Personal Designer

User Programming Language (UPL)

Revision 6.0

User Reference Guide

Chapter 4

Statements and Intrinsics

Statements and Intrinsics

Abs

Type

Intrinsic Function Arithmetic

Purpose

Converts an integer, integer4 or real expression to its absolute value. This function returns an integer, integer4 or real value, depending on the input.

Syntax

Abs*(expr)*

Parameters

expr: Integer, integer4 or real expression (input)
Specifies the integer or real expression to convert.

Accept

Type

Statement Input/Output (Window)

Purpose

Allows the user to input numerical, coordinate, or string data and assign it to a given variable. You can qualify the data by selecting keywords and expressions.

Syntax

Accept *var dataqual(expr,...),...*

Keyword modifiers

var: An integer, integer4, real, coordinate, or string variable. The input data is returned in the *var* variable. The data type declared for *var* determines the type of data the program can accept. When the program is run, the data returned in var will be echoed in different ways, depending on the data type of *var*. When *var* is integer, integer4, real, or string, the input data is echoed to the ACCEPT window, which is the command window by default. Use the **AccptWin** systern variable to change the ACCEPT window. See **DefineAW** and Appendix B, "System Variabies," for more information.
When the ENTANY and ENTLIST keywords are selected, the user must digitize an entity in the graphics window. This is echoed by highlighting the entity. In this case, *var* is declared as an integer4, which is an exception to the rule noted above.
When *var* is declared as a coordinate variable, the user must digitize a location in the graphics window. This is echoed by a digitizing mark, which is the small x in the window. The user cannot enter coordinates through the keyboard, tablet, or on-screen menus-see **GetDig** for more information. The value returned in *var* is relative to the currently selected view rather than model space.

The ASCII value of the last character input into an **Accept** statement is automatically placed in the system variable **LastChar.** For more information about system variables, refer to Appendix B, "System Variables."

dataqual: Optional keyword. These keywords allow you to qualify the data to be accepted. The keywords must be separated by commas or spaces, and can be given in any order. Refer to the tabie below to see which keywords can be used with the different variable types.

Box*(cexpr1, cexpr2)* Coordinate
Limits acceptable digitized points to the inside of the box specified by *cexpr1* and *cexpr2*. These are the lower left and upper right corners of the box. Coordinate values are interpreted as the currently selected view space coordinates.

Entany Integer4
Allows selection of entities by digitizing in the graphics window. **Entany** returns an MIB number for the entity you digitize. This MIB number can then be used in the database statements and the database access intrinsic procedures and functions. See Chapter 3, Functional Listing of Statements and Intrinsics for more information.

Entlist*(i4expr1, i4array(iexpr2))* Integer4
Allows selection of specific entities by digitizing in the graphics window. Only the entities with MIB numbers specified in the *i4array* parameter may be selected; *i4array* must be declared as an integer array with at least *i4expr1* elements. Replace *iexpr2* with the first element that is to be checked in *i4array*. When the program is run. if any entity is picked which is not in the list *i4array*, the system sounds a beep and then waits for the user to select the next entity. Otherwise, the MIB number of the selected entity is returned. This MIB number can then be used in the database statements and the database access intrinsic procedures and functions. See Chapter 3, Functional Listing of Statements and Intrinsics.

Exact*(sexpr)* String
Accepts the exact sequence of characters given in the string expression, up to 250 characters. When you use this keyword, no other keywords can be given.

You can use two consecutive **Exact** keywords. This allows you to accept an answer choice such as "yes" and "no." It also allows you to specify the same *sexpr* twice-in upper and lower case-so that the program can be run with the CAPS lock key on or off. If the user types a space, the next character in the string expression is automatically accepted; this feature is useful for writing tutorials. If the user types a character that is not in the expression, the system sounds a beep and then waits for the next character to be entered.

In*(sexpr)* String
Each character entered must be one of the characters specified in a string expression up to 100 characters. If the user types a character that is not in the expression, the system sounds a beep and then waits for the next character to be entered.

Last*(sexpr)* Any Type
Speeifies the character(s) that will end data input. Input is terrninated when the user enters any of the charaeters listed in *sexpr*. Note that the terminating character is not put in *var*; rather the system variable **LastChar** is assigned the ASCII value of the last character entered. The *sexpr* string may contain a maximum of 100 characters.

Macro*(iexpr)* Integer
Designates the keyboard macro set to be used by the **Accept** statement. Keyboard macro sets define key assignments for the keyboard. If the *iexpr* macro set is not found, the program will use macro set one. Refer to the PDMAC.DEF file in the Personal Designer directory for a description of keyboard macro sets.

Max*(sexpr)* String
Accepts only characters whose ASCII value is less than or equal to the first character specified in the string expression. The string expression may contain a maximum of 100 characters.

Min*(sexpr)* String
Accepts only characters whose ASCII value is greater than or equal to the first character specified in the string expression. The string expression may contain a maximum of 100 characters.

Newline Any Type
Starts a new line in the Accept window after data is input. Data input after this statement will be echoed on this new line. All prompts output after this statement will also be output on the new line.

Note that the **Print, Display, and Send** windows default to the command window. The output from these windows will also appear on the new line if their system variables have not been changed.

Prompt*(sexpr)* Any Type
Before data is accepted, the program prints *sexpr* in the **Accept** window to prompt the user for input. The string may contain a maximum of 500 characters.

Size*(sexpr)* String
Limits the number of characters the user can enter.

Examples

```
ACCEPT S1 SIZE(1), IN(“ABCDEPQ“)

ACCEPT X PROMPT("Input X dimension") NEWLINE

ACCEPT S1 SIZE(1), MIN("A"), MAX("Z")\
     Prompt ('Enter a letter of the alphabet')
```

```
------------------------------------------------
-- The following program a displalys a menu and
-- then prompts the user to enter a one-character
-- menu choice. only one of the letters in the
-- menu is accepted. The program will not continue
-- until one of them is entered.

proc main
    string S1:1
    real X
    integer Num
    display
    Menu:
     -------
    (A) Open Part
    (B) Draw B size border
    (C) Draw C size border
    (D) Draw D size border
    (E) Draw E size border
    (P) Print text file
    (Q) Quit
$
    accept S1 in('ABCEDPQ') prompt('Choice: ´)\
                                          newline
    --  code to execute menu choices goes here
end proc
------------------------------------------------
------------------------------------------------
--  This program demonstrates an ACCEPT statement
--  using error checking on input.

proc main
    integer num
    loop
        accept num prompt('Enter number (1-10): ´)
        exit when (num => 1) and (num <= 10)
        print
        print 'Number out of range. Try again.'
    end loop
end proc
------------------------------------------------
```

AccessCode

Type

Intrinsic Function Operating System

Purpose

Returns a unique access code when given a key and a serial number. **AccessCode** provides a method for UPL program developers to protect their programs from unauthorized use. This function returns a 32-character string which contains only uppercase letters.

Syntax

AccessCode*(keystr, serialstr)*

Parameters

keystr: *String* expression of 32 characters (input)
This parameter specifies the key used to make the access code. Only the first 32 characters in the string will be used. The string can be shorter, however, it is recommended that it be at least 10 characters in length. The *ke.ystr parameter* can contain any characters between the ASCII values 32 (space), and 126 (~).

serialstr: String expression (input)
This specifies the six-digit serial number. On DOS systems it is the Computervision guard box number. On UNIX workstations, it may be the *hostid or* a portion of the Ethemet address. The serial number can be retrieved using the **GetSerialNum** intrinsic function.

AccessCode protects programs in the following way:

1 . The vendor of the UPL program creates the keystring, which is known only to him.
2. When the UPL program is installed, the user provides the vendor with the number of his Computervision guard box.
3. The vendor creates a program that uses this routine, the box number, and the keystring to generate the user code. He returns this code to the user who puts it in a operating system (DOS or UNIX) environment
variable or the CVOPTION.FIL file.

4. The program should contain a series of statements which test the user code. If the test fails, the program should give an error message and abort. See the example below.

Examples

```
------------------------------------------------
-- This program demonstrates how to use the
-- AccessCode routine to protect your software.
-- It contains two functions. The first one uses
-- an environment variable to hold the User Code.
-- The second one uses the file CVOPTION.FIL.
-- Either method may be used.

func CheckCodesEnvVar return Boolean

string KeyStr:32
string UserCode:32

KeyStr = 'This is my keystring'
EnvVar('USERCODE', UserCode)

if UserCode <> AccessCode( KeyStr,\
                           GetSerialNum() ) then
    print 'invalid guard box or access code'
    return False
else
    return True
end if

end func

func CheckCodesCVOpt return Boolean
    string KeyStr:32
    string UserCode:32
    string AccCode:32
    file OptFile
```

```
    KeyStr = 'This is my keystring'
    AccCode = AccessCode( KeyStr, GetSerialNum())
    open OptFile, '\cvoption.fil'
    loop
        read OptFile, UserCode
        exit when OptFile.EOF
        if UserCode = AccCode then
            return True
        end if
    end loop

    print 'invalid guard box or access code'
    return False
end func

proc main
    If Not CheckCodesEnvVar() then
        return
    endif
    If Not CheckCodesCVOpt() then
        return
    endif
end proc
-------------------------------------------------
-- This program demonstrates how to generate a
-- User Code to put in an enviro=ent variable
-- or the file, CVOPTION.FIL

proc main
    string SerialNumber:6
    string KeyString:32
    accept SerialNumber last('#13##3#') size(6)\
             prompt('Enter the box/host id number: ') \
             newline
    return when LastChar = 3
    accept KeyString last('#13##3#') size(32) \
             prompt('Enter your key string: ')
             newline
    return when LastChar = 3
    print 'The User Code for box or host id ',
    print SerialNumber,': '
    print AccessCode( KeyString, SerialNumber)
end proc
-------------------------------------------------
```

ACos

Type

Intrinsic Function Trigonometric

Purpose

Returns the arccosine of a real expression. This function returns a real value in radians.

Syntax

ACos*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the real expression whose arccosine is returned.

AddFnExt

Type

Intrinsic Procedure Operating System

Purpose

Adds an extension to a file name.

Syntax

AddFnExt*(filename, ext, iopt)*

Parameters

filename: String variable (input/output)
On input, this parameter specifies a file name. On output, the new file name is returned in this variable.

ext: String expression of 3 characters (input/output)
This parameter specifies the file name extension that is added to *filename*. Only the first three characters are used.

NOTE: Do not use a period in the extension.

iopt: Integer expression (input)
The *iopt* parameter specifies the conditions under which *ext* is added to the *filename* parameter.

Values are:

0 Tells the program to add *ext filename* only if it does not already have an extension.

1 Tells the system to replace the existing extension with the one specified in the *ext* parameter.

Example

```
AddFnExt(FN, 'DAT', 1)
```

Ang3P

Type

Intrinsic Function Geometric

Purpose

Returns the angle between three points in three-dimensional space. This is the smaller of two angles formed by the two imaginary lines which connect the origin with the other two points. It will always be less than pi radians. The returned real value is in radians.

Syntax

Ang3P*(pnt1, pnt2, pnt3)*

Parameters

Pnt1: Coordinate expression (input)
This parameter specifies the endpoint of the first imaginary line.

pnt2: Coordinate expression (input)
This parameter specifies the vertex point of the two imaginary lines.

pnt3: Coordinate expression (input)
This specifies the endpoint of the second imaginary line.

Example

```
A1 = Ang3P(C1, ORG1, C2)
```

ASCII

Type

Intrinsic Function Data Conversion

Purpose

Returns the ASCII value of the first character in a string. This function returns an integer.

Syntax

ASCII*(sexpr)*

Parameters

sexpr: String expression (input)
This parameter specifies the string whose ASCII value is returned.

ASin

Type

Intrinsic Function Trigonometric

Purpose

Returns the arcsine of a real expression. This function returns a real value in radians.

Syntax

ASin*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the real expression whose arcsine is returned.

AskModifiers

Type

Intrinsic Procedure User Interface

Purpose

Allows your UPL program to accept input in the modifier format used by Personal Designer commands. Transfers controi of the UPL program to the Personal Designer modifier processor which accepts modifiers and values from the user. To exit the modifier processor, the user must type a colon, carriage return, or control C. This character will then be stored in the **LastChar** system variable. See the LastChar system variable, Appendix B, "System Variables." When using **AskModifiers,** the modifier processor behaves the same way as it would with a Personal Designer command. For example, the modifier processor will only allow legal modifiers and context sensitive help is available. See Appendix H, "Writing Personal Designer Commands," and **DefineModifier and GetModifier** for more information.

Syntax

AskModifiers*(modset)*

Parameters

modset: *Integer* expression (input)
This parameter specifies the modifier index number of the modifier set to use. Set *modset* to zero to use the modifiers you created with the **DefineModifier** procedure. Any other number you input will use Personal Designer's modifier table. See Appendix H, "Writing Personal Designer Commands," for more information.

Example

```
AskModifiers(0)
```

AsmInt

Type

Intrinsic procedure Operating System

Purpose

Performs a DOS software interrupt. Allows UPL programs to interface with assembly language programs or the DOS operating system.

Syntax

AsmInt*(IntNo,RegData())*

Parameters

IntNo: Integer expression (input)
Intel interrupt number to execute (0 to 255)

RegData Integer array of nine elements
RegData(1) = AX register
RegData(2) = BX register
RegData(3) = CX register
RegData(4) = DX register
RegData(5) = SI register
RegData(6) = DI register
RegData(7) = DS register
RegData(8) = ES register
RegData(9) = CPU flags

Examples

```
Regs(1) = 12288          --AH = 30h, AL = 00h

AsmInt(33, Regs(1))      --DOS int21h, get DOS version
                           number
```

Assignment

Type

Statement Assignment

Purpose

Gives a value to a variable, variable attribute, or array element.

Syntax

var = *expr*

where:

var: Name of the variable to assign a value to. Variables of file type may not be assigned; however, file attribute variables may be assigned.

expr: Expression of the same data type as *var*.

NOTE: *expr* must be the same data type as *var*. Use the data type conversion intrinsic functions to convert *expr* to the correct data type.

Example

```
-------------------------------------------

-- This program demonstrates some of the many
-- different forms an assignment may take.

PROC MAIN
INTEGER I(10), J, PTR(20)
REAL R, X, Y, Z, RR(15), THETA
STRING S1:80, S2:80
BOOLEAN B1, B2
COORD C1, C2, C
FILE F1

B1 = TRUE
S1 = "BEGIN" + S2 + "END"
R = X**2.0+5.0*Y/(4.0+Z)
I(2)=10
B2=FALSE
B1=NOT B1 OR B2
C.X=2.69
C=[4.0,7.93,-4.68]-C2
C=C1+C2
C.Z=-10.2
RR(J/3+1)=SIN(DEG_RAD(THETA))/4.5
X=C.Y/Z**2.0
F1.POSITION=PTR(J)
F1.POSITION=F1.POSITION+10
R=REAL(J)+2.5+REAL(5)
END PROC
```

ATan

Type

Intrinsic Function Trigonometric

Purpose

Returns the arctangent of a real expression. This function returns a real value in radians.

Syntax

ATan*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the real expression whose arctangent is returned.

ATan2

Type

Intrinsic Function Trigonometric

Purpose

Returns the arctangent of the r11r2 (or sin/cos) parameters. This function will produce valid results even if r2 is zero. It returns a real value in radians.

Syntax

ATan2*(r1, r2)*

Parameters

r1: Real expression (input)
This parameter specifies the sine of the angle.

r2: Real expression (input)
This parameter specifies the cosine of the angle.

AWinClear

Type

Intrinsic Procedure Input/Output (Window)

Purpose

Clears the specified alphanumeric window with the currently defined background color for that window number. The cursor is reset to the upper left corner of the window. See DefineAW for more information.

Syntax

AWinClear*(iwin)*

Parameters

iwin: Integer expression (input)
Specifies the alpha window number to be cleared. Values are 1 through 20.

Example

```
-- The following example clears alphanumeric window
-- 11,the message window. Use the CLEAR statement
-- or GWinClear to clear the graphics window.

AWinClear(11)
```

BigMibList

Type

Intrinsic Function User Interface

Purpose

Returns the integer4 Master Index Block (MIB) number of a digitized entity. **BigMibList** must be used with the GetEnt procedure. **BigMibList** extends the capability of GetEnt. GetEnt returns a speeific number of digitized entities, whereas **BigMibList** allows GetEnt to return an unlimited number of entities. A call to **BigMibList** is equivalent to accessing the *miblist* parameter in the **GetEnt** procedure.

Before you use **BigMibList,** make a call to GetEnt with parameters set as described below:

maxmib:	*-1*
miblist:	a dummy (one element) integer4 array
nent:	an integer4 variable
iend:	an integer variable

If you want to insert a Send statement between calls to **BigMibList** and **GetEnt,** or between successive calls to **BigMibList,** make sure the **Send** statement does not invoke Personal Designer commands that select entities.

Syntax

BigMibList*(ient)*

Parameters

ient: *Integer4* expression (input)
This parameter specifies a position in the internal entity list. This position holds the MIB number of the next entity selected by the GetEnt procedure.

The value of *ient* should not be greater than the *nent* value returned in the previous call to GetEnt. If *ient is* greater than nent, the returned value will be invalid.

Example

```
-----------------------------------------------

-- The following example shows how BigMibList is
-- used with GetEnt. See the GetEnt example which
-- prompts the user for entities without using
-- BigMibList. Note that the BigMibList example can
-- access an unlimited number of entities,
-- whereas GetEnt does not:

proc main

integer4 NEnt, DummyMIBList(1), Mib, I
integer Iend, Ierr

GetEnt(-1, NEnt, DummyMIBList(1), Iend)

print ´you digged the following entities:´,
Loop I = 1 to NEnt
        Mib = BigMibList(I)
        print Mib
End Loop

RpntEnt(DummyMIBList(1),0,Ierr)

end proc
```

Boolean

Type

Statement Declaration

Purpose

Declares the name, data type, aggregate type, and initial value of a Boolean variable. An initial value is optional. Array variables must be declared with their maximum subscripts.
Variables of Boolean data type contain logical values of either true or false.

Syntax

Boolean *bvarname = bconst 1 bvarname(iconst,...)....*

Keyword modifiers

bvarname: Name of the Boolean variable. Only the first 16 characters are used.

bconst: Optional initial value for a Boolean scalar variable. This value must be a literal or named Boolean constant. If the variable is declared in the Group section, it will be set to this value once at the beginning of the program. If the variable is declared in a procedure or function, it will be set to this value each time the procedure or function is called.

iconst: Array subscripts. These declare the variable to have aggregate type array. Up to five subscripts may be declared. The subseripts must be enclosed in parentheses. Array variables may not be given an initial value.

All declaration statements must occur after the **Proc, Func, and Group** statements. They must appear before any other type of statements inside a procedure or function and are the only statements allowed inside the Group section. For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Format."

Examples

```
Boolean Done = False, BoolArr(10)
Boolean ErrorOccured, TimeOut
Boolean WaitFor = TRUE
```

Boolean

Type

Intrinsic Function Data Conversion

Purpose

Converts an integer, real, or string expression to a Boolean value.

Syntax

Boolean*(expr)*

Parameters

expr: Integer, real, or string expression (input)
This parameter specifies the integer, real, or string expression to convert.

For Integers
= 0: false
<> 0: true

For Reals
= 0: false
<> 0: true

For strings
True if the first character is a "T," otherwise false.

Char

Type

Intrinsic Function Data Conversion

Purpose

Returns a one-character string which has the ASCII value of an integer expression.

Syntax

Char*(iexpr)*

Parameters

iexpr: String expression of 1 character (input)
This parameter specifies the integer expression to be converted to an ASCII character. See Appendix F, "ASCII Character Set," for more information.

Clear

Type

Statement Input/Output (Window)

Purpose

Clears a window.

Syntax

Clear *iexpr*

Keyword modifiers

iexpr: Optional expression which specifies the window number you want to clear. If no *iexpr* is given, all windows are cleared. If *iexpr* is 11, the graphics window is cleared. To clear the alpha/text window number 11, use a -11 or use **AWinClear.** Values 1 through 10 clear the corresponding UPL windows. To clear windows 11 through 20, use the corresponding negative values, -1 through -10. See DefineAW for a list of window number assignments.

Examples

```
CLEAR
CLEAR 2
CLEAR PRINT_WIN
```

Close

Type

Statement Input/Output (File)

Purpose

Closes a file that has been opened with the Open statement.

Syntax

Close *flvar*

Keyword modifiers

flvar: File variable.
After you close a file, you can use the file and file variable again in another Open statement. If you do not close a file before exiting a function or procedure where you declared the file variable, the file closes automatically.

Example

```
CLOSE DATA_FL
```

CntrlToNum

Type

Intrinsic Function Data Conversion

Purpose

Converts any non-printabie ASCII characters in a string to the format: #ascii num# and returns the new string. This is the same form used by Personal Designer and the UPL compiler. See **NumToCntrl** for more information.

The UPL program may encounter non-printable ASCII characters when reading from a file created by another prograrn. With many editors, you can create ASCII characters by holding down the Ctrl key and typing the character or by holding down the alt key and typing the numeric ASCII value.

Syntax

CntrlToNum*(str)*

Parameters

str: String expression (input)
This parameter specifies the characters in a string to be converted to ASCII. Non-printable ASCII characters have decimal values less than 32 and greater than 159. See Appendix F, "ASCII Character Set," for a compiete list of ASCII characters used by the system.

Example

```
NewStr = CntrlToNum(0ldStr)
```

$CodeSize

Type

Statement Compiler Directive

Purpose

Sets the number of bytes of UPL code that will reside in virtual memory.

Syntax

$Codesize *iconst*

Keyword modifters

iconst: Number of bytes you want in fast 10. This directive may occur anywhere in the UPL program. If more than one **$CodeSize** directive is specified, the last one is used. The default is the smaller of: a) the actual size of the UCD file or b) the amount of virtual memory set aside for UPL. The latter value is specified using the Configurator.

Use this directive only if you are using the Process statement to invoke another UPL program which is larger than the program you are writing. In this case, set **$CodeSize** to the larger of the two.

Example

```
$CODESIZE 10000
```

Const

Type

Statement Declaration

Purpose

Declares the name, data type, and value of a named constant. The data types are the same for variables except that a file may not be a constant. Literal constants rnust also be scalars; there are no array constants.

A named constant may appear anywhere in a program that a literal constant does.

Syntax

Const *datatype constname = const, constname = const....*

Keyword modifiers

datatype:	Data type of the constant. This may be Integer, Integer4, Real, Coord, Boolean, or String.
constname:	Name of the constant. Only the first 16 characters are used.
const:	Value of the constant. This is not optional. The value must be a literal constant with the data type given by *datatype.* Regardless of whether the constant is declared in the Group section, a procedure, or a function, it will always be set to this value once at the beginning of the program.

All declaration statements must occur after the **Proc, Fune, and Group** statements. They must appear before any other type of statements inside a procedure or function and are the only statements allowed inside the **Group** section.

For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Format."

Examples

```
Const Integer MaxSize = 10, MaxInt = 32767
Const Integer Biggest = 499

Const Real Tolerance = 0.0001, Diam = 2.50

Const Coord Origin = []
Const Coord UnitVec =  [0.57735, 0.57735, 0.57735]

Const Boolean Yes = True, No = False

Const String Title = "Wigit Design #345x"
Const String Dots = “ ..............  “
```

Coord

Type

Statement Declaration

Purpose

Declares the name, data type, aggregate type and initial value of a coordinate variable. An initial value is optional. Array variables must be declared with their maximum subscripts.

Variables of coordinate data type are made up of three real components in the range of -1.0E+38 to -1.0E-37, 1.0E-37 to 1.0E+38, and 0.0.

Syntax

Coord *cvar= [constx, consty constz] | carray(iconst,...)....*

Keyword modifters

cvar: Declares the name of the coordinate variable. Only the first 16 characters are used.

constx: Optional literal or named real constant. It specifies the initial value for the ".X" attribute of a coordinate scalar variable.

consty: Optional literal or named real constant. It specifies the initial value for the ".Y" attribute of a coordinate scalar variable.

constz: Optional literal or nameil real constant. It specifies the initial value for the ".Z" attribute of a coordinate scalar variable.

carray: Declares the name of the coordinate array variable. Only the first 16 charaeters are used.

iconst: Array subscripts. These declare the variable to have aggregate type array. Up to five subscripts may be declared. The subscripts must be enclosed in parentheses. Array variables may not be given an initial value.

If the variable is declared in the Group section, it will be set to these values once at the beginning of the program. If the variable is declared in a procedure or function, it will be set to these values each time the procedure or function is called.

All declaration statements must occur after the **Proc**, **Func**, and **Group** statements. They must appear before any other type of statements inside a procedure or function and are the only statements allowed inside the Group section.

For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Format."

Examples

```
Coord EndPoint1, Endpoint2, Vertices(100)

Coord ThisPoint = [0.0,0.0,0.0]

Coord ThatPoint = [3.25,7.789,415.23]
```

Coord

Type

Intrinsic Function Data Conversion

Purpose

Converts three real expressions to a single coordinate value and returns the value. You cannot use the [X, Y, Z] notation unless X, Y, and Z are constants. This function allows you to convert the X, Y, and Z expressions to one coordinate expression.

Syntax

Coord*(x, y, z)*

Parameters

x: Real expression (input)
This parameter specifies the X component of the returned coordinate value.

y: Real expression (input)
This parameter specifies the Y component of the returned coordinate value.

z: Real expression (input)
This parameter specifies the Z component of the returned coordinate value.

Examples

```
C = Coord(R, 3.5, ZVal/Rad+10.0)

C = C2 + Coord(X1, Y2*Fac, Sin(ZRad)/Sqrt(X1))

SEND "INS POINT:", \
         DigStr(Coord(XI,Y2 + COS(THETA),0.0))
```

Cos

Type

Intrinsic Function Trigonometric

Purpose

Returns the cosine of a real expression. This function returns a real value in radians.

Syntax

Cos*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the real expression whose cosine is returned.

Date

Type

Intrinsic Function Systern Interface

Purpose

Returns the current system date in the format: MM/DD/YY. This function returns a string value (eight characters long).

Syntax

Date*()*

Parameters

The Date function has no parameters.

Example

```
Print "Today's date is", Date()
```

DefineAW

Type

Intrinsic Procedure Input/Output(Window)

Purpose

Defines or returns the characteristics of an alphanumeric window. This procedure allows greater control of window parameters than the Window statement. See Chapter 3, Functional Listing of Statements and Intrinsics under Window Input/Output, for more information.

Syntax

DefineAW*(iwin, iboxl(1), bkg, page, sflag, priority)*

Parameters

iwin: Integer expression (input)
This parameter allows you to specify the alpha window, 1 through 20, to get infortnation about or to set parameters for. If iwin is greater than zero, the parameters given are used to specify the characteristics of the specified window. If iwin is less than zero, the eurrent characteristics of the window are returned in the parameters. See the list below for window numbers and their definitions.

1	UPL window and command/prompt window
2-10	UPL windows 2-10.
11	Message window.
12	Warning message window.
13	Error message window.
14	X,Y coordinates of cursor location window.
15	Status window.
16	Help window.
17	General window to list data as fast as possible.

18 window for the Personal Designer DOS command and the UPL DOS procedure; do not use.

ibox: Integer array of 4 elements (input/output). This parameter specifies and returns the location and size of the window in screen pixel coordinates. See **PixToRowCol and RowColToPix** for more information.
Array values are:.

ibox(1)	left pixel value.
ibox(2)	lower pixel value.
ibox(3)	right pixel value.
ibox(4)	upper pixel value.

ibkg: Integer variable (input/output). This number specifies and retums the background color of the window. See the SELECT COLOR command in the Personal Designer and microDRAFT Revision 6. 0 User Reference Guide for a definition of color numbers.

page: Integer variable (input/output)
This specifies and returns the graphics page that the window is on. For alphanumeric windows, the page parameter usually equals one. This parameter is dependent on the graphics device driver.

sflag: Integer variable (input/output)
This specifies and returns the autowrap flag. Values are:

0	disabies a line autowrap.
1	enabies a line autowrap.

priority: Integer variable (input/output)
This parameter specifies and returns the priority of the window; the greater the number, the higher the priority. The window with the higher priority overwrites the window with the lower priority. See Chapter 3, Functional Listing of Statements and Intrinsics, for a definition of priorities.

Example

```
-----------------------------------------------

This program demonstrates the use of DefineAW

proc main

integer awindow, corners(4), backgrnd
integer page, wrap, priority

awindow = 5
backgrnd = 2
page = 1
wrap = 1
priority = 20
corners(1) = 5
corners(2) = 20
corners(3) = 60
corners(4) = 25

RowColToPix(corners(1), page)
DefineAW(awindow, corners(1),backgrnd, page, \
                                  wrap, priority)
AWinClear(awindow)
end proc
```

DefineModifier

Type

Intrinsic Procedure User Interface

Purpose

Allows your UPL program to accept input in the modifier format used by Personal Designer. The **DefineModifier** procedure is used with the **AskModifiers** and **GetModifier** procedures. **DefineModifier** defines all modifiers to be used and their initial values. Each set of modifiers can contain a maximum of 60 words. **AskModifiers** invokes the Personal Designer modifier processor to scan the modifiers input by the user. **GetModifier** returns the modifier values input by the user.

Syntax

DefineModifier*(iword, word, type, selected, value, immediate)*

Parameters

iword: Integer expression (input)
This parameter specifies an integer expression to be assigned to the modifier word. Set iword to zero if you want to clear existing modifier words and insert new ones.

word: String expression of 12 characters (input)
The word parameter specifies the modifier word. The parameter has a 12 character maximum which can be abbreviated in upper case. If you abbreviate the word, the user only needs to type the upper case letters to select the modifier. For example, if you set a modifier called TOLerance, the user only types TOL to select the word. However, when the user enters a "?" to get a list of the modifiers and their values, the system displays the whole word.

type: String expression of 1 character (input)
This parameter specifies one letter. This letter defines the data type of the modifier; and it also defines whether or not the modifier is exclusive of other modifiers with the same data type. If a modifier is exclusive, the user can select only one modifier out of many that have the same letter assigned to it. There are four types of data: boolean, integer, real, and string.

The following list of letters show which modifiers are exclusive and non-exclusive by data type:

A to H	Boolean, exclusive.
I	Integer non-exclusive.
J to M	Integer, exclusive.
N	Boolean, non-exclusive.
O to P	String, for file names, 64 characters maximum. NOTE: type 0 and P ean only be used once in a set of modifiers.
Q	String, for text, 1.000 characters maximum. NOTE: type Q can only be used only once in a set of modifiers.
R	Real, non-exclusive.
S to V	Real, exclusive.
W to Z	Boolean, exclusive.

selected: Boolean expression (input)
This parameter specifies whether the modifier is active by default. For non-exclusive modifiers, the selected parameter is usually set to false. For exclusive modifiers, only one of the modifiers with the same letter assignment is set to true.

value: Real expression (input)
This parameter specifies the default value for real and integer data. Integers must be converted to real numbers. You cannot set a default value for 0, P and Q modifiers.

immediate: Integer expression (input)
This parameter specifies whether control should be passed back to the calling routine immediately after this modifier has been entered by the user. A non-zero value returns immediately. A zero value passes control to the next line in the UPL program. The user is prompted for modifiers after the AskModifiers call is made. Control flow continues to the next line.

Example

```
------------------------------------------------------------
-- The following procedure shows how to set up
-- the modifiers for a command written in UPL
-- which uses the same modifiers as the Personal
-- Designer command MEASURE MPROP.

proc DefineModifierValues

DefineModifier( 0,''            ,' ',False,0.0,0)
DefineModifier( 1,'SIZE'        ,'R',False,0.0,0)
DefineModifier( 2,'DENSity'     ,'S',True,1.0,0)
DefineModifier( 3,'MASS'        ,'S',False,1.0,0)
DefineModifier( 4,'ADDPoint'    ,'I',False,1.0,0)
DefineModifier( 5,'ADDCplane'   ,'I',False,7.0,0)
DefineModifier( 6,'FILE'        ,'P',False,0.0,0)
DefineModifier( 7,'PRECision'   ,'I',False,5.0,0)
DefineModifier( 8,'TOTalonly'   ,'A',False,0.0,0)
DefineModifier( 9,'ALL'         ,'A',True,0.0,0)
DefineModifier(10,'CHTolerance' ,'R',False,0.1,0)
DefineModifier(11,'TOLerance'   ,'R',False,0.005,0)
DefineModifier(12,'BOundary'    ,'B',False,0.0,0)
DefineModifier(13,'NOBOundary'  ,'B',True,0.0,0)

end proc
------------------------------------------------------------
```

DegRad

Type

Intrinsic Function Data Conversion

Purpose

Converts a real expression frorn degrees to radians. It returns the real value.

Syntax

DegRad*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the real expression to convert.

Delete

Type

Statement Input/Output (File)

Purpose

Deletes a file. Do not delete an open file.

Syntax

Delete *filename*

Keyword modifiers

filename: String expression which gives the name of the file to be deleted.

Example

```
DELETE "TEST.DRW"
```

DigStr

Type

Intrinsic Function Data Conversion

Purpose

Returns a string which represents the X, Y, Z coordinate data of any point in space. You can use DigStr to simplify the specification of coordinate data in the Send statement.
Note that DigStr adds a comma to the end of the text string being created. This is used by the GetData processor in Personal Designer to end coordinate input for that point. See the Personal Designer and microDRAFT Revision 6.0 UserReference Guide, Chapter 2 for more ififormation on Getdata.

Syntax

DigStr*(pnt)*

Parameters

pnt: Coordinate expression (input)
This parameter specifies the coordinate value to be converted to a string.

Example

```
C1 = [-3.0,4.01 Send "ins
lin:",DigStr([1.0,2.0,3.01),DigStr(C1)
This would send the following to the command processor:

ins lin: X1.0 Y2.0 Z3.0, X-3.0 Y4.0 ZO.0,
```

DirFn

Type

Intrinsic Procedure Operating System

Purpose

Retrieves directory information about a DOS or UNIX file. All types of directory entries may be queried including directories, hidden files, and normal files.

Syntax

DirFn*(filename, iatt, timestr, datestr rsize, flag)*

Parameters

filename: String variable of 64 characters (input/output)
Thefilename parameter specifies the file name that will be searched for in the input mode; and the file name that will be returned in the output mode. On input, include path names and drive specifiers. On output, no path names or drive specifiers will be returned with the name of the file. See the flag parameter for more information.

iatt: Integer variable (input/output)
On input, iatt specifies the file attributes you want to find. On output, iatt returns the file attributes that were aetually found. Attribute values are:

0	normal file.
1	read only.
2	hidden file.
4	system file.
8	volume label.
16	subdirectory.
32	archived file.

All other values of iatt represent a combination of the numbers above. For example, iatt = 7 is a read only, hidden system file. See your operating system manual for more information on files.

timestr: String variable of 8 characters (input/output)
This parameter returns the time the file was last modified in the following format: HH:MM:SS.

datestr String variable of 8 characters (input/output)
This returns the date the file was last written in the following format: MM/DD/YY.

rsize: Real variable (input/output)
This parameter returns the number of bytes in the file. This returns a real value, not an integer.

flag: Integer variable (input/output)
On input, setflag = 1 to have the prograrn search for the first matching file name. Setflag = 0 to search for the next matching file name if a wildcard character was used.
On output, ifflag = 0, a file was found. If flag = 1, no additional files could be found.

Examples

```
------------------------------------------------
-- This example prints out a list of all EXE
-- files in the current directory.

proc main
    integer Flag, IAtt
    string FN:64
    string TimeStr:8, DateStr:8
    real RSize
    Flag = 1
    FN = '*.EXE'
    loop
        Iatt = 0
        DirFN(FN,IAtt,TimeStr,DateStr,RSize,Flag)
        exit when Flag <> 0
        print FN:-14,DateStr:9,TimeStr:9,RSize:8:0
    end loop
end proc
------------------------------------------------
```

```
------------------------------------------------
-- This example prints out a list of all
-- the files in the root directory.

proc main
    integer Flag, IAtt
    string FN:64 string TimeStr:8, DateStr:8
    real RSize

    Flag = 1
    FN = '\*.*'
    loop
        Iatt = 16
        DirFN(FN,IAtt,TimeStr,DateStr,RSize,Flag)
        exit when Flag <> 0
        print FN:-14,DateStr:9,TimeStr:9,RSize:8:0
    end loop
end proc
```

DiskFree

Type

Intrinsic Function Operating System

Purpose

The IDiskFree intrinsic is intended to supercede DiskFree. To obtain the number of free bytes as a long integer, use IDiskFree.

Returns the number of free bytes on the specified disk drive. DiskFree returns a real value, not an integer. Do not convert this result to an integer because the returned value may be greater than 32,767 bytes which will produce an incorrect result.

Syntax

DiskFree*(idrive)*

Parameters

idrive:	Integer expression (input) This parameter specifies the drive you want to query. In the table below, drive one starts with A and each successive drive is matched with the next letter in the alphabet. If you do not have a drive B, drive C will still be matched with number three. Values are:
0	current drive
1	A
2	B
3	C
etc.	etc.

Examples

```
DFree = Disk-Free(3)
```

Display

Type

Statement Input/Output (Window)

Purpose

Writes a block of text to a window on the user's screen.

Syntax

Display

Text

$

Keyword modifiers

text: Starts on the line following the **Display** statement and ends when a dollar sign is encountered. To use dollar signs in the text, key in two dollar signs to get one. Be sure to use only the dollar sign character to end the text.
The text is displayed in the **Display** window which is the command window by default. To change the display window, use **the DispWin** system variable to specify the window number. For more information, see Window and the **DispWin** variable in Appendix B, "System Variables."

Example

```
DISPLAY

       Welcome to the Tutorial

       Main Menu

A) Getting started
B) Simple demonstration
C) How to insert new Geometry

Choose option - $


ACCEPT STR IN(-ABC-) SIZE(1)
```

DOS

Type

Intrinsic Procedure Operating System

Purpose

Provides access to the native operating system on DOS systems. Currently supported operating systems are DOS and UNIX. The program can execute operating system level commands or escape to DOS temporarily. The DOS command specified in the cmdstring parameter is executed by the operating system if there is enough memory available.

Syntax

DOS*(cmdstring)*

Parameters

cmdstring: String expression (input)
This parameter specifies the DOS command to be executed. If you set the parameter to the null string, the system will switch to the DOS window which is window 18 when the program is run. Once in the DOS, the user can type in all commands needed. To return control to the program, the user must type the DOS command "Exit."

Examples

```
DOS('COPY'+FileName1+' '+FileName2) --Copies file.

DOS('')--Switches to DOS level.

--Creates the file DIRBAT which contains a
--sorted list of all batch files.

DOS(´DIR *.BAT | SORT > DIRBAT´)
```

Draw

Type

Intrinsic Procedure Graphics

Purpose

Draws on the graphics screen without putting entities into the database.

Syntax

Draw*(what, color, font, npnts, pnts(1))*

Parameters

what: Integer expression (input)
This parameter specifies what to draw. Values are:
1 Dot at each coordinate in the pnts parameter.
2 Line connecting each coordinate in pnts.
3 Boxes with lower left corner given by pnts(1) and upper right corner given by pnts(2).
Add 100 to what to have a small x drawn at the first point.
Add 200 to what to have a small x drawn at the last point.
Add 300 to what to have a small x drawn at all points.

color: Integer expression (input)
Specifies the color number. See SELECT COLOR in the Personal Designer documentation for a list of colors.
>0 Draws the color on.
= 0 Turns the color off.
<0 Uses a complementary color.

font: Integer expression (input)
This parameter specifies the line font to draw with. A negative one uses the currently selected font.

npnts: Integer expression (input)
This parameter specifies the number of points in the pnts array. If what = 3, then each pair of points specifies the opposite corners of a box.

pnts: Coordinate array variable (input)
This specifies the array of view space coordinates.

DrawText

Type

Intrinsic Procedure Graphics

Purpose

Draws graphic text on the graphics screen. This text is not added to the drawing database.

Syntax

DrawText*(str org, angle, hgt, wdt, color)*

Parameters

str	String expression (input) This parameter specifies the characters to draw. There is a maximum of 500 characters.
0rg:	Coordinate expression (input) This parameter specifies the view space origin to draw the text at. The text is always drawn leftjustified.
angle:	Real expression (input) This parameter specifies the angle in degrees (0 to 360) to draw the text in.
hgt:	Real expression (input) This parameter specifies the height in inches to draw the character on the screen. This is independent of the drawing scale.
wdt:	Real expression (input) This parameter specifies the width in inches to draw the character on the screen. This is independent of the drawing scale.
color:	Integer expression (input) This parameter specifies the color number to draw the text with. Zero erases text. See the SELECT COLOR command in the Personal Designer and microDRAFTRevision 6.0 User Reference Guide for a list of color numbers.

Dsubrec

Type

Intrinsic Procedure Database Access

Purpose

Deletes a specific subrecord from an entity.

Syntax

DSubrec*(mib,occur,error,srtype)*

Parameters

mib: Integer4 expression (input) (integer4 expression in Rev, 5.0 or later);
This parameter specifies the MIB number of the entity from which the subrecord will be deleted.

occur: Integer expression (input)
Specifies which occurrence of the srtype subrecord will be deleted. It is used if the Part Data File (PDF) portion of the entity record contains more than one srtype subrecord. If the PDF portion of the entity record does not have more than one srtype subrecord, set the occurrence to one.

error: Integer variable (output)
This parameter retums the error condition:

0	no errors.
1	an 10 error was found.
3	no srtype subrecord was found.
4	an invalid MIB number was given.

srlype: String expression of 2 characters (input)
Specifies the type of subrecord to delete. If the parameter is an empty string, the procedure will delete a subrecord of any type found in the position specified by the parameter occur.

Example

```
DSubrec(mib(i),occur,ierr,dmtype)
```

Echo

Type

Statement Input/Output (Window)

Purpose

Controls the echoing of text and digitize marks from the **Accept, Display, Print, and Send** statements on your screen. Personal Designer commands are echoed in the command window; digitize marks are echoed in the graphics window.

Syntax

Echo On | Off All

Keyword modifiers

The Echo statement is initially set to On, which echoes commands and digitize marks.
If you select Off, digitize marks will be echoed, but not commands. All is an optional keyword; Off All inhibits all echoing. Off All is especially useful when executing commands with the Send statement.

Examples

```
ECHO ON
ECHO OFF
ECHO OFF ALL
```

EntIntOf

Type

Intrinsic Procedure Geometric

Purpose

Finds the intersecting point of two entities.

Syntax

EntIntOf*(mibl, mib2, viewno, sends(1), digpnt, intpnt, error)*

Parameters

mibl: Integer4 expression (input)
This specifies the MIB number of the first entity.

mib2: Integer4 expression (input)
Specifies the MIB number of the second entity.

viewno: Integer expression (input)
This parameter specifies the view to locate the intersection in. If the two entities appear to intersect, this is a view space intersection. However, if the two entities really do intersect in 3D space, this is referred to as a model space intersection. Values are:

-1 model space intersection (in any view).
0 view space intersection in the current view.
>0 view space intersection in a given view.

If the view number cannot be found, the current view is used.

sends: Integer array of 2 elements (input/output)
This parameter is necessary if one of the possible intersecting entities is a string entity. The parameter is necessary because the string might represent a shape like a spiral which may have many intersections with another entity. If no entities are strings, use a dummy array as a place holder.

The array elements specify the string vertex numbers to find the intersection for. You can use the iend parameter in the **GetEnt** procedure to determine which vertex is closest to the digitized point. The array returns the following values:

sends(1) the vertex number for the *mib1* entity.

sends(2) the vertex number for the *mib2* entity if the entity is also a string.

digpnt: Coordinate expression (input)
This parameter specifies the intersecting reference point. If two entities intersect in more than one place, the closest intersecting point to *digpnt* will be the point returned in the *intpnt* parameter.

intpnt: Coordinate variable (input/output)
This parameter specifies the intersection point.

error: Integer variable (input/output)
This parameter returns the error condition returned to the user. Values returned are:

0	at least one intersecting point was found.
10	parallel lines were found. Depending on the view, this could either indicate an infinite number of intersecting points, (the lines occupy the same space), or no intersection at all.
11	no intersection was found.
12	intersecting point cannot be determined for the two entity types given.

Example

```
Ent_Int_Of(Mib1, Mib2, 7, SEnds(1), C1, Pnt, IErr)
```

EntMask

Type

Intrinsic Procedure User Interface

Purpose

Specifies the types of entities that can be accepted as input by GetDig, GetEnd, or GetEnt.
The first call to **EntMask** should be made with a value of zero to reset entity masking. The next call to EntMask with a value other than zero allows only that entity type to be accepted. Successive non-zero calls add new entities to the list of acceptable entity types.

Syntax

EntMask*(ienttype)*

Parameters

ienttype: Integer expression (input)
This parameter specifies the entity types that will be accepted by the **GetDig, GetEnd, and GetEnt** procedures. Values are:

0 resets parameter so that all entity types can be picked.
1 adds Line entity type.
2 adds String entity type.
3 adds Arc entity type.
4 adds Text entity type.
5 adds Point entity type.
6 adds Linear Dimension entity type.
7 adds Label, Point Dimension entity type.
8 adds Radius Dimension entity type.

9	adds Angular Dimension entity type.
10	adds Cross-Hatching entity type.
11	adds Figures entity type.
12	adds Diameter Dimension entity type.
13	adds MView.
14	adds Ellipse entity type.
16	adds Curve entity type (Cpole).
17	adds Surface entity type (Spole).
18	adds Plane entity type.
30	adds NURB curve entity type.
31	adds NURB surface entity type.
35	adds 3-D tool path entity type.
36	adds 2 1/2 D tool path entity type.

Example

```
------------------------------------------------
-- This example will only allow Line and
-- String entities to be picked by GetEnt.

proc main
    integer endd
    integer4 MibList(100), NEnt

    EntMask(0)
    EntMask(1)
    EntMask(2)

    GetEnt( 100, NEnt, MibList(1), endd)

end proc
```

EntPntOn

Type

Intrinsic Procedure Geometric

Purpose

Determines the point on an entity which is closest to the point given by the digpnt parameter.

Syntax

EntPntOn*(mib, send, digpnt, transform(1), onpnt, error)*

Parameters

mib: Integer4 expression (input)
This parameter specifies the MIB number of the entity to find the point on.

send: Integer expression (input)
If mib is a string entity, send specifies the segment to search for the point. This parameter is necessary if the entity is a string entity. The parameter is necessary because the string may represent a shape like a spiral which may have many points near the given point. Negative values cause EntPntOn to find the actual point closest to digpnt on the string. Positive values return a point which is the projection of digpnt onto the send segment, even if that segment must be extended in space to allow the projection. If no entities are strings, use a zero as a place holder.

digpnt: Coordinate expression (input)
The digpnt parameter specifies the proximity reference point. The point on the entity which is closest to this point becomes the onpnt parameter. This point is given in the coordinate system defined by the transform parameter.

transform: Real array of 15 elements (input/output)
This transfor7n specifies the view orientation used when determining the closest point on the entity. When working in model space, use the **NullTransform** procedure.

onpnt: Coordinate variable (input/output)
This returns the point which is on the given entity that is closest to the *digpnt* parameter. It is given in model space coordinates.

error: Integer variable (input/output)
This parameter specifies the following error condition:

0 no errors were found.
1 an 10 error was found.
2 there are not enough bytes to read (nbytes is too big).
3 the subrecord was not found.
4 an invalid MIB number was given.

EnvVar

Type

Intrinsic Procedure Operating System

Purpose

Returns the value of a operating system (OS) environment variable. Currently supported operating systems are DOS and UNIX. Environment variabies can be set with the DOS or UNIX set command; refer to your operating system manual for more information.

Syntax

EnvVar*(envvarname, envval)*

Parameters

envvarname:	String expression (input) This parameter specifies the OS environment variable name to look for and return the value of
envval:	String variable (input/output) This parameter retums the value of the environment variable. lf envvarname is not found, envval is returned as an empty string, EnvVal.Length = 0.

Example

```
Env Var("PATH", PathStr)
If PathStr = " " Then
    print "no path name found"
Else
    print "path = ",PathStr
End If
```

Erase

Type

Statement Database Access

Purpose

Removes an existing entity from the drawing database.

Syntax

Erase *entloc* **Rpnt***(bexpr)*

Keyword modifiers

entloc: Specifies which entity to delete from the database. You must use an entloc keyword in this statement. The entloc keyword can be used in two ways; the one you use depends on whether you know the MIB number of the entity to be deleted.

If you know the entity's MIB number, use this form for *entloc:*

EntId*(i4expr):*
Replace i4expr with an integer4 expression for the MIB number. You may find the MIB number by using the Verify statement or by using intrinsic functions. Some intrinsics, such as GetEnt, allow the user to digitize entities in the graphics window. Their MIB numbers are then available to the program. Other functions, such as **FindProp and TagMib,** will return an MIB number when given non-graphical information such as the entity's properties or tags. It is recommended that the MIB number be obtained before using the Erase statement.

If you do not know the entity's MIB number, you may use one of the following keywords for *entloc*:

First:
Deletes the first entity in the database.

Next:
Deletes the next entity in the database. This keyword allows the program to step through the database sequentially and delete each entity. Each time an Erase Next statement is executed, the next entity in the database is deleted. An **Erase Next** statement may also be used after an **Erase EntId***(iexpr)* statement. The database search will then start at the *iexpr* entity instead of the first entity.

Last:
Deletes the last entity in the database. This keyword allows the program to erase the last entity inserted into the database without searching the database from the beginning.

When the end of the database is reached, DBStatus is set to two. See Appendix B, "System Variables," for more information.

Rpnt*(bexpr):*
Optional clause to specify repainting of entity.
Replace bexpr with the Boolean expression after the keyword **Rpnt.** If **bexpr** evaluates to true, then the entity is erased from the graphics window after it is deleted from the database. Otherwise, it is not.

Example

```
ERASE ENT_ID(E_ENT)

ERASE LAST RPNT(TRUE)
```

Exist

Type

Intrinsic Function System Interface

Purpose

Returns true if a specified file exists; otherwise returns false. This function returns a Boolean value.

Syntax

Exist*(sexpr)*

Parameters

sexpr: String expression (input)
This parameter specifies the file name. The file name may include path names and a drive specifier.

Exit

Type

Statement Flow Control

Purpose

The Exit statement unconditionally or conditionally exits the current Loop or If block structure.

Syntax

Exit Loop | **If** *iexpr* | **All When** *bexpr*

Keyword modifiers

Loop | **If:** Optional clause that specifies the type of structure you want the program to exit.

iexpr | **All:** An optional clause that specifies the number of levels in the structure you want the program to exit. If you choose All, the program exits all loop and/or if structures. If you replace iexpr with an integer expression, the program exits that number of structures.

When *bexpr:* Optional clause that specifies the Boolean expression to satisfy in order to exit. Otherwise the program continues execution on the next line of code.

Examples

```
Exit   --(Exits 1 if or loop structure)
Exit 2 --(Exits 2 nested if or loop structures)
Exit All     --(Exits all if or loop structures)
Exit If      --(Exits 1 if structure)
Exit If All--(Exits all if structures)
--(Exits all if structures if i = 1)
Exit If All When i = 1
```

```
--(Exits 1 if structure if J = 1)

Exit If When J = 1

Exit Loop    --(Exits 1 loop structure)

Exit Loop 2  --(Exits 2 nested loop structures)

Exit Loop All --(Exits all nested loop structures)

--(Exits all nested loop structures if i = 1) Exit Loop
All When J = 1

--(Exits 1 loop structure if J = 1)
Exit Loop When J = 1

--(Exits 1 loop or if structure if i = 1)
Exit When J = 1
```

Extract

Type

Intrinsic Function String Handling

Purpose

Extraets a substring from a given string. This function returns a string value.

Syntax

Extract*(sexpr, iexpr,numchar)*

Parameters

sexpr: String expression (input)
This parameter specifies the string.

iexpr: Integer expression (input)
This parameter specifies the starting position in the string sexpr. If iexpr is greater than the length of sexpr, a null string is returned.

numchar: Integer expression (input)
This parameter speeifies the number of characters to extract from the string sexpr. If iexpr + numchar is greater than the length of sexpr, the extracted string will end with the last character in sexpr.

File

Type

Statement Declaration

Purpose

Declares the name and aggregate type of a file variable. File variables represent the file to the program and have various attributes which relate to the actual file. File variables use the same name for their data type and aggregate type: file.
File variables may not be assigned a constant value and may not be arrays.

Syntax

File *filevar, filevar...*

Keyword modifiers

filevar: Name of the file variable. Only the first 16 characters are used.

All declaration statements must occur after the **Proc, Func, and Group** statements. They must appear before any other type of statements inside a procedure or function and are the only statements allowed inside the Group section.
For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Format."

Examples

```
File InputFile, OutFile

File DataFile
```

FillPoly

Type

Intrinsic Procedure Graphics

Purpose

Creates a filled polygon with a given color. The polygon is only filled on the screen; it does not become part of the part database.

Syntax

FillPoly*(color pattern, ixy(1), nverts)*

Parameters

color: Integer expression (input)
This parameter specifies which color to fill the polygon with. Specify colors with a Personal Designer color number. For a definition of color numbers, see the SELECT COLOR Command in the Personal Designer and microDRAFT Revision 6.0 User Reference Guide. A positive number for color obscures all geometry inside the polygon. A negative number leaves the geometry visible. A second call to **FillPoly** with a negative color fills the polygon with the original background color. See the example.

pattern: Integer expression (input)
This parameter specifies the pattern to fill the polygon with. If the pattern number is zero or one, the polygon is filied with a solid pattern, If the pattern number is greater than one, the polygon is filled by a pattern specified by the graphics device driver.

ixy: Integer array of nvert elements (input/output)
This parameter specifies the X, Y coordinates of the polygon's vertices. The parameter must be specified in pairs of X and Y pixel coordinates. To convert from two-D and 3D coordinates, use the **Map2PxN** procedure.

nverts: Integer expression (input)
This specifies the number of X,Y coordinate pairs in the ixy parameter. A maximum of 6.000 pairs may be specified.

Example

```
Proc Main
Integer IXY(2,4)
Coord Pnts(4)

Pnts(1) = [2,21
Pnts(2) = [2,-21
Pnts(3) = [-2,-21
Pnts(4) = [-2,21

Map2PxN(Pnts(1), IXY(1), 4)

Fillpoly(9,1, IXY(1), 4)

Fillpoly(-9,1, IXY(1), 4)

End Proc
```

FindFn

Type

Intrinsic Procedure Operating System

Purpose

Finds the directory of a specific file. The current directory is checked first. If the file is not found there, the file paths given by the paths parameter are searched. The **EnvVar** procedure ean be used to get the value of the paths parameter directly from the operating system's **path** environment variable.

Syntax

FindFn*(filename, paths, ifound)*

Parameters

filename: String of 64 characters (input/output)
Thefilename parameter specifies the file name that will be searched for on input; and the file name that will be returned on output. On input, do not include path names and drive specifiers. On output, path names will be returned with the name of the file.

paths: String expression (input)
This specifies the drive and path names to search for the filename parameter. The paths parameter must be specified in the format <path>;<path>;. This is the same fortnat used in the **path** environment variable.
An example of a typical path string is:
`C:\DOS;C:\BIN;C:\PD5`

ifound: Integer variable (input/output)
This parameter retums whether or not the file was found. Values returned are:

0 the file was not found.
1 the file was found.

Example

```
Proc Main
    Integer Ifound
    String PathSTR:80,MyDrawing:64
        :
    EnvVar("PATH",PathStr)
    MyDrawing = "GEOM.DRW"
    FindFn(MyDrawing,PathStr,IFound)
    IF Ifound <> 0 Then
        Print "Drawing name and Path is", MyDrawing
    Else
        Print "Drawing not founds"
    End If
End Proc
```

FindMenu

Type

Intrinsic Procedure User Interface

Purpose

Determines the on-screen icon number that is at a screen location. This icon number can then be used with other intrinsie procedures such as **GetMenuInfo and HilightMenu.**
This procedure rnay be used by UPL programmers who will be writing tutorial programs which have the user selecting an icon menu box. It may also be used to simply determine which icon the cursor is over.

Syntax

FindMenu*(menunum, idigpnt(1))*

Parameters

menunum: Integer variable (input/output)
This parameter returns the icon menu number that the idigpnt parameter is over. The menunum parameter returns a negative one if idigpnt is not over any icon menu.

idigpnt: Integer array (input/output)
This parameter, given in pixel coordinates, speeifies what screen location to look for the menu icon. To get the current X, Y cursor position in pixel coordinates, use the GetC procedure or funetion 11 in the **SysVarl** procedure. You must use these procedures since idigpnt is not in the graphics window.

Example

```
Find-Menu(MenuNum, IPnt(1))
```

FindProp

Type

Intrinsic Procedure Database Access

Purpose

Quickly finds properties of entities.

If you have a property name and a starting MIB number, this procedure will return the property type, property value, and the MIB number of the next entity the property name was found on.

You may use the intrinsie procedure **EntMask** to restrict the types of entities to be searched.

Syntax

FindProp*(propname, proptype, propval, mib)*

Parameters

propname:	String of 8 characters (input/output) On input, this parameter specifies which property name to find. The name can include the wildcard character "?" in any or all of the character positions. If a wildcard is specified, this parameter returns, on output, the name of the property found. Note that if **FindProp** is called in a loop, the value of propname can change. You may need to explicitly reset the value of propname inside the loop to find the exact propname you want. See the example below.
proptype:	String of 7 characters (input/output) If a property is found, the property type is retumed in this parameter.
propval:	String variable of 100 characters (input/output) The property value is returned in this parameter.

mib: Integer4 variable (input/output)
On input, this parameter specifies the MIB number to start the search at. To search the entire part, the mib parameter should initially be set to zero. If a property is found on an entity, the MIB number will be returned in this parameter. If no matching property name is found, a zero will be returned. To continue searching where the last call to **FindProp** left off, call **FindProp** again. Set mib to -1 and propname to the name of the desired property. If negative one is returned, an error has occurred in reading the part database.
Note that the MIB numbers may not be returned in aseending order. To force them into order the part must be packed before calling **FindProp.**

Example

```
-- The following is a program fragment that will
-- find all properties on all entities that
-- match the name "PART????"

PName1 = IPART????"
MIB = 0
Loop
    --This assignment is necessary
    --because FindProp changes PName2
    PName2 = PName1
    FindProp(PName2, PType, PVa1, MIB)
    exit when MIB <= 0
    Print MIB:6,PName2,',",PType,',",PVa1
    MIB = -1
End Loop
```

FlushInput

Type

Intrinsic Procedure User Interface

Purpose

Clears the Personal Designer input buffer. When this procedure is called, all pending user input is cleared from the buffer and the UPL program can start obtaining new input.
This can be used in conjunction with the InputStr procedure.

Syntax

FlushInput

Parameters

This procedure has no parameters.

Func

Type

Statement Program Structure

Purpose

Declares the name and returned data type of a user-defined function, as well as the name, data, and storage types of the parameters. All statements between the **Fune and End Func** keywords form the body of the function.

Syntax

Func *funcname(parameterlist)* **Return** *retdatatype*

Keyword modifiers

funcname: Declares the name of the user-defined function. Only the first 16 characters are used. Usingfuncname in an expression in the program will cause the function to be called. Expressions enclosed in parentheses after the function name will be passed as parameters. The function will return a value that will take the place of the function call in the expression. See Chapter 2, Program Structure, for more information.

parameterlist: Contains parameter declarations. Parameters are optional, and any number of them may be declared, but they must be enclosed in parentheses. If there are no parameters, you must still include parentheses. Parameter declarations are equivalent to variable declarations inside a function- their names are local to the function. However, paratneter names may not be the same as any variables declared in the Group section. A parameter list takes the form of:

mode datatype paramname

mode

Keyword which declares the parameter mode. Functions may only have input parameters. Iherefore, mode must always be replaced with "in."

datatype

Keyword that declares the data type of the parameter. It must be one of the UPL data types: Integer, Integer4, Real, Coord, String, or Boolean. There is no initial default data type, but, the most recently used data type becomes the default after the first parameter is declared.

paramname

Parameter name. Only the first 16 characters will be used. Arrays and files may not be passed as parameters to functions.

If a string parameter is declared, the maximum length must be given preceded by a colon.

There are shortcuts for declaring parameters. If the data type of the parameters has not changed, they may simply be separated by commas. If the data type changes, do the following:

1. separate the declarations with a ; or start a new line
2. list the new data type
3. list the new parameter names separated by commas.

retdatatype: Keyword that specifies the data type of the return values computed by the function. This value must be returned from the function using the **Return** statement. It rnust be one of the UPL data types: Integer, Integer4, Real, Coord, String, or Boolean.

For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Format."

Example

```
Func ProcessData(In Integer IScalar; Real Delta) \
Return Integer
```

GetBit

Type

Intrinsic Procedure Arithmetic

Purpose

Returns the value of a binary bit in the bittable parameter. This value is located at the offset specified by the ibit parameter.

This procedure is useful when you want to store and manipulate large amounts of simple True/False or On/Off data. See SetBit for more information.

Syntax

GetBit*(bittable(1), ibit, bit)*

Parameters

bittable: Integer variable or array (input/output)
This parameter specifies a table of bits. Each bit can have a value of zero or one. Each integer in bittable can store up to 16 binary bit values.

ibit: Integer expression (input)
This parameter specifies the offset in the bittable(1) parameter you want returned. The first bit is ibit = 0.

bit: Integer variable (input/output)
This parameter returns the value of the bit. It will have a value of zero or one.

Example

```
Table(1)     8192--- equal to "0010 0000 0000 0000"
in binary Get_Bit(Table(1), 13, BitVal)
Print "Bit 13 is",BitVal
```

GetC

Type

Intrinsic Procedure User Interface

Purpose

Returns the next character from user input and the current crosshair location. GetC uses the macro set currently selected by Personal Designer.

Syntax

GetC*(char ixy(1))*

Parameters

char: String variable of 1 character (input/output)
This parameter returns the retrieved charaeter. If the user digitizes a location that is not over a character (: ;) in a menu, a one is returned. Otherwise char returns the input character.

ixy: Integer array of 2 elements (input/output)
This parameter returns the crosshair position (in pixel coordinates) at the time the character was retrieved. The first element of ixv is the X value; the second element is the Y value.

GetCPL

Type

Intrinsic Procedure Geometric

Purpose

Returns the transformation matrix for a CPL number. Only the transformation and offset portions of the transformation matrix (the first 12 elements), are set. The offset portion of the matrix is not set if the CPL number given is predefined by Personal Designer as a view number. See Appendix E, "Internal Data Format," under Transformation Matrix, for more information.

Syntax

GetCPL*(cplno, transform(1))*

Parameters

cplno: Integer expression (input)
This speeifies the construction plane number to get the transformation matrix for. If the CPL is not defined, the current view transforrn is returned.

transform: Real array of 15 elements (input/output)
This returns the transformation matrix for the given CPL number. Only the first 12 elements of transfonn are filled by this procedure.

Getcur

Type

Intrinsic Procedure Input/Output (Window)

Purpose

Returns the cursor column and row position relative to the upper left corner of the specified window.

Syntax

GetCur*(iwin, row, col)*

Parameters

iwin: Integer expression (input)
This parameter specifies the window number to get the cursor position for.

row: Integer variable (input/output)
This parameter returns the row number of the current cursor position relative to the window specified by iwin.

col: Integer variable (input/output)
This parameter returns the column number of the current cursor position relative to the window specified by iwin.

GetDig

Type

Intrinsic Procedure User Interface

Purpose

Allows your program to get coordinate data using the Getdata processor. This is the same format used by Personal Designer commands. See Appendix H, "Writing Personal Designer Commands", for more information.

Syntax

GetDig*(max, gleep, ndigs, xyz(1))*

Parameters

max: Integer expression (input)
This parameter specifies the number of coordinates to retrieve. A maximum of 100 is allowed. When the program is run, if a colon, semi-colon, carriage return, or control C is input from a menu or the keyboard, **GetDig** will stop accepting digitizes and the systern variable **LastChar will** be given that character's ASCII value. Otherwise, **LastChar** will be set to one after max digitizes.

gleep: Integer expression (input)
This parameter specifies how to mark each digitize:

0	No mark.
1	Small x at coordinates.
2	Lines between coordinates.
3	Lines between coordinates and x at coordinates.

ndigs: Integer variable (input/output)
This parameter returns the number of digitized coordinates.

xyz: Coordinate array variable of max elements (input/output)
This parameter returns the model space coordinates.

GetEnd

Type

Intrinsie Procedure User Interface

Purpose

Allows your program to get coordinate data at entity endpoints using the Getdata processor. This is the same format used by Personal Designer commands. See Appendix H, "Writinc, Personal Desi ner Commands," 9 for more information.

Syntax

GetEnd*(max, gleep, nend, xyz(1))*

Parameters

max: Integer expression (input)
This parameter specifies the number of coordinates to retrieve. A maximum of 100 is allowed.
When the program is run, if a colon, semi-colon, earriage return, or control C is input from a menu or the keyboard, **GetEnd** will stop accepting digitizes and the system variable **LastChar** will be given that character's ASCII value.
Otherwise, **LastChar will** be set to one after max digitizes.

gleep: Integer expression (input)
This parameter specifies how to rnark each digitize:

0	No mark.
1	Small x at coordinates.
2	Lines between coordinates.
3	Lines between coordinates and x at coordinates.

nend: Integer variable (input/output)
This parameter returns the number of endpoints. These endpoints are stored in the xyz parameter.

xyz: Coordinate array variable of max elements (input/output)
This parameter returns the model space coordinates.

GetEnt

Type

Intrinsic Procedure User Interface

Purpose

Allows the user to digitize entities in the same manner as with Personal Designer. Using the Getdata processor, GetEnt returns a list of the MIB numbers of digitized entities. Use **EntMask** to limit the types of entities that GetEnt can select. Make any necessary calls to EntMask before a call to GetEnt. Digitized entities will highlight in the same manner as when digitized in Personal Designer. See Appendix H, "Writing Personal Designer Commands," for more information.

Syntax

GetEnt*(maxmib, nent, miblist(1), iend)*

Parameters

maxmib: Integer expression (input)
This parameter specifies the maximum number-of entities the user can digitize. Any maxmib value greater than zero specifies the number of entities the user can select. The maximum is 1.000. If the user wants to choose more than 1.000 entities, set maxmib equal to negative one and use the **BigMibList** intrinsic function instead of the miblist parameter.
When the program is run, if a colon, semi-colon, carriage return, or control C is input from a menu or the keyboard, GetEnt will stop accepting digitizes and the system variable LastChar will be given that character's ASCII value. Otherwise, LastChar will be set to one after max digitizes.

nent: Integer4 variable (input/output)
This parameter returns the number of entities actually digitized.

miblist: Integer4 array of maxmib elements (input/output)
If maxmib is greater than zero, this array returns the MIB numbers; if maxmib is less than zero, it is a signal that the **BigMibList** function is used to retrieve the MIB numbers. In that case, declare miblist to be one element long.

iend: Integer variable (input/output)
For a line or arc, iend returns the end that is closest to the digitized point. For a string, it returns the vertex that is closest to the digitized point. For crosshatching, it returns the closest end of the closest crosshatching line. For all other entities, it returns one.
The order the user digitizes points when creating an entity corresponds to the number that will be assigned to the endpoint or string vertex.

Example

```
-----------------------------------------------------
-- This example shows how to call GetEnt

proc main

    integer4 NEnt, MIBList(100), MIB
    integer I, Iend

    print 'digitize entities:1',
    GetEnt( 100, NEnt, MIBList(1), Iend)
    print
    print 'You digged these entities:'
    loop I=1 to integer(NEnt)
        MIB = MIBList(I)
        print MIB
    end loop

end proc
-----------------------------------------------------
```

GetHelp

Type

Intrinsic Procedure User Interface

Purpose

Allows the user to access the on-line help systern while in a UPL program. Users stay in the help system until they press the eseape key to exit. See **SetHelp** and Appendix H, "Writing Personal Designer Commands," for more information

Syntax

GetHeip*(helpindex)*

Parameters

helpindex: Integer expression (input)
This parameter specifies the help index number in the help definition file you want displayed. If helpindex is not defined in the help file, the message "No Documentation Available" appears.

Example

Get_Help(1001)

GetKbdChar

Type

Intrinsic Function Operating System

Purpose

Checks whether or not a character has been input to the keyboard. If it has, the function returns an integer keyboard code. If not, it can wait for a character and then return the code. **GetKbdChar** bypasses all of Personal Designer's other input sources, (including tablet and execute files), and the keyboard macros.

Syntax

GetKbdChar*(iopt)*

Parameters

iopt: Integer expression (input)
This parameter specifies how the UPL program waits for keyboard input.
-1 checks the keyboard for a character. If there is a character in the keyboard buffer, the function returns its ASCII value. If there are no characters, the function immediately returns a negative one.
0 waits until a character is input on the keyboard. If there is a character in the keyboard buffer, the function returns its ASCII value or it waits until a character is typed in and then returns its ASCII value.
> 0 waits a specified number of seconds for a character to be entered. If there is a character in the keyboard buffer, the function returns its ASCII value, or it waits iopt seconds for a character to be typed in. If no character has been typed in iopt seconds, it retums a negative one.

Examples

```
exit when char(GetKbdChar(-1» = "s" IDummy = GetKbdChar(0)
```

GetLayer

Type

Intrinsic Function Graphics

Purpose

Determines if the display of a particular layer is currently active; and, optionally, if the layer is used.
If the ilayer parameter is positive, GetLayer determines whether the layer is active. If ilayer is negative, GetLayer determines whether the layer is active and/or used. GetLayer returns the following integer codes:

0 Layer is inactive and not used (if ilayer was less than 0).
1 Layer is active and not used (if ilayer was less than 0).
2 Layer is inactive but the layer is used.
3 Layer is active and is used.

Note that testing for an active layer is faster than testing for a used layer. If you only need to know whether a layer is active, use positive values for ilayer to allow your UPL program to run faster.

Syntax

GetLayer*(ilayer)*

Parameters

ilayer: Integer expression (input)
Allows you to get information about a specific layer number.

Example

```
ISet = Get_Layer(22)
```

GetMenuInfo

Type

Intrinsic Procedure User Interface

Purpose

Returns information about an on-screen menu icon. An on-screen menu area can contain an icon or an icon set. The information includes:

- the icon set (layer) number the area belongs to.
- the pixel coordinates of the areas lower left and upper right corner.
- the menu command string associated with the area. For more information, see the Personal Designer and microDRAFT Revision 6.0 User Reference Guide..

Syntax

GetMenuInfo*(areanumber, setnumber,areacorners, cmdstring)*

Parameters

areanumber:	Integer expression (input) Allows you to get information about a specific icon menu number. To retrieve the area number, use FindMenu or **SysVarI** with function nine.
Setnumber:	Integer variable (input/output) Specifies the icon set (layer) which the on-screen menu area belongs to. If the area number is not found, a negative one is returned.
areacorners:	Integer array of 4 integers (input/output) This parameter returns the lower left and upper right X, Y pixel coordinates of the on-screen menu area associated with areanumber. Use **PixToRowCol** to convert the pixel to row and column coordinates.
cmdstring:	String variable (input/output) Returns the command string associated with areanumber.

Example

```
Get_Menu_Info(234, GNum, PickArea(1), CStr)
```

GetModifier

Type

Intrinsic Procedure User Interface

Purpose

Returns the modifier values the user has input with the AskModifers procedure. See **AskModifers and DefineModifier** for more information.

Syntax

GetModifier*(iword, selected, numvalue, strvalue)*

Parameters

iword: Integer expression (input)
This parameter specifies which modifier to return the value for. The modifier number is defined in the DerineModirier procedure.

selected: Boolean variable (input/output)
This parameter returns true if the modifier is selected by the user or if it is initialized to true by the **DefineModifier** procedure. Otherwise, it is returned as false.

numvalue: Real variable (input/output)
If the modifier type is an integer or real, the numerical value of the modifier is retumed in numvalue. This value is always returned as a real even if it is defined as an integer.
If the modifier type is a boolean, numvalue returns 0.0 for false and 1.0 for true.
If the modifier type is a string, numvalue returns the number of characters in the strvalue parameter.

strvalue: String variable (input/output)
If the modifier type is a string, (0, P, or Q), this parameter retums the modifier's string value. Otherwise a null string is retumed.

Example

```
Get_Modifier(2, Sel, NVal, Sval)
```

GetSerialNum

Type

Intrinsic Function Operating System

Purpose

Retrieves the six-digit serial number. On DOS systems it the Computervision guard box number. On UNIX workstations, it may be the hostid or a portion of the Ethernet address. It is usually used with the AccessCode function. The function returns the serial number as a 6-character string.

Syntax

GetSerialNum

Parameters

The GetSerialNum function has no parameters.

Examples

```
print "The serial number is:",GetSerialNum()
Acode = AccessCode(KeyStr, GetSerialNum())
```

GetTagField

Type

Intrinsic Procedure Database Access

Purpose

Returns the value of an entity's tag field. See **SetTagField** for more information.

Syntax

GetTagField*(mib, fieldnumber, fieldstring, errorflag)*

Parameters

mib:	Integer4 expression (input) This parameter is the MIB number of the entity with the tag.
fieldnumber:	Integer expression (input) This parameter specifies which tag field to read. If the fieldnumber parameter equals zero, the system tag value is returned.
fieldstring:	String variable (input/output) This is the value of the tag field. This parameter specifies the text string to put into the field. There is a total of 996 bytes for all tag fields. Each tag field uses two bytes. Each character given in fieldstring uses one byte. Be sure you do not exceed this limit.
errorflag:	Integer variable (input/output) If there is no tag attached to the entity, errorflag is returned as negative two. If thefieldnumber is greater than the number of fields currently on the tag, errorflag is returned as negative one. Otherwise, errorflag is returned as zero.

Examples

```
Get_Tag_Field(Mib1, 1, F1dStr, IErr)
Get_Tag_Field(Mib, 0, SysTag, IErr)
```

GetView

Type

Intrinsic Procedure Geometric

Purpose

Returns a transformation matrix for a given view.

Syntax

GetView*(viewno, transform(1))*

Parameters

viewno: Integer expression (input)
Specifies the view to get a transformation matrix for. If the view for the number given is not defined, the current view transform is returned.

transform: Real array of 15 elements (input/output)
This parameter returns the transformation matrix for the viewno parameter. Only the transform portion of the transformation matrix (the first 12 elements), is modified. The XIY/Z offset portion (elements 10, 11, and 12), is also modified if the viewno parameter is also a CPL number. The scaling factor portion of the transformation matrix is set to zero.

Examples

```
Get_View(8, View8Trans(1))
Get_View(76, Trans(1))
```

GoTo

Type

Statement Flow Control

Purpose

Transfers controi of the program to a specified label.

Syntax

GoTo *label* **When** *bexpr*

Keyword modifiers

label: Specifies the label to transfer control to. A label may be any legal UPL identifier name followed by a colon. It may have a maximum of 16 characters. Define label in the current procedure or function by using the label to start a statement. The GoTo statement cannot transfer control outside the procedure or function in which it is contained.

When *bexpr:* An optional clause that causes the GoTo statement to be executed only when bexpr evaluates to true.

Examples

```
GOTO ASK

GOTO LAB2 WHEN I = J

LAB2: Print "Hello"
```

Group

Type

Statement Program Structure

Purpose

Deelares the name, data, aggregate, and storage types as well as the initial values of global variabies and global named constants. The variabies and constants declared in this section are available to all procedures and functions in the programs. In addition, the variables and constants in this section may also be accessed by other UPL programs which have been invoked by the **Process** statement.
All declaration statements between the **Group and End Group** keywords form the body of the group section.
The syntax of all the declarations is exactly the same as for local variables and constants.

Syntax

Group

Keyword modifiers

The variable and constant names used in the Group section cannot be the same as any parameter names for user-defined procedures and functions.
In order to share the variables and constants with another UPL program, the variables must be defined in the same order in both Group seetions. That is, the variable or constant to be shared between the programs must have the same byte offset from the beginning of the Group section. This offset will depend upon the number and data types of variables in the section before the shared ones. Using the example below, to access Diam from another UPL program, that program must have a Group section which included identical declarations. Alternately, a dummy declaration could be made to fill the gap between the beginning of the Group section to Diam. Such a declaration would be: Integer Dummy(3). MaxInt, MinInt, and Tolerance will occupy the same amount of space which is eight bytes.

The compiler directive $Include is useful in creating group section declarations. A file of shared data declarations can be created and inserted into your programs. Since the programs use the exact same declarations, the correct offsets are assured.
For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Format."

Example

```
Group
      Const Integer MaxInt = 32767, MinInt = -32767
      Real Tolerance = 0.0001, Diam = 2.50
      String Title = "Wigit Design #345x", \
                                 Dots = '.........'
End Group
```

GText

Type

Intrinsic Procedure Graphics

Purpose

Prints alphanumeric text anywhere on the graphics screen. Note that this text is not inserted in the database. This text uses the character font used in Personal Designer alphanumeric windows. The font is limited to the font size specified by your system's graphics device .
See **DrawText** for related information.

Syntax

GText*(color, ixloc, iyloc, hjust, vjust, text)*

Parameters

color: Integer expression (input)
This parameter specifies the color of text. Values are zero through 15. See the SELECT COLOR command in the Personal Designer and microDRAFT Revision 6. 0 User Reference Guide for a definition of color numbers.

ixloc: Integer expression (input)
This parameter specifies the screen X pixel location where the text will be drawn.

iyloc: Integer expression (input)
This parameter specifies the screen Y pixel location where the text will be drawn.

hjust: Integer expression (input)
'Mis specifies the horizontal justification of the text.
1 Leftjustification.
2 Centerjustifieation.
3 Rightjustification.

vjust: Integer expression (input)
This specifies the vertical justification of the text.

1 Bottom justification.
2 Centerjustification.
3 Top justification.

text String expression (input)
Text specifies the string to be displayed on the graphics screen. No special characters such as a carriage return (ASCII 13) or line feed (ASCII 10) are interpreted.

Examples

```
-- This messalge will print on one line
GText(1, 10, 300, 1, 1,\
      "some text on the screen #13"
      " more text on the screen")


GText(1, 10, 500-Iline*16, 1, 1, Line(Iline))
```

GWinClear

Type

Intrinsic Procedure Input/Output (Window)

Purpose

Clears the graphics window with the current color of the background.

Syntax

GWinClear*(iwin)*

Parameters

iwin: Integer expression (input)
This parameter specifies the graphics window number to clear. Presently, graphics window one is the only valid value for iwin. Graphics window one is the sarne as window 11 in the Window statement. See DerineAW for more information.

Example

```
GWinClear(1)
```

HilightEnt

Type

Intrinsic Procedure User Interface

Purpose

Highlights a]ist of entities on the screen. An entity is highlighted in the same manner as when it is digitized by the user in Personal Designer. Highlighting turns off if you repaint the selected entities with RpntEnt or make another call to **HilightEnt.**

Syntax

HilightEnt*(miblist(1), nent, error)*

Parameters

miblist: Integer4 variable or array (input/output)
This specifies the]ist of entity MIB numbers to be highlighted.

nent: Integer4 expression (input)
This parameter is the number of entity MIBs in miblist.

error: Integer variable (input/output)
This parameter is the error flag:
0 No error was found.

Example

```
HilightEnt(MList(1), 4, Ierr)
```

HilightMenu

Type

Intrinsic Procedure User Interface

Purpose

Highlights an on-screen menu icon with a specified color.

Syntax

HilightMenu*(color,menuno)*

Parameters

color: Integer expression (input)
'Mis parameter specifies the color number (0 through 15), to highlight the on-screen menu icon with. See the SELECT COLOR command in the Personal Designer and microDRAFT Revision 6.0 User Reference Guide for color definitions.
If the color parameter is greater than zero, the area is filled with the given color and the icon graphics disappear. If the parameter is less than zero, the area is filled with the given color and the icon graphics will remain, but the graphics will change to another color. A second calt to **HilightMenu** with a negative number turns off the highlighting. The suggested negative value to use is -15.

menuno: Integer expression (input)
This parameter specifies the menu nuniber of the on-screen icon to highlight. The menu number can be retrieved by using the **FindMenu** procedure or by calling SysVarl with a value of nine for the ivar parameter.

Example

```
Hilight_Menu(-15, 342)
```

IDiskFree

Type

Intrinsic Function Operating System

Purpose

Returns the number of free bytes on the specified disk drive. **IDiskFree** returns an integer4 value. Since the maximum value of an integer4 is about 2 billion, it is can be used for drives containing less than 2 gigabytes of storage. This routine replaces the **DiskFree** routine.

Syntax

IDiskFree*(idrive)*

Parameters

idrive: Integer expression (input)
This parameter specifies the drive you want to query. In the table below, drive one starts with A and each successive drive is matched with the next letter in the alphabet. If you do not have a drive B, drive C will still be matched with number three. Values are:

0	current drive
1	A
2	B
3	C
etc.	etc.

Examples

```
Free = IDiskFree(3)

Free = IDisk_Free(0)
```

If-Then-Else-End If

Type

Statement Flow Control

Purpose

Conditionally executes a group of statements.

Syntax

If *bexpr1* **Then**

statements executed if *bexpr1* is true

Else If *bexpr2* **Then**

statements executed if *bexpr2* is true

Else

statements executed if none of the *bexpr* are true

End If

Keyword modifiers

Alt statements between the **If and EndIf** keywords form the body of the **If** statement.

End If can be typed as shown or as one word, **EndIf.**

When the program encounters an **If** statement, it evaluates the Boolean expression. If the expression is true, the program executes the statements following **Then** and up to either the optional **Else** portion or the **EndIf** keyword. If the Boolean expression evaluates to false, the program skips the statements after **Then** and executes the **Else** portion instead. In either case, only one alternative is taken. See the second example.

There are several variations on the general format. First, you can use an **If** statement without the **Else** portion to let a program decide whether to perform a certain action, as shown in the first example. If A is less than or equal to B in this example, the program simply skips this step.

You can also nest **If** statements which allows you to make further conditional tests by putting a whole **If** statement inside another one. If you indent each level of nesting, it is easier to tell which **If** and **End If** keywords are paired together.
If and **Loop** statements combined can be nested up to 50 levels. See the third example.
To test one variable for many different values, use **Else If**, which is equivalent to an **End If** keyword followed by another **If** statement. When you use **Else If**, you only need one **End If** at the end of your code. Only one set of statements is executed between the **If**, **Else If**, **Else**, and **End If** keywords. This is always the first set of statements whose bexpr evaluates to true. If none are true, the set following the final **Else** is executed. If there is not a final Else keyword, no statements are executed. Examples three and four show the same **If** statement coded two ways.

Examples

```
--Example 1: The simplest form of an if statement.

IP A > B     THEN
    PRINT "GOOD VALUE"
ENDIF

--Example 2: A simple If-Then-Else statement.

IF X<5 OR X>100 THEN
    PRINT Y
ELSE
    PRINT X
ENDIF

--Example 3: Nested If-Then-Else statements.

IF A=B THEN
    A=C
ELSE
    IF A=C THEN
        A=D
    ELSE
        IF A=D THEN
        A=Z
        ELSE
        PRINT "A doesn't equal B,C,D"
        ENDIF
    ENDIF
ENDIP
```

```
--Example 4:
--If statement using Else-If clause.
--This is equivalent to the If statement in
--Example 3.

IF A=B THEN
    A=C
ELSE IF A=C THEN
    A=D
ELSE IP A=D THEN
    A=Z
ELSE
    PRINT "A doesn't equal B,C,D"
ENDIF
```

$Include

Type

Statement Compiler Directive

Purpose

Directs the compiler to read UPL program source code from the specified file.

Syntax

$Include *filename*

Keyword modifiers

filename: Specifies the name of the file to read UPL source from. It must be a literal or named string constant which specifies a legal file name for your operating system (DOS or UNIX
When the end of this file is reached, the compiler resumes reading program source in the file from which it encountered the previous **$Include** directive.
Included files may be nested up to four levels deep. That is, an included file may contain other **$Include directives,** and so on.
You can develop libraries of subroutines and declarations to use in building more complex UPL programs. This directive allows you to easily insert them. Doing so ensures that you use the same code in each program. This can reduce problems in your programs.

Examples

```
$include 'vardec.ins'

$INCLUDE 'MATH.LIB'
```

Index

Type

Intrinsic Function String Handling

Purpose

Locates the position of a substring within a string. This function returns an integer value which is the position of the first occurrence of the substring. If no match is found, zero is returned.

Syntax

Index*(sexpr1, iexpr, subsexpr)*

Parameters

sexpr1:	String expression (input) This parameter specifies the string to be searched.
iexpr:	Integer expression (input) This parameter specifies the position in sexpri to start searching for the substring. By updating this value, you may search for additional occurrences of subsexpr.
subsexpr:	String expression (input) This parameter specifies the substring to find.

InputStr

Type

Intrinsic Procedure User Interface

Purpose

Inputs a given character string to the Personal Designer input buffer. This procedure can be used to set up the input buffer so that a subsequent call to GetDig, GetEnd, or GetEnt will read a GetData modifier or reference such as **end, win, mib,** etc.
This can be used in conjunction with the **Flushluput** procedure.

Syntax

InputStr*(str)*

Parameters

str: String expression (input)
Character string to 'feed' to 'feed' to Personal Designer. It may be up to 400 characters long.

Example

```
-------------------------------------------------
-- This example shows how to call
-- InputStr cind FlushInput.

proc main
       integer4 NEnt, MIBList(100), MIB
       integer I, Iend
       print 'digitize corners of window:',
       FlushInput()
       InputStr(' win ')
       GetEnt( 100, NEnt, MIBList(1), Iend)
       print
       print 'You windowed these entities:'
       loop I=1 to integer(NEnt)
              MIB = MIBList(I)
              print MIB
       end loop
end proc
-------------------------------------------------
```

Insert

Type

Statement Database Access

Purpose

Allows a new entity to be added to the current drawing database.
Entities are referenced by Master Index Block (MIB) numbers. The number identifies an entity and gives its location in the database. An MIB is assigned when an entity is inserted into the part database by Personal Designer or a UPL program. The number remains valid untit the part is filed or exited with the pack database option.
After the **Insert** statement is executed, the system variable DBStatus is set. DBStatus gives the result of the insertion. See Appendix B, "System Variabies," for more information.

Syntax

Insert *enttype entloc entatts entdata* **Rpnt***(bexpr)*

Keyword modifiers

enttype: Specifies the type of entity to insert: Line, String, Arc, Text or Point. The enttype must immediately follow the **Insert** keyword.

entloc: Optional clause that returns the location of the inserted entity. It uses the **EntId** keyword:
EntId*(i4var)*
Returns the MIB number of the inserted entity. Replace *i4var* with an integer4 variable. When the statement is executed, *ivar* returns the MIB number of the entity inserted.

entatts: Optional keywords that let you specify the attributes common to all entities. All entatts keywords have a default value; if you use a keyword, the value specified in the expression becomes the new default value. The default value changes when the keyword is used again in another **Insert** or **Modify** statement. You can use the following entatts keywords in any order and with all entity types:

Color*(iexpr)* where *iexpr* is an integer expression which gives the color number for the new entity. The default is the color currently selected by Personal Designer.

Font*(iexpr)* where *iexpr* gives the font number for the new entity. The default is the font currently selected in Personal Designer.

Group*(iexpr)* where *iexpr* gives the group number of inserted entities. There is no default value.

Layer*(iexpr)* where *iexpr* gives the layer for the new entity. The default is the currently selected layer in Personal Designer.

View*(iexpr)* where *iexpr* gives the view you will insert your geometry relative to. Without this keyword, the geometry will be inserted in model space which is view one. If this keyword is used, it affects the data specified in the entdata keywords.

Vvis*(iexpr)* where *iexpr* gives the view number that the added entity is visible in (zero to be visible in all views). The default is zero.

entdata: Lets you specify data which is specific to different entity types. These keywords also have a default value; if you use an *entdata* keyword, the value given in the expression becomes the new default value. The default value is changed when you use the keyword again in another **Insert** or **Modify** statement.
Replace *entdata* with keywords for the specific enttype. The following list describes the keywords for each entity type:

For Lines:

Ends*(cexpr1, cexpr2)*
Specify the endpoint coordinates after the keyword Ends. Replace *cexpr1* with end one and *cexpr2* with end two of the line. Both are coordinate expressions. The default for *cexpr1* and *cexpr2* is [0.0,0.0,0.0]

For Strings:

Verts*(iexpr1, carray(iexpr2))* Specifies the coordinates for the vertices of a string entity in an array after the keyword **Verts**. Replace carray with the name of a coordinate array which speeifies the coordinates for the string vertices. Replace *iexpr1* with the number of vertices to get from carray, and *iexpr2* with the first element to get out of carray.

For Arcs:

Specify the characteristics of the arc after one of the following keywords. Arcs are drawn counterclockwise.

Org*(cexpr)* Replace *cexpr* with the origin of the arc. The default is [0.0,0.0,0.01.

Radius*(rexpr)* Replace *rexpr* with the radius of the arc. The default value is 1.0.

AB(*rexpr*) Replace *rexpr* with the beginning angle of the arc in degrees. The default is 0.0.

AE*(rexpr)* Replace *rexpr* with the ending angle of the arc in degrees. The default is 360.0.

ArcEnds*(cexpr1, cexpr2)* If this keyword is given, it replaces **AB** and **AE**. Either **AB** and **AE** or **ArcEnds** determine the beginning and ending angles of the arc. The arc is drawn such that a line from the origin (as specified by **Org**) to *cexpr1* determines the beginning angle of the arc, and a line from the origin (as specified by Org) to *cexpr2* determines the ending angle of the arc. The arc has a radius specified by **Radius** above. There is no default for **AreEnds**; values for **AB** and **AE** are used to find the default beginning and ending angles of the arc.

For Text:

Specify the characteristics of the text to be inserted after one of the following keywords:

Ang*(rexpr)* Replace *rexpr* with the angle the text is to be inserted at. The angle is in degrees. The default is zero.

Hgt*(rexpr)* Replace *rexpr* with the text character height. The default is 0.5.

Just*(iexpr)* Replace *iexpr* with one of the text justification codes:

1 left justification.
2 right justification.
3 center justification.

Lnsp*(rexpr)* Replace *rexpr* with the text line spacing factor. The default is 1.5.

Org*(cexpr)*
Replace *cexpr* with the text origin. The default is [0.0,0.0,0.0]

Txt*(sexpr)* Replace *sexpr* with a string expression that gives the actual text to be inserted. Maximum is 1000 characters. The default is an empty string.

Wdt*(rexpr)* Replace *rexpr* with the text character width. The default is 0.5.

For points:

Loc*(cexpr)* Replace *cexpr* with the coordinate of the point to be inserted. The default is [0.0,0.0,0.0].

For all above entities:

> **Rpnt***(bexpr)* An optional clause; replace *bexpr* with the Boolean expression after the keyword Rpnt. If bexpr evaluates to true, then the entity is repainted after it is added to the drawing data base. Otherwise, it is not. The default is false.

All entatts and entdata keywords may be used in any order, and may be separated by commas (,) or spaces

Examples

```
INSERT LINE ENDS([], [1.0, 1.0]), COLOR(IC), \
                                  FONT(3),RPNT(TRUE)

INSERT STRING VERTS(12, VERTICES(1))

INSERT ARC ORG(ORIGIN + [1.0,-1.0]) AB(45.0)\
                                  AE(135.0) RADIUS(R)

INSERT ARC RADIUS(D/2.0), ORG([]), ARCENDS(C1, C2)\
                                  COLOR(5)

INSERT TEXT HGT(H), WDT(H * 2.0), ORG(TORG),\
                                  TXT("Some text")

INSERT POINT LOC(PNTS(I + 1)) LAYER(10)
```

Integer

Type

Statement Declaration

Purpose

Declares the name, data type, aggregate type, and initial value of an integer variable. An initial value is optional. Array variabies must be declared with their maximum subscripts.
Variables of integer data type contain whole number values ranging from -32768 to 32767.

Syntax

Integer *ivar* = iconstl | *iarray(iconst2,...),...*

Keyword modifiers

ivar: Integer variable name. Only the first 16 characters are used.

Iconst1: Optional initial value for a sealar variable. This value must be a literal or named integer constant. If the variable is declared in the Group section, it will be set to this value once at the beginning of the program. If the variable is declared in a procedure or function, it will be set to this value each time the procedure or function is called.

ivar: Integer array variable name. Only the first 16 characters are used.

iconst2: Array subscripts. These declare the variable to have aggregate type array. Up to five subscripts may be declared. The subscripts must be enclosed in parentheses. Array variables may not be given an initial value.
All declaration statements must occur after the **Proc, Func, and Group** statements. They must appear before any other type of statements inside a procedure or function and are the only statements allowed inside the Group section. For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Format."

Example

```
Integer Int, Count=0, i, IntArray(10)
Integer Maximum = 32767
```

Integer

Type

Intrinsic Function Data Conversion

Purpose

Converts a Boolean, integer4, real or string expression to an integer value. Returns an integer. For Boolean values, false equals 0 and true equals 1. For real values, the decimal portion is truncated. Integer4 and real values must be converted with care. If the exceed the limits of integer (-32,768 to +32,767) the most significant places are truncated.

Syntax

Integer *(expr)*

Parameters

expr: Boolean, integer4, real or string expression (input) to be converted.

Integer4

Type

Statement Declaration

Purpose

Declares the name, data type, aggregate type, and initial value of an integer4 variable, also often called a long integer. It is represented internally by a 4-byte variable. An initial value is optional. Array variables must be declared with their maximum subscripts.

Variabies of integer4 data type contain whole number values ranging from -2,147,483,648 to 2,147,483,647 or approximately plus or minus 2 billion.

Syntax

Integer4 *ivar* = i4const 1 | *iarray(iconst2,...)*....

Keyword modifiers

ivar: Name of the integer4 variable. Only the first 16 characters are used.

i4const1: Optional initial value for a scalar variable. This value must be a literal or named integer4 constant. If the variable is declared in the Group section, it will be set to this value once at the beginning of the program. If the variable is declared in a procedure or function, it will be set to this value each time the procedure or function is called.

iarray: Name of the integer4 array variable. Only the first 16 characters are used.

iconst2: Array subscripts. These declare the variable to have aggregate type array. Up to five subscripts may be declared. The subscripts must be enclosed in parentheses. Array variables may not be given an initial value.

All declaration statements must occur after the Proc, Func, and Group statements. They must appear before any other type of statements inside a procedure or function and are the only statements allowed inside the Group section.
For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Format."

Example

```
Integer4 Int, Count=0, i, IntArray(10)

Integer4 Maximum = 2,147,483,647
```

Integer4

Type

Intrinsic Function Data Conversion

Purpose

Converts a Boolean, real, integer or string expression to an integer4, or long integer, value. Returns an integer4. For Boolean values, false equals 0 and true equals 1. For real values, the decimal portion is truncated.

When an [shortl integer is converted to an integer4, the value is not changed. The storage format is converted from 2-byte format to 4-byte format. This conversion is useful in assigning an integer to an integer4 variable or passing an integer to a function that requires an integer4 argument.

Syntax

Integer4 *(expr)*

Parameters

expr: Boolean, real, integer or string expression (input). Expression to be converted.

LastDig

Type

Intrinsic Function User Interface

Purpose

Returns the last digitized location that the system received from the user. The location can be digitized in several Getdata modes such as DIG, ENT, END, and ORG- for more information, see Getdata Capabilities in Chapter2of thePersonalDesignerandmicroDRAFTRevision6.OUser Reference Guide. The location returned is in model space coordinates.

Syntax

LastDig()

Parameters

The LastDig function has no parameters.

Examples

```
if vlen([], LastDig()) <= 4.0 then
    print "the last digitize was within 4 units of the
    origin"
endif

C1 = LastDig()
Print "you last digitized", C1
```

LinIntOf

Type

Intrinsic Procedure Geometrie

Purpose

Determines the true and apparent intersecting points of two lines.

Syntax

LinIntOf*(l1end1, l1end2,l2end1, l2end2, int1, int2)*

Parameters

l1end1: Coordinate expression (input)
This speeifies the first end of line one.

l1end2: Coordinate expression (input)
This parameter specifies the second end of line one.

l2end1: Coordinate expression (input)
This specifies the first end of line two.

l2end2: Coordinate expression (input)
This parameter specifies the second end of line two.

Int1: Coordinate variable (input/output)
This returns the intersection point on line one.

int2: Coordinate variable (input/output)
This returns the intersecting point on line two. If the lines truly intersect in model space, int1 equals int2. If you are interested in intersections within some tolerance value, use the **VLen** function (**VLen**(int1, int2)) and compare its result against your tolerance value.

Ln

Type

Intrinsic Function Arithmetic

Purpose

Returns the natural logarithm of a real expression. This funetion returns a real value.

Syntax

Ln*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter speeifies the real expression whose natural logarithm will be returned.

Log

Type

Intrinsic Function Arithmetic

Purpose

Returns the base 10 logarithm of a real expression. This function returns a real value.

Syntax

Log*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the real expression whose base 10 logarithm will be returned.

Loop-End Loop

Type

Statement Flow Control

Purpose

Executes a group of statements zero or more times.

Syntax

Loop *var* = *expr1* **To** *expr2* **By** *expr3*
Statements
End Loop

Keyword modifiers

Each **Loop** statement keyword has a matching **End Loop** keyword. All statements between the **Loop** and **End Loop** keywords form the body of the loop. End Loop can be typed as shown or as one word, Endloop.

var = *expr1* To *expr2* By *expr3*

Optional clause. Replace *var* with an integer, integer4 or real variable that will count the number of loops executed. You can use *var* anywhere inside the loop body. However, you should not make any assignment to this variable.

Expr1, *expr2*, and *expr3*
Replace with the sarne data type (integer, integer4 or real) as *var*. They are evaluated onee at the beginning of the loop. The default for *expr3* is one if *var* is an integer or integer4; and 1.0 if *var* is a real number.

NOTE: **By** *expr3* is an optional clause within the first clause, *var* = expri **To** *expr2*.

When executing a loop, the loop index *var* is assigned the value of *expr1*. The statements are then executed. When **End Loop** is encountered, control passes back to the **Loop** keyword. The *var* variable is then incremented **by** *expr3*. If *var* <= *expr2*, the statements are executed again and var is incremented **by** *expr3*. This sequence repeats until *var* > *expr2* at which point control passes to the statement immediately following the **End Loop** keyword. (See Example 1 below.)

This logic is reversed if *expr1* is greater than *expr2* and *expr3* is negative. See Example 2 below.

Loop statements may be nested; each **Loop** keyword is matehed with the closest **End Loop** keyword. The **End Loop** must follow the **Loop** and not be preceded by another unmatched **Loop** keyword. (See Example 3 below.)

If you do not give the optional *var* and *expr*, the **Loop** statement will loop until an **Exit** statement is encountered, or until a **GoTo** statement transfers control to outside of the loop. See Example 4 below. **GoTo** must not transfer control into the middle of a **Loop** statement.

Examples

--***Example*** 1:

```
LOOP J=1 TO NUM COUNT
    S=S+1
    PRINT S
END LOOP
```

--***Example*** 2: A "count down" loop

```
LOOP J=100 TO 1 BY -1
    PRINT J
END LOOP
```

--***Example*** 3:

```
loop i = 0 to 90 by 10
    loop j = 1 to 10
        print "the number is", i + j
        k = i + j
    end loop
end loop
print "the last number was",k
```

--***Example*** 4:

```
Loop
    accept inum prompt( "enter an integer 1-10")
    exit when (inum > 0) and (inum < 11)
end loop
```

Map2Px

Type

Intrinsic Procedure Geometric

Purpose

Maps view coordinates to screen pixel coordinates.

Syntax

Map2Px*(xy, ixy(1)*

Parameters

xy: Coordinate expression (input)
This parameter specifies the view space coordinates to convert to pixel space coordinates.

ixy: Integer array of 2 elements (input/output)
This parameter returns the pixel space coordinates. Element one is the X pixel value; element two is the Y pixel value.

Map2PxN

Type

Intrinsic Procedure Geometric

Purpose

Maps view coordinates to screen pixel coordinates.

Syntax

Map2PxN*(pnts(1), ixy(1), npnts)*

Parameters

pnts: Coordinate array of npnts (input/output)
This parameter specifies the view coordinate points to be mapped.

ixy: Integer array of *npnts**2 elements (input/output)
This parameter retums pairs of mapped X and Y coordinates in screen pixels. The first element of a pair is the is the X pixel value; the second is the Y pixel value.

npnts: Integer expression (input)
This specifies the number of points in *pnts*.

MapCPLM

Type

Intrinsic Function Geometric

Purpose

Maps a point from CPL coordinates to model space coordinates and returns the model space coordinates. The currently selected CPL is used. If no CPL is selected, the value of pnt is returned.

Syntax

MapCPLM*(pnt)*

Parameters

pnt: Coordinate expression (input)
This parameter specifies the point to be mapped.

MapFrom

Type

Intrinsic Procedure Geometric

Purpose

Maps a set of points from a transformed coordinate system represented by the transform parameter to the model coordinate system. **MapFrom** is a generalized version of **MapVM**. (The latter routine specifically maps points from view to model coordinates.) **MapFrom** performs the inverse function of **MapTo**. It is equivalent to calling Transpose to invert a transformation previously used by **MapTo** and then calling **MapTo** again.

The transform parameter may be obtained from intrinsics such as **Mat3P, RotMat,** or even GetView or GetCPL.

Syntax

MapFrom *(transform(1), pnts(1), npnts)*

Parameters

transform: Real array of 15 elements (input/output)
This parameter specifies the transformation matrix used to map the points. Only the first nine elements of transform are used.

pnts: Coordinate array of npnts elements (input/output)
On input, this parameter specifies the points to be mapped. On output, it retums the mapped points.

npnts: Integer expression (input)
This parameter specifies the number of points in *pnts* to be mapped.

Examples

```
------------------------------------------------

This exelmple demonstrates the use of MapTo and
MapFrom. It inserts a line in the current view, maps
the lines endpoints from the current view to a view
of your choice, and prints out the endpoints in the
view you chose.
```

```
proc main

   real Transform(15)
   coord EndPnts(2)
   integer CurView, WhichView

   SysVarI(4, CurView)
   EndPnts(1) = [0,0,0]
   EndPnts(2) = [1,0,0]
   insert line ends(EndPnts(1), EndPnts(2)) \
                          view(CurView) rpnt(true)
   print 'In this view the line runs from '
   print EndPnts(1),' to ',EndPnts(2)

   accept WhichView newline \
              prompt('Enter an existing view #: ')

   GetView( CurView, Transform(1))
   MapFrom( Transform(1), EndPnts(1), 2)
   GetView( WhichView, Transform(1))
   MapTo( Transform(1), EndPnts(1), 2)
   print 'In view ',WhichView,' the line runs',
   print 'from ',EndPnts(1),' to ',EndPnts(2)

end proc
```

MaPix2

Type

Intrinsic Procedure Geometric

Purpose

Maps pixel space coordinates to view space coordinates.

Syntax

MaPix2*(ixy(1), xy)*

Parameters

ixy: Integer array of 2 elements (input/output)
This parameter specifies the pixel space coordinates to convert to view space coordinates. Elernent one is the X value; elernent two is the Y value.

xy: Coordinate variable (input/output)
This parameter returns the view space coordinates. The Z value represents the currently selected depth.

MaPix2N

Type

Intrinsic Procedure Geometric

Purpose

Maps an array of points frorn pixel coordinates to current view coordinates.

Syntax

MaPix2N*(ixy(1), pnts(1), npnts)*

Parameters

ixy: Integer array of *npnts**2 elements (input/output)
This parameter specifies the points to be mapped from pixel coordinates. The pixel coordinates are in pairs. The first is the pixel X coordinate and the second is the pixel Y coordinate.

pnts: Coordinate array of *npnts* elements (input/output)
This parameter returns the points mapped to the currently selected view.

npnts: Integer expression (input)
This specifies the number of points in *pnts* to be mapped.

MapMCPL

Type

Intrinsic Function Geometric

Purpose

Maps a point from model space coordinates to CPL space coordinates and returns the CPL space coordinates. The currently selected CPL is used. If no CPL is selected, the value of pnt is returned.

Syntax

MapMCPL*(pnt)*

Parameters

pnt: Coordinate expression (input)
This parameter specifies the point to be mapped.

MapMV

Type

Intrinsic Function Geometric

Purpose

Maps a point from model space coordinates to view space coordinates. The currently selected view is used. This function returns a coordinate value.

Syntax

MapMV*(pnt)*

Parameters

pnt: Coordinate expression (input)
This parameter specifies the point to be mapped.

MapTo

Type

Intrinsic Procedure Geometric

Purpose

Maps a set of points from the model coordinate System to a new coordinate system represented by the transform parameter. **MapTo** is a generalized version of **MapMV**. (The latter routine specifically maps points from model to view coordinates.)
MapTo and **MapFrom** are inverse functions. The points may be restored to their previous coordinate system via a call to **MapFrom**. It performs the inverse operation of **MapTo**.
The transform parameter may be obtained from intrinsics such as **Mat3P**, **RotMat**, or even **GetView** or **GetCPL**.

Syntax

MapTo*(transform(1), pnts(1), npnts)*

Parameters

transform:	Real array of 15 elements (input/output) This specifies the transformation matrix used to map the points. Only the first nine elements of transform are used.
pnts:	Coordinate array of *npnts* (input/output) On input, this parameter specifies the points to be mapped. On output, it returns the mapped points.
npnts:	Integer expression (input) This specifies the number of points in *pnts* to be mapped.

Examples

```
-----------------------------------------------

-- This example demonstrates the use of MapTo and
-- MapFrom. It inserts a line in the current view,
-- maps the lines endpoints from the current view
-- to a view of your choice, and prints out the
-- endpoints in the view you chose.

proc main

   real Transform(15)
   coord EndPnts(2)
   integer CurView, WhichView

   SysVarI(4, CurView)
   EndPnts(1) = [0,0,0]
   EndPnts(2) = [1,0,0]
   insert line ends(EndPnts(1), EndPnts(2)) \
                          view(CurView) rpnt(true)

   print 'In this view the line runs from '
   print EndPnts(1),' to ',EndPnts(2)

   accept WhichView newline \
              prompt('Enter an existing view #:')

   GetView( CurView, Transform(1))
   MapFrom( Transform(1), EndPnts(1), 2)
   GetView( WhichView, Transform(1))
   MapTo( Transform(1), EndPnts(1), 2)
   print 'In view ',WhichView,' the line runs',
   print 'from ',EndPnts(1),' to ',EndPnts(2)

end proc
```

MapTT

Type

Intrinsic Procedure Geometric

Purpose

Maps points from one coordinate system to another coordinate system. Coordinate systems can be defined by the transformation for views or construction planes- see **GetCPL, GetView,** Mat3P and RotMat for more information. This procedure is equivalent to:

MapFrom*(transform1(1), pnts(1), npnts)*
MapTo*(transform2(1), pnts(1), npnts)*

Syntax

MapTT*(transform1(1), transform2(1), pnts(1), npnts)*

Parameters

transform1:	Real array of 15 elements (input/output) This specifies the transforrn frorn which to map points.
transform2:	Real array of 15 elements (input/output) This specifies the transform to which to map points.
pnts:	Coordinate array of npnts elements (input/output) On input, this parameter specifies the points to be mapped. On output, it returns the mapped points.
npnts:	Integer expression (input) This specifies the number of points in *pnts* to be mapped.

MapVM

Type

Intrinsie Function Geometric

Purpose

Maps a point from view space coordinates to model space coordinates. This function returns the model space coordinates. The currently selected view is used.

Syntax

MapVM*(pnt)*

Parameters

pnt: Coordinate expression (input)
This parameter specifies the point to be mapped.

Mat3P

Type

Intrinsic Procedure Geometric

Purpose

Produees a transformation matrix for a coordinate system from three points in space.
The coordinate system is defined as follows: the X-axis is defined by the vector form pntl parameter to pnt2 parameter. The Y-axis is defined by a line perpendicular to the X-axis and passing through the pnt3 parameter. The Z-axis is perpendicular to the XY plane. The positive Z direction is defined by the right hand rule.

Syntax

Mat3P*(pnt1, pnt2, pnt3, transform (1))*

Parameters

pnt1, pnt2, pnt3: Coordinate expressions (input) These specify the points used to create the transformation matrix.

transform: Real array of 15 elernents (input/output)
This parameter returns a view transformation matrix for the coordinate system which is defined by three points.

Max

Type

Intrinsic Function Arithmetic

Purpose

Returns the value of the largest integer, integer4, real, or string parameter. The number of parameters allowed is unlimited but they must all have the same data type.
NOTE: Strings are compared in the following way: all strings are extended to the length of the largest string with blanks (ASCII character 32 in decimal). Strings are then compared character by character by their ASCII value. Lower ASCII values are considered to be less than higher ASCII values; i.e. A < B< C.

Syntax

Max*(expr1, expr2,...)*

Parameters

Expr1: Integer, integer4, real, or string expression (input)
This parameter specifies the first integer, integer4, real, or string expression.

expr2: Integer, integer4, real, or string expression (input)
This parameter specifies the second integer, integer4, real, or string expression.

Examples

```
A = Max(X,Y,Z)

Largest = Max(i, j, 10, k, 1 x 2)
```

MemAvail

Type

Intrinsic Function Operating System

Purpose

Returns the number of 128 byte blocks available on the UPL data stack. The data stack contains local variable data, parameters, procedure, and function return addresses. This function returns an integer.

Syntax

MemAvail()

Parameters

The MemAvail function has no parameters.

MenuCmd

Type

Intrinsic Procedure User Interface

Purpose

Sends an on-screen menu command string to the menu command processor. Refer to the Personal Designer and mierodraft Revision 6.0 User Reference Guide, for a description of the menu commands.

Syntax

MenuCmd*(menucmdstr)*

Parameters

Menucmdstr: String expression (input)
This specifies the menu command string to send to the menu command processor. The "\" character usually found in menu files cannot be used in the menucmdstr parameter. The above remark has one exception, that is if it is used in a string with the P command.

Example

```
--turn off menu 17, put up menu 18.

MenuCmd('M17-M18+')
```

MibTag

Type

Intrinsic Procedure Database Access

Purpose

Returns a unique entity tao, value, given the MIB number of an entity. If the entity has no tag, the tagstr parameter is returned with a length of zero.
There are two parts to an entity tag. The first part is the entity tag value. This is an integer which starts at zero and goes to values over four million. UPL cannot support such a large number as an integer, or integer4 so the number's equivalent representation is returned in a string of 10 characters. An entity can have only one entity tag value at a time.
The second part of an entity tag is the tag field. It is also a string of characters and it can hold any information associated with the tagged entity. See SetTagField and GetTagField for more information on tag fields.

Syntax

MibTag*(mib, tagvalstr)*

Parameters

mib:	Integer4 expression (input) This parameter specifies the MIB number of the entity to get the tag value for.
tagvalstr:	String variable of 10 characters (input/output) This returns the tag value.

Min

Type

Intrinsic Function Arithmetic

Purpose

Returns the value of the smallest integer, integer4, real, or string parameter. The number of parameters allowed is unlimited but they must all have the same data type.
NOTE: Strings are compared in the following way: all strings are extended to the length of the largest string with blanks (ASCII character 32 in decirnal). Strings are then compared character by character by their ASCII value. Lower ASCII values are considered to be less than higher ASCII values; i.e. A < B< C.

Syntax

Min*(exprl, expr2,...)*

Parameters

expr1: Integer, integer4, real, or string expression (input)
This parameter specifies the first integer, integer4, real, or string expression.

expr2: Integer, integer4, real, or string expression (input)
This parameter specifies the second integer, real, or string expression.

Examples

```
B = Min(r, s, t)

smallest = (1, m, 2, n, 5 * i)
```

MirEnt

Type

Intrinsic Procedure Database Access

Purpose

Mirrors the entities about a given plane. The plane is defined by a transformation matrix which can be generated by calling **Mat3P** or similar functions.

Syntax

MirEnt*(miblist(1), nent, transform(1))*

Parameters

miblist: Integer4 array of nent elements (input/output)
This parameter specifies the entities to be mirrored.

nent: Integer4 expression. Specifies the number of entities to be mirrored.

transform: Real array of 15 elements (input/output)
This specifies the transformation matrix that defines the plane on which to map the points. Elements 10, 11, and 12 specify the origin of the plane.

MirEntCopy

Type

Intrinsic Procedure Database Access

Purpose

Mirrors a copy of entities about a given plane. The plane is defined by a transformation matrix which can be generated by calling **Mat3P** or similar functions. The copied entities are added to the end of the database.

Syntax

MirEnt*(miblist(1), nent, transform(1))*

Parameters

miblist: Integer4 array of nent elements (input/output)
This parameter specifies the entities to be mirrored.

nent: Integer4 expression. Specifies the number of entities to be mirrored.

transform: Real array of 15 elements (input/output)
This specifies the transformation matrix that defines the plane on which to map the points. Elements 10, 11, and 12 specify the origin of the plane.

MirPnt

Type

Intrinsic Procedure Geometric

Purpose

Mirrors a set of points about a given plane. The plane is defined by a transformation matrix which can be generated by calling either the **GetCPL**, **GetView**, or **Map3P** procedures.

Syntax

MirPnt*(transform(1), pnts(1), npnts)*

Parameters

transfom:	Real array of 15 elements (input/output) This specifies the transformation matrix that defines the plane on which to map the points. Elements 10, 11, and 12 specify the origin of the plane.
pnts:	Coordinate array of *npnts* elements (input/output) On input, this parameter specifies the points to be mirrored. On output, it returns the mirrored points.
npnts:	Integer expression (input) This parameter specifies the number of points in the *pnts* array.

Modl

Type

Intrinsic Function Arithmetic

Purpose

Returns the modulo, (or remainder), of the num and divisor parameters. These parameters must be integers; the function also returns an integer.

Syntax

Modl*(num, divisor)*

Parameters

num: Integer expression (input)
This specifies the input number, or the dividend.

divisor: Integer expression (input)
This specifies the divisor for the input number.

Modl4

Type

Intrinsic Function Arithmetic

Purpose

Returns the modulo, (or remainder), of the num and divisor parameters. These parameters must be of type integer4; the function also returns an integer4.

Syntax

Modl*(num, divisor)*

Parameters

num: Integer4 expression (input)
This specifies the input number, or the dividend.

divisor: Integer4 expression (input)
This specifies the divisor for the input number.

Modify

Type

Statement Database Access

Purpose

Modifies existing entities in the drawing database. Only the current part file may be modified.
Entities are referenced by Master Index Block (MIB) numbers. The number identifies an entity and gives its location in the database. An MIB is assigned when an entity is inserted into the part database by Personal Designer or a UPL program. The number remains valid until the part is filed or exited with the pack database option.
After the **Modify** statement is executed, the system variable DBStatus is set. DBStatus gives the result of the insertion. See Appendix B, "System Variables," for more information.

Syntax

Modify enttype *entloc* entatts entdata **Rpnt***(bexpr)*

Keyword modifiers

enttype: Optional keyword that gives the type of entity to be modified. Replace enttype with one of the following keywords: **Line, String, Arc, Text or Point.** The enttype keyword must directly follow the **Modify** keyword.

entloc: Specifies which entity to modify. You must use an *entloc* keyword in this statement. The *entloc* keyword can be used in two ways; the one you use depends on whether you know the MIB number of the entity to be modified.
If you know the entity's MIB number, use this form for *entloc:*

EntId*(i4expr)*
Replace *i4expr* with an integer expression for the MIB number. You may find the MIB number by using the Verify

statement or by using intrinsic functions. Some intrinsics, such as GetEnt, allow the user to digitize entities in the graphics window. Their MIB numbers are then available to the program. Other functions, such as **FindProp and TagMib,** will return an MIB number when given non-graphical information such as the entity's properties or tags. It is recommended that the MIB number be obtained before using the **Modify** statement.

If you do not know the entity's MIB number, you may use one of the following keywords for entloc:

First
Modifies the first entity in the database. This will initialize the database search.

Next
Modifies the next entity in the database. This keyword allows the program to step through the database sequentially and modify each entity. Each time a **Modify Next** statement is executed, the next entity in the database with the matching enttype is modified. If no enttype is given, any entity is matched. A **Modify Next** statement may also be used after an Entld(i4expr) statement. The database search will then start at the i4expr entity instead of the first entity.

Last Modifies the last entity in the database. This keyword allows the program to modify the last entity inserted into the database without searching the database from the beginning.

If the type of entity specified by the enttype keyword does not match the type found using the entloc keyword, the DBStatus variable is set to three. When the end of the database is reached, DBStatus is set to two. See Appendix B, "System Variables," for more information.

entatts: Optional keywords that let you change the data of an entity. You can use the following entatts keywords with all entity types:

Color*(iexpr)*
where *iexpr* gives the new color number for the modified entity.

Font*(iexpr)* where *iexpr* gives the new font number for the modified entity.

Group*(iexpr)* where *iexpr* gives the modified group number of inserted entity.

Layer*(iexpr)* where *iexpr* gives the layer for the modified entity.

Vvis*(iexpr)* where *iexpr* gives the new view number that the modified entity is visible in,

entdata: Optional keywords that provide the data to be modified for a specified enttype. The keywords for each enttype are:

For Lines:

Ends*(cexpr1, cexpr2)*
Specify the endpoint coordinates after the keyword Ends. Replace cexpr1 with end one and cexpr2 with end two of the line. Both are coordinate expressions which represent model space coordinates. The default for cexpr1 and cexpr2 is [0.0,0.0,0.0].

For Strings:

Verts*(iexpr1, carray(iexpr2))*
Specify the coordinates for the vertices of a string in carray. Replace carray with the name of a coordinate array which contains the model space coordinates for the string vertices. Replace iexpr1 with the number of vertices in carray, and iexpr2 with the first element of carray to be used.

For Arcs:

Specify what you want to change after one of the following keywords:

Org*(cexpr)* Replace *cexpr* with the new model space origin of the modified arc.

Radius*(rexpr)*
Replace *rexpr* with the new radius of the arc. Arcs are drawn counterclockwise.

AB*(rexpr)*Replace *rexpr* with the new beginning angle of the modified arc in degrees.

AE*(rexpr)* Replace *rexpr* with the new ending angle of the modified arc in degrees.

ArcEnds*(cexpr1, cexpr2)*
If this keyword is given, it replaces **AB** and **AE**. Either **AB** and **AE** or ArcEnds determine the beginning and ending angles of the arc. The arc is drawn such that a line from the origin (as specified by Org) to cexprl determines the beginning angle of the arc, and a line from the origin (as specified by Org) to cexpr2 determines the ending angle of the arc. The arc has a radius specified by **Radius** above. There is no default for **ArcEnds**; values for **AB** and **AE** are used to find the default beginning and ending angles of the arc.

For Text:

Specify the new characteristics of the text to be modified after one of the following keywords:

Ang*(rexpr)*
Replace *rexpr* with the new angle for the text.

Hgt*(rexpr)*
Replace *rexpr* with the new text character height.

Just*(iexpr)*
Replace *iexpr* with the new text i ustification code:
1 - left justification
2 - right justification
3 - center j ustification

Lnsp*(rexpr)*
Replace *rexpr* with the new text line spacing factor.

Org*(cexpr)*
Replace cexpr with the new text origin in model space.

Txt*(sexpr)*
Replace *sexpr* with a string expression that gives the new text.

Wdt*(rexpr)*
Replace *rexpr* with the new text character width.

For Points:

Loc*(cexpr)*
Replace *cexpr* with the new model space coordinate of the point to be modified.

For all entity types:

Rpnt*(bexpr)*
Optional clause. Replace *bexpr* with the Boolean expression after the keyword **Rpnt**. If *bexpr* evaluates to true, then the entity is repainted after it is modified. Otherwise, it is not.

Example

```
MODIFY LINE ENDS(C1,C2) ENT ID(MIB NUM) COLOR(IC)\
                                        RPNT(TRUE)
```

ModR

Type

Intrinsic Function Arithmetic

Purpose

Returns the modulo, (or remainder), of the *num* and *divisor* parameters. These parameters must be real numbers; the function will also return a real number.

Syntax

ModR*(num, divisor)*

Parameters

num: Real expression (input)
This specifies the input number, or dividend.

divisor: Real expression (input)
This specifies the divisor for the input number.

MouseInp

Type

Intrinsic Procedure User Interface

Purpose

Returns information about input from the pointing device. It may be called in a loop to track mouse movements and check for presses to mouse buttons. The procedure returns immediately after the call and does not wait for input.

Syntax

MouseInput*(digchar, moved, locpix, locdev)*

Parameters

digchar: Integer variable (input/output)
Returns the ASCII value of the button(s) pressed on the input device. This is sent from the device driver back to Personal Designer. A value of 1 indicates a button has been pushed. All other values are device dependent.

moved Boolean variable (input/output)
Returns a boolean value telling whether the mouse has moved from the coordinates specified in locpix.

locpix Integer array of 2 elements (input/output)
On output, specifies the latest mouse position in pixel coordinates. This is clipped to the graphics device page coordinates, not the graphics window coordinates. See Chapter 3, Functional Listing, under Window Input/Output Intrinsics for more information.

locdev Integer4array of 2 elements (input/output)
Returns the latest mouse position in input device coordinates. These depend upon the type of input device and its current resolution settings. On input, if *locdev*(1) is <0, the system uses the current rnouse position as the initial position. If you set *locdev* to -1, this tells the system to start tracking the mouse at its current position. On input, if *locdev* is not -1 (or <0) then use the location specified by *loedev* as the mouse's starting point.

MovEnt

Type

Intrinsic Procedure Database Access

Purpose

Moves a set of entities from one location to another. With **MovEnt,** only the location changes. The graphics of the entity's previous location are erased.

Syntax

MovEnt*(miblist(1), nent, deltaxyz(1))*

Parameters

miblist:	Integer4 array of *nent* elements (input/output) This parameter specifies the MIB numbers of the entities to be moved.
nent:	Integer4 expression (input) This specifies the number of entities in miblist.
deltaxyz:	Real array of 3 elements (input/output) This parameter specifies the amount to move the entity in each direction of X, Y and Z respectively. In a transformation matrix, these deltas are stored in elements 10, 11, and 12.

MovEntCopy

Type

Intrinsic Procedure Database Access

Purpose

Copies a set of entities and moves it to a new location. The only difference between the new entities and their copies is the location. The new entities are added to the end of the database.

Syntax

MovEntCopy*(miblist(1), nent, deltaxyz(1))*

Parameters

miblist: Integer4 array of *nent* elernents (input/output)
This parameter specifies the MIB numbers of the entities to be moved.

nent: Integer4 expression (input)
This is the number of entities in miblist.

deltaxyz: Real array of 3 elements (input/output)
This parameter specifies the amount to move the entity in each direction of X, Y and Z respectively. In a transfonnation matrix, these deltas are stored in elements 10, 11 and 12.

NullTransform

Type

Intrinsic Procedure Geometric

Purpose

Returns the null transformation matrix. This function should be used to initialize a transformation matrix before using the matrix in a UPL program. The values returned by the procedure are:

The null transform values are:

1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0

The X, Y, Z offset values are:

0.0 0.0 0.0

The X, Y, Z scaling values are:

1.0 1.0 1.0

Syntax

NullTransform*(transform(1))*

Parameters

transform: Real array of 15 elements (input/output)
This parameter returns the null transformation matrix to be initialized. After this procedure is executed, this matrix will contain the values listed above.

NumToCntrl

Type

Intrinsic Function Data Conversion

Purpose

Converts characters in the form #ascii num# to ASCII control characters and returns the new string value. This is the same format used by Personal Designer and the UPL compiler. See CntrlToNum for more inforrnation. For a complete list of the ASCII character set, see Appendix F, "ASCII Character Set."

Syntax

NumToCntrl*(str)*

Parameters

str: String expression (input)
This parameter specifies the string containing the character sequence #ascii num#. Characters that are not in this form are not changed.

Open

Type

Statement Input/Output

Purpose

Opens a file for reading and/or writing. If the file you open does not exist, a new file is created.
UPL has two kinds of files: text and binary. There are also two kinds of file access: sequential and random. The default is a sequential access text file.
lt is easier to create and verify data in a text file, so you should try to use text files in most of your UPL programs. A text file may be examined or created by any program or command capable of reading or writing a text file; for example, a text editor. However, if the UPL programs will read and write files for programs other than Personal Designer, you may have to use binary files. This should allow your files to be compatible with the other programs.
All files have afile pointer. This is the position in the file at which the next Read or Write will occur. The file pointer is moved about differently for sequential and random access files.

Syntax

Open *flvar fn* **Binary Reclen***(iexpr)*

Keyword modifiers

flvar: File variable. See the File declaration statement for rnore information.

fn: String expression which gives the name of the file to be opened.

Binary: Optional keyword. Specifies the kind of file to be opened. If you do not use the **Binary** keyword, the file will be a text file. A text file is made up of ASCII characters. These characters are the ASCII representation of integer, real, coordinate, Boolean, or string data.

Text files are divided into lines. The lines are separated by an end-of-line sequence. For DOS these are the carriage return (ASCII 13) and the line feed (ASCII 10) characters together. On UNIX, it is just the line feed character (ASCII 10). Text files may end with an end-of-file mark. On DOS, the end-of-file mark is ^Z (ASCII 26); on UNIX, it is AD (ASCII 4). Do not read or write these special characters as part of your data: the **Read** and **Write** statements will do this for you. The **Read** and **Write** statements will transfer data and automatically convert data between the ASCII character format in the file and the internal data storage format of the variable. For more information, see the **Read** and **Write** statements, Appendix F, "ASCII Character Set," and Appendix E, "Internal Data Format."

Ifyou do use the **Binary** ke.yword, the file will be a binary file. A binary file is made up of data in the internal storage format for integer, real, string, Boolean, and string data. There is no line separation in a binary file. It is just a series of bytes. The **Read** and **Write** statements merely transfer the appropriate number of bytes between the file and the program's variables.

Reclen: Optional keyword. Specifies the method of file access.

Ifyou do not use the **Reclen** keyword, the program will use sequential access. Sequential access starts at the beginning of the file and reads or writes data at the file pointer. The file pointer is then advanced. The file pointer cannot move backward. It may, however, be set to the beginning of the file. This can only be done by closing the file and opening it again. For more information see the **Close** statement.

If you do use the **Reclen** keyword, the file will use random access. A random access file may position the file pointer to the beginning of any record within the file. Setting the file variable's POSITION attribute sets the file pointer. The file pointer may be moved to a position within a record using the Read statement.

The record length is also set by the **Reclen** keyword . It may be checked using the file variable's.RECLEN attribute. Replace *iexpr* with an integer expression for the record length in bytes. In text files, the record length must include the end-of-line sequence. Thus, add 2 to the record length when running under DOS; add 1 to the record length when running under UNIX. **Reclen** only affects how much the file pointer is moved; it does not affect how much data is read.

The file pointer may also moved using the POS4 attribute. Setting this 4-byte integer value moves the file pointer to the specified offset from the beginning of the file. It is not effected by the POSITION attribute or the record length specified in the **Reclen** keyword.

At most, UPL can have four files open at one time. If UNDO is on, or if a journal file is on, this is reduced to three. In addition, some Personal Designer commands such as PLOT, EXEC, INSERT TFILE, and INSERT FIGURE require a file to execute. If you have used all available files and invoke one of these commands using the **Send** statement, your files may be damaged. These restrictions exist because DOS limits the number of files that can be open to 20, and Personal Designer uses 16 of these.

Examples

```
OPEN F1 FNAME BINARY RECLEN(32)

Open Txt "\DATA\PROPERTY.DAT"

string FileName:60
file FileVar
       :
       :
accept FileName prompt("Enter name of file: ")
open FileVar FileName
```

PageInfo

Type

Intrinsic Procedure Operating System

Purpose

Returns information about the graphics display device. For more information, see "Input/Output (Window) Intrinsics in the Functional Listing of Statements and Intrinsics chapter.

Syntax

PageInfo*(infono, pageno, pagedata(1))*

Parameters

infono: Integer expression (input)
This parameter speeifies the information to return about the graphics driver. The four input values are:

1 page type.
2 pixel width of characters.
3 pixel height of characters.
4 page size in pixels.

pageno: Integer expression (input)
This parameter speeifies the graphics device page to query. Values are:

1-6 These query pages 1-6. If the graphics device does not support a page, *pagedata* returns a zero.
-1 Returns information about the page containing the window most recently accessed by the **Accept**, **Display**, **Print**, or **Send** statements.
-2 Returns information about the page containing the graphics window.

pagedata: Integer array of 4 elements (input/output)
This parameter returns the page inforniation:

If infono = 1, *pagedata(1)* contains integer code telling what kind of information is on the page:

0 invalid page
1 alpha/text only
2 graphics only
3 alpha/text and graphics

If infono = 2, *pagedata(1)* holds the pixel width (X direction) for each character in alpha/text.

If infono = 3, *pagedata(1)* holds pixel height (Y direction) for each character in alpha/text

If infono = 4, *pagedata(4)* holds pixel coordinates of corners of the graphics device page:

pagedata(1) X coordinate of lower left corner
pagedata(2) Y coordinate of lower left corner
pagedata(3) X coordinate of upper right corner
pagedata(4) Y coordinate of upper right corner

Pi

Type

Intrinsic Function Trigonometric

Purpose

Returns the real value equal to 3.141593.

Syntax

Pi()

Parameters

The Pi function has no parameters.

PixToRowCol

Type

Intrinsic Procedure Input/Output (Window)

Purpose

Converts pixel coordinates to the corresponding row and column coordinates. This procedure can be used with **DerineAW** to convert the dimensions of your alphanumeric window. See **RowColToPix** for more information. For more information, see "Input/Output (Window) Intrinsics in the Functional Listing of Statements and Intrinsics chapter.

Syntax

PixToRowCol*(ixy(1), ipage)*

Parameters

ixy: Integer array of 4 elements (input/output)
On input, this parameter specifies the two X, Y pixel locations. On output, it returns the equivalent row/column position.
This parameter ean be used to represent the lower left and upper right corners of a window in the **DerineAW** procedure:

ixy(1) Left boundary.
ixy(2) Lower boundary.
ixy(3) Right boundary.
ixy(4) Upper boundary.

ipage: Integer expression (input)
This parameter specifies the graphies device page number for the conversion.

PntPrp

Type

Intrinsic Procedure Geometric

Purpose

Returns the projection of a point onto a line. It returns the intersecting point of the two imaginary lines. The first line passes through two points in space. The second line starts at a given point in space and intersects the first line so that the two lines are truly perpendicular. This point of intersection is the projection of the given point on the given line. All points must be given using the same coordinate system.

Syntax

PntPrp*(lend1, lend2, pnt, prppnt)*

Parameters

lend1: Coordinate expression (input)
This parameter specifies one end of the first line.

lend2: Coordinate expression (input)
This parameter specifies the other end of the first line.

pnt: Coordinate expression (input)
This parameter specifies the point to project onto the line specified by *lend1* and *lend2*.

prppnt: Coordinate variable (input/output)
This parameter returns the projection of *pnt* onto the line specified by *lend1* and *lend2*.

PntPrpV

Type

Intrinsic Procedure Geometric

Purpose

Returns the projection of a point onto a line in a given view. It returns the point of intersection of two imaginary lines seen as perpendicular in the given view. The first line passes through two points in space. The second line starts at a given point in space and interseets with the first line so that the two lines appear to be perpendicular when seen from the given view. This point of intersection is the projection of the point on a line in the given view.
The lines in this procedure may only appear to be perpendicular, whereas the lines in the PntPrp procedure are actually perpendicular. All points in space are given in mode] coordinates.

Syntax

PntPrpV*(viewno, lend1, lend2, pnt, prppnt)*

Parameters

viewno: Integer expression (input)
This parameter specifies the view number in which the *pnt* parameter appears to be projected.

lend1: Coordinate expression (input)
This parameter specifies one end of the first line.

lend2: Coordinate expression (input)
This parameter specifies the other end of the first line.

pnt: Coordinate expression (input)
This parameter specifies the point to project in the given view.

prppnt: Coordinate variable (input/output)
This parameter returns the projeetion of pnt onto the line given by *lend1* and *lend2* as it appears in the given view.

PolyArea

Type

Intrinsic Procedure Geometric

Purpose

Calculates the area of a polygon given by the vertices parameter. The polygon can be open or closed.

Syntax

PolyArea*(vertices(1), nvert, area)*

Parameters

vertices: Coordinate array of *nvert* elements (input/output)
This parameter specifies the vertices of the polygon. If the polygon is not closed, the line between the first and last vertex will act as the closing side.

nvert: Integer expression (input)
This parameter specifies the number of vertiees in the *vertices* parameter.

area: Real variable (input/output)
This returns the area of the polygon in squared database units.

Polywin

Type

Intrinsic Procedure Geometric

Purpose

Determines if a point lies within the boundaries of a polygon. The point and the polygon must exist in the same plane.

Syntax

PolyWin*(vertices(1), nver, pnttocheck, pntin)*

Parameters

vertices:	Coordinate array of *nvert* elements (input/output) This parameter specifies the vertices of a polygon. A maximum of 1.000 vertices may be specified.
nvert:	Integer expression (input) This parameter specifies the number of vertices in the *vertices* Parameter.
pnttocheck:	Coordinate expression (input) This parameter specifies the point to check.
pntin:	Boolean variable (input/output) This parameter returns true if *pnttocheck* is bounded by the polygon; otherwise it returns false. If a point is on a boundary defined by the vertices, or is the same as a vertex, *pntin* returns true.

Print

Type

Statement Input/Output (Window)

Purpose

Prints numerical or string expressions to a window on the screen.

Syntax

Print expr :f1:f2 expr:f1:f2...,

Keyword modifiers

expr: Optional expression. Replace *expr* with any resulting data type except file. After the expression is evaluated, the result is printed at the current cursor location in the Print window. This is the Personal Designer command window by default. To change the window use the **PrintWin** system variable to specify the window number. For more information, refer to **DefineAW** and Appendix B, "System Variables."

f1: Optional expression that can be used with *expr*. This lets you format the output of the expression by indicating the field width. Replace *f1* with the field width the *expr* value is to be printed in. The expression is printed rightjustified if the field width is positive and left justified if the field width is negative. Always choose a field width that will accommodate the length of the longest expression to be printed. If you try to print an expression that is longer than the given field width, the expression is truncated on the right. If no field width is given, the statement uses any field width that is necessary to print the value.

f2: Opüonal expression that can be used with *f1* to format the decimal spacing for real and coordinate values. Replace *f2* with the number of places to the right of the decimal point. The decimal point uses one decimal place. If *f2* is negative, the number is printed in exponential

form. Note that trailing zeros will be added to fill out the field if the value can be exactly expressed in fewer decimal places than specified by *f2*. If the number cannot be expressed in exactly *f2* decimal places, it will be rounded up. The default value is to print all significant digits and one trailing zero. Optional punctuation. If the last item in the **Print** statement is a comma, the cursor is left at the end of the current line. (i.e. no end-of-line sequence is printed.) Otherwise, it is moved to the beginning of a new line. An empty (no expression) **Print** statement moves the cursor to the beginning of a new line.

Examples

```
PRINT  'Select Option ':30,

PRINT A/(3.0+B):12:2, 'A string', B1 OR B2:10
```

Proc

Type

Statement Program Structure

Purpose

Declares the name of a user-defined procedure, as well as the name, data, aggregate, and storage types of the parameters. All statements between the **Proc** and **End Proc** keywords form the body of the procedure.

Syntax

Proc procname(parameterlist)

Keyword modifiers

procname: Declares the name of the user-defined procedure. Only the first 16 characters are used. Using procname as a statement in the program causes the procedure to be called. Variables or expressions enclosed in parentheses after the procedure call will be passed as parameters. See Chapter 2, Program Structure, for more inforrnation.

parameterlist: Contains parameter declarations. Any number of parameters may be declared, but they must be enclosed in parentheses if they exist. If there are no parameters, do not include parentheses. Parameter deelarations are equivalent to variable deelarations inside a procedure: their names are local to the procedure. However, parameter names may not be the same as any variables declared in the Group section. A parameter list takes the form of
mode datatype paramname.

mode
Parameter mode. It must be either **In** for input parameters or **InOut** for input/output parameters. The Initial default mode is input. The default mode becomes the most recently used mode thereafter.

paramname
Name of the parameter. Onty the first 16 characters will be used.

datatype:
Data type of the parameten It may be any of the UPL data types: Integer, Integer4, Real, Coord, String, Boolean, or File. There is no initial default data type, but, the most recently used data type becomes the default after the first parameter is declared.

Arrays may only be passed as input/output parameters. The array parameter's declaration must include the maximum subscripts enclosed in parentheses.

If a string parameter is declared as input, the maximum length must be given, preceded by a colon. If a string parameter is declared as input/output, the maximum length used is the maximum length of the string variable passed as a parameter to the procedure.

There are shortcuts for declaring parameters. If the mode and data type of the parameters has not changed, they may simply be separated by commas. If either the mode or data type changes, do the following:

1. separate the declarations with a ; or start a new line

2. list the new mode and data type if the mode changes or list the new data type only if the data type changes.

3. list the new parameter names separated by commas.

Example

For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Storage Format."

Process

Type

Statement Flow Control

Purpose

Allows other UPL programs to be executed inside a running UPL program.
When a Process statement is encountered, the flow of control passes from your program to another program. After this program is executed, the flow of control returns to your program and continues execution on the staternent immediately following the **Process** statement.
Data may be passed from the calling program to the called program with variables which are declared identically in the Group section of each program. These variables must be declared in the exact same order in each program. The offset of each shared variable from the beginning of the Group seetion must be the same. The offset is determined by the variable's data type and the order it is deelared. See the Group statement and Appendix E, "Internal Data Format," for more information.
If the new program uses more code than your current program, use **$CodeSize** to allocate enough memory for the new program. Note this directive is only effective if either program is larger than the default value set by the configurator.

Syntax

Process *sexpr*

Keyword modifiers

sexpr: Name of the UCD file to be executed. The UCD extension is automatically appended to the end of *sexpr* to complete the file name.

Examples

```
PROCESS "LESSON1"

PROCESS "MENU"+OPTION
```

Product

Type

Intrinsic Procedure Operating System

Purpose

Returns information about the software that is running the UPL program. The type of information returned includes product identification number, product version number, version number, and graphics device identification.

Syntax

Product*(proddata(1))*

Parameters

proddata: Integer array of 19 elements (input/output)
The list below shows the information proddata returns about the software.

proddata(1)	Product id code. If *proddata(1)* = 1 the software running the UPL program is Personal Designer. If *proddata(1)* =2 The software running the UPL program is microDraft.
proddata(2)	Personal Designer version number (500 means version 5.00).
proddata(3)	Database version.
proddata(4)	Graphics device driver id number.
proddata(5)	Graphics device driver version number.
proddata(6)	Graphics device driver size in bytes
proddata(7)	Input device driver id number.
proddata(8)	Input device driver version number.
proddata(9)	Input device driver size in bytes.

proddata(10)	Plot device driver id number.
proddata(11)	Plot device driver version number.
proddata(12)	Plot device driver size in bytes.
proddata(13)	current UPL program version number (i.e., the version of the UPL compiler that compiled the program).
proddata(14)	UPL interpreter version nurnber
proddata(15)	Operating System flag 1 =DOS 2 = DOS Extender 3 = UNIX
proddata(16)	CPU architecture flag 1 = SPARC 2 = Intel
proddata(17)	productusage 1 = demo 2 = production
proddata(18)	Surfacing flag. 0 = not installed 1 = installed
proddata(19)	Machining flag. 0 = not installed 1 = installed

PutCur

Type

Intrinsic Procedure Input/Output (Window)

Purpose

Puts the cursor in a specified row and column relative to the upper left corner of a window.

Syntax

PutCur*(iwin, row, col)*

Parameters

iwin: Integer expression (input)
This parameter specifies the window number to position the cursor in.

row: Integer expression (input)
This parameter specifies the row number in *iwin* to put the cursor. If row is zero, the cursor's row position does not change.

col: Integer expression (input)
This parameter specifies the column number in *iwin* to put the cursor. If col is zero, the cursor's column position does notchange.

RadDeg

Type

Intrinsic Function Data Conversion

Purpose

Converts a real expression from radians to degrees. This function returns a real value.

Syntax

RadDeg*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the real expression to convert.

Read

Type

Statement Input/Output

Purpose

Transfers data from a file to a variable in the program. The file must be opened with the Open statement.
Each **Read** operation is done in 3 steps:
1. Data is scanned from the file starting at the file pointer.
2. Data is interpreted and placed in the specified variables.
3. The file pointer is advanced to point to new data.
These steps are performed differently depending on the type of file, the data type of the variables, and the way the statements syntax is used.

Syntax

Read *flvar, var:iexpr,var:iexpr ....,*

Keyword modifiers

flvar: File variable that must have been opened using the **Open** statement. The file may have been opened as a text or binary file, and may use sequential or random access. See the **Open** statement for more information.

var: Optional expression. Replace *var* with a variable of any data type except file. It must be a variable, variable attribute, or array element. The *var* expressions must be separated by commas.
If the file is a **text file,** characters are scanned starting at the file pointer. The **Read** statement scans until it encounters a character which could not be in the text representation for *var*. See Appendix E, "Intemal Data Storage Format," for more information. The **Read** statement then converts these

characters to the equivalent value in var's data type and stores that value in *var*. The file pointer is then advanced to the next unscanned character. This process repeats for the next variable in the statement. After all variables in the statement have been read, the file pointer advances to the beginning of the next line. It ignores any data left on that line.

iexpr: Optional integer expression, to be used with *var*, that allows you to alternately specify how many characters to scan and convert. The *iexpr* expression allows you to set up a format in your text file. For example, you could read a file of numbers arranged in columns.

Replace *iexpr* with a field width which is the number of characters you want to scan, convert, and store in *var*. if *iexpr* is specified, but the **Read** statement encounters a character which would not convert to *var*'s data type, it stops scanning and convert the value there. A colon must precede *iexpr*.

Optional punctuation. If you want to leave the file pointer within a line, put a comma at the end of the **Read** statement. It leaves the file pointer just after the character last scanned. When the file pointer has reached the end of the line, the file attribute *flvar*.EOLN is set to true; the file pointer does not advance further. Any subsequent **Read** statement ending with a comma does not advance the file pointer. Any **subsequent Read** statement without a comma moves the file pointer to the beginning of the next line. For example, a statement of the form "READ *flvar*" moves the file pointer to the beginning of the next line and does not scan any characters.

If the **Read** statement tries to scan more variables than there is data on the line, the remaining variables receive default values as follows:

Data Type	Default Value
REAL	0.0
INTEGER	0
COORD	[0.0,0.0,0.0]
BOOLEAN	FALSE
STRING	"" (empty string)

If the file is a **binary file, the Read** statement scans and interprets the values in the file as being in the internal data storage format for binary data. See Appendix E, "Internal Data Format," for more information. No data type conversion is done. Each variable is given the value in the file starting at the file pointer and contained in the subsequent bytes. The number of bytes scanned depends on the variable's data type and its internal data storage format. The file pointer is then advanced to the next unscanned byte.

The field width *iexpr* and the comma at the end of the **Read** statement have no significance with binary files.

If random file access is used, the file pointer may also be repositioned using the *flvar*.POSITION or *flvar*.POS4 attribute. See the **Open** statement or Appendix B, "System Variables," for more information.

Examples

```
------------------------------------------------
proc main

integer i1, i2, i3, i4
integer iarr1(10), iarr2(10), iarr3(10), iarr4(I0)
string s1:16, s2:20, s3:20, s4:20
rea1 r1, r2, r3, r4, x1, x2, x3, x4
coord c1, c2, c3, c4
boolean b1, b2, b3, b4
       :
       :
open datafl filename
       :
read datafl, i1, s1, r1, c1, iarr1(5), b1, x1
       :
read datafl, i2, s2:16, r2, c2, iarr2(5):3, x2, b2
       :
read datafl, i3, s3, r3, c3, iarr3(5), x3, b3
       :
read datafl, i4:5, s4:20, r4:10, c4:15, iarr4(5):5, \
                                        x4:5, b4:1
       :
end proc
------------------------------------------------
```

If the file contained the following data (an underscore _ denotes a blank and the <CR><LF> denotes the end of line sequence):

```
33This_is_a_string12345.6[3.0,4.0,5.5]435T2.2<CR><LF>
33This_is_a_string12345.6[3.0,4.0,5.514352.2F<CR><LF>
33This_is_a_string12345.6[3.0,4.0,5.5]4352.2F<CR><LF>
33___This_is_a_string____12345.6___[3.0,4.0,5.51__4
35__2.2__F<CR><LF>
```

The first Read statement would produce the following results:

```
i1 = 33, s1 = 'This_is_a_string', r1 = 12345.6,
c1 = [3.0,4.0,5.51 iarr1(5) = 435, b1 = TRUE,
x1 = 2.2
```

In each case the characters in the data file imply a boundary between the variables' data. When the Read statement sees the "T"' it knows it must stop scanning characters for an integer. Note that the declared length of the string variable tells it when to stop scanning.

The second **Read** statement shows how an optional field width can help clarify the implied boundaries. This is necessary for data following strings. It is also necessary when integer and real data are adjacent to each other. In this example, variables s2 and j2 need a field width,

```
i2 = 33, s2 = 'This_is_a_string, , r2 =12345.6,
c2 = [3.0,4.0,5.5], iarr2(5) = 435, x2 = 2.2,
b2 = FALSE
```

The third **Read** statement shows what would happen without the field widths to clarify the boundaries. This is probably not what is wanted.

```
i3 = 33, s3 = 'This_is_a_string1234' , r3 = 5.6,
c3 = [3.0,4.0,5.5), iarr3(5) = 4352, x3 = 0.0,
b3 = FALSE
```

The fourth READ statement uses a field width for each of the variables. Since trailing blanks are ignored for all but string data, they may be used to pad out the fields of data. This would be the way to read data arranged in columns.

```
i4 = 33, s4 = 'This_is_a_string____' ,
r4 = 12345.6, c4 = [3.0,4.0,5.5], iarr4(5) = 435,
x4 = 2.2, b4 = FALSE
-------------------------------------------------

proc main
    :
integer a, b, c1 d, e
integer v, w, x, y, z, final
    :
open dfile datafile
    :
read dfile, a, b, c, d, e
read dfile, v,
read dfile
read dfile, w, x,y,

Read

read dfile, z
read dfile, final
read dfile
    :
end proc
-------------------------------------------------
```

If the file contained the following data:

```
5_10_15<CR><LF>
20_25<CR><LF>
30_35<CR><LF>
40<CR><LF>
<CR><LF>
<EOF>
```

The first READ yields the following: a = 5, b = 10, c = 15, d = 0, e = 0. The second READ makes v = 20 and leaves the file pointer between the 20 and the 25 on the second line. The third READ moves the file pointer to the beginning of the third line. The fourth READ makes w = 30, x =35, y = 0, and sets dfile.EOLN to true. The fifth READ makes z = 0 and moves the file pointer to the beginning of the next line. The sixth READ makes final = 40 and moves the file pointer to the beginning of the last line. The last READ would skip the empty line and set dfile.EOF to true.

ReadCArray, ReadIArray, ReadRArray

Type

Intrinsic Procedure Input/Output (Window)

Purpose

Allows fast retrieval of integer, real, or coordinate data from a binary file. It is also useful for programs which need more than 32,767 bytes of data, the maximum amount which can be declared in a UPL program.
Your program may store large amounts of data in a file. This routine can be used to read the data into a buffer array, and then access and modify the data. The program can then write the data back to the file using the **WriteCArray, WriteIArray, and WriteRArray** intrinsic procedures.

Syntax

ReadCArray*(file, array(1))*
ReadIArray*(file, array(1))*
ReadRArray*(file, array(1))*

Parameters

file: File variable (input/output)
This parameter is the file variable for the data file. See the Open statement for more information on file IO. The file must be opened as a binary file. Sequential or random file access may be used. It is suggested that you read files written by the **WriteCArray, WriteIArray, and WriteRArray** intrinsic procedures, as using these procedures with other files is complicated.

array: Coordinate, integer, or real array of any length (input/output)
This parameter specifies the array which the program reads the data into. It should be declared to be large enough to hold all the data to read in one call to **ReadCArray, ReadIArray, and ReadRArray.** The amount of data read is determined by the number of elements declared in the array. Specifically, each call reads the number of bytes equal to the number of

elements in the array multiplied by the number of bytes per element. It must always be passed with a subscript of one, for example: array(1).

The program starts reading data at the file pointer and puts it into *array*. The file pointer is then placed immediately after the last byte read.

If you are using random file access, the file pointer may point to any array in the file. This is done by setting the file.POSITION or file.POS4 attribute.

Setting the file. POSITION attribute moves the file pointer to the position which equals the value of the file.POSITION attribute multiplied by the value of the file.RECLEN attribute. That is, the file.POSITION attribute tells the program what file record to point to. The file.RECLEN attribute specifies how many bytes are in the record.

If your program is reading a file whose arrays are all of the same size and data type, simply declare your record length to be the size of that array in bytes. This is done in the **Open** statement. Repositioning the file pointer is then simply a matter of setting file.POSITION to the array you want.

Setting thefile.POS4 attribute, places the file pointer at the given byte offset from the beginning of the file. (It is not effected by the file.POSITION or file.RECLEN attributes.)

If you are mixing arrays of different data types in the same file, you may find it easier to set the record length to 1 (using **Reclen keyword in Open** statement) and use the file.POS4 attribute.

When calculating file.POS4, take into account the difference in array element sizes. That is, a real element takes up as much as two integer elements and, a coordinate element takes six times as much as an integer element.

Since the file.POSITION attribute is itself an integer value, it can only be set as high as 32,767 bytes. Files are therefore limited to 32,767 * file.RECLEN bytes. If you want to read a larger file, use the file.POS4 attribute which will allow a byte offset of up to 2,147,483,647.

Examples

```
-----------------------------------------------
-- RRArray.upl
-- This program demonstrates use of ReadRArray.
-- The use of ReadIArray and ReadCArray are very
-- similar.
-- See WriteCArray, WriteIArray, WriteRArray for
-- the program WRArray.upl that will create an
-- appropriate file.
-----------------------------------------------
Proc Main

    integer I
    integer SavePos
    real RealBuffer(100)
    file DataFile

    -- Open the data file with the length of the
    -- data record: 400 = (4 bytes per real) * 100)

    open DataFile, 'File.Dat' binary reclen(400)

    -- Position the file pointer to some record
    -- in the file, say #25 and read it and print it.

    DataFile.POSITION = 25

    ReadRArray( DataFile, RealBuffer(1))

    loop I = 1 to RealBuffer(1).SIZE
        print RealBuffer(I),' ',
    end loop

end proc
```

```
------------------------------------------------
-- RXArray.upl
-- This program demonstrates use of ReadCArray,
-- ReadIArray, ReadRArray. It reads a file with
-- blocks of 1000 integers, 500 reals and 200
-- coord data in the same file. A 12 byte header
-- points to the beginning of each section. The
-- program uses the POS4 attribute for
-- positioning the file pointer. See WriteCArray,
-- WriteIArray, WriteRArray for the program
-- WXArray.upl which writes the data file.
------------------------------------------------
proc main

integer IntegerSize = 2
integer RealSize = 4
integer CoordSize = 12
integer HeaderSize = 12

-- Header information
integer4 StartIntegerData
integer4 StartRealData
integer4 StartCoordData

-- Data buffers
integer IntegerBuffer(100)
real   RealBuffer(50)
coord  CoordBuffer(20)

integer I
integer4 DataOffset
file DataFile

-- start of code –

open DataFile,'Data.fil' binary reclen(1)

-- Read in values for header.
-- Reset file pointer to beginning of file.

DataFile.POS4 = 0
read DataFile, StartIntegerData, StartRealData, \
                                        StartCoordData

-- Read in a buffer full random integer values
-- starting after integer value 47.
-- Print them out.
```

```
DataOffset = 47 * integer4( IntegerSize)
DataFile.POS4 = StartIntegerData + DataOffset

ReadIArray( DataFile, IntegerBuffer(1»

print 'IntegerBuffer='
loop I = 1 to IntegerBuffer(1).SIZE
    print IntegerBuffer(I),' ',
end loop
print

-- Read in a buffer full random real values
-- starting after real value 150.
-- Print them out.

Dataoffset = 150 * integer4(RealSize)
DataFile.POS4 = StartRealData + DataOffset

ReadRArray( DataFile, RealBuffer(1))

print 'RealBuffer='
loop I = 1 to RealBuffer(1).SIZE
    print RealBuffer(I),' ',
end loop
print

-- Read in a buffer full random coord values
-- starting after coord value 10.

Dataoffset = 10 * integer4(CoordSize)
DataFile.POS4 = StartCoordData + DataOffset

ReadCArray(DataFile, CoordBuffer(1))

print 'CoordBuffer='
loop I = 1 to CoordBuffer(1).SIZE
    print CoordBuffer(I),' ',
end loop

close DataFile

end proc
-------------------------------------------------
```

Real

Type

Statement Declaration

Purpose

Declares the name, data type, aggregate type, and initial value of a real variable. An initial value is optional. Array variables must be declared with their maximum subscripts.
Variables of real data type contain real number values in the ranges -1.0E+38 to -1.0E-37, 1.0E-37 to 1.0E+38, and 0.0.

Syntax

Real *rvar* = rconst | *rarray(iconst,...),...*

Keyword modifiers

rvar: Name of the real variable. Only the first 16 characters are used.

rconst: Optional literal or named real constant. This is the initial value for a scalar variable. If the variable is declared in the Group section, it will be set to this value once at the beginning of the program. If the variable is declared in a procedure or function, it will be set to this value each time the procedure or function is called.

rarray: Name of the real array variable. Only the first 16 characters are used.

iconst: Array subscripts. These declare the variable to have aggregate type array. Up to five subscripts may be declared. The subscripts must be enclosed in parentheses. Arrays may not be given an initial value.

All declaration statements must occur after the **Proc, Func, and Group** statements. They must appear before any other type of statements inside a procedure or function and are the only statements allowed inside the Group section.

For more information, see Chapter 2, Program Structure, and Appendix E, "Internal Data Storage Forrnat."

Examples

```
Real X, Y, Z

Real Delta = 0.000001, Diameter, Radius, Offset

Real Values(10,20,30)
```

Real

Type

Intrinsic Function Data Conversion

Purpose

Converts a Boolean, integer, integer4 or string expression to a real expression and returns the value. For Booleans, false = 0.0 and true = 1.0.

Syntax

Real*(expr)*

Parameters

expr. Boolean, integer, or string expression (input)
This parameter specifies the Boolean, integer, integer4, or string expression to convert.

Return (for Functions)

Type

Statement Flow Control

Purpose

Returns a value and passes flow of controi back to the calling procedure or function. The returned value may be used in an expression after control is passed to the calling function or procedure. Execution continues in the same statement the function call occurred in.

Syntax

Return *expr*

Keyword modifiers

expr: Replace with an expression of the same data type as defined by the **Return** keyword in the Func statement.

Examples

```
RETURN R

RETURN SQRT(X^2.0 + Y^2.0 + Z^2.0)

RETURN ANS = "Y" OR ANS = "y"

RETURN BOOL
```

Return (for Procedures)

Type

Statement Flow Control

Purpose

Returns control to the calling procedure or function from the current procedure. Execution continues in the calling procedure on the line immediately following the procedure call.

Syntax

Return When *bexpr*

Keyword modifiers

When *bexpr*. Optional expression. If you use the optional When keyword, the program returns only if *bexpr* evaluates to true. Otherwise execution continues in the current procedure on the statement immediately following this Return statement.

Examples

```
RETURN
```

```
RETURN WHEN I > 10 OR X = 0.0
```

RmvChr

Type

Intrinsic Function String Handling

Purpose

Removes all occurrences of the given characters from a string. This function returns a new string without the characters.

Syntax

RmvChr*(sexpr, setsexpr)*

Parameters

Sexpr: String expression (input)
This parameter specifies the string to remove the characters from.

Setsexpr: String expression (input)
This parameter specifies the charaeters to remove.

Example

```
-- This example will print: A string of characters.

Print RmvChr("A &Str!ing $ of vcharxacters.", \
                                        "$vx!&")
```

Rnd

Type

Intrinsic Function Arithmetic

Purpose

Returns a random number between 0.0 and 1.0. This function returns a real value.

Syntax

Rnd()

Parameters

The Rnd function has no parameters.

RotEnt

Type

Intrinsic Procedure Database Access

Purpose

Rotates the entities given using a transformation matrix. **RotMat** may be used to create the transformation matrix.

Syntax

RotEnt*(miblist(1), nent, transform(1))*

Parameters

miblist: Integer4 array of *nent* elements (input/output)
This parameter specifies the MIB numbers of the entities to be rotated.

nent: Integer4 expression (input)
This parameter specifies the number of entities to be rotated.

transform: Real array of 15 elements (input/output)
This parameter specifies the rotation of the entities. The first nine elements of the array are used to rotate the entities. Elements 10, 11, and 12 are the point about which the entities are rotated.

RotEntCopy

Type

Intrinsic Procedure Database Access

Purpose

Rotates a copy of the entities given using a transformation matrix. RotMat may be used to create the transformation matrix. The copied entities are added to the end of the database.

Syntax

RotEntCopy*(miblist(1), nent, transform(1))*

Parameters

miblist:	Integer4 array of *nent* elements (input/output) This parameter specifies the MIB numbers of the entities to be rotated.
nent:	Integer4 expression (input) This is the number of entities to be rotated.
transform:	Real array of 15 elements (input/output) This parameter specifies the rotation of the entities. The first nine elements of the array are used to rotate the entities. Elements 10, 11, and 12 are the point about which the entities are rotated.

RotMat

Type

Intrinsic Procedure Geometric

Purpose

Returns a transformation matrix. This matrix describes the rotation about an axis or vector. It describes the rotation only and not a point about which the entities are rotated. **RotMat** can provide the rotational transformation matrix for routines such as **RotEnt, RotEntCopy, and RotPnt.**

Syntax

RotMat*(rotangle, rotvec, transform(1))*

Parameters

rotangle: Real expression (input)
This parameter specifies the angle of rotation in radians. The positive direction is counterclockwise while looking along the *rotvec* vector.

rotvec: Coordinate expression (input)
This parameter specifies the vector to rotate about. You can define a vector from [0,0,0] to *rotvec*; or you can define it by subtracting the two coordinate values that represent points in space.

transform: Real array of 12 elements (input/output)
This returns the transformation matrix that will give the rotation described by the *rotangle* and *rotvec* parameters.

RotPnt

Type

Intrinsic Procedure Geometric

Purpose

Rotates a set of points using a transformation matrix.

Syntax

RotPnt*(transform(1), pnts(1), npnts)*

Parameters

transform:	Real array of 15 elements (input/output) This parameter specifies the rotation of the points. The first nine elements of the array are used to rotate the points. Elements 10, 11, and 12 are the location about which the points are rotated. The **RotMat** procedure may be used to create the transforrn array.
pnts:	Coordinate array of *npnts* elements (input/output) On input, this parameter specifies the points to be rotated. On output, it returns the rotated points.
npnts:	Integer expression (input) This parameter specifies the number of points in the *pnts* parameter.

RowColAW

Type

Intrinsic Procedure Input/Output (Window)

Purpose

Returns the size of an alphanumeric window as the number of rows and columns in the window. For more information, see "Input/Output (Window) Intrinsics" in the Functional Listing of Statements and Intrinsics chapter.

Syntax

RowColAW*(iwin, irow, icol)*

Parameters

iwin: Integer expression (input)
This parameter specifies the alpha window number to query. If *iwin* is greater than zero, the current size is returned. This could be smaller than the defined size if other windows with a higher priority overlay *iwin*, or if the on-screen menus are on. These cases cause the window to shrink.
If *iwin* is less than zero, the maximum possible window size is returned.

irow: Integer variable (input/output)
This parameter returns the number of rows in the window.

icol: Integer variable (input/output)
This parameter retums the number of columns in the window.

RowColToPix

Type

Intrinsie Procedure Input/Output (Window)

Purpose

Converts row and column coordinates to the corresponding pixel coordinates. This procedure can be used with the DerineAW procedure to convert the dimensions of your alphanumeric window. See the the **PixToRowCol** procedure and "Input/Output (Window) Intrinsics" in "Functional Listing, " Chapter 3. for more information.

Syntax

RowColToPix*(ixy(1), ipage)*

Parameters

ixy: Integer array of 4 elements (input/output)
On input, this parameter specifies two sets of row and column coordinates. On output, it returns the equivalent pixel coordinates.
This parameter may represent the lower left and upper right corners of a window for the DerineAW procedure:

ixy(1) Left boundary
ixy(2) Lower boundary.
ixy(3) Right boundary
ixy(4) Upper boundary.

ipage: Integer expression (input)
This specifies the graphics device page for the conversion.

RpntEnt

Type

Intrinsic Procedure Graphics

Purpose

Repaints entities on the graphics sereen. This procedure allows newly inserted or modified entities to be displayed. To be repainted, the entity must be on a layer or in a view of visibility which is selected on.

Syntax

RpntEnt*(miblist(1), nent, error)*

Parameters

miblist: Integer4 array of nent elements (input/output)
This parameter specifies the MIB numbers of the entities to be repainted.

nent: Integer4 expression (input)
This parameter specifies the number of entities in miblist. If *nent* is zero, all visible entities in the part are repainted. If you set *nent* to zero, you must give a dummy array with one element to the *miblist* parameter.

error: Integer variable (input/output)
This parameter retums the error condition:

0 no errors were found.

< > 0 a database error was found.

SclEnt

Type

Intrinsic Procedure Database Access

Purpose

Scales a set of entities by a specified amount.

Syntax

SclEnt*(miblist(1), nent, scale(1))*

Parameters

miblist: Integer4 array of *nent* elements (input/output)
This parameter specifies the list of MIB numbers of the entities that are to be scaled.

nent: Integer4 expression (input)
This parameter specifies the number of entities to be scaled

scale: Real array of 6 elements (input/output)
This array specifies a point to scale about and the X, Y, and Z scale factors. This array should have the same values as elements 13 through 15 of the transforination matrix. The scale parameter could be replaced with transform(13).

SclEntCopy

Type

Intrinsic Procedure Database Access

Purpose

Scales a copy of a set of entities by a specified amount.

Syntax

SclEntCopy*(miblist(1), nent, scale(1))*

Parameters

miblist: Integer4 array of *nent* elements (input/output)
This parameter specifies the list of MIB numbers of the entities that are to be scaled.

nent: Integer4 expression (input)
This parameter specifies the number of entities to be scaled.

scale: Real array of 6 elements (input/output)
This array specifies a point to scale about and the X, Y, and Z scale factors. This array should have the same values as elements 10 through 15 of the transforrnation matrix. The scale parameter could be replaced with transform(10).

Send

Type

Statement Input/Output (Window)

Purpose

Executes a Personal Designer command from within a UPL program by sending expressions to the Personal Designer command processor. After Personal Designer processes the expression, the UPL program continues.
The **Send** statement, is the easiest way to manipulate the database and graphics. A faster, but slightly more difficult, method is to use the database access statements **Erase, Insert, Modify, and Verify.** See these statements for more information. An even faster method is to use the Database Access intrinsic procedures. See Chapter 3, "Functional Listing," for more information.

Syntax

Send *expr: f1: f2,...,*

Keyword modifiers

expr: an optional expression of any data type except file. After the expression is evaluated, the result is sent to the Personal Designer command processor as a stream of characters. The syntax of the commands sent to the command processor is not checked for the correct Personal Designer command syntax. If the command is incorrect, the system rejects the command and continues executing the UPL program.

f1: Optional field width to be used with *expr*. It specifies the field width to set *expr* in. If the field width is positive, the expression is printed rightjustified. If the field width is negative, the expression is printed leftjustified.

f2: Optional expression to be used with *f1*. This expression formats the decimal places for real and coordinate values. Replace *f2* with the number of decimal places you want the value sent in. If *f2* is negative, the number is printed in exponential form.

When you run a program with the **Send** statement, you see the expressions displayed on the screen as they are sent. You can control the display of characters with the **Echo** statement. Use **Echo Off** to suppress the display and **Echo** On to resume the display after the Send statement is executed. Initially, **Echo** is set to **On**.

To change the window which the commands are echoed in, use the **SendWin** system variable to specify the window number. For more information, see **DefineAW** and the **SendWin** variable in Appendix B, "System Variabies."

When writing expressions for the Send statement, use quotes around each string, and commas to separate each expression. Verify the correct syntax for the command before sending it, since even one misplaced comma will prevent the command from being executed.

Before you use the Send statement in a UPL program, first execute an empty **Send** statement to send back the first character, which is ignored. This allows the UPL program to have control before Personal Designer requests user input.

The last character you send in a Send statement is stored in the **LastChar** system variable.

If the last item in the Send statement is a comma, then no <CR> is sent. Otherwise, it is.

It is possible to send part of a Personal Designer command in one **Send** statement and the remainder of the command in another **Send** statement. However, there are some statements and procedures which should not be used while only a portion of a Personal Designer statement has been sent. These include all database access statements and intrinsic procedures, and the user interface intrinsic procedures which allow access to the Getdata processor. See Chapter 3, "Functional Listing," for more information.

Many Personal Designer commands, such as INSERT TFILE, require access to files. If you have three or four files open and invoke these commands with the **Send** statement, you may receive an error message such as "VNP table file read/write error" or "Modifier file read/write error." You must close one of the open files temporarily to allow the command to be executed.

Examples

```
SEND 'INS LIN:X ',X,'Y ',Y,
SEND 'Z ',Z:10:3
SEND   -- Sends only a <CR>
```

SetBit

Type

Intrinsic Procedure Arithmetic

Purpose

Sets the value of a bit in the bittable parameter. This value is located at the offset specified by the *ibit* parameter. See the GetBit procedure for more information.

Syntax

SetBit*(bittable(1), ibit, ival)*

Parameters

bittable: Integer variable or array (input/output)
On input, this parameter specifies a table of bits. Each bit can have a value of zero or one. Each integer element in bittable can store 16 binary bit values. On output, it returns the bit table with the new bit value set.

ibit: Integer expression (input)
This parameter specifies the offset in the bittable(1) Parameter you want returned. Bit 0 is the least significant bit of the first integer in bittable.

ival: Integer expression (input)
This parameter specifies the value to set the bit to:

-1	The bit changes to the opposite of whatit currently is. This is an "exclusive OR" operation.
0	sets the bit to 0.
1	sets the bit to 1.

SetHelp

Type

Intrinsic Procedure User Interface

Purpose

Allows you to set up on-line help for a UPL program. With **SetHelp,** you can customize your help system to behave like the Personal Designer help system. See the HELP command in the Personal Designer and microDRAFT Revision 6.0 User Reference Guide for more information. Make a call to **SetHelp** before each call to the **AskModifiers, GetDig, GetEnd, or GetEnt** procedures. This ensures that your users can access the appropriate help screen any time they are prompted for data. See Appendix H, "Writing Personal Designer Commands," for more information.

Syntax

SetHeip*(helpfn, vnpindex, modifierindex, getdataindex)*

Parameters

helpfn: String Expression (input)
This parameter specifies the name of the help file to use to get the help information from. Before you exit your UPL program, you should use **SetHeip** with the *helpfn* parameter set to a blank string (""). This will cause the help system to use the default Personal Designer help file.

vnpindex: Integer expression (input)
This parameter specifies the verb/noun table index number. This is usually set to one for UPL program help files. See Appendix H, "Writing Personal Designer Commands", for more information.

You should set the verb/noun table index number even though you will not pass control of your UPL program to the verb/noun processor for a long period of time. All modifier

and Getdata processor help is associated with the verb/noun processor index, and the user may need help with these portions of the command as well as on the verb/noun portion of your customized command.

modifierindex: Integer expression (input) This parameter specifies the modifier tabie index number. This is usually set to one for UPL program help files.

getdataindex: Integer expression (input)
This parameter specifies the Getdata processor help index number.

SetLayer

Type

Intrinsic Procedure Graphics

Purpose

Sets the echo display of a given layer or all layers. If the layernumber parameter is zero, all layers are set. Otherwise only the single layer specified will be set. For more information, see GetLayer and the Personal Designer command ECHO LAYER in the Personal Designer and microDRAFT Revision 6. 0 User Reference Guide.

Syntax

SetLayer*(layernumber, howtosetlayer)*

Parameters

layernumber: Integer expression (input)
This parameter specifies the layer number to set. Input values are 1 through 256. A zero sets all layers.

howtosetlayer: Integer expression (input)
This parameter specifies how to set the layer:

- 0 turns layer off.
- 1 turns layer on.
- 2 turns off layers on, and on layers off.

SetMenuInfo

Type

Intrinsic Procedure User Interface

Purpose

Defines a new on-screen menu area or updates an existing one. An on-screen menu area may contain either an icon or an icon set. For more information, see the Personal Designer and microDRAFT Revision 6. 0 User Reference Guide.

The menu area to create or update is determined by areacorners. If this matches an existing area in setnumber, the menu area is updated with the new cmdstring; its menu area number is then returned in areanumber. If the area does not match, a new menu area is created in setnumber with *cmdstring*; this is assigned a new area number which is returned in *areanumber*.

Note that the menu area is limited to the be within the area defined for *setnumber*. Be aware that if you define a menu area outside of the *setnumber* area, the menu area will shrink to be within the *setnumber* area.

Syntax

SetMenuInfo*(areanumber, setnumber, areacorners, cmdstring)*

Parameters

areanumber:	Integer variable (input/output) On input, this parameter specifies the menu area number to be created or modified. On output, it specifies the newly created or modified menu area number.
setnumber:	Integer expression (input) This parameter specifies the icon set number of the new or modified area. The set number is also known as the layer number.
areacorners:	Integer array of 4 elements (input/output) This parameter specifies the lower left and upper right corners of the menu area.

cmdstring: String expression (input)
This parameter specifies the new or modified command string to be associated with the new area.

SetTagField

Type

Intrinsic Procedure Database Access

Purpose

Sets a tag field on an entity. If the entity does not have a tag, the procedure will automatically add one to it.

There are two parts to an entity tag. The first part is an entity tag value. Entities may have only one tag value. See **TagMib** and **MibTag** for more information.

The second part of an entity tag is the tag field. This is a text string associated with an entity tag. An entity may have many tag fields.

Syntax

SetTagField*(mib, fieldnumber,fieldstring)*

Parameters

mib: Integer4 expression (input)
This parameter specifies the MIB number of the entity to set the tag field for. If the entity does not have a tag, the procedure will automatically add one to it. The tag will be the next available number in the sequence. See the **GetTagField** procedure for more information.
An entity MIB number can be determined in several ways, one of which is with the **TagMib** procedure.

fieldnumber: Integer expression (input)
This parameter specifies the field number to set for the entity. Field zero cannot be set because it holds the system tag number. If there are less tags than specified by *fieldnumber*, the system fills in empty fields until it reaches the number specified in *fieldnumber*.

fieldstring: String expression (input)
This parameter specifies the text string to put into the field. There is a total of 996 bytes for all tag fields. Each tag field uses two bytes. Each character given in *fieldstring* uses one byte. Be sure you do not exceed this limit.

ShadeColor

Type

Intrinsic Function Graphics

Purpose

Returns an integer which is the color index number for a given color and shade. This value may be used in routines that use a color index number.

Syntax

ShadeColor*(color shade)*

Parameters

color: Integer expression (input)
This parameter specifies the color index number. Values are 0 through 7:

0 black.
1 red.
3 blue.
4 yellow.
5 cyan.
6 magenta.
7 white.

shade: Integer expression (input)
This parameter specifies the shade value. Values range from 1 through 127; 1 is the darkest and 127 is the lightest. Shade 64 is the brightest.
From 64 through 127, equal amounts of the two colors complimentaty to the given color are added on an increasing basis until the color reaches white.

Sin

Type

Intrinsic Function Trigonometric

Purpose

Returns the sine of an angle. This function returns a real value.

Syntax

Sin*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the angle in radians whose sine will be returned.

Size

Type

Intrinsic Function System Interface

Purpose

This intrinsie is superceded by Size4 for UPL version 5.0 and later. It is retained for compatibility with programs written under earlier versions. Returns the number of 128 byte blocks in a given file. This function returns an integer.

Syntax

Size*(sexpr)*

Parameters

sexpr: String expression (input)
This parameter specifies a string expression that is the name of a file. Zero is returned if the file does not exist or has a length of zero.

Size4

Type

Intrinsic Function System Interface

Purpose

Returns the number of bytes in a file. This function returns an integer4. It supercedes the intrinsic Size in UPL version 5.0 and later.

Syntax

Size*(sexpr)*

Parameters

sexpr: String expression (input)
This parameter specifies a string expression that is the name of a file. Zero is returned if the file does not exist or has a length of zero.

Sleep

Type

Statement Flow Control

Purpose

Causes a UPL program to "go to sleep" or wait in the background for a specified number of verb/noun processor (VNP) commands. No other UPL program may be run while a UPL program is in the sleep mode.

Syntax

Sleep *n, vnp,...*

Keyword modifiers

n: Optional expression. This specifies the number of Personal Designer commands to execute before the UPL program "awakes" or begins to run. If n is a negative one, the UPL program will not begin running until the Personal Designer AWAKE command is given. Note that the AWAKE command can also be used when n is not negative one. The default value for n is one.

vnp: Optional expression that can be used only if *n* is given. These are VNP numbers. If all the numbers are positive, only those commands may be input by the user. If any of the numbers are negative, all commands except the ones given will be accepted. The AWAKE command cannot be excluded.
VNP numbers can be determined for any Personal Designer command by using the Personal Designer command SELect MESSage HELP. If this command is used, the VNP number for each command is displayed before the command is executed. To turn off the display, use the Personal Designer command SELect MESSages NORMal.
You can also obtain VNP index numbers by looking at the file PDVNP.DEF, or by creating the file with the BLDF utility and using the following file as input:

```
BEGIN VNP
FILE \PD5\GCD3.VNP
DUMP PDVNP.DEF
END VNP
END
```

Examples

```
Sleep

Sleep 10

Sleep J*20

--allows only INS CIR, INS LIN,
--INS STG and REPAint

Sleep 5, 203, 2001, 2022, 1002

--allow all commands
--except INS CIR, INS LIN, INS STG and REPAint

Sleep K+2, -1002, -203, -2001, -2022
```

String

Type

Statement Declaration

Purpose

Declares the name, data type, aggregate type, and initial value of a string variable. An initial value is optional. Array variables must be declared with their maximum subscripts.

Variables of string data type contain strings of characters from the ASCII Character Set. See Appendix F, "ASCII Character Set," for more information.

Syntax

String *svar: iconst1 = sconst | sarray(iconst2,...): iconst1,...*

Keyword modifiers

svar: Name of the string variable. Only the first 16 characters are used.

iconst1: Maximum length of the string. This specifies how many characters can be stored in the string. This must be a literal or named integer constant.

sconst: Optional literal or named string constant. It is the initial value for a scalar variable. If the variable is declared in the Group section, it will be set to this value once at the beginning of the program. If the variable is declared in a procedure or function, it will be set to this value each time the procedure or function is called.

sarray: Name of the string array variable. Only the first 16 characters are used.

iconst2: array subscripts. These declare the variable to have aggregate type array. Up to five subscripts may be declared. The subscripts must be enclosed in parentheses. Array variables may not be given an initial value.

All declaration statements must occur after the **Proc**, **Func**, and **Group** statements. They must appear before any other type of statements inside a procedure or function and are the only statements allowed inside the **Group** section.

For iyiore information, see Chapter 2, "Program Structure," and Appendix E, "Internal Data Format."

Examples

```
String FileName:60

String CommandPrompt:18 = 'Type next command:'
String Answer:1

String NameTable(1000):16
```

String

Type

Intrinsic Function Data Conversion

Purpose

Converts real, coordinate, integer, integer4, and Boolean expressions to a string value and returns the value. The field width and the number of decimal places are optional; if used, they must be integer constants. If there are too many characters for the specified field width, characters are dropped from the right of the field.

Syntax

String*(expr: f 1: f2)*

Parameters

expr: Real, coordinate, integer, integer4 or Boolean expression (input). This specifies the real, coordinate, integer, integer4 or Boolean expression that is converted to a string.

f1: Optional expression that can be used with *expr*. This lets you format the output of the expression by indicating the field width. Replace *f1* with the field width the *expr* value is to be printed in. The expression is printed rightjustified if the field width is positive and left justified if the field width is negative. Always choose a field width that will accommodate the length of the longest expression to be printed.

If you try to print an expression that is longer than the given field width, the expression is truncated on the right. If no field width is given, the statement uses any field width that is necessary to print the value.

f2: Optional expression that can be used with *f1* to format the decimal spacing for real and coordinate values. Replace *f2* with the number of places to the right of the decimal point. The decimal point uses one decimal place. If *f2* is negative,

the number is printed in exponential form. Note that trailing zeros will be added to fill out the field if the value can be exactly expressed in fewer decimal places than specified by *f2*. If the number cannot be expressed in exactly *f2* decimal places, it will be rounded up. The default value is to print all significant digits and one trailing zero.

StrWide

Type

Intrinsic Procedure Database Access

Purpose

Returns a new set of string vertices at an offset from an existing set of vertices. The new set of vertices defines a "wide string."

Syntax

StrWide*(vertices(1), nvert, width, just, viewno, newveritces(1), newnverts)*

Parameters

vertices: Coordinate array of *nvert* elements (input/output)
This parameter specifies the existing set of vertices.

nvert: Integer expression (input)
This parameter specifies the nurnber of vertices in the *vertices* parameter. The maximum is 333 vertices.

width: Real expression (input)
This specifies the width of the wide string. This parameter is the offset from the existing vertices that will be used to generate the new set of vertices.

just: Integer expression (input)
This parameter specifies string justification:

-1 leftjustification.
0 center justification.
1 rightjustification.

A vector between the first and second vertices establishes the direction for justification.

viewno: Integer expression (input)
This parameter specifies the view transformation to use for generating the new set of points. The wide string will be generated in the X/Y plane of this view number. View one is model coordinates.

newvertices: Coordinate array of newnvert elements (input/output)
This parameter returns the wide string vertices.

newnvert: Integer variable (input/output)
This returns the number of vertices in the *newnvert* Parameter.

SqRt

Type

Intrinsic Function Arithmetic

Purpose

Returns the square root of a real expression. This function returns a real value.

Syntax

SqRt*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the real expression whose square root will be returned.

SysVarI

Type

Intrinsic Procedure Operating System

Purpose

Returns or sets the values of Personal Designer system variables whose data type is integer.

Syntax

SysVarI*(iexpr ival)*

Parameters

iexpr: Integer expression (input)
This parameter speeifies the system variable to return or set If its value is positive, the variable is returned in *ival*. If its value is negative, the variable is set to the value in *ival*.
The variables are shown in the list below. All of the variables can be read. Variables marked with an asterisk may be set as well. Only advanced users should set these system variables. Incorrect use could damage or destroy your part database. It is best not to change any variables you do not understand.Refer to the lists below for Personal Designer system variables and system colors.

ival: Integer array (input/output)
This parameter returns the values of the Personal Designer system variabies if *ivar* is positive. If *ivar* is negative, it specifies the input value. The size of the array depends on the value of the ivar parameter. The size of the array depends upon the information being returned or set

1 *	currently selected color
2 *	currently selected font
3	currently selected layer

4 currently selected view
5 no. of entities in part, including erased entities
6 on-screen menu on/off flag
7 echo command flag
0 = all echoing on
1 = equivalent to Personal Designer 'echo comm off
2 = equivalent to UPL 'echo off
8 journal file flag
9 menu no. of menu that cursor is currently over
10 current number of active icon boxes
11 current graphics cursor location returned as
X and Y pixel coordinates.
12 current CPL number returned; 0 is returned if no CPL
is active)
13 network flag
0 = no network active
1 = network active
14 drawing read only flag
0 = read only
1 = read/write

System Colors

101* digitize marker color.
102* window entity box color.
103* normal cross hair.
104* grid dot color for quadrants + + and --.
105* default entity color.
106* system text color.
107* user text color.
108* entity identification cross hair color.
109 * grid dot color for quadrants -+ and +-.

110* notused.
111* icon highlight color used by edit menu command.
112* notused.
113* on-screen icon menu cursor color.
114* message text color.
115* warning message text color.
116* error message text color.
117* cross-hatched X,Y coordinate color.
118* helptextcolor.
119 * general window text color.
120* menu icon highlight color.

1016 Read status of autosave on = 1, off = 0
1100 Read currently selected font number

1140 - 1155* A 16-byte buffer which may be used to pass data between UPL programs. This buffer must be accessed two bytes at a time, treated as integers. Other values may be stored by using based variables.
1217 - 1222* database header bit flags (bytes 144-127 of header)

1233* XH (crosshatch) solid fill flag
0 - normal, depends on pattern #
1 - XH solid fill off
2 - all XHs are solid filled
1242 maximum number of entities in the Active Entity Table
1249* beep control, 0 = on, 1 = off
1254 UNDO, 1 = on, 0 = off
1256 coordinate display, 1 = on, 0 = off
1286 last MIB number read in database search
1301 CPL indicator axes displayed, 1 = on, 0 = off

1302 MV number containing most recent entity pick (if multiple entity pick such as a WIN, MV number containing last entity found is returned)
1303 MVs, 1 = on, 0 = off
1311 MIB number of last entity that was stored in the memory portion of the display list
1312" Getdata angle lock, 1 = on, 0 = off
1313* Getdata color mask
1314* figure activate flag, 0 = no, 1 = yes, 2 = ask
1333* next available group number
1375 perspective, 1 = on, 0 = off
1379* Read and/or set/clear "SEL TEXT OFF/ON"; display text, 1 = on, 0 = off
1385 hard fonts, 1 = on, 0 off
1387* database pack flag, 0 no, 1 yes, 2 = ask
1474* visibility flag, 1 = VCON, 0 VALL
1511* Read and/or set/clear "SEL GRID ON/OFF'; grid on = 1, grid off = 0
1512* Read and/or set MIB number where next entity pick search will start.

Note that SysVarl variables 2001-2038 are dimensioning variables. They contain the values currently in effect in the system. This data is initialized when the part is opened by reading the MD subrecord of the Part Parameter Entity (PPE). When the part is filed, Personal Designer updates the PPE entity. Between part initialization and filing, use these SysVarl variables to obtain the correct values.

2001 * dimensioning arrow head type
2002* dimensioning precision
2003* dimensioning tolerance precision
2004* diameter dimensioning type
2005* suppress both witness lines? no 0, yes <> 0
2006* suppress first witness lines? no 0, yes <> 0
2007* suppress second witness lines? no = 0, yes <> 0
2008* display both witness lines? no = 0, yes <> 0
2009* VALL for dimensions? no 0, yes <> 0
2010* VCON for dimensions? no 0, yes <> 0

2011* point to point dimension? no = 0, yes <> 0
2012* horizontal dimension? no = 0, yes <> 0
2013* vertical dimension? no = 0, yes <> 0
2014* dimension arrows in? no = 0, yes <> 0
2015* dimension arrows out? no = 0, yes <> 0
2016* auto justify dimension text? no = 0, yes <> 0
2017* leftjustify dimension text? no = 0, yes <> 0
2018* center dimension? no = 0, yes <> 0
2019* no dimension centering? no = 0, yes <> 0
2020* prefix dimension text? no = 0, yes <> 0
2021* suffix dimension text? no = 0, yes <> 0
2022* no diameter symbol? no = 0, yes <> 0
2023* diameter symbol? no = 0, yes <> 0
2024* diameter symbol word? no = 0, yes <> 0
2025* align dimension? no = 0, yes <> 0
2026* do not align dimension? no = 0, yes <> 0
2027* ANSI dimensioning? no = 0, yes <> 0
2028* JIS dimensioning? no = 0, yes <> 0
2029* dimension feet mode? no 0, yes <> 0
2030* dimension inch mode? no 0, yes <> 0
2031* dimension tolerance type
2032* use comma in dimension numbers?
no = 0, yes <> 0
2033* use decimal point in dimension numbers?
no = 0, yes <> 0
2034* have trailing zeros in dimension numbers?
no = 0, yes <> 0
2035* do not have trailing zeros in dimension numbers?
no=O,yes<>0
2036* DIN dimensioning? no = 0, yes <> 0
2037* have leading zeros in dimension numbers?
no = 0, yes <> 0
2038* do not have leading zeros in dimension numbers?
no = 0,yes<>0

SysVarI4

Type

Intrinsic Procedure Operating System

Purpose

Returns or sets the values of Personal Designer system variables whose data type is integer4.

Syntax

SysVarI4*(iexp, i4val)*

Parameters

iexpr: Integer expression (input)
This parameter specifies the system value to return.
Currently there is only one Integer4 system value defined.
1 = number of entities in the part database.

i4val: Integer4 array (input/output)
This parameter returns the values of the Personal Designer system variable specified by *iexpr*.

SysVarR

Type

Intrinsic Procedure Operating System

Purpose

Returns the values of Personal Designer system variables whose data type is real.

Syntax

SysVarR*(rexp, rval)*

Parameters

rexp: Integer expression (input)
This parameter specifies the system value to return:

1 current screen scaie factor.

2 current screen extents in view space.

3 current construction depth.

4 current drawing extents in model space.

Additional values are listed below.

rval: Real array (input/output)
This parameter returns the value of the system variables. The size of the array depends on the value of *rexp*. Refer to the following system variable list.

When *rvar* = *1, rval (1)* screen scale factor

When *rvar* = 2,

rval (1) minimum X

rval (2) maximum X

rval (3) minimum Y

rval (4) maximum Y

When rvar = 3,	rval (1)	construction depth
When rvar = 4,		
	rval(1)	minimum X extent
	rval(2)	minimum Y extent
	rval(3)	minimum Z extent
	rval(4)	maximum X extent
	rval(5)	maximum Y extent
	rval(6)	maximum Z extent

Notes

Text, Dimension, and Grid Variables:
SysVarR variabies 1239-1241, 1252-1255, and 2000-2001 contain the values currently in effect in the system. This data is initialized when the part is opened by reading the MD subrecord of the Part Parameter Entity (PPE). When the part is filed, Personal Designer updates the PPE entity. Between part initialization and filing, use these SysVarR variabies to obtain the correct values.

Other System Variables

Note that all variables can be read. An asterisk indicates a variable can also be set..

1113	soft fonts scale factor
1118	trap size in screen inches
1119	Read and/or set digitize mark ('gleep') size in screen inches
1123	plotting point entity size in inches
1124	Read and/or set screen point entity size in inches
1158-1163	Read drawing extents X,YZ min and X,YZ max
1181 -1183	model space coordinates of perspective eye point
1184-1186	view space coordinates of perspective eye point
1187	perspective depth

1192*	Read and/or set ZOOM ALL border percent factor
1200	global scale factor
1223 - 1225*	Read and/or set X,YZ model space location of most recent digitize
1239*	Read and/or set default text entity line spacing
1240*	Read and/or set default text entity height
1241 *	Read and/or set default text entity width
1252 - 1253*	Read and/or set grid origin x,y
1254 - 1255*	grid increment x,y
2001 *	dimension angle
2002*	dimension text height
2003*	dimension text width
2004*	dimension arrow head size
2005*	dimension offset
2006*	dimension scale
2007*	dimension both tolerances
2008*	dimension positive tolerance
2009*	dimension minus tolerance
2010*	dimension tolerance text height
2011 *	dimension text line spacing
3001 - 3016*	soft font definition arrays. Each soft font definition contains 10 numbers, so a REAL array dimension to at least 10 must be used.

Part Extents:

SysVarR variables 1158-163 are the part extents. These values are not initialized until a ZOOM ALL or REGEN is executed. Until then, the program must directly access the EX subrecord of the PPE to obtain the part extents.

Other System Variables:

SysVarR variables 1113-1119, 1123-1124, 1158-1163, 1181-1187, 1192, and 1200-1225 are not stored in the part database. Instead they are initialized form the file PD.CFG but are not written back to it.

SysVarS

Type

Intrinsic Procedure Operating System

Purpose

Returns or sets the values of Personal Designer system variables whose data type is string.

Syntax

SysVarS*(ivar, sval)*

Parameters

ivar: Integer expression (input)
This parameter specifies the system value to return or set. If the number is positive the value is returned as speeified in the list below. If the number is negative, you can set the value specified in the list below. Currently only number 50 can be set; all the numbers can be returned:

1 - 20 = file names of device drivers, support, and temporary data files:

1	Graphics deviee driver.
2	Input device driver.
3	Initial plot device driver.
4	Low level 10 driver.
5	Verb/noun command table.
6	Prompts/message file.
7	Help file.
8	Macro definitions.
9	Initial tablet menu file.
10	Initial color/pen table.

11	Undo (OOPS) transaction file.
12	Start-up UPL program.
13	Initial text font definition.
14	Modifier words.
15	GetData command words.
17	MIB file name.
18	PDF file name.
19	Initial on-screen menu definition file.
20	Entity list file.
50	Current part file name; this can be set and returned using sval. It is the name the file will be written to if you file the part.
51	User/login name
52	Network name

sval: String variable (input/output)
This parameter retums the string system data if *ivar* is positive. It specifies the string value if *ivar* is negative. The size of the string depends on the value of the *ivar* parameter. The string may be up to 64 characters long.

Tan

Type

Intrinsic Function Trigonometric

Purpose

Returns the tangent of an angle. This function returns a real value.

Syntax

Tan*(rexpr)*

Parameters

rexpr: Real expression (input)
This parameter specifies the angle (in radians) whose tangent will be returned.

TagMib

Type

Intrinsic Procedure Database Access

Purpose

Returns the MIB number of a tagged entity.
There are two parts to an entity tag. The first part is an entity tag value. This is an integer which starts at zero and grows to over four million. An integer this size cannot be supported in UPL, so the number is specified as a string of 10 characters. An entity may have only one tag value.
The second part of an entity tag is the tag field. This is a text string associated with an entity tag. An entity may have many tag fields. See **SetTagField** and **GetTagField** for more information.

Syntax

TagMib*(tagvalstr,mib)*

Parameters

tagvalstr: String expression of 10 characters (input)
This parameter specifies the tag value to find an MIB number for.

mib: Integer4 variable (input/output)
This parameter retums the MIB number of the entity with the tag value. A zero is returned if no entity can be found with the given tag value.

TextColor

Type

Intrinsic Procedure Input/Output (Window)

Purpose

Sets the color of the text drawn in the alphanumeric windows. The Print, Accept (with a Prompt modifier), and Display statements use this color when displaying text.

Syntax

TextColor*(color)*

Parameters

color: Integer expression (input)
This parameter specifies the color number to use for the text. Values are 0 through 15 and 101 through 120. For color values 0 through 15, refer to the SELECT COLOR command in the Personal Designer and microDRAFT Revision 6. 0 User Reference Guide.
If *color* is greater than 100, the actual color number is obtained from the color table given in the configuration file. For example, if *color* = 106, the sixth value from the color table would be used as the color value. For color values 101 through 120 see SysVarl.

Time

Type

Intrinsic Function System Interface

Purpose

Returns the current system time in the format: HH:MM:SS.SS. This function returns a string value 11 characters long.

Syntax

Time()

Parameters

The Time function has no parameters.

Transpose

Type

Intrinsic Procedure Geometric

Purpose

Returns a matrix transposed about its diagonal elements. This procedure is useful for transposing a transformation matrix. See Appendix E, "Internal Data Format" for information on the Transformation matrix.

Syntax

Transpose*(transform(1))*

Parameters

transform: Real array of 15 elernents (input/output)
On input, this specifies the transforrnation matrix to be transposed. Only the first nine elements of *transform* are modified. On output, this parameter returns the transposed matrix.

TwoPi

Type

Intrinsic Function Trigonometrie

Purpose

Returns the real value equal to 6.283185.

Syntax

TwoPi()

Parameters

The TwoPi function has no parameters.

UpperCase

Type

Intrinsic Function String Handling

Purpose

Returns a string with all lower case letters to converted upper case letters.

Syntax

UpperCase*(str)*

Parameters

str: String expression (input)
This parameter specifies the string to convert. Uppercase letters will stay in uppercase. Any characters that are not letters will stay the same.

Example

```
String Command: 20
       :
Command = "INSert LINE:"

-- this will print: "INSert LINe:"
Print Command

-- this will print: "INSERT LINE:"
Print Uppercase (Command)
```

VCross

Type

Intrinsic Funetion Geometric

Purpose

Returns a vector (or cross) product of two vectors represented by coordinate expressions and returns a vector represented by a coordinate value equal to (*cexpr2* x *cexpr1*).

Syntax

VCross(cexpri, cexpr2)

Parameters

Cexpr1: Coordinate expression (input)
This parameter specifies the endpoint of a veetor (starting at [0,0,0]) that is to be cross multiplied.

cexpr2: Coordinate expression (input)
This parameter specifies the endpoint of the other vector (starting at [0,0,0]) that is to be multiplied.

NOTE: A coordinate value representing a vector is obtained by subtracting the coordinate of the starting point of a vector from its ending point. For example, if a vector is directed from coordinate CI to coordinate C2, it can be represented by the coordinate C2 - C 1.

VDot

Type

Intrinsic Function Geometric

Purpose

Returns a real equal to a dot (scalar) product of *cexpr1* and *cexpr2*.

Syntax

VDot*(cexpr1, cexpr2)*

Parameters

cexpr1: Coordinate expression (input)
This parameter specifies the endpoint of a vector (starting at [0,0,0]) that is to be dot multiplied.

cexpr2: Coordinate expression (input)
This parameter specifies the endpoint of the other vector (starting at [0,0,0]) that is to be dot multiplied.

NOTE: A coordinate value representing a veetor is obtained by subtracting the coordinate of the starting point of a vector from its ending point. For example, if a vector is directed from coordinate C 1 to coordinate C2, it can be represented by the coordinate C2 - C 1.

Verify

Type

Statement Database Access

Purpose

Returns information about existing entities in the current drawing database. Only the current file may have entities verified.
Entities are referenced by Master Index Block (MIB) numbers. The number identifies an entity and gives its location in the database. An MIB number is assigned when an entity is inserted into the part database by Personal Designer or a UPL program. The number remains valid until the part is filed or exited with the pack database option.

Syntax

Verify *enttype entloc entatts entdata*

Keyword modifiers

enttype: Optional keyword that specifies or returns the type of entity to be verified. If you know what type of entity you want to verify, replace enttype with one of the following keywords: **Line, String, Arc, Text** or **Point.** The *enttype* keyword must directly follow the **Verify** keyword. The **Verify** statement will only execute entities of the given type.

If you want to find out what type of entity is being verified, specify the following enttype keyword instead:

EntTyp*(ivar)*
Replace *ivar* with an integer variable which returns the following codes for the verified entity's type:

1 Line.
2 String.
3 Arc.
4 Text.

5 Point.

6 Linear dimension.

7 Label point dimension.

8 Radial dimension.

9 Angular dimension.

10 Cross-hatching.

11 Figure instance.

12 Diameter dimension.

13 Multiple view.

14 Ellipse.

15 Construction line.

16 Curve (cpole).

17 Surface (spole).

18 Plane.

30 NURB curve

31 NURB surface

35 3-D Tool path

36 2 1/2-D Tool path

145 Display image.

146 View.

147 Figure imagelist.

148 Extents.

entloc: Specifies which entity to verify. You must use an *entloc* keyword in this statement. The *entloc* keyword can be used in two ways; the one you use depends on whether you know the MIB number of the entity to be modified.

If you know the entity's MIB number, use this forrn for *entloc:*

EntId*(i4expr)*
Replace *i4expr* with an integer4 expression for the MIB number. Some intrinsie procedures and functions, such as **GetEnt,** allow the user to digitize entities in the graphics window. Their MIB numbers are then available to the program. Others, such as **FindProp and TagMib,** will return an MIB number when given non-graphical information such as the entity's properties or tags.
If you do not know the entity's MIB number, you may use one of the following keywords for entloc:

First
Verifies the first entity in the database. This will initialize the database search.
Next Verifies the next entity in the database. This keyword allows the program to step through the database sequentially and verify each entity. Each time a **Verify Next** statement is executed, the next entity in the database is verified. A Verify Next statement may also be used after a **Verify EntId***(i4expr)* statement. The database search will then start at the *i4expr* entity instead of the first entity.

Last
Verifies the last entity in the database. 'Mis keyword allows the program to verify the last entity inserted into the database without searching the database from the beginning.

If the type of entity specified by the enttype keyword does not match the type found using the entloc keyword, the DBStatus variable is set to three. When the end of the database is reached, DBStatus is set to two. See Appendix B, "System Variables," for more information.

Only enttype and *entloc* keywords can specify which entities to verify. The entdata keywords may only retum data about entities.

entatts: Optional keywords that allow you to verify the data of an entity. You can use the following *entatts* keywords with all entity types:

Color*(ivar)*
where *ivar* returns the color number for the verified entity.

Font*(ivar)*
where *ivar* returns the font number for the verified entity.

Group*(ivar)*
where *ivar* returns the group number of the verified entity.

Layer*(ivar)*
where *ivar* returns the layer for the verified entity.

Vvis*(ivar)*
where *ivar* returns the view number that the verified entity is visible in. A value of zero means that the entity is visible in all views.

entdata: Optional keywords that provide the data to be verified for a specified enttype. Note that only line, string, arc, text, and point entity types are supported. If the enttype keyword does not match the entity found by the entloc keyword, the value of the variables given in these entdata keywords will not change. The keywords for each enttype are:

For Lines:

Ends*(cvarl, cvar2)*
Returns the endpoint coordinates after the keyword Ends. The *cvar1* variable returns end one and *cvar2* returns end two of the line. Both variables return model space coordinates.

For Strings:

Verts*(ivar1, carray(iexpr1))*
The coordinates for the vertices
of a string will be returned in an array after the keyword **Verts.** Replace *carray* with the name of a coordinate array which returns the model space coordinates for the string vertices. The *ivar* variable retums the number of vertices in *carray*, and *iexpr1* with the first element to put vertices into.

For Arcs:

Org*(cvar)*
Replace *cvar* with a coordinate variable which will return the model space origin of the arc.

Radius*(rvar)*
Replace *rvar* with a real variable which will return the radius of the arc. Arcs are drawn counterclockwise.

AB*(rvar)*
Replace *rvar* with a real variable which will return the beginning angle of the arc in degrees.

AE*(rvar)*
Replace *rvar* with a real variable which will return the ending angle of the arc in degrees.

For Text:

Ang*(rvar)*
Replace *rvar* with a real variable which will return the angle the text was inserted at.

Hgt*(rvar)*
Replace *rvar* with a real variable which will return the text character height.

Just*(ivar)*
Replace *ivar* with an integer variable which will return the textjustification code: 1 - left j ustification 2 - rightjustification 3 - center j ustification

Lnsp*(rvar)*
Replace *rvar* with a real variable which will return the text line spacing factor.

Org*(cvar)*
Replace *cvar* with a coordinate variable which will return the text origin in model space.

Txt*(svar)*
Replace *svar* with a string variable that returns the actual text. Be sure to declare *svar* to be large enough to hold the data to be returned.

Wdt*(rvar)*
Replace *rvar* with a real variable which will return the text character width.

For Points:

Loc*(cvar)*
Replace *cvar* with a coordinate variable which will return the model space coordinate of the point.

Examples

```
VERIFY ARC ENT ID(WHICHENT) ORG(ORIG) RADIUS(RAD)
PROC MAIN
INTEGER MIB, icolor
    MIB = 1
    VERIFY FIRST
    LOOP
        MIB = MIB + 1
        VERIFY LINE NEXT COLOR(icolor)
        IF (DBStatus = 0) AND (icolor <> 12)
        MODIFY LINE ENTID(MIB) COLOR(12)
        ELSE IF (DBStatus = 2) THEN
        EXIT 2
        END IF
    END LOOP
END PROC
```

VLen

Type

Intrinsie Function Geometric

Purpose

Returns a real value equal to the distance between two points in three-dimensional space.

Syntax

VLen*(cexpr1, cexpr2)*

Parameters

cexpr1: Coordinate expression (input)
This parameter specifies the first point.

cexpr2: Coordinate expression (input)
This parameter specifies the second point.

Vunit

Type

Intrinsic Function Geometrie

Purpose

Returns the unit vector as a coordinate expression. The vector is one unit long, starting at the origin, parallel to the vector cexpr.

Syntax

VUnit*(cexpr)*

Parameters

cexpr: Coordinate expression (input)
This parameter specifies the coordinate expression whose unit vector will be returned.

NOTE: A coordinate value representing a veetor is obtained by subtracting the coordinate of the starting point of a vector from its ending point. For example, if a vector is directed from coordinate Cl to coordinate C2, it can be represented by the coordinate C2 - C 1.

Window

Type

Statement Input/Output

Purpose

Defines the dimensions of the UPL windows on your screen. It is easier to use the **Window** statement but it allows less control than the intrinsic procedure **DerineAW.** See **DerineAW** and Chapter 3, "Functional Listing", for more information.

Syntax

Window *iwin, itop, ibot, ileft, iright*

Keyword modifiers

iwin: Replace with the window number you want to define.

itop, ibot, ileft, and iright:

Allow you to change the top, bottom, left and right dimensions of the window. The ileft and iright expressions are optional. You must use commas to separate each expression.

The top line of the screen is line one and the bottom line is determined by the graphics device used on the user's system. The intrinsic procedure **PageInfo** will return this value. It usually is between 25 and 42.

For an alphanumeric window, if ileft and iright are not given, the window will fill the width of the screen and the graphics window will fill the largest remaining rectangular region. lf ileft and iright are given and are non-zero, then the graphics window is not changed.

UPL has 10 rectangular windows and a graphics window that allow you to input and output data. Other windows are reserved for data output by Personal Designer. The following list describes each window and its function:

1	General purpose alphanumeric window and Personal Designer command window
2 to 10	General purpose alphanumeric windows
11	Graphics window

The initial setting for all alphanumeric windows when a UPL program is started is a window the current size of the Personal Designer command window, which is window one. The graphics window will occupy the area left by the command window and the on-screen icon menu.

Windows 1 through 10 are used by the **Accept, Display, Print, and Send** statements. When you use one of these statements, your input data is placed in a window determined by the corresponding system variable. The system variables are **AccptWin, DisplayWin, PrintWin, and SendWin.** For more information, see Appendix B, "System Variables."

Each window has a cursor which starts in the upper left-hand corner of the window. When you use the **Accept, Display, Print, and Send** statements, the cursor will move to the position immediately following the last output character. UPL remembers the last eursor position for each window. When the cursor reaches the bottom of a window, it automatically scrolls to the window. To move the cursor to a specific location, see **PutCur.**

You can overlap as many windows as you like. The window with the highest priority overlaps the others. For windows 1 through 10, the window most recently used by the the **Accept, Display, Print, or Send** statement assumes the highest priority. For the rest of the windows you must specify the priority using the **DefineAW** intrinsic procedure. See DefineAW and Chapter 3, "Functional Listing", for more information.

Examples

```
WINDOW  2,1,5,50,80
WINDOW  IWIN,ITOP,ITOP+4
WINDOW  3,10,10,30,30    -- a very small window1

-- graphics window at top of screen
WINDOW  11,0,24,1,80
```

Write

Type

Statement Input/Output

Purpose

Transfers data from an expression to a file. The file must be opened with the **Open** statement.
Each **Write** operation is done in 3 steps:

1. Data in an expression is evaluated.

2. Data is then placed in the file starting at the file pointer

3. The file pointer is advanced to the point just after that data.
These steps are performed differently depending on the type of file, the data type of the expressions, and the way the statement's syntax is used.

Syntax

Write *flvar, expr: iexprl: iexpr2....,*

Keyword modifiers

flvar: File variable that must have been opened using the **Open** statement. The file may have been opened as a text or binary file, and may use sequential or random access. See the **Open** statement for more information.

expr: Optional expression of any data type except file, that can be used with *flvar*. The *expr* expressions must be separated by commas.
If the file is a **text file,** the **Write** statement first evaluates the expression and then converts the values from *expr*'s data type to the equivalent character string value. It then stores the string value in the file. Characters are written to the file starting at the file pointer. The file pointer then advances to point to the position immediately following the last character

written. This process repeats for the expressions in the statement. After all expressions in the statement have been written, an end-of-line sequence is written to the file. This starts a new line in the file. The file pointer is advanced to the beginning of the new line.

Iexpr1: Optional expression that can be used with *expr* to specify how many characters to write. This allows you to set up a format in your text file. For example, you could write a file of numbers arranged in columns.
Replace iexprl with a field width. This is an integer expression for the number of charaeters to use when outputting the value. If iexprl is positive, the field is right justified. If *iexpr1* is negative, the field is leftjustified. The unused portion of a field is written as blanks (ASCII 32). If *iexpr1* is specified but the value cannot be written in a field of *iexpr1* characters, the value is truncated on the right. A colon must precede *iexpr1*.

iexpr2: Optional field width for decimal places to be used with *iexpr1*. This may be used for real and coordinate values only. It specifies the number of characters in the *iexpr1* field which will be used for the decimal places. The "." counts for one place. A number of greater preeision is rounded up to fit in this field and, a number of less precision is padded with zeros. If *iexpr2* is negative, exponential notation will be used. A colon must precede *iexpr2*.
Optional punctuation. If you do not want to leave an end-of-line sequence after writing data, put a comma at the end of the **Write statement.** It will leave the file pointer just after the last written character. Any **subsequent Write** statement ending without a comma leaves an end-of-line sequence after writing its expressions. In partieular, a statement of the form "WRITE flvar" writes a blank line. That is, it writes an end-of-line sequence and moves the file pointer to the beginning of the new line.

If the file is a **binary file,** the **Write** statement simply transfers data using the internal data storage format for binary data. See Appendix E, "Internal Data Format," for more information. No data type conversion is done. Each expression's value is transferred to the file starting at the file pointer and continuing in the following bytes. The number of bytes written depends on the expression's data type. The file pointer then advances to the byte immediately following the last byte written. No end-of-line sequence is written to a binary file.

The field widths *iexpr1*, *iexpr2*, and the comma at the end of the **Write** statement have no significance with binary files, except when writing a string expression. In this case iexprl specifies how many characters to write. If iexprl is not given, the current length of the string expression is written.

lf writing to a sequential access file, the current file pointer position determines the end of the file when you close it. If you are writing to the middle of an existing file, move the file pointer to the end of the file before closing it to avoid truncating the file. To move the pointer to the end of the file, simply keep reading data until the file.EOF attribute is TRUE. Then close the file.

If Random File Access is used, the file pointer may also be repositioned using the flvar.POSITION or flvar.POS4 attribute. See the **Open** statement and WriteCArray, WriteIArray, WriteRArray intrinsics for more information.

Example

```
------------------------------------------------
proc main
integer i = 5, j = 321
integer ifld = 6, k = -2212
real a = 23.3256
     :
open f1, "data"
     :
write f1, i, j*3, a:10:3, i:fld, ' ':5,'A
string ':-12, k:(ifld*2)
     :
write f1, i:8, j:8,
     :
write f1, 'Another string':20
     :
write f1

end proc
------------------------------------------------
```

This program would write a file which looks like the following three lines (note: x represents a blank; <cr><lf> is an end-of-line sequence):

```
5963xxxx23.326xxxxx5xxxxxA
stringxxxxxxxxxxx-2212<cr><lf>
xxxxxxx5xxxxx321xxxxxxAnother string<cr><lf>
<cr><lf>
```

WriteCArray,WritelArray, WriteRArray

Type

Intrinsic Procedure Input/Output (File)

Purpose

Allows fast storage of integer, real or coordinate data to a binary file. It is useful for programs which need more than 32,767 bytes of data, the maximum amount which can be declared in a UPL program. This routine can be used to write large amounts of data to a file from a buffer array. The program can then read the data back from the file using the **ReadCArray, ReadArray, and ReadRArray** intrinsic procedures.

Syntax

WriteCArray*(file, array(1))*
WritelArray*(file, array(1))*
WriteRArray*(file, array(1))*

Parameters

file: File variable (input/output)
This parameter specifies the file variable for the data file. See the **Open** statement for more information on file IO. The file must be opened as a binary file. Sequential or random file access may be used.

array: Coordinate, integer, or real array of any length (input/output)
This parameter specifies the array from which you write the data into the file. It should be declared to be large enough to hold all the data to write in one call to **WriteCArray, WritelArray, and WriteRArray.**

The routine takes all the data in the array and writes it to the file starting at the file pointer. The file pointer is then placed immediately after the last byte written. The amount of data written is determined by the number of elements declared in the array. Specifically, each call to **WriteCArray,**

WritelArray, and **WriteRArray** writes a number of bytes equal to the number of elements in the array muttiplied by the number of bytes per element. The array must always be passed with a subscript of 1: `array (1).`

If you are using random file access, the file pointer may be changed to point to any byte in the file. This is done by setting the file.POSITION or file.POS4 attribute.

Setting the file. POSITION attribute moves the file pointer to the position equal to the value of file.POSITION multiplied by the value of file.RECLEN. That is, the file.POSITION attribute teils the program what file record to point to. The file.RECLEN attribute says how many bytes are in the record.

lf your program is writing a file whose arrays are all of the same size and data type, simply declare your record length to be the size of that array in bytes. Repositioning the file pointer is then simply a matter of setting file.POSITION to the array you want.

Setting the file.POS4 attribute moves the file pointer to the given byte offset into the file. (It is not affected by file.RECLEN or file.POSITION).

lf you are mixing arrays of different data types in the same file, you may find it easier to set the record length to 1 (using the **Reclen** keyword in the **Open** statement) and use the file.POS4 to set the file pointer as a byte offset into the file.

When calculating file.POS4, take into account the difference in array element sizes. That is, a real element takes up as much as two integer elements and, a coordinate element takes up six times as much as an integer element.

Since the file.POSITION attribute is itself an integer value, it can only be set as high as 32,767. Files are therefore limited to 32,767 * file.RECLEN bytes. lf you want to make a larger file, use the file.POS4 attribute which will allow a byte offset of up to 2,1147,483,647.

Examples

```
-----------------------------------------------
-- WRArray.upl
-- This program demonstrates use of WriteRArray.
-- The use of WriteIArray and WriteCArray are
-- very similar.
-- See ReadCArray, ReadIArray, ReadRArray for
-- the program RRArray.upl that will read the
-- file created here.
-----------------------------------------------

Proc Main

integer I, J, K
integer SavePos
real RealBuffer(100)
file DataFile

-- Open the data file with the length of the
-- data record: 400 = (4 bytes per real) * 100)

open DataFile, 'File.Dat' binary reclen(400)

-- Initialize buffer with random values
-- (for purposes of this demonstration).

loop I=1 to RealBuffer(1).SIZE
    RealBuffer(I) = Rnd()
end loop

-- Write data out to a file.

loop J = 1 to 30

-- Note: the loop below and the call to
-- WriteArray below it write the same
-- data to the same place in the file,
-- however, the call to WriteRArray is
-- MUCH faster!!

    RealBuffer(1) = Real(J) -- some calculation
    SavePos DataFile.POSITION -- save record #
    loop K  1 to RealBuffer(1).SIZE
        write DataFile, RealBuffer(K)
    end loop
    DataFile.POSITION = SavePos -- restore rec. #
    WriteRArray( DataFile, RealBuffer(1))

end loop
end proc
-----------------------------------------------
```

```
------------------------------------------------
-- WXArray.upl
-- This program demonstrates use of WriteCArray,
-- WriteIArray, WriteRArray. It creates a file
-- with blocks of 1000 integers, 500 reals and
-- 200 coord data in the same file.
-- A 12 byte header points to the beginning of
-- each section.
-- The program uses the POS4 attribute for
-- positioning the file pointer.
-- See ReadCArray, ReadIArray, ReadRArray for
-- the program RXArray.upl which reads this data
-- file.
------------------------------------------------

proc main

integer IntegerSize = 2
integer RealSize = 4
integer CoordSize = 12
integer HeaderSize = 12

-- Header information
integer4 StartIntegerData
integer4 StartRealData
integer4 StartCoordData

-- Data buffers
integer IntegerBuffer(100)
real    RealBuffer(50)
coord   CoordBuffer(20)

integer4 Dataoffset
integer  I, BufferCnt = 10

file DataFile

-- start of code –
open DataFile 'Data.fil' binary reclen(1)

-- Write data to the file.
-- First write placeholders for header values to
-- be filled in later.
-- Next write the data out. (In this example all
-- the data will be zero values.)
-- Keep track of where each block of data starts.
```

```
write DataFile, StartIntegerData, StartRealData, \
                                  StartCoordData
StartIntegerData = DataFile.POS4
loop I=1 to BufferCnt
    WriteIArray( DataFile, IntegerBuffer(1))
end loop

StartRealData = DataFile.POS4
loop I=1 to BufferCnt
    WriteRArray( DataFile, RealBuffer(1))
end loop

StartCoordData = DataFile.POS4
loop I=1 to BufferCnt
    WriteCArray( DataFile, CoordBuffer(1))
end loop

-- Write out updated values for header. Must
-- reset file pointer to beginning of file.

DataFile.POS4 = 0
write DataFile, StartIntegerData, StartRealData, \
                                  StartCoordData
-- Initialization completed.
-- Now write out some values.

-- Write out a buffer full random integer values
-- starting after integer value 47.

loop I = 1 to IntegerBuffer(1).SIZE
    IntegerBuffer(I) = integer(Rnd()*10.0)
end loop

DataOffset = 47 * integer4( IntegerSize)
DataFile.POS4 = StartIntegerData + DataOffset

WriteIArray( DataFile, IntegerBuffer(1))

-- Write out a buffer full random real values
-- starting after real value 150.

loop I = 1 to RealBuffer(1).SIZE
    RealBuffer(I) = Rnd()
end loop

DataOffset = 150 * integer4(RealSize)
DataPile.POS4 = StartRealData + DataOffset

WriteRArray( DataFile, RealBuffer(1))

-- Write out a buffer full random coord values
-- starting after coord value 10.
```

```
loop I = 1 to CoordBuffer(1).SIZE
    CoordBuffer(I).X = (Rnd() * 10.0) * real(I)
    CoordBuffer(I).Y = (Rnd() * 10.0) * real(I)
    CoordBuffer(I).Z = (Rnd() * 10.0) * real(I)
end loop

DataOffset = 10 * integer4( CoordSize)
DataFile.POS4 = StartCoordData + DataOffset

WriteCArray( DataFile, CoordBuffer(1))

close DataFile

end proc
-----------------------------------------------------
```