *Variable*

U_LTIME() *Function*

Function

Access the time of day that a user last logged on.

Syntax

```
U_LTIME()
```

No arguments are required

Return Type & Value

TIME Returns the time of day of the current users last log on.

Remarks

PCBoard tracks the last time of day of the last log on for each user. This function will return that time for the user currently online. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You last logged on at ",U_LTIME(),"."
```

See Also

U_LDATE() *Function*, **U_LDIR()** *Function*

U_MSGRD() *Function*

Function

Access the total number of messages read by the current user.

Syntax

U_MSGRD()

No arguments are required

Return Type & Value

INTEGER Returns the current users total messages read.

Remarks

PCBoard tracks the total number of messages read by each user. This function will return that number for the user currently online. One quick idea for use: a message/file ratio enforcement door. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
IF ((U_MSGRD()+U_MSGWR())/U_FDL() > 10) THEN
  PRINTLN "You need to do more messaging!!!"
  END
ENDIF
```

See Also

U_MSGWR() *Function*

U_MSGWR() *Function*

Function

Access the total number of messages written by the current user.

Syntax

U_MSGWR ( )

No arguments are required

Return Type & Value

INTEGER Returns the current users total messages written.

Remarks

PCBoard tracks the total number of messages written by each user. This function will return that number for the user currently online. One quick idea for use: a message/file ratio enforcement door. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
IF ((U_MSGRD()+U_MSGWR())/U_FDL() > 10) THEN
  PRINTLN "You need to do more messaging!!!"
  END
ENDIF
```

See Also

U_MSGRD() *Function*

U_NAME() *Function*

Function

Access the current users name.

Syntax

U_NAME()

No arguments are required

Return Type & Value

STRING Returns a string with the current users name.

Remarks

Perhaps the most important piece of information about a caller is his name. The user name differentiates a user from every other user on the BBS and can be used to track PPE user information that must be kept separate from all other users information. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
IF (U_NAME() = "JOHN DOE") THEN
  PRINTLN "I know who you are!  Welcome!"
  GETUSER
  LET U_SEC = 110
  PUTUSER
  PRINTLN "Automatically upgraded!"
ENDIF
```

See Also

CURCONF() ***Function***, **MESSAGE** ***Statement***

U_NOTES() *VARIABLE ARRAY*

Function

Allow reading and writing of current user notes.

Type & Value

STRING

Subscript 0-4 SysOp defineable user notes (60 characters max).

Remarks

This array is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that the array is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Additionally, the array will only have meaningful information if the notes PSA is installed. The existence of the notes PSA may be checked with the **PSA()** function.

Examples

```
INTEGER i
IF (PSA(6)) THEN
  GETUSER
  FOR i = 0 TO 4
    PRINTLN "Note ",i+1,":   ",U_NOTES(i)
  NEXT
ENDIF
```

See Also

GETUSER *Statement*, **PSA()** *Function*, **PUTUSER** *Statement*

U_PAGELEN *VARIABLE*

Function

Allow reading and writing of the current users page length setting.

Type & Value

INTEGER The current users page length (0 - 255).

Remarks

This integer is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Your page length was ",U_PAGELEN
LET U_PAGELEN = 20
PRINTLN "Your page length is now ",U_PAGELEN
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*

U_PWD *VARIABLE*

Function

Allow reading and writing of the current users password.

Type & Value

STRING The current users password (12 characters max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. A mask of valid characters that may be used in the **U_PWD** variable is available via the **MASK_PWD** function.

Examples

```
STRING s
INPUT "Enter Password",s,@X0E,12,MASK_PWD(),UPCASE
GETUSER
IF (s <> U_PWD) THEN
  PRINTLN "Sorry, hanging up"
  HANGUP
ENDIF
```

See Also

GETUSER *Statement*, **MASK_PWD()** *Function*, **NEWPWD** *Statement*, **PUTUSER** *Statement*, **U_PWDEXP** *Variable*, **U_PWDHIST()** *Function*, **U_PWDLC()** *Function*, **U_PWDTC()** *Function*

U_PWDEXP *VARIABLE*

Function

Allow reading and writing of the current users password expiration date.

Type & Value

DATE The current users password expiration date.

Remarks

This **DATE** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Additionally, it will only have meaningful information if the password PSA is installed. The existence of the password PSA may be checked with the **PSA()** function.

Examples

```
IF (PSA(4)) THEN
  GETUSER
  PRINTLN U_PWDEXP-DATE()," until current password expiration"
  LET U_PWDEXP = DATE()+30
  PRINTLN "You now have 30 days until you *MUST* change you password."
  PUTUSER
ENDIF
```

See Also

GETUSER *Statement*, **NEWPWD** *Statement*, **PSA()** *Function*, **PUTUSER** *Statement*, **U_PWD** *Variable*, **U_PWDHIST()** *Function*, **U_PWDLC()** *Function*, **U_PWDTC()** *Function*

U_PWDHIST() *Function*

Function

Access the last three passwords used by the current user.

Syntax

```
U_PWDHIST(num)
```

num — The number of the password from the history to return (1 through 3).

Return Type & Value

STRING — Returns the specified password from the history (1 for the most recent, 3 for the least recent).

Remarks

PCBoard has the ability to track the last three passwords used by each user. This function will return one of those passwords from the history for the user currently online. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information. However, it does require that the password PSA has been installed to return meaningful information. The existence of the password PSA may be checked via the **PSA()** function.

Examples

```
INTEGER i
IF (PSA(4)) THEN
  FOR i = 1 TO 3
    PRINTLN "Password history ",i,":  ",U_PWDHIST(i)
  NEXT
ENDIF
```

See Also

NEWPWD *Statement*, **PSA()** *Function*, **U_PWD** *Variable*, **U_PWDEXP** *Variable*, **U_PWDLC()** *Function*, **U_PWDTC()** *Function*

U_PWDLC() *Function*

Function

Access the last date the user changed his password.

Syntax

U_PWDLC ( )

No arguments are required

Return Type & Value

DATE Returns the last date the user changed his password.

Remarks

PCBoard has the ability to track the last date of a password change for each user. This function will return that date for the user currently online. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information. However, it does require that the password PSA has been installed to return meaningful information. The existence of the password PSA may be checked via the **PSA()** function.

Examples

```
IF (PSA(4)) PRINTLN "You last changed your password on ",U_PWDLC(),"."
```

See Also

NEWPWD *Statement*, **PSA()** *Function*, **U_PWD** *Variable*, **U_PWDEXP** *Variable*, **U_PWDHIST()** *Function*, **U_PWDTC()** *Function*

U_PWDTC() *Function*

Function

Access the number of times the user has changed his password.

Syntax

U_PWDTC()

No arguments are required

Return Type & Value

INTEGER Returns the number of times the user has changed his password.

Remarks

PCBoard has the ability to track the total number of times each user changes his password. This function will return that count for the user currently online. Unlike the predefined U_... user variables, this function does not require the use of **GETUSER** to return valid information. However, it does require that the password PSA has been installed to return meaningful information. The existence of the password PSA may be checked via the **PSA()** function.

Examples

```
IF (PSA(4)) THEN
  PRINTLN "You have changed your password ",U_PWDTC()," times."
ENDIF
```

See Also

NEWPWD *Statement*, **PSA()** *Function*, **U_PWD** *Variable*, **U_PWDEXP** *Variable*, **U_PWDHIST()** *Function*, **U_PWDLC()** *Function*

U_RECNUM() *Function*

Function

Determine if a user is registered on the system and what the record number is.

Syntax

U_RECNUM(user)

user	A string expression with the user name to search for.

Return Type & Value

INTEGER	Returns the record number of the user in the USERS file if found or -1 if not found.

Remarks

This function serves two purposes. The first is to determine whether or not a given user name is registered on the system. If the value -1 is returned the user isn't in the user files. The second use is to get the users record number for the **U_INCONF()** function to determine whether or not a user is registered in a given conference.

Examples

```
INTEGER i,rec
STRING  un,ynStr(1)
LET ynStr(0) = "NO"
LET ynStr(1) = "YES"
INPUT "User name",un
NEWLINE
LET rec = U_RECNUM(un)
FOR i = 1 TO 10
  PRINTLN un," in conf ",i,":  ",ynStr(U_INCONF(i,rec))
NEXT
```

See Also

U_INCONF() *Function*

U_SCROLL *VARIABLE*

Function

Allow reading and writing of the current users multi-screen message scroll flag.

Type & Value

BOOLEAN The current users scroll flag status (**TRUE** or **FALSE**).

Remarks

This **BOOLEAN** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Turning off message scrolling..."
LET U_SCROLL = FALSE
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*, U_CLS *Variable*, U_LONGHDR *Variable*

U_SEC *VARIABLE*

Function

Allow reading and writing of the current users security level.

Type & Value

INTEGER The current users security level (0 - 255).

Remarks

This **INTEGER** is set with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is undefined until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed.

Examples

```
GETUSER
PRINTLN "Raising your security to level 20..."
LET U_SEC = 20
PUTUSER
PRINTLN "Automatic upgrade complete!"
```

See Also

CURSEC() *Function*, **GETUSER** *Statement*, **PUTUSER** *Statement*, **U_EXPSEC** *Variable*

U_STAT() *Function*

Function

Access a statistic about the current user.

Syntax

```
U_STAT(stat)
```

`stat`	The statistic to retrieve (1 through 15).

Return Type & Value

DATE	Returns the first date the user called the system if **stat** is 1.
INTEGER	Returns one of the following for all other values of **stat**:
2	The number of times the user has paged the SysOp;
3	The number of group chats the user has participated in;
4	The number of comments left by the user;
5	The number of 300 bps connects by the user;
6	The number of 1200 bps connects by the user;
7	The number of 2400 bps connects by the user;
8	The number of connects by the user greater than 2400 bps and less than or equal to 9600 bps (9600 bps >= connect speed > 2400 bps);
9	The number of connects by the user greater than 9600 bps and less than or equal to 14,400 bps (14,400 bps >= connect speed > 9600 bps);
10	The number of security violations by the user;
11	The number of "not registered in conference" warnings to the user;
12	The number of times the user's download limit has been reached;
13	The number of "file not found" warnings to the user;
14	The number of password errors to access the user's account;
15	The number of verify errors to access the user's account.

Remarks

PCBoard has the ability to track a number of statistics about the user. This function will return the desired statistic for the user currently online. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information. However, it does require that the statistics PSA has been installed to return meaningful information. The existence of the statistics PSA may be checked via the **PSA()** function.

Examples

```
STRING  label
INTEGER i
FOPEN 1,PPEPATH()+"STATTEXT",O_RD,S_DN
FOR i = 1 TO 15
  FGET 1,label
  PRINTLN label," - ",U_STAT(i)
NEXT
FCLOSE 1
```

See Also

PSA() ***Function***

U_TIMEON() *Function*

Function

Access the users time online today in minutes.

Syntax

U_TIMEON ( )

No arguments are required

Return Type & Value

INTEGER Returns the users time online today in minutes.

Remarks

PCBoard tracks the users time online each day. This function will return the elapsed time for the user currently online. Unlike the predefined **U_...** user variables, this function does not require the use of **GETUSER** to return valid information.

Examples

```
PRINTLN "You have been online for ",U_TIMEON(),"  total minutes today."
```

See Also

ADJTIME *Statement*, **MINLEFT()** *Function*, **MINON()** *Function*

U_TRANS *VARIABLE*

Function

Allow reading and writing of the current users default transfer protocol letter.

Type & Value

STRING The current users default transfer protocol letter (1 character max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Valid characters that may be used in the **U_TRANS** variable are A through Z and 0 through 9.

Examples

```
GETUSER
PRINTLN "Your default file transfer protocol letter is ",U_TRANS
LET U_TRANS = "N" ' Set to no default protocol
PRINTLN "Default file transfer protocol letter set to None"
PUTUSER
```

See Also

GETUSER *Statement*, **PUTUSER** *Statement*

U_VER *VARIABLE*

Function

Allow reading and writing of the current users verification string.

Type & Value

STRING The current users verification string (25 characters max).

Remarks

This **STRING** is filled with information from the current users record when the **GETUSER** statement is executed. It may then be changed and written back to the users record with the **PUTUSER** statement. Note that it is empty until a **GETUSER** statement is processed and that changes are not written until a **PUTUSER** statement is processed. Additionally, it will only have meaningful information if the verification PSA is installed. The existence of the verification PSA may be checked with the **PSA()** function.

Examples

```
STRING s
IF (PSA(2)) THEN
  GETUSER
  PRINTLN "Enter verification information"
  INPUT "",s
  IF (s <> U_VER) HANGUP
ELSE
  PRINTLN "No verification information available"
ENDIF
```

See Also

GETUSER *Statement*, **PSA()** *Function*, **PUTUSER** *Statement*

VALCC() *Function*

Function

Tests a string for credit card number format validity.

Syntax

VALCC(sexp)

sexp	Any string expression.

Return Type & Value

BOOLEAN	Returns **TRUE** if the string is a valid credit card number format, **FALSE** otherwise.

Remarks

This function will take a string and attempt to identify it as a credit card number. If the number is invalid for any reason (insufficient digits or bad checksum, primarily) then this function will return **FALSE**, otherwise it will return **TRUE**.

Examples

```
STRING s
WHILE (!VALCC(s)) DO
  INPUT "CC #",s
  NEWLINES 2
ENDWHILE
PRINTLN CCTYPE(s)," - ",FMTCC(s)
```

See Also

CCTYPE() ***Function*****, FMTCC()** ***Function*****, VALDATE()** ***Function*****, VALTIME()** ***Function***

VALDATE() *Function*

Function

Tests a string for date format validity.

Syntax

VALDATE(sexp)

sexp Any string expression.

Return Type & Value

BOOLEAN Returns **TRUE** if the string is a valid date format, **FALSE** otherwise.

Remarks

PPL does it best to convert incompatible types, as needed, automatically. Converting a **STRING** type to a **DATE** type is particularly problematic because of the virtually unlimited numbers of strings possible. This function checks to make sure that the hour is from 0 to 23, the minute is from 0 to 59, and the second (optional) is from 0 to 59. Also, each field (hours/minutes/seconds) must be separated by a colon. If the string matches these requirements then the string is considered valid and **TRUE** is returned. Any other string will result in a **FALSE** value being returned.

Examples

```
STRING s
WHILE (!VALTIME(s)) DO
  INPUT "Time",s
  NEWLINES 2
ENDWHILE
TIME t
LET t = s
PRINTLN s," ",t
```

See Also

VALCC() *Function*, **VALDATE()** *Function*

VALTIME() *Function*

Function

Tests a string for time format validity.

Syntax

VALTIME(sexp)

sexp	Any string expression.

Return Type & Value

BOOLEAN	Returns **TRUE** if the string is a valid time format, **FALSE** otherwise.

Remarks

PPL does it best to convert incompatible types, as needed, automatically. Converting a **STRING** type to a **TIME** type is particularly problematic because of the virtually unlimited numbers of strings possible. This function checks to make sure that the hour is from 0 to 23, the minute is from 0 to 59, and the second (optional) is from 0 to 59. Also, each field (hours/minutes/seconds) must be separated by a colon. If the string matches these requirements then the string is considered valid and **TRUE** is returned. Any other string will result in a **FALSE** value being returned.

Examples

```
STRING s
WHILE (!VALTIME(s)) DO
  INPUT "Time",s
  NEWLINES 2
ENDWHILE
TIME t
LET t = s
PRINTLN s," ",t
```

See Also

VALCC() ***Function*****, VALDATE()** ***Function***

VARADDR *Statement*

Function

Sets a variable to the complete address of another variable.

Syntax

VARADDR src,dest

src	The variable to get the address of.
dest	The variable to store the address in.

Remarks

This statement is primarily useful in conjunction with the **DOINTR** statement. It may be necessary to give an interrupt the address of a memory location that can be used to store information. This statement will allow you to get the address of a specified variable to pass to the **DOINTR** statement.

Examples

```
' Create subdirectory - DOS function 39h
INTEGER addr
STRING  path
LET path = "C:\$TMPDIR$"
VARADDR path,addr
DOINTR 21h,39h,0,0,addr%10000h,0,0,0,addr/10000h,0
IF (REGCF() & (REGAX() = 3)) THEN
  PRINTLN "Error:  Path not found"
ELSE IF (REGCF() & (REGAX() = 5)) THEN
  PRINTLN "Error:  Access Denied"
ELSE IF (REGCF()) THEN
  PRINTLN "Error:  Unknown Error"
ELSE
  PRINTLN "Directory successfully created..."
ENDIF
```

See Also

DOINTR *Statement*, **MKADDR()** *Function*, **PEEKB()** *Function*, **PEEKDW()** *Function*, **PEEKW()** *Function*, **POKEB()** *Function*, **POKEDW()** *Function*, **POKEW()** *Function*, **VAROFF** *Statement*, **VARSEG** *Statement*

VAROFF *Statement*

Function

Sets a variable to the offset address of another variable.

Syntax

```
VAROFF src,dest
```

`src`	The variable to get the offset address of.
`dest`	The variable to store the offset address in.

Remarks

This statement is primarily useful in conjunction with the **DOINTR** statement. It may be necessary to give an interrupt the address of a memory location that can be used to store information. This statement will allow you to get the offset address of a specified variable to pass to the **DOINTR** statement.

Examples

```
' Create subdirectory - DOS function 39h
INTEGER saddr, oaddr
STRING  path
LET path = "C:\$TMPDIR$"
VARSEG path,saddr
VAROFF path,oaddr
DOINTR 21h,39h,0,0,oaddr,0,0,0,saddr,0
IF (REGCF() & (REGAX() = 3)) THEN
  PRINTLN "Error:  Path not found"
ELSE IF (REGCF() & (REGAX() = 5)) THEN
  PRINTLN "Error:  Access Denied"
ELSE IF (REGCF()) THEN
  PRINTLN "Error:  Unknown Error"
ELSE
  PRINTLN "Directory successfully created..."
ENDIF
```

See Also

DOINTR *Statement*, **MKADDR()** *Function*, **PEEKB()** *Function*, **PEEKDW()** *Function*, **PEEKW()** *Function*, **POKEB()** *Function*, **POKEDW()** *Function*, **POKEW()** *Function*, **VARADDR** *Statement*, **VARSEG** *Statement*

VARSEG *Statement*

Function

Sets a variable to the segment address of another variable.

Syntax

VARSEG src,dest

src	The variable to get the segment address of.
dest	The variable to store the segment address in.

Remarks

This statement is primarily useful in conjunction with the **DOINTR** statement. It may be necessary to give an interrupt the address of a memory location that can be used to store information. This statement will allow you to get the segment address of a specified variable to pass to the **DOINTR** statement.

Examples

```
' Create subdirectory - DOS function 39h
INTEGER saddr, oaddr
STRING  path
LET path = "C:\$TMPDIR$"
VARSEG path,saddr
VAROFF path,oaddr
DOINTR 21h,39h,0,0,oaddr,0,0,0,saddr,0
IF (REGCF() & (REGAX() = 3)) THEN
  PRINTLN "Error:  Path not found"
ELSE IF (REGCF() & (REGAX() = 5)) THEN
  PRINTLN "Error:  Access Denied"
ELSE IF (REGCF()) THEN
  PRINTLN "Error:  Unknown Error"
ELSE
  PRINTLN "Directory successfully created..."
ENDIF
```

See Also

DOINTR *Statement*, **MKADDR()** *Function*, **PEEKB()** *Function*, **PEEKDW()** *Function*, **PEEKW()** *Function*, **POKEB()** *Function*, **POKEDW()** *Function*, **POKEW()** *Function*, **VARADDR** *Statement*, **VAROFF** *Statement*

VER() *Function*

Function

Get the version of PPL available.

Syntax

VER ( )

No arguments are required

Return Type & Value

INTEGER Returns the version number of PPL running.

Remarks

As time passes, new features will be added to PCBoard and PPL. Of course, in order to utilize the new features, you must be running a version of PCBoard that supports them. This function will return the version of PCBoard (and PPL). For PCBoard version 15.0 this value will be 1500. In other words, the major version will be accessable via **VER**()/100, and the minor version will be available via **VER**()%100. Everything documented herein will be available for all versions greater than or equal to 1500. Future PPL features will be documented with the required version.

Examples

```
IF (VER() < 1600) THEN
  PRINTLN "PCBoard Version 16.0 required for this PPE file"
  END
ENDIF
FOO a,b,c,d,e ' Obviously, this is not a 15.0 statement
```

See Also

PSA() *Function*

WAIT *Statement*

Function

Wait for the user to hit ENTER.

Syntax

WAIT

No arguments are required

Remarks

It is often necessary to pause in the display of information and wait for the user to catch up. This statement allows you to wait for the user to hit ENTER before continuing. It displays prompt number 418 from the PCBTEXT file for the current language to let the user know what is expected.

Examples

```
PRINTLN "Your account has expired!"
PRINTLN "You are about to be logged off"
WAIT
```

See Also

DISPTEXT *Statement*, **INKEY$()** *Function*, **MORE** *Statement*, **PROMPTSTR** *Statement*

WAITFOR *Statement*

Function

Wait for a specific string of text to come in from the modem.

Syntax

```
WAITFOR str,flag,sec
```

`str`	Any string expression.
`flag`	A variable to return the status.
`sec`	An integer expression with the maximum number of seconds to wait.

Remarks

This statement can be used to wait for specific replies to questions, responses from terminal emulators and modem result codes. If the text that is needed isn't received within the specified time period, or if there is not a remote caller online, flag will be set to **FALSE**. If the text is found, then flag will be **TRUE**. If a remote caller is online this statement will wait up to the maximum time for the text and return **TRUE** or **FALSE** as appropriate. If the caller is local, it will immediately return **FALSE**. Also, the text to wait for is not case sensitive. "connect" will match "CONNECT".

Examples

```
BOOLEAN flag
KBDCHKOFF
CDCHKOFF
DTROFF
DELAY 18
DTRON
SENDMODEM "ATDT5551212" ' Please don't really dial this number!
WAITFOR "CONNECT",flag,60
IF (!flag) SPRINLN "No connect found in 60 seconds"
CDCHKON
KBDCHKON
```

See Also

DELAY *Statement*, **MGETBYTE()** *Function*, **SENDMODEM** *Statement*

WHILE/ENDWHILE *Statement*

Function

Execute one or more statments while a condition is true.

Syntax

```
WHILE (bexp) statement
-or-
WHILE (bexp) DO
  statement(s)
ENDWHLLE
```

`bexp` Any boolean expression.

`statement` Any valid PPL statement.

Remarks

Computers are known for their ability to perform monotonous tasks quickly, efficiently, and accurately. What better way to implement monotony than through a **WHILE** loop? The **WHILE** statement supports two types of loops: logical and block. A logical **WHILE** loop is a single statement; if a condition is **TRUE**, execute a single statement and check again. A block **WHILE** loop can be one or more statements. The start of a block **WHILE** loop is differentiated from a logical **WHILE** loop by the **DO** keyword immediately after the condition. At some point in the loop some action must be taken that will make the condition **FALSE**. If the condition never changes from **TRUE** to **FALSE** you have what is known as an infinite loop; your computer will appear to be hung, even though it is rapidly executing things just as fast as it can. Be sure to thoroughly test all programs, but especially programs with loops!

Examples

```
INTEGER i
LET i = 0
WHILE (i < 10) GOSUB sub
END
:sub
PRINTLN "i is ",i
INC i
RETURN

INTEGER i
LET i = 0
WHILE (i < 10) DO
  PRINTLN "i is ",i
  INC i
ENDWHILE
```

See Also

GOSUB *Statement*, **GOTO** *Statement*, **FOR/NEXT** *Statement*, **IF/ELSEIF/ELSE/ENDIF** *Statement*, **RETURN** *Statement*

WORDWRAP *Constant*

Function

Set the word wrap flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

512 = 1000000000b = 1000o = 200h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to word wrap from one input statement to the next input statement. If you reach the end of the input field PCBoard will automatically save the last word from the input field in an internal buffer. The next input statement will use that saved word if both statements used the **WORDWRAP** constant. If the passed variable isn't empty or if an input statement is used that doesn't have the **WORDWRAP** flag set then the saved word will not be used.

Examples

```
STRING  s(5)
INTEGER i
CLS
FOR i = 1 TO 5
  INPUTSTR "Line "+STRING(i),s(i),@X0E,40,MASK_ASCII(),WORDWRAP+NEWLINE
NEXT
CLS
FOR i = 1 TO 5
  PRINTLN "Line ",i,":  ",s(i)
NEXT
```

See Also

INPUTSTR *Statement*, **PROMPTSTR** *Statement*

WRUNET *Statement*

Function

Write information to the USERNET file for a specific node.

Syntax

```
WRUNET node,stat,name,city,oper,br
```

`node`	An integer expression with the node to update.
`stat`	A string expression with the new node status.
`name`	A string expression with the new node user name.
`city`	A string expression with the new node city.
`oper`	A string expression with the new node operation text.
`br`	A string expression with the new node broadcast text.

Remarks

To facilitate internode communications, a file named USERNET.XXX is maintained with an entry for each node on the system. This file is used by the BROADCAST command of PCBoard and to prevent multiple simultaneous logins, among other things. This statement may be used to change information for the current node (for example, to update operation text during PPE execution) or other nodes (for example, to broadcast a message).

Examples

```
RDUNET PCBNODE()
WRUNET PCBNODE(),UN_STAT(),UN_NAME(),UN_CITY(),"Running "+PPENAME(),""
RDUNET 1
WRUNET 1,UN_STAT(),UN_NAME(),UN_CITY(),UN_OPER(),"Hello there node 1"
```

See Also

BROADCAST *Statement*, **RDUNET** *Statement*, **UN_...()** *Functions*

WRUSYS *Statement*

Function

Write a USERS.SYS file out to disk.

Syntax

WRUSYS

No arguments are required

Remarks

Some DOOR applications require a USERS.SYS file to access information about the caller. This statement allows you to create that file prior to running an application via the **SHELL** statement. Should the DOOR make changes to the USERS.SYS file, you should use the **RDUSYS** statement after the **SHELL** to read the changes back into memory. It should be noted that it is not possible to create the USERS.SYS file with a TPA record with this statement.

Examples

```
INTEGER ret
WRUSYS
SHELL FALSE,ret,"MYAPP.EXE",""
RDUSYS
```

See Also

RDUSYS *Statement*, **SHELL** *Statement*

XOR() *Function*

Function

Calculate the bitwise XOR (exclusive or) of two integer arguments.

Syntax

```
AND(iexp1,iexp2)
```

iexp1	Any integer expression.
iexp2	Any integer expression.

Return Type & Value

INTEGER	Returns the bitwise XOR of **iexp1** and **iexp2**.

Remarks

This function may be used to toggle selected bits in an integer expression by XORing the expression with a mask that has the bits to toggle set to 1 and the bits to ignore set to 0.

Examples

```
' Toggle the bits in the low byte
PRINTLN XOR(1248h,00FFh)
' Toggle a flag
INTEGER flag
LET flag = XOR(flag,1)
```

See Also

AND() ***Function***, **NOT()** ***Function***, **OR()** ***Function***

YEAR() *Function*

Function

Extracts the year from a specified date.

Syntax

```
YEAR(dexp)
```

dexp	Any date expression.

Return Type & Value

INTEGER	Returns the year from the specified date expression (**dexp**). Valid return values are from 1900 to 2079.

Remarks

This function allows you to extract a particular piece of information about a **DATE** value, in this case the year of the date.

Examples

```
PRINTLN "This year is:  ",YEAR(DATE())
```

See Also

DATE() *Function*, **DAY()** *Function*, **DOW()** *Function*, **MONTH()** *Function*

YESCHAR() *Function*

Function

Get the yes response character for the current language.

Syntax

YESCHAR()

No arguments are required

Return Type & Value

STRING Returns the yes character for the current language.

Remarks

Support for foreign language yes/no responses can be easily added by using this function to determine what an affirmative response should be instead of hardcoding the english "Y" character.

Examples

```
STRING ans
LET ans = YESCHAR()
INPUTSTR "Run program now",ans,@X0E,1,"",AUTO+YESNO
IF (ans = NOCHAR()) END
```

See Also

NOCHAR() *Function*, YESNO *Constant*

YESNO *Constant*

Function

Set the international yes/no response flag in an **INPUTSTR** or **PROMPTSTR** statement.

Value

16384 = 100000000000000b = 40000o = 4000h

Remarks

The **INPUTSTR** and **PROMPTSTR** statements have the ability to allow a yes/no response to be entered in addition to any valid characters passed to the statement. The extra characters allowed are Y/N (or whatever characters were defined for the current language; spanish would use S/N, french would use O/N, etc). Note that you do not need to pass any valid characters to use this flag; regardless of the other legal characters the international Y/N characters will be allowed.

Examples

```
STRING ans
LET ans = NOCHAR()
INPUTSTR "Run program now",ans,@X0E,1,"",AUTO+YESNO
IF (ans = NOCHAR()) END
```

See Also

INPUTSTR *Statement*, **NOCHAR()** *Function*, **PROMPTSTR** *Statement*, **YESCHAR()** *Function*

Index

A

ABORT(), 57
ABS(), 58
ADJTIME, 59
AND(), 60
ANSION(), 61
ANSIPOS, 62
Application
    Installation, 12
    Testing, 14
ASC(), 63
Assignment
    LET, 178
AUTO, 64

B

B2W(), 65
BACKUP, 66
Basics, 47
BELL, 67
BLT, 68
BOOLEAN, 69
Branching
    ELSE, 156
    ELSEIF, 156
    ENDIF, 156
    ENDWHILE, 345
    FOR, 128
    GOSUB, 146
    GOTO, 147
    IF, 156
    NEXT, 128
    RETURN, 251
    WHILE, 345
BROADCAST, 70
BYE, 71

C

CALL, 72
CALLID(), 73
CALLNUM(), 74
CARRIER(), 75
CCTYPE(), 76
CDCHKOFF, 77
CDCHKON, 78
CDON(), 79
CHAT, 80
CHR(), 81
CLOSECAP, 82
CLREOL, 83
CLS, 84
Code Statements, 48
COLOR, 85
Color Control
    COLOR, 85
    CURCOLOR(), 88
    DEFCOLOR, 97
    DEFCOLOR(), 98
Commands, 12, 29
Comments, 47
Compiler
    Errors, 20
    Exit Codes, 23
    Warnings, 19
Compiling Source Code, 11, 18
Conference
    CONFFLAG, 86
    CONFUNFLAG, 87
    CURCONF(), 89
    F_EXP, 134
    F_MW, 135
    F_REG, 136
    F_SEL, 137
    F_SYS, 138
    U_INCONF(), 312
CONFFLAG, 86
CONFUNFLAG, 87
Connection Information
    CALLID(), 73
    CALLNUM(), 74
    CARRIER(), 75
    LOGGEDON(), 182
    MODEM(), 198
    ONLOCAL(), 211
Constant List, 55
Constants, 49
CPU Access
    DOINTR, 107
    MKADDR(), 196
    PEEKB(), 223
    PEEKDW(), 224
    PEEKW(), 225
    POKEB, 226
    POKEDW, 227
    POKEW, 228
    REGAH(), 245
    REGAL(), 245
    REGAX(), 245
    REGBH(), 245
    REGBL(), 245
    REGBX(), 245
    REGCF(), 245
    REGCH(), 245
    REGCL(), 245
    REGCX(), 245
    REGDH(), 245
    REGDI(), 245
    REGDL(), 245
    REGDS(), 245
    REGDX(), 245
    REGES(), 245
    REGF(), 245
    REGSI(), 245
    VARADDR, 339
    VAROFF, 340
    VARSEG, 341
Creating Source Code, 11
Credit Cards
    CCTYPE(), 76
    FMTCC(), 125
    VALCC(), 336
CURCOLOR(), 88
CURCONF(), 89
CURSEC(), 90
Cursor
    ANSIPOS, 62

Index

BACKUP, 66
FORWARD, 129
GETX, 143
GETY, 144

D

DATE, 91
Date
DATE(), 92
DAY(), 93
DOW(), 108
MKDATE(), 197
MONTH(), 200
VALDATE(), 337
YEAR(), 351
DATE(), 92
DAY(), 93
DBGLEVEL, 94
DBGLEVEL(), 95
Debugging
DBGLEVEL, 94
DBGLEVEL(), 95
LOG, 181
DEC, 96
DEFCOLOR, 97
DEFCOLOR(), 98
DEFS, 99
DELAY, 100
DELETE, 101
DELUSER, 102
Developing PPL Applications, 11
DIR, 103
DISPFILE, 104
DISPFILE Flags
GRAPH, 149
LANG, 174
SEC, 258
Display Files, 13, 39
Display Menus, 14, 42
DISPSTR, 105
DISPTEXT, 106
DISPTEXT Flags
BELL, 67
LFAFTER, 179
LFBEFORE, 180
LOGIT, 183
LOGITLEFT, 184
NEWLINE, 204
DOINTR, 107
DOW(), 108
DTROFF, 109
DTRON, 110

E

ECHODOTS, 111
ELSE, 156
ELSEIF, 156
END, 112
ENDIF, 156
ENDWHILE, 345
ERASELINE, 113
Errors, 20
EXIST(), 114
Exit Codes, 23
Expressions, 49

F

F_EXP, 134
F_MW, 135
F_REG, 136
F_SEL, 137
F_SYS, 138
FALSE, 115
FAPPEND, 116
FCL, 117
FCLOSE, 118
FCREATE, 119
FERR(), 120
FGET, 121
FIELDLEN, 122
File
DELETE, 101
EXIST(), 114
FAPPEND, 116
FCLOSE, 118
FCREATE, 119
FERR(), 120
FGET, 121
FILEINF(), 123
FOPEN, 127
FPUT, 130
FPUTLN, 130
FPUTPAD, 131
FREWIND, 133
READLINE(), 244
RENAME, 247
FILEINF(), 123
FMTCC(), 125
FNS, 126
FOPEN, 127
FOR, 128
FORWARD, 129
FPUT, 130
FPUTLN, 130
FPUTPAD, 131
FRESHLINE, 132
FREWIND, 133
Function List, 55
Functions, 50

G

GETENV(), 139
GETTOKEN, 140
GETTOKEN(), 141
GETUSER, 142
GETX(), 143
GETY(), 144
GOODBYE, 145
GOSUB, 146
GOTO, 147
GRAFMODE(), 148
GRAPH, 149
Graphics
ANSION(), 61
GRAFMODE(), 148

Index

GUIDE, 150

H

HANGUP, 151
Hello, World!, 27
HELPPATH(), 152
HIGHASCII, 153
HOUR(), 154

I

I2S(), 155
IF, 156
INC, 158
INKEY(), 159
INPUT, 160
Input
    INPUT, 160
    INPUTCC, 161
    INPUTDATE, 161
    INPUTINT, 161
    INPUTMONEY, 161
    INPUTSTR, 163
    INPUTTEXT, 165
    INPUTTIME, 161
    INPUTYN, 161
    PROMPT, 234
Input Flags
    AUTO, 64
    ECHODOTS, 111
    ERASELINE, 113
    FIELDLEN, 122
    GUIDE, 150
    HIGHASCII, 153
    LFAFTER, 179
    LFBEFORE, 180
    NEWLINE, 204
    NOCLEAR, 209
    STACKED, 269
    UPCASE, 291
    WORDWRAP, 347
    YESNO, 353
Input Masks
    MASK_ALNUM(), 187
    MASK_ALPHA(), 187
    MASK_ASCII(), 187
    MASK_FILE(), 187
    MASK_NUM(), 187
    MASK_PATH(), 187
    MASK_PWD(), 187
INPUTCC, 161
INPUTDATE, 161
INPUTINT, 161
INPUTMONEY, 161
INPUTSTR, 163
INPUTTEXT, 165
INPUTTIME, 161
INPUTYN, 161
Installing PPLC, 7
Installing Your Application, 12
INSTR(), 166
INTEGER, 167
Interactive Welcome Screens, 39
Internationalization
    NOCHAR(), 208
    YESCHAR(), 352
Introduction to PPL, 3

J

JOIN, 168

K

KBDCHKOFF, 169
KBDCHKON, 170
KBDFILE, 171
KBDSTUFF, 172
Keyboard
    INKEY(), 159
    KBDFILE, 171
    KBDSTUFF, 172
    KINKEY(), 173
    MGETBYTE(), 190
    MINKEY(), 193
KINKEY(), 173

L

LANG, 174
LANGEXT(), 175
LEFT(), 176
LEN(), 177
LET, 178
LFAFTER, 179
LFBEFORE, 180
LOG, 181
LOGGEDON(), 182
LOGIT, 183
LOGITLEFT, 184
Logon Language Prompt, 37
LOWER(), 185
LTRIM(), 186

M

MASK_ALNUM(), 187
MASK_ALPHA(), 187
MASK_ASCII(), 187
MASK_FILE(), 187
MASK_NUM(), 187
MASK_PATH(), 187
MASK_PWD(), 187
MAXNODE(), 188
Menus, 14
MESSAGE, 189
MGETBYTE(), 190
MID(), 191
MIN(), 192
MINKEY(), 193
MINLEFT(), 194
MINON(), 195
Miscellaneous Constants
    DEFS, 99
    FALSE, 115
    TRUE, 289
MKADDR(), 196
MKDATE(), 197

Index

Modem
- CDON(), 79
- DTROFF, 109
- DTRON, 110
- SENDMODEM, 260
- WAITFOR, 344

MODEM(), 198
MONEY, 199
MONTH(), 200
MORE, 201
MPRINT, 202
MPRINTLN, 202

N

NC, 203
NEWLINE, 204, 205
NEWLINES, 206
NEWPWD, 207
NEXT, 128
NOCHAR(), 208
NOCLEAR, 209
Node
- RDUNET, 242
- UN_CITY(), 290
- UN_NAME(), 290
- UN_OPER(), 290
- UN_STAT(), 290
- WRUNET, 348

Node Specific Display Files, 39
NOT(), 210
Numerical
- ABS(), 58
- AND(), 60
- B2W(), 65
- DEC, 96
- INC, 158
- NOT(), 210
- OR(), 214
- RANDOM(), 241
- XOR(), 350

O

O_RD, 215
O_RW, 216
O_WR, 217
ONLOCAL(), 211
Open Flags
- O_RD, 215
- O_RW, 216
- O_WR, 217

OPENCAP, 212
Operator Page, 30
Operators, 51
- Precedence, 52

OPTEXT, 213
OR(), 214

P

PAGEOFF, 218
PAGEON, 219
PAGESTAT(), 220
Password Expiration Warning, 36
PCBDAT(), 221
PCBNODE(), 222
PCBOARD.DAT Information
- HELPPATH(), 152
- PCBDAT(), 221
- PCBNODE(), 222
- SLPATH(), 265
- SYSOPSEC(), 276
- TEMPPATH(), 281

PCBoard Commands
- BLT, 68
- BROADCAST, 70
- BYE, 71
- DIR, 103
- GOODBYE, 145
- JOIN, 168
- MESSAGE, 189
- QUEST, 240

PCBTEXT Display Prompts, 13, 36
PEEKB(), 223
PEEKDW(), 224
PEEKW(), 225
POKEB, 226
POKEDW, 227
POKEW, 228
POP, 229
PPE Files
- Commands, 12
- Display Files, 13
- Display Menus, 14
- PCBTEXT Display Prompts, 13
- Script Questionnaires, 13

PPE Information
- PPENAME(), 231
- PPEPATH(), 232

PPENAME(), 231
PPEPATH(), 232
PPL
- Commands, 29
- Developing Applications, 11
- Display Files, 39
- Display Menus, 42
- Introduction, 3
- PCBTEXT Display Prompts, 36
- Reference, 55
- Script Questionnaires, 34
- Structure, 47
- Tutorial, 27

PPLC
- Errors, 20
- Exit Codes, 23
- Installing, 7
- Running, 17
- Using, 17
- Warnings, 19

PRINT, 233
PRINTLN, 233
Process
- CALL, 72

Index

END, 112
RDUSYS, 243
SHELL, 261
STOP, 271
WRUSYS, 349
PROMPTSTR, 234
PSA(), 236
PUSH, 237
PUTUSER, 239

Q

QUEST, 240

R

RANDOM(), 241
RDUNET, 242
RDUSYS, 243
READLINE(), 244
REGAH(), 245
REGAL(), 245
REGAX(), 245
REGBH(), 245
REGBL(), 245
REGBX(), 245
REGCF(), 245
REGCH(), 245
REGCL(), 245
REGCX(), 245
REGDH(), 245
REGDI(), 245
REGDL(), 245
REGDS(), 245
REGDX(), 245
REGES(), 245
REGF(), 245
REGSI(), 245
RENAME, 247
REPLACE(), 248
RESETDISP, 249
RESTSCRN, 250
RETURN, 251
RIGHT(), 252
RTRIM(), 253
Running PPLC, 17

S

S_DB, 277
S_DN, 278
S_DR, 279
S_DW, 280
S2I(), 254
SAVESCRN, 255
Screen
ABORT(), 57
CLOSECAP, 82
CLREOL, 83
CLS, 84
DISPFILE, 104
DISPSTR, 105
DISPTEXT, 106
FCL, 117
FNS, 126
FRESHLINE, 132
MORE, 201
MPRINT, 202
MPRINTLN, 202
NC, 203
NEWLINE, 205
NEWLINES, 206
OPENCAP, 212
OPTEXT, 213
PRINT, 233
PRINTLN, 233
RESETDISP, 249
RESTSCRN, 250
SAVESCRN, 255
SHOWOFF, 262
SHOWON, 263
SHOWSTAT(), 264
SPRINT, 268
SPRINTLN, 268
STARTDISP, 270
WAIT, 343
Script Questionnaires, 13, 34
SCRTEXT(), 256
SEC, 258
SEC(), 259
SENDMODEM, 260
Share Flags
S_DB, 277
S_DN, 278
S_DR, 279
S_DW, 280
SHELL, 261
SHOWOFF, 262
SHOWON, 263
SHOWSTAT(), 264
SLPATH(), 265
SOUND, 266
Source Code
Compiling, 11, 18
Creating, 11
Specifying the File to PPLC, 17
SPACE(), 267
Specifying the Source Code File, 17
SPRINT, 268
SPRINTLN, 268
Stack
POP, 229
PUSH, 237
STACKED, 269
Start, 33
STARTDISP, 270
Statement List, 56
Statements
Code, 48
Variable Declaration, 47
STOP, 271
STRING, 272
String
ASC(), 63
CHR(), 81
I2S(), 155
INSTR(), 166
LEFT(), 176

Index

LEN(), 177
LOWER(), 185
LTRIM(), 186
MID(), 191
REPLACE(), 248
RIGHT(), 252
RTRIM(), 253
S2I(), 254
SCRTEXT(), 256
SPACE(), 267
STRING(), 273
STRIP(), 274
STRIPATX(), 275
TRIM(), 288
UPPER(), 292
STRING(), 273
STRIP(), 274
STRIPATX(), 275
Structure, 47
Sub-Expressions, 50
SysOp Chat
CHAT, 80
PAGEOFF, 218
PAGEON, 219
PAGESTAT(), 220
SYSOPSEC(), 276
System
CDCHKOFF, 77
CDCHKON, 78
GENENV(), 139
HANGUP, 151
KBDCHKOFF, 169
KBDCHKON, 170
MAXNODE(), 188
PSA(), 236
SOUND, 266
VER(), 342

T

TEMPPATH(), 281
Testing Your Application, 14
TIME, 282
Time
ADJTIME, 59
DELAY, 100
HOUR(), 154
MIN(), 192
SEC(), 259
TIME(), 283
TIMEAP(), 284
VALTIME(), 338
TIME(), 283
TIMEAP(), 284
TOKCOUNT(), 285
TOKENIZE, 286
Tokens
GETTOKEN, 140
GETTOKEN(), 141
TOKCOUNT(), 285
TOKENIZE, 286
TOKENSTR(), 287
TOKENSTR(), 287
TRIM(), 288
TRUE, 289
Tutorial, 27
Display Menus, 42
Hello, World!, 27
Interactive Welcome Screens, 39
Logon Language Prompt, 37
Node Specific Display Files, 39
Operator Page, 30
Password Expiration Warning, 36
Script Questionnaire, 34
Start, 33
Type List, 56
Types
BOOLEAN, 69
DATE, 91
INTEGER, 167
MONEY, 199
STRING, 272
TIME, 282

U

U_ADDR(), 293
U_ALIAS, 294
U_BDL(), 295
U_BDLDAY(), 296
U_BDPHONE, 297
U_BUL(), 298
U_CITY, 299
U_CLS, 300
U_CMNT1, 301
U_CMNT2, 302
U_DEF79, 303
U_EXPDATE, 304
U_EXPERT, 305
U_EXPSEC, 306
U_FDL(), 307
U_FSE, 308
U_FSEP, 309
U_FUL(), 310
U_HVPHONE, 311
U_INCONF(), 312
U_LDATE(), 313
U_LDIR(), 314
U_LOGONS(), 315
U_LONGHDR, 316
U_LTIME(), 317
U_MSGRD(), 318
U_MSGWR(), 319
U_NAME(), 320
U_NOTES(), 321
U_PAGELEN, 322
U_PWD, 323
U_PWDEXP, 324
U_PWDHIST(), 325
U_PWDLC(), 326
U_PWDTC(), 327
U_RECNUM(), 328
U_SCROLL, 329
U_SEC, 330
U_STAT(), 331

Index

U_TIMEON(), 333
U_TRANS, 334
U_VER, 335
UN_CITY(), 290
UN_NAME(), 290
UN_OPER(), 290
UN_STAT(), 290
UPCASE, 291
UPPER(), 292
User Information
    CURSEC(), 90
    DELUSER, 102
    GETUSER, 142
    LANGEXT(), 175
    MINLEFT(), 194
    MINON(), 195
    NEWPWD, 207
    PUTUSER, 239
    U_ADDR(), 293
    U_ALIAS, 294
    U_BDL(), 295
    U_BDLDAY(), 296
    U_BDPHONE, 297
    U_BUL(), 298
    U_CITY, 299
    U_CLS, 300
    U_CMNT1, 301
    U_CMNT2, 302
    U_DEF79, 303
    U_EXPDATE, 304
    U_EXPERT, 305
    U_EXPSEC, 306
    U_FDL(), 307
    U_FSE, 308
    U_FSEP, 309
    U_FUL(), 310
    U_HVPHONE, 311
    U_INCONF(), 312
    U_LDATE(), 313
    U_LDIR(), 314
    U_LOGONS(), 315
    U_LONGHDR, 316
    U_LTIME(), 317
    U_MSGRD(), 318
    U_MSGWR(), 319
    U_NAME(), 320
    U_NOTES(), 321
    U_PAGELEN, 322
    U_PWD, 323
    U_PWDEXP, 324
    U_PWDHIST(), 325
    U_PWDLC(), 326
    U_PWDTC(), 327
    U_RECNUM(), 328
    U_SCROLL, 329
    U_SEC, 330
    U_STAT(), 331
    U_TIMEON(), 333
    U_TRANS, 334
    U_VER, 335
Using PPLC, 17

V

VALCC(), 336
VALDATE(), 337
VALTIME(), 338
VARADDR, 339
Variable Declaration Statements, 47
Variable List, 56
VAROFF, 340
VARSEG, 341
VER(), 342

W

WAIT, 343
WAITFOR, 344
Warnings, 19
WHILE, 345
WORDWRAP, 347
WRUNET, 348
WRUSYS, 349

X

XOR(), 350

Y

YEAR(), 351
YESCHAR(), 352
YESNO, 353