OVERVIEW

PASTERP is a PASCAL-like interpreter, with an embedding interface to C++ and Borland PASCAL programs. Now you can enhance your applications by providing a powerful and easy to use extension language to your projects.

The language parsed by the interpreter is a subset of PASCAL, with syntax enhancements that make it easier to read. In most of the constructs the PASTERP language is also more forgiving than PASCAL.

This document describes the language syntax, the supported run-time library, and the interface provided to embed the language in your application projects.

Related Topics :
PASTERP Language
PASTERP Run-Time Library

PASTERP Language

The PASTERP Language is based on PASCAL with some enhancements that simplify parsing (both for the machine and the users), and is more forgiving than PASCAL.

The language support is described in the following sections :

Statements
Varaibles
Expressions

To understand the language capabilities please refer to the Run-Time Library as well.

STATEMENTS

The following statements are recognized in the PASTERP Language :

GLOBAL Statement

The GLOBAL statement is used to define a global variable, that can be accessed from all the procedures of the PASTERP program being executed. It is important to call the GLOBAL statement only ONCE, or the interpreter will not be able to recognize it as a valid variable definition.

It is a good idea to call this statement in an initialization routine that can be modified by your users. Global statements that are defined out of any procedure in the source file are automatically evaluated and placed in the PASTERP symbol table when the source file is loaded.

The GLOBAL statement syntax is as follows :

```
GLOBAL Var-Name : Var-Type [= Initialization-Value] [;]
        [Var-Name ... ]
ENDVAR
```

Where Var-Name is the name of the variable, Var-Type is the type of the variable, and optionally, an Initialization-Value can be specified, using an expression.

Multiple Var-Names can be specified, each one of them will be allowed only if there is no previous variable defined with the same name.

You can also create arrays of variables using the following syntax :

```
varName : ARRAY [low .. high] of VarType
```

Where low and high are the array boundries.

e.g. - myArray : array [1 .. 5] of byte;

An array with 5 elements of byte type was created and assigned to myArray, to access the 4th element in this array we will have to reference myArray[4] .

Please note that since PASTERP is an interpreted language, the optional Initialization-Value can be an expression that references functions, variables etc.. However, if you want to translate your PASTERP sources to PASCAL, you should restrict yourself to constant Initialization-Values only.

Related Topics :

LOCAL Statement
Expressions
Variables

LOCAL Statement

The LOCAL statement is used to define a local variable to the currently executing procedure. This variable can NOT be accessed from any other procedure of the PASTERP program being executed. It is important to call the LOCAL statement only ONCE in the procedure, or the interpreter will not be able to recognize it as a valid variable definition.

It is a good idea to call this statement in an initialization part of your routine, and use it later. The LOCAL statement is closer to the C/C++ variable definition that is performed in the code (and not out of it as in PASCAL). However, unlike C/C++, the variable is not local to a block, but to the entire procedure, from the point of it's declaration.

The LOCAL statement syntax is as follows :

```
LOCAL|VAR Var-Name : Var-Type [= Initialization-Value] [;]
        [Var-Name ... ]
ENDVAR
```

For easier translations from PASCAL to PASTERP, the keyword VAR can be used instead of LOCAL. Notice that if the keyword VAR is used in a procedure/function definition - it is considered to be a local variable, if it is defined out of any procedure/function it is considered to be a global variable and is defined in the PASTERP symbol table when the source file is loaded.

Where Var-Name is the name of the variable, Var-Type is the type of the variable, and optionally, an Initialization-Value can be specified, using an expression.

Multiple Var-Names can be specified, each one of them will be allowed only if there is no previous variable defined with the same name.

Please note that since PASTERP is an interpreted language, the optional Initialization-Value can be an expression that references functions, variables etc. However, if you want to translate your PASTERP sources to PASCAL, you should restrict yourself to constant Initialization-Values only.

Related Topics :
GLOBAL Statement
Expressions
Variables

WRITE Statement

The WRITE statement is used to write a list of expressions. This statement is very close to the PASCAL Write procedure.

The WRITE statement syntax is :

WRITE([File,]Expr-1 [: format-length][[,] Expr-2[: format- length] [[,] Expr-3]])[;]

Where Expr-1, Expr-2 .. are expressions that produce an output. In this version of PASTERP these are Numeric and String Expressions.

The optional File parameter is the name of the file that the output will be directed to. In this version of PASTERP, only TEXT files are supported.

After every expression, a format length parameter can be specified to created columns in the output.

Related Topics :
WRITELN Statement
Expressions

WRITELN Statement

The WRITELN statement is used to write a list of expressions. This statement is very close to the PASCAL Writeln procedure. This statement will write a newline character at the end of the arguments list.

The WRITELN statement syntax is :

WRITELN([File,]Expr-1 [: format-length][[,] Expr-2 [:format- length][[,] Expr-3]])[;]

or

WRITELN[;]

Where Expr-1, Expr-2 .. are expressions that produce an output. In this version of PASTERP these are Numeric and String expressions.

The optional File parameter is the name of the file that the output will be directed to. In this version of PASTERP, only TEXT files are supported.

After every expression, a format length parameter can be specified to created columns in the output.

Related Topics :
WRITE Statement
Expressions

Assignment Statement

The ASSIGNMENT statement assigns a value to a variable. The variable to be assigned is called the LVALUE of the assignment, and the expression that is being evaluated is called the RVALUE of the assignment.

The ASSIGNMENT statement Syntax is as follows :

Variable := Expression

Where Variable is a variable defined before, as a GLOBAL or LOCAL variable, or was defined by the application program that set the Variable.

The Expression is a Numeric/String/Logical expression that is legal for the LVALUE variable it will be assigned to.

Related Topics :
Expressions
Varaibles
PASTERP <-> Borland PASCAL Interface

IF Statement

The IF statement is used to choose code execution according to a set of rules that is correct (evaluated to TRUE) when the IF statement is executed. This statement is semantically equal to the PASCAL IF statement.

The IF statement syntax is :

```
IF (Conditional-Expression) [THEN]
        ... commands to do if conditional-expression is evaluated to TRUE
[ELSE
        ... commands to do if conditional-expression is evaluated to FALSE]
ENDIF
```

Where Conditional-Expression is a logical expression that can be evaluated to a Boolean value.

Unlike in PASCAL, every IF statement must end with the ENDIF keyword. An optional ELSE keyword defines the end of the statement block that should be evaluated when the Conditional-Expression is evaluated to TRUE, and the start of the statement block that should be executed if the Conditional-Expression is evaluated to FALSE.

Related Topics :
Expressions

WHILE Statement

The WHILE statement is used to create loops that are executed during the time a specific condition is true. The condition is re-evaluated at the beginning of the loop, and if the logic evaluation returns TRUE, a block of commands is executed, until a ENDWHILE (or WEND) keyword is reached.

The WHILE statement syntax is :

```
WHILE (Conditional-Expression)
        ... block of statements
ENDWHILE
```

Where Conditional-Expression is a logical expression that can be evaluated to a Boolean value.

Related Topics :

Expressions
REPEAT Statement
FOR Statement

FOR Statement

The FOR statement is used to loop through a block of instructions a fixed number of times.

The Start/End and Step conditions of the loop are evaluated only once,
when the FOR statement starts, this is different from the WHILE and REPEAT statement that are re-evaluated with each iteration.

This statement is close to the standard PASCAL FOR statement.
It adds a STEP parameter that defines how the loop's control variable is incremented/decremented.
Notice that PASTERP Pascal can use REAL (Floating Point) Variables as control variables.

The FOR statement syntax is :

```
FOR Control-Variable := Start-Value TO|DOWNTO End-Value [STEP Step- Value]
        ... Block of statement
ENDFOR
```

Where Control-Variable is the Numeric variable that will be used as a control variable for the loop, Start-Value is the initial value assigned to the control variable, End-Value is the value that the Control-Variable will be tested against. If the optional Step-Value is supplied, this is the value that will be added to the Control- Variable.

The TO and DOWNTO keywords are used for the same purpose, if the STEP parameter is specified, the Step-Value sets the value that will be added to the Control-variable. If the STEP parameter is not supplied, using TO will assign 1.0 to the Step-Value, and DOWNTO will assign - 1.0 to this value.

Related Topics :
Expressions
REPEAT Statement
WHILE Statement

REPEAT Statement

The REPEAT statement is used to create loops that are executed during the time a specific condition is false. The condition is re-evaluated at the end of the loop, and if the logic evaluation returns FALSE, a block of commands is executed.

This statement is different from the WHILE statement, because the command block will be performed at least once, until the first time the conditional-expression is evaluated. In the WHILE statement, the command block might not be executed even once.

The REPEAT statement syntax is :

```
REPEAT
        ... block of statements
UNTIL (Conditional-Expression)
```

Where Conditional-Expression is a logical expression that is re- evaluated at the end of the loop, and the loop is executed while it is evaluated to FALSE.

Related Topics :

Expressions
WHILE Statement
FOR Statement

READ Statement

The READ statement is used to get input from the keyboard, or a file. Unlike the PASCAL READ statement, PASTERP READ statement receives only one argument to read.

The READ statement syntax is :

READ([File,] Variable)[;]

Where Variable is the variable the data will be read into. The optional File parameters is the source file of the input, if no file is specified, the input is received from the keyboard, otherwise, it arrives from the specified file. In this version of PASTERP the only files supported are TEXT files.

Related Topics :

Variables
READLN Statement

READLN Statement

The READLN statement is used to get input from the keyboard, or a file. Unlike the PASCAL READLN statement, PASTERP READLN statement receives only one argument to read.

The READLN statement syntax is :

READLN([File,] Variable)[;]

Where Variable is the variable the data will be read into. The optional File parameters is the source file of the input, if no file is specified, the input is received from the keyboard, otherwise, it arrives from the specified file. In this version of PASTERP the only files supported are TEXT files.

Related Topics :

Variables
READ Statement

Procedure Call Statement

PROCEDURE (and FUNCTIONS) CALL are recognized as statements by the PASTERP language. When a procedure call is recognized, the PASTERP interpreter passes control to the specified procedure/function, and continues execution in that function/procedure. When the called procedure exits, execution is resumed after the call to the procedure/function.

The PROCEDURE/FUNCTION CALL syntax is :

Procedure-Name[(Parameter-1, Parameter-2)]

Where Procedure-Name is the name of the procedure/function, that had been defined either by the calling application, or in the PASTERP code.

The optional Parameters are the parameters defined in the procedure definition.

Related Topics :
Procedure/Function Definition

RETURN Statement

The RETURN statement is used to exit a procedure/function, and according to the function/procedure return type, return a value.

The RETURN statement is close to the C/C++ statement, that has no equivalent in PASCAL.

An alternative to the RETURN statement is to set the function value, by assignment, end exit when the ENDPROC keyword is reached, this method is equivalent to the PASCAL return model.

The RETURN statement syntax is :

RETURN [Expression] [;]

Where Expression is the expression that defines the value the function will return, if the function/procedure does not return a value (return type = void), the expression is not necessary.

An alternate syntax is :

```
FUNCTION myfunc(Parameter-List) : Return-Type
 [... some code]
 myfunc := expression
 [... some code]
ENDPROC
```

Related Topics :

Expressions
Procedure/Function definition

Procedure/Function Definition Statement

There are four (4) types of procedures/functions that PASTERP recognizes, of these two are implemented/registered by the Host Application, and 2 are implemented by the user in PASTERP code.

The procedures/functions that are implemented by the host application programmer are described in the : Extending the PASTERP syntax/system library section.

Dynamic binding functions are described in the :
Dynamic Binding functions/procedures section.

This section describes procedures and functions definition that are defined in PASTERP source. The other types are defined elsewhere in this document.

Procedures or functions that are defined in PASTERP source must be defined on a new source line. You can not start a procedure/function definition on a line that has any previous statement, or even remarks.

The following syntax is used to define procedures and functions :

```
PROCEDURE Proc-Name[(Parameters-List)] [BEGIN]
 .. procedure code
ENDPROC
```

or

```
FUNCTION Func-Name[(Parameter-List)] : Return-Type [BEGIN]
 .. function code
ENDPROC
```

Where Proc-Name/Func-Name is the name of the procedure. Please note that this name must be unique, or a problem might occur.

The optional Parameter-List is a list of parameters that should be passed to the procedure/function, using the following syntax :

Parameter-Name : Parameter-Type [, Parameter-Name : Parameter-Type [..]]

Where Parameter-Name is the name the parameter will be called in the procedure, and Parameter-Type is the type of the parameter.

Return-Type in a FUNCTION definition is the type of the result returned by the function.

Please note that unlike in PASCAL, procedures and functions that are recognized in expressions/statements even if they are declared and defined after the procedure/function call. This can be done, because the interpreter updates the internal procedure table while it loads the source file to be interpreted.

Another important issue to notice, is that PASTERP procedures or functions CANNOT be nested in other procedures/functions. This is more like the C/C++ functions scope rules.

Related Topics :
RETURN Statement
Extending the PASTERP syntax/system library

Dynamic Binding procedures/functions

Dynamic Binding procedures/functions are a way that allow PASTERP code to extend the PASTERP run-time environment and syntax.

Dynamic binding procedures and functions are defined in DLLs, and must be exported by the DLL. (Notice that technically, dynamic binding procedures and functions can also be defined in applications, and used, if the application programmer exported them).

The definition of a dynamic binding procedure or function in PASTERP code is :

EXTERNAL standard procedure/function definition MODULE module- name[;]

Where "standard procedure/function definition" is described in the Procedure/Function Definition Statement topic.

module-name is the name of the module exporting the procedure/function.

Assuming we have a DLL called "MYDLL.DLL", that exports a procedure called Beep that is defined to receive one integer parameter, the PASTERP definition of this procedure will be :

external procedure Beep(i : integer); module "mydll.dll";

This procedure definition can not be nested within other procedures or functions.

The use of this procedure is like any other procedure.

Related Topics :

Procedure/Function Definition Statement

CONTINUE Statement

The CONTINUE statement is used to start a new iteration of a WHILE, REPEAT or FOR statement. The CONTINUE statement is evaluated as a ENDWHILE, UNTIL or ENDFOR keyword is for the relevant statements.

If no loop is defined, CONTINUE will result in an error code.

The CONTINUE syntax is :

CONTINUE

Related Topics :
WHILE Statement
FOR Statement
REPEAT Statement

SWITCH Statement

The SWITCH Statement is used to choose one option from a list options, based on an expression that is evaluated first.

Unlike PASCAL CASE Statement, PASTERP SWITCH Statements can compare expressions to expressions, and not just an expression to a list of constants. Notice that this is possible because PASTERP is an interpreted language. If you want to translate your PASTERP programs to PASCAL, avoid such constructs.

The SWITCH Statement is syntax is :

```
SWITCH expression [OF]
        CASE expr1 : ....
        ENDCASE
        [CASE expr2 : ....
         ENDCASE ...]
        [ELSE ....
         ENDCASE]
ENDCASE
```

Where expression is the expression that expr1, expr2 .. will be tested against. When a match between exprN to expression is found, the statements specified until the ENDCASE keyword, are executed, and execution continues after the ENDSWITCH keyword.

If no expression from expr1 .. exprN was matched to expression, and an ELSE option is specified, the statements between the ELSE and the ENDCASE keywords will be executed.

Notice that in PASTERP SWITCH Statements can not be nested.

VARIABLES

PASTERP variables must be declared before they can be used. Variables can be declared either in the PASTERP source code, or in the host application.

PASTERP variables are either GLOBAL, where every procedure can access them (they have a global scope), or LOCAL to the procedure that executes them.

GLOBAL variables can be defined either from the PASTERP source, or the host application code, LOCAL variables can be defined only in the PASTERP source code, or as parameters to procedures that can be defined by the host application that registers the procedure/function.

This version of PASTERP supports only the built-in variables types. New types can not be created. Arrays and records are not supported in this version of PASTERP.

The supported variable types are :

BYTE	- Equal to PASCAL BYTE, a 0-255 integer type.
INTEGER	- Equal to PASCAL INTEGER, a -32K .. + 32K integer type.
WORD	- Equal to PASCAL WORD, a 0 .. 64K integer type.
LONGINT	- Equal to PASCAL LONGINT, a -2 Billion .. + 2 Billion integer type.
REAL	- Equal to PASCAL REAL, a 2.9*10-39 .. 1.7*10.38 float.
STRING	- Equal to PASCAL STRING, a 255 Character string.
CHAR	- Equal to PASCAL CHAR, one character.
PCHAR	- Equal to PASCAL PCHAR, an AsciiZ pointer.
BOOLEAN	- Equal to PASCAL BOOLEAN, a TRUE/FALSE logical variable.
BOOL	- Equal to PASCAL BOOL, A TRUE/FALSE word size logical
TEXT	- Equal to PASCAL TEXT, a text mode file.
POINTER	- Equal to PASCAL POINTER type, a generic pointer.

Please note that while PASTERP does not support most of the other PASCAL types, the keywords for all the standard PASCAL types are reserved by PASTERP for a future release that might support them.

EXPRESSIONS

PASTERP supports expressions that are either Numeric, String or Logical expressions. These expressions are evaluated by the interpreter according to the type of function return, parameter or variable assignment.

In PASTERP all Numeric expressions are evaluated as REAL expressions, and data is converted back and forth if needed between REALs and the Variable/ Parameter used.

All PASTERP String expressions are evaluated as AsciiZ expressions, and data is converted back and forth if needed between AsciiZ and STRING variables/parameters.

The Expressions Definitions are :

Numeric Expressions

In PASTERP all Numeric expressions are evaluated as REAL expressions, and data is converted back and forth if needed between REALs and the Variable/ Parameter used.

The PASTERP Numeric Expressions will be described in a simple structure :

A Numeric Expression supports the standard math operations (+, -, *, /, %) it also supports the POWER operator, parenthesis, and unary minus.

The Primitive elements of a numeric expression are numeric constants,
variables of a numeric type, and functions that return a numeric value.

The operators in decreasing evaluation order are :

Primitives
Parenthesis
Unary Minus
POWER
Mul (*), Div (/), Mod (%)
Add (+), Sub (-)

Operators on the same line are left associative.

Related Topics :

Variables
String Expressions
Logical (Boolean) Expressions

String Expressions

All PASTERP String expressions are evaluated as AsciiZ expressions, and data is converted back and forth if needed between AsciiZ and STRING variables/parameters.

String expressions support string concatenation using the + operator.

The Primitive elements of a string expression are string constants,
variables of a string type, and functions that return a string value.

Please note that you can concatenate AsciiZ (PCHAR) and STRING type strings.

String Constants are delimited either by single or double quotes, the
matching quote is determined by the first quote, this way it is easy to create strings that include the "other" quote character. Like PASCAL, PASTERP Strings can also include the quote character by doubling it.

e.g. - "This string has a single quote right here : ' "

e.g. - 'And this one has a double quote here : " '

Related Topics :

Variables
Numeric Expressions
Logical (Boolean) Expressions

Logical (Boolean) Expressions

PASTERP logical expressions return a Boolean value - TRUE or FALSE.

The supported logical operators are AND, OR, XOR, NOT and parenthesis.

The Primitive Boolean values are TRUE, FALSE, variables of a Boolean type, and functions that return a Boolean type.

The operators in decreasing evaluation order are :

Primitives
Parenthesis
NOT
AND
XOR
OR

Related Topics :

Variables
Numeric Expressions
String Expressions

PASTERP Library

The PASTERP Standard Library is based on the Standard PASCAL library, with some modifications needed to support the extended PASTERP features, and some procedures/functions missing because PASTERP does not support all the PASCAL features.

An extended library will be supplied in a future version, and will support functions that are more related to the PC environment.

The library support is described in the following sections :

PASTERP Standard Library

The PASTERP Standard Library is based on the Standard PASCAL library, with some modifications needed to support the extended PASTERP features, and some procedures/functions missing because PASTERP does not support all the PASCAL features.

The following functions and procedures are defined in the standard library :

Related Topics :

Standard Library Function : PI

function pi : real;

The PI function returns the PI value.

Standard Library Function : EXP

function exp(r : real) : real;

Returns the exponent of (r).

Related Topics

Standard Library Function : LN
Standard Library Function : LOG10
Standard Library Function : LOG2

Standard Library Function : SIN

function sin(r : real) : real;

Returns the Sin of (r).

Related Topics

Standard Library Function : COS
Standard Library Function : TAN
Standard Library Function : COTAN
Standard Library Function : ARCSIN

Standard Library Function : RANDOM

function random(l : longint) : longint;

Returns a LONGINT in the range 0 .. l .

Standard Library Function : COS

function cos(r : real) : real;

Returns the Cos of (r).

Related Topics

Standard Library Function : SIN
Standard Library Function : TAN
Standard Library Function : COTAN
Standard Library Function : ARCCOS

Standard Library Function : LN

function ln(r : real) : real;

Return the Ln of (r).

Related Topics

Standard Library Function : EXP
Standard Library Function : LOG10
Standard Library Function : LOG2

Standard Library Function : LOG10

function log10(r : real) : real;

Returns the log (base 10) of (r).

Related Topics

Standard Library Function : EXP
Standard Library Function : LN
Standard Library Function : LOG2

Standard Library Function : LOG2

function log2(r : real) : real;

Returns the log (base 2) of (r).

Related Topics

Standard Library Function : EXP
Standard Library Function : LN
Standard Library Function : LOG10

Standard Library Function : ABS

function abs(r : real) : real;

Returns the absolute value of (r).

Standard Library Function : ARCTAN

function arctan(r : real) : real;

Returns the Arctan of (r).

Related Topics

Standard Library Function : TAN
Standard Library Function : COTAN

Standard Library Function : SQR

function sqr(r : real) : real;

Returns the square of (r).

Related Topics

Standard Library Function : SQRT

Standard Library Function : SQRT

function sqrt(r : real) : real;

Returns the square root of (r).

Related Topics

Standard Library Function : SQR

Standard Library Function : TAN

function tan(r : real) : real;

Returns the Tan of (r).

Related Topics

Standard Library Function : SIN
Standard Library Function : COS
Standard Library Function : ARCTAN
Standard Library Function : COTAN

Standard Library Function : COTAN

function cotan(r : real) : real;

Returns the COTAN of (r).

Related Topics

Standard Library Function : SIN
Standard Library Function : COS
Standard Library Function : TAN

Standard Library Function : ARCSIN

function arcsin(r : real) : real;

Returns the Arcsin of (r).

Related Topics

Standard Library Function : SIN
Standard Library Function : ARCCOS

Standard Library Function : ARCCOS

function arccos(r : real) : real;

Returns the Arccos of (r).

Related Topics

Standard Library Function : COS
Standard Library Function : ARCSIN

Standard Library Function : CHR

function chr(b : byte) : char;

Returns the CHAR representation of (b).

Related Topics

Standard Library Function : ORD

Standard Library Function : ORD

function ord(c : char) : byte;

Returns the ordinal number (representation) of (c).

Related Topics

Standard Library Function : CHR

Standard Library Function : TRUNC

function trunc(r : real) : longint;

Returns (r), truncated.

Related Topics

Standard Library Function : ROUND

Standard Library Function : ROUND

function round(r : real) : longint;

Returns (r), rounded.

Related Topics

Standard Library Function : TRUNC

Standard Library Function : COPY

function copy(s : string, i : byte, l : byte) : string;

Returns the substring of (s), that start and index (i), for (l) bytes.

Related Topics

Standard Library Function : LENGTH
Standard Library Function : INSERT
Standard Library Function : DELETE
Standard Library Function : POS

Standard Library Function : LENGTH

function length(s : string) : byte;

Returns the length of (s).

Related Topics

Standard Library Function : COPY
Standard Library Function : INSERT
Standard Library Function : DELETE
Standard Library Function : POS

Standard Library Function : INSERT

function insert(s : string, var d : string, i : index);

Inserts (s) into (d), after position (i).

Related Topics

Standard Library Function : COPY
Standard Library Function : LENGTH
Standard Library Function : DELETE
Standard Library Function : POS

Standard Library Function : DELETE

procedure delete(var s : string, i : byte, c : byte) : char;

Delete (c) bytes from position (i) of (s).

Related Topics

Standard Library Function : COPY
Standard Library Function : LENGTH
Standard Library Function : INSERT
Standard Library Function : POS

Standard Library Function : POS

function pos(s : string, d : string) : integer;

Returns the position of (d) in (s), 0 if not found.

Related Topics

Standard Library Function : COPY
Standard Library Function : LENGTH
Standard Library Function : INSERT
Standard Library Function : DELETE

Standard Library Function : VAL

function val(s : string, var r : real) : integer;

Returns the value of (s), in (r). If the function returns 0, the conversion was successful, otherwise it points to the index in (s), where the conversion failed.

Related Topics

Standard Library Function : STR

Standard Library Function : STR

procedure str(r : real, var s : string);

Returns the string representation of (r) in (s).

By default the STR procedure creates a scientific representation of the value in s, if you want to use standard representation the following syntax is provided :

str(r [: numOfDigits [: numOfDecimalPlaces]], s)

Where numOfDigits is the number of digits to display, and numOfDecimalPlaces is the number of decimal places to present.

0 is used to indicate that the size is the number of digits in the number with no padding or truncation.

Examples

```
str(2, s);
```

s will include the scientific representation 2.00000000E+00

```
str(2 : 0 : 0, s);
```

s will inclde the standard representation 2

```
str(2 : 0 : 2, s);
```

s will include the value 2.00

Related Topics

Standard Library Function : VAL

Standard Library Function : ASSIGN

procedure assign(t : text, s : string);

Associates the text file (t), with the file name specified in (s).

Please note that PASTERP supports automatic assignment of a file name to a text variable during the variables definition.

The following two code fragments are equivalent :

Figure A :

```
var
        t : text;
endvar
        assign(t, "myfile.txt");
```

Figure B :

```
var
        t : text = "myfile.txt";
endvar
```

Related Topics

Standard Library Function : RESET
Standard Library Function : CLOSE
Standard Library Function : APPEND
Standard Library Function : REWRITE
Standard Library Function : EOF

Standard Library Function : RESET

function reset(t : text) : byte;

Resets (t) for input, and returns an error code. If the function returns 0, the reset operation was successful

Related Topics

Standard Library Function : ASSIGN
Standard Library Function : CLOSE
Standard Library Function : APPEND
Standard Library Function : REWRITE
Standard Library Function : EOF

Standard Library Function : CLOSE

function close(t : text) : byte;

Closes the text file (t), and returns an error code. If the function returns 0, no error occurred.

Related Topics

Standard Library Function : ASSIGN
Standard Library Function : RESET
Standard Library Function : APPEND
Standard Library Function : REWRITE
Standard Library Function : EOF

Standard Library Function : APPEND

function append(t : text) : byte;

Opens (t) for output, from the end of the file. Returns an error code. If the function returns 0, no error occurred.

Related Topics

Standard Library Function : ASSIGN
Standard Library Function : RESET
Standard Library Function : CLOSE
Standard Library Function : REWRITE
Standard Library Function : EOF

Standard Library Function : REWRITE

function rewrite(t : text) : byte;

Opens (t) for output, rewriting over any previous file with the same name.
The function returns an error code, or 0 if no error occurred.

Related Topics

Standard Library Function : ASSIGN
Standard Library Function : RESET
Standard Library Function : CLOSE
Standard Library Function : APPEND
Standard Library Function : EOF

Standard Library Function : EOF

function eof(t : text) : Boolean;

Returns TRUE if the file pointer of (t) is at the end of file.

Related Topics

Standard Library Function : ASSIGN
Standard Library Function : RESET
Standard Library Function : CLOSE
Standard Library Function : APPEND
Standard Library Function : REWRITE

Standard Library Function : INC

procedure inc(var v[, by : real]);

Increments the variable (v) that must be of a numeric type. If the optional (by) parameter is specified, (v) is incremented using (by). Otherwise, (by) is assumed to be 1.

Related Topics

Standard Library Function : DEC

Standard Library Function : DEC

procedure dec(var v[, by : real]);

Decrement the variable (v) (must be of a numeric type). If the optional (by) parameter is specified, (v) is decremented using (by). Otherwise, (by) is assumed to be 1.

Related Topics

Standard Library Function : INC

Standard Library Function : GETMEM

,groups stdlib
.list stdlib

function getmem(size : longint) : pointer;

Allocate size bytes from the heap, and returns a pointer to this block.

Related Topics

Standard Library Function : FREEMEM

Standard Library Function : FREEMEM

procedure freemem(p : pointer; size : longint);

De-Allocates a block of size bytes, pointed by the pointer (p). Notice that if (p) does not point to a valid memory block, a memory corruption may occur.

Related Topics

Standard Library Function : GETMEM

PASTERP Extended Library

The Extended library provides procedures and functions that are specific to the PC environment.

The following functions and procedures are defined in the extended library :
function chdir
function getDir
function mkDir
function rmDir

procedre getDate
procedure setDate

procedure getTime
procdeure setTime

function gregToJul
procedure julToGreg

procedure forEachFile

Related Topics :
Standard Library

function chdir

function chdir(s : string) : byte;

The chdir function changes the current directory to the directory specified in the s parameter.

Parameters

s - This parameter contains directory that will be the current directory after the function has been performed.

Return

The function returns 0 on success or an error code otherwise.

function getDir

function getDir(d : byte) : string

The getDir function returns the current directory of the drive specified in the parameter d.

if d is 0, the drive is the default drive, otherwise d is the drive using the table :

d = 1 : Drive = A
d = 2 : Drive = B
d = 3 : Drive = C

... etc ...

function mkDir

function mkDir(s : string) : byte;

The mkDir function is used to create a new directory, using the parameter s to specify the new directory path.

The function returns 0 for success, or an error code for failure.

function rmDir

function rmDir(s : string) : byte;

The rmDir function is used to remove the directory specified in the s parameter.

The function returns 0 for success, or an error code for failure.

procedre getDate

procedure getDate(var year : word, var month : word, var day : word, var dayOfWeek : word);

The getDate procedure gets the current date from the operating system and sets the parameters passed to it.

The variables day, month and year will will be set to the Gregorean date components, and the dayOfWeek parameter will contain the day of the week.

procedure setDate

procedure setDate(year : word, month : word, day : word);

The setDate procedure sets the current date in the operating system to the date specified in the 3 parameters provided.

procedure getTime

procedure getTime(var hour : word, var minute : word, var second : word, var sec100 : word);

The getTime procedure gets the current time from the operating system into the parameters provided.

procdeure setTime

procedure setTime(hour : word, minute : word, second : word, sec100 : word);

The setTime procedure sets the current time in the operating system using the paramters provided.

function gregToJul

function gregToJul(month : longint, day : longint, year : longint) : longint;

The gregToJul function returns a Julian date from the Gregorian date provided in the parameters.

procedure julToGreg

procedure julToGreg(jul : longint, var month : longint, var day : longint, var year : longint);

The julToGreg procedure converts a Julian date to the day, month and year components of a Gregorian date.

procedure forEachFile

procedure forEachFile(mask : string, fileFuncName : string, recurse : boolean);

The forEachFile procedure allows you to automatically traverse all the files that match a valid dos mask in the current directory (and optionaly in all the sub-directories) and perform a common operation on each one of these files.

Parameters

mask - This is the dos mask that will be matched. e.g. - '*.pas' will match all the files with the extension pas.

fileFuncName - The name of the function that will be called for every file that matches the mask.

recurse - TRUE will recures into the sub-directories of the current directory, FALSE will not.

The fileFuncName parameter points to a PASTERP function that must have the following declaration :

```
function name(fName : string, attr : byte, time : longint, size : longint) : boolean;
```

This function receives the file name in the fName parameter, the attributes of the file in the attr parameter, the DOS time stamp in the time parameter and the file size in the size parameter.

The function should return TRUE to contine the file retrieval for the next file, or FALSE to abort the forEachFile process.

Take a look at the supplied ETEST7.TRP file for an example of using the forEachFile procedure and a file handling function.