12d® Model™

12d Model Programming Language Manual

Version 10

October 2013

12D SOLUTIONS PTY LTD

ACN 101 351 991

PO Box 351 Narrabeen NSW Australia 2101

Australia Telephone (02) 9970 7117 Fax (02) 9970 7118

International Telephone 61 2 9970 7117 Fax 61 2 9970 7118

email support@12d.com web page www.12d.com

12d Model Programming Manual

12d Model Programming Manual V10

This book is the programming manual for the software product 12d Model.

Disclaimer

12d Model is supplied without any express or implied warranties whatsoever.

No warranty of fitness for a particular purpose is offered.

No liabilities in respect of engineering details and quantities produced by 12d Model are accepted.

Every effort has been taken to ensure that the advice given in this manual and the program 12d Model is correct. However, no warranty is expressed or implied by 12d Solutions Pty Ltd.

Copyright

Introduction **11**
The Mouse 11
Compiling and Running a 12dPL Program 12

Basic Language Structure **15**
Names 15
Reserved Names 15
White Space 16
Comments 16
Variables 17
Variable Names 17
Variable Declarations 17
Variable Types 17
Constants 35
Assignment and Operators 36
Assignment 36
Binary Arithmetic Operators 36
Binary Arithmetic Operators for Vectors and Matrices 36
Relational Operations 38
Logical Operators 38
Increment and Decrement Operators 38
Bitwise Operators 38
Assignment Operators 38
Statements and Blocks 40
Flow Control 41
Logical Expressions 41
12dPL Flow Controls 41
if, else, else if 42
Conditional Expression 43
Switch 43
While Loop 45
For Loop 46
Do While Loop 47
Continue 47
Break 47
Goto and Labels 48
Precedence of Operators 49
Preprocessing 50

Functions **51**
Functions 51
Main Function 52
User Defined Functions 53
Return Statement 53
Array Variables as Function Arguments 54
Function Prototypes 55
Automatic Promotions 56
Passing by Value or by Reference 57
Overloading of Function Names 59
Recursion 60
Assignments Within Function Arguments 61
Blocks and Scopes 62

Locks **65**

12dPL Library Calls **67**
Creating a List of Prototypes 68
Function Argument Promotions 69
Automatic Promotions 69
Function Return Codes 70

Command Line-Arguments 71
Array Bound Checking 72
Exit 73
Angles 74
    Pi 74
    Types of Angles 74
Text 76
    Text and Operators 76
    General Text 76
    Text Conversion 79
Textstyle Data 83
Maths 96
Random Numbers 98
Vectors and Matrices 100
Triangles 121
System 123
Uid's 131
    Uid Arithmetic 131
    Uid Functions 132
Input/Output 138
    Output Window 138
    Clipboard 140
    Files 141
    12d Ascii 150
Menus 152
Dynamic Arrays 155
    Dynamic Element Arrays 156
    Dynamic Text Arrays 158
    Dynamic Real Arrays 161
    Dynamic Integer Arrays 163
Points 165
Lines 167
Arcs 169
Spirals and Transitions 172
Parabolas 184
Segments 185
Segment Geometry 189
    Length and Area 189
    Parallel 190
    Tangents 192
    Intersections 193
    Offset Intersections 194
    Angle Intersect 195
    Distance 196
    Locate Point 197
    Drop Point 198
    Projection 199
    Change Of Angles 200
Colours 201
User Defined Attributes 203
Folders 213
12d Model Program and Folders 215
Project 219
Models 229
Views 245
Elements 250
    Types of Elements 251
    Parts of 12d Elements 253

Tin Element ... 272
Triangulate Data ... 273
Tin Functions ... 274
Null Triangles ... 283
Colour Triangles ... 286
Super String Element ... 288
Super String Dimensions ... 288
Basic Super String Functions ... 296
Super String Height Functions ... 308
Super String Tinability Functions ... 311
Super String Segment Radius Functions ... 316
Super String Point Id Functions ... 318
Super String Vertex Symbol Functions ... 321
Super String Pipe/Culvert Functions ... 328
Super String Vertex Text and Annotation Functions ... 344
Super String Segment Text and Annotation Functions ... 362
Super String Fills - Hatch/Solid/Bitmap/Pattern/ACAD Pattern Functions ... 381
Super String Hole Functions ... 394
Super String Segment Colour Functions ... 397
Super String Segment Geometry Functions ... 399
Super String Extrude Functions ... 402
Super String Interval Functions ... 406
Super String Vertex Attributes Functions ... 409
Super String Segment Attributes Functions ... 420
Super String Uid Functions ... 431
Super String Vertex Image Functions ... 434
Super String Visibility Functions ... 438
Examples of Setting Up Super Strings ... 444
2d Super String ... 445
2d Super String with Arcs ... 446
3d Super String ... 448
Polyline Super String ... 449
Pipe Super String ... 451
Culvert Super String ... 453
Polyline Pipe Super String ... 455
4d Super String ... 457
Super Alignment String Element ... 459
Arc String Element ... 461
Circle String Element ... 467
Text String Element ... 469
Pipeline String Element ... 483
Drainage String Element ... 485
Underlying Drainage String Functions ... 488
General Drainage String Functions ... 492
Drainage String Pits ... 496
Drainage Pit Type Information in the drainage.4d File ... 513
Drainage String Pit Attributes ... 519
Drainage String Pipes ... 530
Drainage Pipe Type Information in the drainage.4d File ... 543
Drainage String Pipe Attributes ... 544
Drainage String House Connections - For Sewer Module Only ... 555
Feature String Element ... 562
Interface String Element ... 564
Face String Element ... 568
Plot Frame Element ... 575
Strings Replaced by Super Strings ... 585
2d Strings ... 586
3d Strings ... 590

4d Strings 594
Pipe Strings 608
Polyline Strings 613
Alignment String Element 618
General Element Operations 629
Selecting Strings 629
Drawing Elements 630
Open and Closing Strings 631
Length and Area of Strings 631
Position and Drop Point on Strings 632
Parallel Strings 634
Self Intersection of String 634
Loop Clean Up for String 634
Check Element Locks 635
Miscellaneous Element Functions 635
Creating Valid Names 637
XML 639
Map File 645
Macro Console 649
Panels and Widgets 662
Cursor Controls 666
Panel Functions 667
Horizontal Group 670
Vertical Group 673
Widget Controls 676
General Widget Commands and Messages 684
Widget Information Area Menu 685
Widget Tooltip and Help Calls 686
Panel Page 689
Input Widgets 691
Message Boxes 830
Log_Box and Log_Lines 835
Buttons 843
GridCtrl_Box 853
Tree Box Calls 862
General 868
Quick Sort 869
Name Matching 870
Null Data 871
Contour 873
Drape 875
Drainage 876
Volumes 882
Interface 884
Templates 885
Applying Templates 886
Strings Edits 889
Place Meshes 892
Utilities 893
Affine Transformation 894
Chains 895
Convert 896
Cuts Through Strings 897
Factor 898
Fence 899
Filter 900
Head to Tail 901
Helmert Transformation 902

Rotate ... 903
Swap XY ... 904
Translate ... 905
12d Model Macro_Functions ... 906
Processing Command Line Arguments in a Macro_Function ... 907
Creating and Populating the Macro_Function Panel ... 908
Storing the Panel Information for Processing ... 910
Recalcing ... 910
Storing Calculated Information ... 911
Macro_Function Functions ... 911
Function Property Collections ... 936
Plot Parameters ... 946
Undos ... 953
Functions to Create Undos ... 954
Functions for a 12dPL Undo_List ... 956
ODBC Macro Calls ... 959
Connecting to an external data source ... 959
Querying against a data source ... 961
Navigating results with Database_Result ... 963
Insert Query ... 966
Update Query ... 967
Delete Query ... 969
Manual Query ... 970
Query Conditions ... 971
Transactions ... 974
Parameters ... 975

Examples ... 979
Example 1 ... 982
Example 1a ... 983
Example 1b ... 984
Example 2 ... 986
Example 2a ... 987
Example 3 ... 988
Example 4 ... 989
Example 5 ... 990
Example 5a ... 991
Example 5b ... 992
Example 6 ... 993
Example 7 ... 1000
Example 8 ... 1003
Example 9 ... 1005
Example 10 ... 1011
Example 11 ... 1014
Example 12 ... 1018
Example 13 ... 1024
Example 14 ... 1031
Example 15 ... 1039

Appendix - Set_ups.h File ... 1053
General Constants ... 1054
Model Mode ... 1055
File Mode ... 1059
View Mode ... 1066
Tin Mode ... 1069
Template Mode ... 1073
Project Mode ... 1074
Directory Mode ... 1075
Function Mode ... 1076

Linestyle Mode 1077
Symbol Mode 1078
Snap Mode 1079
Super String Use Modes 1080
Select Mode 1082
Widgets Mode 1083
Text Alignment Modes for Draw_Box 1084
Set Ups.h 1085

Appendix - Ascii, Ansi and Unicode 1096

12d Model Programming Language Course 5

1 Introduction

The 12d Solutions Programming Language (12dPL), is a powerful programming language designed to run from within 12d Model. It is also known as 4DML from when the product was called *4d Model*.

Its main purpose is to allow users to enhance the existing 12d Model package by writing their own programs.

12dPL is based on a subset of the C++ language with special extensions to allow easy manipulation of 12d Model data. A large number of intrinsic functions are supplied which cover most aspects of civil modelling.

12dPL has been designed to fit in with the ability of 12d Model to "stack" an incomplete operation.

This reference manual does not try to teach programming techniques. Instead this manual sets out the syntax, restrictions and supplied functions available in 12dPL.

Examples of usage are given for many of the 12dPL supplied functions.

It is assumed that the reader has an understanding of the basic concepts of programming though not necessarily using C++.

Note: 12dPL programs are often referred to as "macros". However 12dPL programs are fully fledged computer programs and should not be confused with say "keyboard macros" which simply record a users keystrokes and then replays them.

When you see the word **macro** in this manual, it refers to a 12dPL program and not a keyboard macro.

See The Mouse
See Compiling and Running a 12dPL Program

The Mouse

The mouse is used extensively in 12d Model and also in 12d Model programs.

Most new PC mice have three buttons (left, middle and right) but on older PC's both two and three button mice exist.

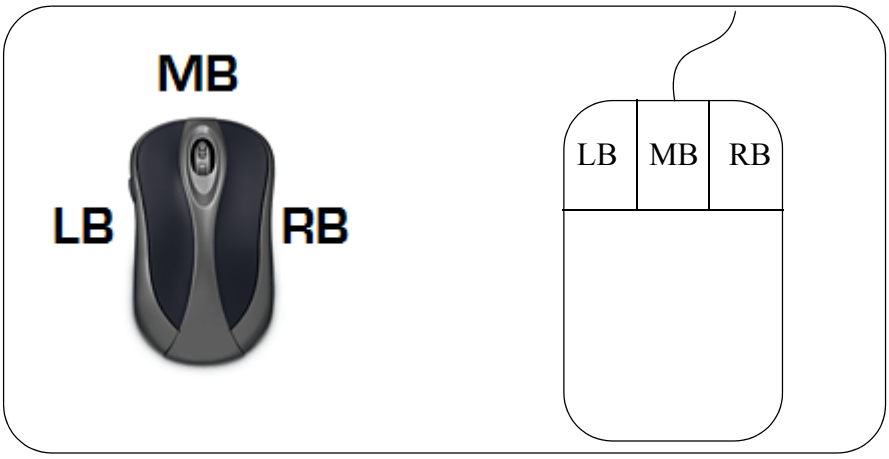
12d Model can be operated with either a two or a three button mouse but a three button mouse is preferred.

In this manual the buttons will be denoted by

LB = the left button

MB = the middle button

RB = the right-button



12d Model monitors the mouse being **pushed down** and when it is subsequently **released** as separate events. Unless otherwise specified in the manual, **clicking** a button will mean **pressing the button down and releasing it again**. The **position of the mouse** is normally taken as being when the **button is released**.

In screen messages, the effect of pressing each button on the mouse is shown by enclosing the effect for each button in square brackets ([]) in left-to-right button order. That is

[left button effect] [middle button effect] [right button effect]

Empty brackets, [], indicate that pressing the button has no effect at that time.

Continue to Compiling and Running a 12dPL Program .

Compiling and Running a 12dPL Program

A 12d Model Programming Language program consists of one file containing a starting function called **main**, and zero or more user defined functions. The complete definition and structure of functions will be specified later in this manual.

The filename containing the program must end in **.4dm**.

Once typed in, the 12dPL program is **compiled**, from either inside or outside of 12d Model, to produce a run-time version of the program (a compiled program).

It is the compiled version of the program that is run from within 12d Model.

To compile a 12dPL program, use either

(a) Compiling from Inside 12d Model

Inside 12d Model use the **compile** or **compile and run** options

Utilities =>Macros =>Compile

Utilities =>Macros =>Compile/run

or

(b) Compiling from Outside 12d Model

Outside 12d Model, the 12dPL compiler is called **cc4d.exe** which is in the **nt.x64** folder for the 64-bit 1*2d.exe* or **nt.x86** for 32-bit *12d.exe*.

To compile the program, run cc4d.exe followed by the name of the file containing the macro.

For example, to compile the program *macro.4dm,* type into a command window:

(a) when running a 64-bit 12d.exe on a 64-bit Microsoft Windows Operating System

"C:\Program Files\12d\12dmodel\10.00\nt.x64**cc4d.exe**" *macro.4dm*

(b) or when running a 32-bit *12d.exe* on a 32-bit Microsoft Windows OS.

"C:\Program Files\12d\12dmodel\10.00\nt.x86**cc4d.exe**" *macro.4dm*

(c) or when running a 32-bit *12d.exe* on a 64-bit Microsoft Windows OS.

"C:\Program Files (x86)\12d\12dmodel\10.00\nt.x86**cc4d.exe**" *macro.4dm*

The compiler first checks the program's syntax and reports any errors to the console window. If there are no errors, a run-time object is created with the same name as the original program but ending in **.4do**.

If you want the errors to be logged to a file rather than going to the console window, then add

-log *log_file_name*

before the program name (a common convention is to use the same file name stem and add ".4dl" for the log file):

For example

"C:\Program Files\12d\12dmodel\10.00\nt.x64**cc4d.exe**" -log *macro.4dl* *macro.4dm*

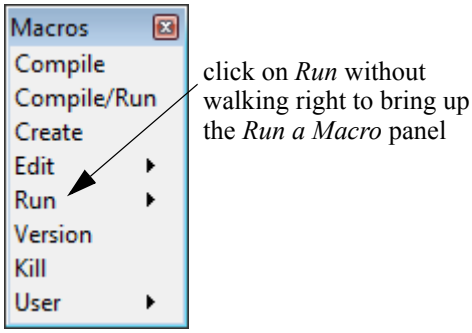
Running a Compiled 12d Model Program

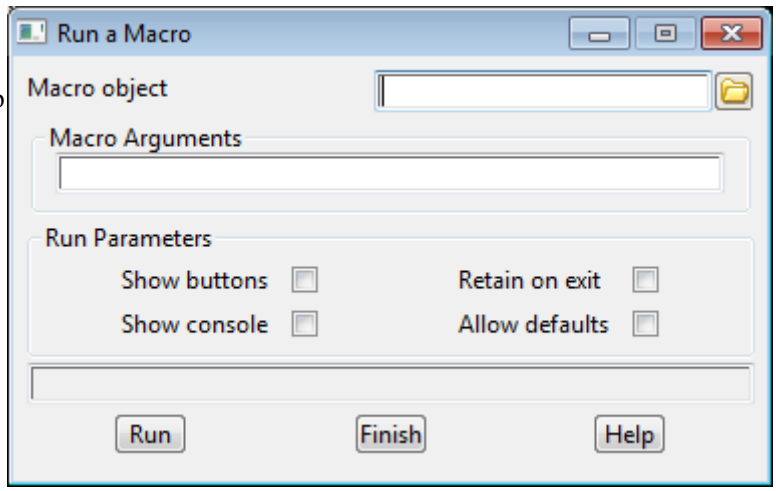
To run a compiled program from within 12d Model, walk-right on the menu option

Utilities =>Macros =>Run

and select the program from the list of available programs.

Alternatively, if the **Utilities =>Macros** menu has been pinned up, then clicking on the **Run** option (and not walking right) brings up the **Run a Macro** panel.





A program is run by entering the name of its compiled object into the Macro object panel field, filling in the Macro arguments field if there are any command-line argument for the program, and then selecting the button **Run**.

The **Run a Macro** panel is then removed from the screen and the program run.

Note: Programs can also be run form functions keys, menus and toolbars. See the Appendix *Function Keys, Manus, Toolbars* in the **12d Mode**l Reference manual for more details.

2 Basic Language Structure

See Names
See Reserved Names
See White Space
See Comments
See Variables
See Assignment and Operators
See Statements and Blocks
See Flow Control
See Precedence of Operators
See Preprocessing

Names

A name (also known as a word) denotes an object, a function, an enumerator, a type, or a value.

A name is introduced into a program by a declaration.

All names must be declared before they can be used.

A name can be used only within a region of program text called its scope (discussed later).

A name has a type that determines its use.

Reserved Names

The following names (words) are reserved and cannot be used for user defined names:

Integer	Real	Text	Element	Model
Point	Line	Segment	Menu	View
Tin	Dynamic_Element	Dynamic_Text		
break	case	char	continue	default
do	double	else	float	for
goto	if	int	integer	long
real	return	short	switch	void
while				
auto	class	const	delete	enum
extern	friend	inline	new	operator
private	protected	public	register	signed
sizeof	static	struct	template	this
throw	try	typedef	union	unsigned
virtual	volatile			

All 12dPL variable types and 12dPL functions and user defined functions are also considered to

be keywords and cannot be used for user defined names.

White Space

Spaces, tabs, newlines (<enter>, <CR>), form feeds, and comments are collectively known as white space.

White space is ignored except for the purpose of separating names or in text between double quotes. Hence blank lines are ignored in a 12dPL program.

For example,

```
    goto          fred     ;
```

is the same as

```
  goto fred;
```

Comments

12dPL supports two styles of comments -

A line oriented comment

all characters after a double slash // and up the end of a line are ignored.

A block comment

all characters between a starting /* and a terminating */ are ignored.

An example of comments in 12dPL is

```
  void main()
  {
    Real y = 1;                    // the rest of this line is comment
/*        this comment can carry
           over many lines until
           we get to the termination characters */
  }
```

Variables

Variables and constants are the basic data objects manipulated in a program.

Declarations list the names of the variables to be used, and state what type they have.

Operators specify what is to be done to variables.

Expressions combine variables and operators to produce new values.

The type of an object determines the set of values it can have and what operations can be performed on it.

Variable Names

In 12dPL, variable names must start with an alphabetic character and can consist of upper and/or lower case alphabetic characters, numbers and underscores (_) and there is no restriction on the length of variable names.

12dPL variable names are **case sensitive**.

Variable Declarations

In 12dPL, all variables must be declared before they are used.

A declaration consists of a variable type and a list of variable names separated by commas and **ending the line** with a **semi-colon ";"**.

For example

Integer fred, joe, tom**;**

where Integer is the variable type and fred, joe and tom are the names of variables of type Integer.

Variable Types

There are a wide variety of *12d Model* variable types supported in 12dPL. For example

(a) void

This is a special type which is only used for functions which have no return value. All other functions must return one variable take as the function return value. The user does not define variables of this type and it is only used in function definitions.

For example:

```
void    Exit(Integer code)
```

(b) Mathematical Variable Types

Standard mathematical variables for calculations using the mathematical operations such as addition, subtraction, multiplication and division.

These variables only exist within the 12dPL program and cease to exist when it finishes.

For example, Integer, Real, Text, Vector2, Vector3, Matrix2, Marix3, Matrix4

For more information on these variables, go to Mathematical Variable Types

(c) Geometric Construction Variable Types

These objects are used within 12dPL macros for geometric calculations. They are only temporary objects and only last for the duration of the program.

For example, Point, Line, Arc, Spiral, Segment.

For more information on these variables, go to Geometric Construction Variable Types

(d) 12d Database Handles

These variable types act as **Handles** to access data stored in the **12d Model** database. This data is retrieved from and stored in the 12d Model database and so exists after the program terminates.

For example, Element, Dynamic_Element, Tin, Model, View, Function, Undo_List

For more information on these variables, go to 12d Model Database Handles

(e) 12d Internal Variable Types

These variables help access data stored in the *12d Model* database handles. This data may be retrieved from and stored in 12d Model database via the handles, and so can exist after the program terminates.

For example, Uid, Attributes, SDR_Attributes, Blobs, Textstyle_Data.

For more information on these variables, go to 12d Internal Variable Types

(f) 12d Interface Variable Types

Variables for building interfaces, such as menus and panels, to communicate with the macro user.

For example, Menu, Panel, Widget, Model_Box.

For more information on these variables, go to 12d Model Interface Variable Types

(g) File Interface Variable Types

Variables for accessing files.

For example, File, Map_File, Plot_Parameter_File, XML_Document, XML_Node.

For more information on these variables, go to File Interface Variable Types

(h) ODBC Database Interface Variable Types

Variables for accessing and manipulating ODBC databases.

For example, Connection, Select_Query, Insert_Query, Update_Query, Delete_Query, Database_Results, Transactions, Parameter_Collection, Query_Condition, Manual_Condition

For more information on these variables, go to ODBC Database Variable Types

(i) Arrays and Dynamic Arrays Types

Arrays are used to allocate a number of storage units that have the same type. Arrays sore a fixed number of items and Dynamic Arrays store a variable number of items.

For example, Real arrays, Integer, Arrays, Text Arrays, Dynamic_Text.

For more information on these variables, go to Array Types

For a quick summary of all the 12dPL variables, go to Summary of 12dPL Variable Types

Mathematical Variable Types

Standard mathematical variables for calculations using the mathematical operations such as addition, subtraction, multiplication and division.

See

Integer
Real
Text
Vector2
Vector3
Vector4

Matrix3
Matrix4

Integer

A 32-bit whole number. It can be positive or negative. For example -1, 0 and 1.

Real

A 64-bit decimal number. It can be positive or negative. For example -1.0, 0.0 and 1.0

Text

A sequence of characters. For example Dog

Vector2

An entity consisting of two Real values. If the two real values of a **Vector2** are X and Y, the values in a Vector2 are often expressed as (X,Y).

Vector3

An entity consisting of three Real values. If the three real values of a **Vector3** are X, Y and Z, the values in a **Vector3** are often expressed as (X,Y,Z).

Vector4

An entity consisting of four Real values. If the four real values of a **Vector3** are X, Y, Z and W, the values in a **Vector4** are often expressed as (X,Y,Z,W).

Matrix3

An entity consisting of nine Real values. The values in the **Matrix3 matrix** are expressed as three rows and three columns and indexed as matrix (row, column) and

matrix (1,1) = a matrix(1,2) = b matrix(1,3) = c

matrix (2,1) = d matrix(2,2) = e matrix(2,3) = f

matrix (3,1) = g matrix(3,2) = h matrix(3,3) = i

where a, b, c, d, e, f, g, h and i are the nine Real values of **matrix**.

where a, b, c and d are the four Real values of **matrix**.

Matrix4

An entity consisting of sixteen Real values. The values in the **Matrix4 matrix** are expressed as four rows and four columns and indexed as matrix(row,column) and

matrix (1,1) = a matrix(1,2) = b matrix(1,3) = c matrix(1,4) = d

matrix (2,1) = e matrix(2,2) = f matrix(2,3) = g matrix(2,4) = h

matrix (3,1) = i matrix(3,2) = j matrix(3,3) = k matrix(3,4) = l

matrix (4,1) = m matrix(4,2) = n matrix(4,3) = o matrix(4,4) = p

where a, b, c, d, e, f, g, h, i, j, k, l, m, n, o and p are the sixteen Real values of **matrix**.

Geometric Construction Variable Types

Construction variables are used within 12dPL macros for geometric calculations but they are temporary objects and only last for the duration of the program.

See

Point
Line
Arc

Spiral (Transition)
Parabola
Segment

Point

A Point is a three dimensional point consisting of x, y and z co-ordinates (x,y,z).

A Point is a construction entity and is not stored in **12d Model** models.

Line

A Line is three dimensional line joining two Points.

A Line is a construction entity and is not stored in **12d Model** models.

Arc

An Arc is a helix which projects onto a circle in the (x,y) plane.

That is, in a plan projection, an Arc is a circle. But in three dimensions, the Arc has a z value (height) at the start of the Arc and another (possibly different) z value at the end of the Arc. The z value varies linearly between the start and end point of the Arc. So an Arc is **NOT** a circle in a plane in 3d space, except when it is in a plane parallel to the (x,y) plane.

In 12dPL an Arc is a construction entity and is not stored in **12d Model** models.

Spiral (Transition)

An spiral is a mathematically defined transition which when projected on to the (x,y) plane, has a continuously varying radius going between a between a line (infinite radius) and an arc for a full spiral, or an arc to another arc for a partial spiral.

Note that in 12d Model, the Spiral covers the traditional clothoid spirals and also other transitions (such as a cubic parabola) which are not spirals in the true mathematical sense.

For more information on Spirals and Transitions, go to Spirals and Transitions in the chapter 12dPL Library Calls

In 12dPL a Spiral is a construction entity and is not stored in **12d Model** models.

Parabola

Parabolas are used in the vertical geometry of an Alignment or Super Alignment. The vertical geometry is defined in the (chainage, height) plane and parabolas can be place on vertical intersection points. So the parabola is defined in the (chainage, height) plane.

In 12dPL a Parabola is a construction entity and is not stored in **12d Model** models.

Segment

A Segment is either a Point, Line, Arc, Parabola or a Spiral.

A Segment has a unique type which specifies whether it is a Point, Line, Arc, Parabola or Spiral.

A Segment is a construction entity and is not stored in **12d Model** models.

See Segments

12d Model Database Handles

Unlike construction entities, the **12d Model** database handle variables are used for data from the **12d Model** project database. They could be handles for Views, Models, Elements, Functions etc.

The handles don't contain the database information but merely point to the appropriate database records.

Hence data created with handle variables can be stored in the **12d Model** database and will exist after the 12dPL program terminates.

Since the handle merely points to the Project data, the handle can be changed so that it points to a different record without affecting the data it originally pointed to.

The 12dPL variables **Element**, **View**, **Model** and **Macro_Function** create and use handles.

Sometimes it is appropriate to set a handle so that it doesn't point to any data. This process is referred to as setting the handle to null.

Note that when setting a handle to null ("nulling" it), no **12d Model** data is changed - the handle simply points to nothing.

See

Element
Model
View
Macro_Function or Function
Undo_List

Element

The variable type **Element** is used to refer to the standard *12d Mode*l entities that can be stored in a *12d Model* models.

Elements act as handles to the data in the *12d Model* database so that the data can be easily referred to and manipulated within a macro.

The different types of **Elements** are

Arc	an arc in the (x,y) plane with linear interpolated z values (i.e. a helix). *See* Arc String Element
Circle	a circle in the (x,y) plane with a constant z value. *See* Circle String Element
Drainage	string for drainage or sewer elements. *See* Drainage String Element
Feature	a circle with a z-value at the centre but only null values on the circumference. *See* Feature String Element
Interface	string with (x,y,z,cut/fill flag) at each vertex. *See* Interface String Element
Pipe	string width (x,y,z) at each point and a diameter. *See* Pipe Strings
Plot Frame	element used for production of plan plots. *See* Plot Frame Element
Pipeline	an Alignment string with a diameter. *See* Pipeline String Element
Super	general string with at least (x,y,z,radius) at each vertex. *See* Super String Element
Super Alignment	a string with separate horizontal geometry defined by using the intersection point methods and other construction methods such as fixed and floating. *See* Super Alignment String Element
SuperTin	a list of Tins that acts as one Tin
Text	string with text at a vertex. *See* Text String Element

Tin triangulated irregular network - a triangulation *See* Tin Element

Superseded Element Types

2d string with (x,y) at each vertex but constant z. *See* 2d Strings

3d string with (x,y,z) at vertex point. *See* 3d Strings

4d string with (x,y,z,text) at each vertex. *See* 4d Strings

Alignment string with separate horizontal and vertical geometry defined only by using the intersection point methods. *See* Alignment String Element

Polyline string with (x,y,z,radius) at each vertex. *See* Polyline Strings

The Element type is given by the Get_type(Element elt,Text text) function.

Model

The variable type **Model** is used as a handle to refer to *12d Model* models within macros. *See* Models

View

The variable type View is used as a handle to refer to *12d Model* views within macros. *See* Views

Macro_Function or Function

The variable type Macro_Function or Function is used as a handle to refer to a 12d Model function within macros. User defined Macro_Functions/Functions can be created from a macro. *See* 12d Model Macro_Functions

12d Internal Variable Types

These variables help access data stored in the *12d Model* database handles. This data may be retrieved from and stored in 12d Model database via the handles, and so can exist after the program terminates.

See

- Uid
- Attributes
- SDR_Attribute
- Blob
- Screen_text
- Textstyle_Data
- Equality_Label
- Undo

Uid

A **U**nique **Id**entifier for entities in a 12d Model database. *See* Uid's

Attributes

The variable type Attributes is used as a handle to refer to an 12d Model attribute structure within macros.

Attributes are user defined and can be attached to Projects, Models, Elements and Macro_Functions/Functions.*See* User Defined Attributes

SDR_Attribute

SDR_Attribute are special attributes used with the *12d* Survey Data Reduction process.

Blob

A binary object.

Screen_text

See Screen_Text .

Textstyle_Data

TextStyle_Data holds information about the text such as colour, textstyle, justifcation, height. *See* Textstyle Data .

Equality_Label

Equality_Label holds information for labelling text as an Equality

Undo

A variable to hold information that is placed on the 12d Model Undo system.*See* Undos

Undo_List

The variable type Undo_List is a handle to a list of Undo's. *See* Undos

12d Model Interface Variable Types

The objects for building interfaces, such as menus and panels, to communicate with the macro user.

All these items are derived from a Widget and so can be used in any argument that is of type **Widget**.

See
Widget

See
Menu
Panel
Overlay_Widget

Objects for Formatting Widgets in a Panel

See
Vertical_Group
Horizontal_Group
Widget_Pages

Control Objects for Placing in Horizontal/Vertical Groups and Panels

See
Button
Select_Button
Angle_Box
Attributes_Box
Attributes_Box
Billboard_Box
Bitmap_Fill_Box
Bitmap_List_Box
Chainage_Box

Choice_Box
Colour_Box
Colour_Message_Box
Date_Time_Box
Directory_Box
Draw_Box
File_Box
Function_Box
Graph_Box
GridCtrl_Box
HyperLink_Box
Input_Box
Integer_Box
Justify_Box
Linestyle_Box
List_Box
ListCtrl_Box
Map_File_Box
Message_Box
Model_Box
Name_Box
Named_Tick_Box
New_Select_Box
New_XYZ_Box
Plotter_Box
Polygon_Box
Real_Box
Report_Box
Select_Box
Select_Boxes
Sheet_Size_Box
Source_Box
Symbol_Box
Tab_Box
Target_Box
Template_Box
Text_Edit_Box
Text_Style_Box
Texture_Box
Tree_Box
Tree_Page ??
Tick_Box
Tin_Box
View_Box
XYZ_Box

Widget

The objects for building interfaces, such as menus and panels, to communicate with the macro user. All these items are derived from a Widget and so can be used in any argument that is of type **Widget**. For the Widget 12dPL calls, see Panels and Widgets

Menu

An object that holds the data for a user defined **12d Model** menu.

Panel

An object that holds the data for a user defined **12d Model** panel. See Panels and Widgets .

Objects for Formatting Widgets in a Panel

Overlay_Widget

Sheet_Panel

Vertical_Group

Used for formatting a panel.

A Vertical_Group holds Widgets that will be placed horizontally in a Panel. *See* Panel Functions

Horizontal_Group

Used for formatting a panel.

A Horizontal_Group holds Widgets that will be placed horizontally in a Panel. *See* Panel Functions

Widget_Pages

A panel can have different pages. *See* Panel Page

Control Objects for Placing in Horizontal/Vertical Groups and Panels

Button

A button on a Panel. *See* Buttons

Select_Button

A button on a Panel for selecting strings. *See* Select_Button

Angle_Box

A box on a Panel for inputting angle information. *See* Angle_Box.

Attributes_Box

See Attributes_Box.

Billboard_Box

A box on a Panel for selecting a billboard name from the pop-up list of project billboards. *See* Texture_Box.

Bitmap_Fill_Box

See Bitmap_Fill_Box.

Bitmap_List_Box

Chainage_Box

See Chainage_Box.

Choice_Box

See Choice_Box.

Colour_Box

A box on a Panel for selecting a colour from the pop-up list of project colours. *See* Colour_Box.

Colour_Message_Box

A box on a Panel for writing messages to. Different background colours for the display area can also be set. *See* Colour_Message_Box .

Date_Time_Box

See Date_Time_Box .

Directory_Box

See Directory_Box .

Draw_Box

See Draw_Box .

File_Box

See File_Box .

Function_Box

See Function_Box .

Graph_Box

See Function_Box .

GridCtrl_Box

See GridCtrl_Box .

HyperLink_Box

See HyperLink_Box .

Input_Box

See Input_Box .

Integer_Box

See Integer_Box .

Justify_Box

See Justify_Box .

Linestyle_Box

A box on a Panel for selecting a linestyle from the pop-up list of project linestyles. *See* Linestyle_Box .

List_Box

See List_Box .

ListCtrl_Box

Map_File_Box

See Map_File_Box .

Message_Box

A box on a Panel for writing messages to. *See* Message_Box . Also *see* Colour_Message_Box

Model_Box

A box on a Panel for creating a new model, or selecting a model from the pop-up list of project models. *See* Model_Box .

Name_Box

See Name_Box .

Named_Tick_Box

See Named_Tick_Box .

New_Select_Box

See New_Select_Box .

New_XYZ_Box

See New_XYZ_Box .

Plotter_Box

See Plotter_Box .

Polygon_Box

See Polygon_Box .

Real_Box

See Real_Box .

Report_Box

See Report_Box .

Select_Box

See Select_Box .

Also see New_Select_Box

Select_Boxes

See Select_Boxes .

Sheet_Size_Box

See Sheet_Size_Box .

Source_Box

See Source_Box .

Symbol_Box

See Symbol_Box .

Tab_Box

See Select_Boxes .

Target_Box

See Target_Box .

Template_Box

See Template_Box .

Text_Edit_Box

See Text_Edit_Box .

Text_Style_Box

See Text_Style_Box .

Texture_Box

See Texture_Box .

Tree_Box

See Tree Box Calls .

Tree_Page ??

Tick_Box

See Tick_Box .

Tin_Box

See Tin_Box .

View_Box

A box on a Panel for selecting a view from the pop-up list of project views. *See* View_Box .

XYZ_Box

Also see New_XYZ_Box

File Interface Variable Types

Variables for accessing files.

See

- File
- Map_File
- Plot_Parameter_File
- XML_Document
- XML_Node

File

A file unit. *See* Files .

Map_File

A file used for mapping element properties. *See* Map File .

Plot_Parameter_File

A file unit. *See* Map File .

XML_Document

The file contents are structured as an XML document. *See* XML .

XML_Node

ODBC Database Variable Types

The variables are used when accessing and querying a ODBC database.

See

- Connection
- Select_Query
- Insert_Query
- Update_Query
- Delete_Query
- Database_Results
- Transactions
- Parameter_Collection
- Query_Condition
- Manual_Condition

Connection

The connection to the database.

Select_Query

Used to retrieve data from the database.

Insert_Query

Used to add data to the database.

Update_Query

Used to update data in the database.

Delete_Query

Used to delete data in the database.

Database_Results

Database results.

Transactions

Database transactions.

Parameter_Collection

Query the database parameters.

Query_Condition

Query conditions

Manual_Condition

Manual condition

Array Types

Arrays are used to allocate a number of storage units that have the same name.

In *12d Model*, there are two types of arrays - fixed and dynamic.

Fixed arrays must have their lengths defined when the array is declared. This can either be at compile time when a number is used (e.g. 10) or when a variable which has been given a specific value before the array declaration (e.g. N).

The length of dynamic arrays can vary at any time whilst the macro is running.

See

Fixed Arrays
Dynamic Arrays

Fixed Arrays

A fixed array is defined by giving the size of the array (the number of storage units being set aside) enclosed in the square brackets **[** and **]** immediately after the variable name.

The size can either be a fixed number or a variable that has been assigned a value before the array is defined.

For example, a Real array of size 100 is defined by

```
Real  real_array[100];
```

and a Real array of size N, where N is an Integer variable, is defined by

```
Real  real_array[N];
```

Note that once the array is defined, the size is fixed by the value of N **at the time when the array is defined** - it does not change if N is subsequently modified.

In a macro, the individual items of an array are accessed by specifying an array subscript enclosed in square brackets.

For example, the tenth item of real_array is accessed by real_array[10].

Warning to C++ Programmers

This is **not** the same as C++ where array subscripts start at zero

Dynamic Arrays

For many 12dPL operations, an array of items is required but the size of the array is not known in advance or will vary as the macro runs.

For example, an array may be needed to hold Elements being selected by the user running the macro. The number of Elements selected would not be known in advance and could overflow any fixed array. Hence a fixed array is inconvenient or impossible to use.

To cover these situations, 12dPL has defined **dynamic arrays** that can hold an arbitrary number of items. At any time, the number of items in a dynamic array is known but extra items can be added at any time.

Like fixed arrays, the items in dynamic arrays are accessed by their unique position number. It is equivalent to an array subscript for a fixed array.

But unlike fixed arrays, the items of a dynamic array can only be accessed through function calls rather than array subscripts enclosed in square brackets.

As for an array, the dynamic array positions go from one to the number of items in the dynamic

array.

The dynamic arrays currently supported in 12dPL are

Dynamic_Element

a dynamic array of Elements

Dynamic_Integer

a dynamic array of Integers.

Dynamic_Real

a dynamic array of Reals.

Dynamic_Text

a dynamic array of Texts.

Summary of 12dPL Variable Types

The 12dPL variable types are:

void - only used in functions which return no value

Mathematical Variable Types

Integer - 32 bit integer

Real - 64 bit IEEE Real precision floating point, 14 significant figures

Text - one or more characters

Vector2, Vector3, Vector4 - contain two, three and four Reals respectively

Matrix3, Matrix4 - nine and sixteen Reals respectively

Geometric Construction Variable Types

Point - a three dimensional point

Line - a line between two points

Arc - a helix

Spiral - a transition

Parabola - a parabola

Segment - a Point, Line, Arc, Parabola or Spiral

12d Model Database Handles

Element - a handle for the *12d Model* strings

Tin - a handle for *12d Model* tins

Model - a handle for *12d Model* models

View - a handle for *12d Model* views

Functions, Macro_Function - a handle for *12d Model* functions

Undo_List - a list to combine Undo's

12d Internal Variable Types

Uid - unique identifier for entities in a 12d Model database

Attributes - used as a handle to refer to a 12d Model attribute structure

SDR_Attribute - special attributes used with the *12d* Survey Data Reduction process

Blob - a binary object

Screen_Text -

Textstyle_Data - holds information about a text such as colour, textstyle, rustication

Equality_Label - holds information for labelling text as an Equality

12d Model Interface Variable Types

Menu -holds the data for a user defined 12d Model menu
Panel - holds the data for a user defined 12d Model panel
Widget -
Vertical_Group - holds Widgets that will be placed horizontally in a Panel
Horizontal_Group - holds Widgets that will be placed vertically in a Panel
Widget_Pages -
Overlay_Widget -
Sheet_Panel -
Button - a button on a Panel.
Select_Button -
Angle_Box -

Attributes_Box -
Billboard_Box -
Bitmap_Fill_Box -
Bitmap_List_Box -
Chainage_Box -
Choice_Box -
Colour_Box -
Colour_Message_Box -
Date_Time_Box -
Directory_Box -
Draw_Box -
File_Box -
Function_Box -
Graph_Box -
GridCtrl_Box -
HyperLink_Box -
Input_Box -
Integer_Box -
Justify_Box -
Linestyle_Box -
List_Box -
ListCtrl_Box -
Map_File_Box -
Message_Box -
Model_Box -
Name_Box -
Named_Tick_Box -
New_Select_Box -
New_XYZ_Box -
Plotter_Box -
Polygon_Box -
Real_Box -
Report_Box -
Select_Box - see also New_Select_Box -
Select_Boxes -
Sheet_Size_Box -
Source_Box -
Symbol_Box -
Tab_Box -
Target_Box - // not yet implemented
Template_Box -
Text_Edit_Box -
Text_Style_Box -
Texture_Box -
Tree_Box -
Tree_Page -??
Tick_Box -
Tin_Box -
View_Box -
XYZ_Box - see also New_XYZ_Box

File Interface Variable Types

File -
Map_File -
Plot_Parameter_File -
XML_Document -
XML_Node -

ODBC Database Variable Types

Connection - the connection to the database.
Select_Query - used to retrieve data from the database.
Insert_Query -used to add data to the database.
Update_Query -used to update data in the database.
Delete_Query - used to delete data in the database.
Database_Results - database results.
Transactions - database transactions.
Parameter_Collection - query the database parameters.
Query_Condition - query conditions
Manual_Condition - manual condition

Array Types

Real Array - Real[num] - a fixed array of Reals
Integer Array - Integer[num] - a fixed array of Integers
Text Array - Text[num]- a fixed array of Texts
Dynamic_Element - a dynamic array of Elements
Dynamic_Text - a dynamic array of Texts
Dynamic_Integer - a dynamic array of Integers
Dynamic_Real - a dynamic array of Reals

Constants

There are three kinds of constants (or literals)

Integer Constant
Real Constant
Text Constant

Integer Constant

An integer constant consists of any number of digits.

All integer constants are assumed to be in decimal notation.

Examples of valid integer constants are

1 76875

Real Constant

A Real constant consists of any number of digits ending in a mandatory decimal point, followed by an optional fractional part and an optional exponent part. The exponent part consists of an e or E, and an optionally signed integer exponent.

There can be no spaces between each part of the Real constant.

Valid floating constants are

6. 1.0 1.0e 1.0e+1 1.0e-1 .1e+2

Note that 1e1 is not a valid floating constant.

Text Constant

A Text constant is a sequence of characters surrounded by double quotes.

Valid Text constants are

"1 ""1234 ""!@#$%^&"

A Text constant can also contain escape characters. For example, if you wish to have the " character in a Text constant, you place a \ character in front of it.

"A silly \" symbol" translates to

A silly " symbol

The following escape characters are supported in Text variables:

new-line	NL(LF)	\n
double quote	"	\"
backslash	\	\\

Assignment and Operators

See

Assignment
Binary Arithmetic Operators and Binary Arithmetic Operators for Vectors and Matrices
Relational Operations
Logical Operators
Logical Operators
Increment and Decrement Operators
Bitwise Operators
Assignment Operators

Assignment

Assignment

=	assignment	e.g. x = y

The Assignment **=** is NOT a mathematical equal.

The *Assignment* is to be interpreted as

the expression on the right hand side is evaluated and then the variable on the left is given that value.

So if the same variable occurs on both sides of the assignment, the current value is used in evaluating the right hand side and then the variable is given the new value. For example, the expression

x = x + 1;

means that x is given the new value that is equal to the original value plus 1.

Binary Arithmetic Operators

The binary arithmetic operators are

+ addition

- subtraction

***** multiplication

/ division - note that integer division truncates any fractional part

% modulus: x%y where x and y are integers, produces the integer remainder when x is divided by y

Binary Arithmetic Operators for Vectors and Matrices

The binary arithmetic operators for vectors and matrices are

+ addition

- subtraction

***** multiplication of matrices

***** dot product of vectors

^ cross product of two vectors

where the following combinations are allowed

Vector2 + Vector2 = Vector2 Vector2 - Vector2 = Vector2
Vector3 + Vector3 = Vector3 Vector3 - Vector3 = Vector3
Vector4 + Vector4 = Vector4 Vector4 - Vector4 = Vector4

Real * Vector2 = Vector2 Vector2 * Real = Vector2 Vector2 / Real= Vector2
Real * Vector3 = Vector3 Vector3 * Real = Vector3 Vector3 / Real= Vector2
Real * Vector4 = Vector4 Vector4 * Real = Vector4 Vector4 / Real= Vector4

Vector2 * Vector2 = Real * is the dot product between the two vectors
Vector3 * Vector3 = Real * is the dot product between the two vectors
Vector4 * Vector4 = Real * is the dot product between the two vectors

Vector2 ^ Vector2 = Vector3 ^ is the cross product between the two Vector2 vectors
Note: to form this cross product, the Vector2's are turned into Vector3's by adding the third dimension with value 0.

Vector3 ^ Vector3 = Vector3 ^ is the cross product between the two Vector3 vectors

Matrix3 + Matrix3 = Matrix3 Matrix3 - Matrix3 = Matrix3 Matrix3 * Matrix3 = Matrix3
Matrix4 + Matrix4 = Matrix4 Matrix4 - Matrix4 = Matrix4 Matrix4 * Matrix4 = Matrix4

Real * Matrix3 = Matrix3 Matrix3 * Real = Matrix3 Matrix3 / Real= Matrix3
Real * Matrix4 = Matrix4 Matrix4 * Real = Matrix4 Matrix4 / Real= Matrix4

Vector3 * Matrix3 = Vector3 Note that the Vector3 is treated as a row vector.
Matrix3 * Vector3 = Vector3 Note that the Vector3 is treated as a column vector.

Vector4 * Matrix4 = Vector4 Note that the Vector4 is treated as a row vector.
Matrix4 * Vector4 = Vector4 Note that the Vector4 is treated as a column vector.

A vector of dimension 2, 3 or 4 can be cast to a vector of a higher or a lower dimension.

If casting to a dimension of one higher, the new component is set by default to 1.0.

For example a Vector2 represented by (x,y) is cast to a Vector3 (x,y,1).

When casting to a dimension of one lower, the vector is homogenized and the last component (which has the value 1) is dropped.

For example, a Vector4 represented by (x,y,z,w) is cast to a Vector3 as (x/w,y/w,z/w).

So for example

Vector2 * Matrix3 = Vector3 requires Vector2 say (x,y) to be cast to a Vector3 so that this make sense and the operation is defined as (x,y,1)*Matrix3

Relational Operations

The relational operators are

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Logical Operators

The logical operators are

==	equal to
!=	not equal to
||	inclusive or
&&	and
!	not

Increment and Decrement Operators

The increment and decrement operators are

++	post and pre-increment	e.g. i++ which is shorthand for i = i + 1
--	post and pre-decrement	e.g. i-- which is shorthand for i = i - 1

Bitwise Operators

The bitwise operators are

&	bitwise and
|	bitwise inclusive or
^	bitwise exclusive or
~	one's complement (unary)

Assignment Operators

assignment operator

For some operators ***op***, the assignment operator ***op=*** is supported where for expressions expr1 and expr2:

expr1 op= expr2

means

expr1 = (expr1) op (expr2)

where the supported assignment operators for op= are

+= -= *= /= %=

For example

x += 2 is shorthand for x = x + 2

x *= 2 is shorthand for x = x * 2

Statements and Blocks

An expression such as x = 0 or i++ becomes a **statement** when it is followed by a semi-colon.

Curly brackets **{** and **}** (braces) are used to group declarations and statements together into a **compound statement**, or **block**, so that they are syntactically equivalent to a **single statement**.

There is no semi-colon after the right brace that ends a block.

Blocks can be nested but cannot overlap.

Examples of statements are

```
x = 0;
i++;
fred = 2 * joe + 9.0;
```

An example of a compound statement or block is

```
{
  x = 0;
  i++;
  fred = 2 * joe + 9.0;
}
```

For more information, see Blocks and Scopes .

Flow Control

In a macro, the normal processing flow is that a statement is processed and then the following statement is processed.

The ***flow control*** statements of a language change the **order** in which statements are processed.

12dPL supports a subset of the C++ flow control statements but before they can be examined, we need to look at logical expressions.

Logical Expressions

Many flow control statements include expressions that must be *logically evaluated*.

That is, the flow control statements use expressions that must be evaluated as being either ***true*** or ***false***.

For example,

a is equal to b	a == b
a is not equal to b	a != b
a is less than b	a < b

Following C++, 12dPL extends the expressions that have a truth value to any expression that can be evaluated arithmetically by the simple rule:

> **an expression is considered to be true if its value is non-zero, otherwise it is considered to be false**.

Hence the truth value of an arithmetic expression is equivalent to:

"value of the expression" is not equal to zero

For example, the expression

a + b

is true when the sum a+b is non-zero.

Any expression that can be evaluated logically (that is, as either true or false) will be called a **logical expression**.

12dPL Flow Controls

The flow control statements supported by 12dPL are listed below and each will be defined in the following sections

if, else, else if

Conditional Expression
Switch
While Loop
For Loop
Do While Loop
Continue
Break
Goto and Labels

if, else, else if

12dPL supports the standard C++ **if, else** and **else if** structures.

if

```
if (logical_expression)
      statement
```

is interpreted as:

If logical_expression is true then execute the statement.

If logical_expression is false then skip the statement.

For example

```
if (x == 5) {
     x = x + 1;
     y = x * y;
}
```

Notice that in this example the **statement** consists of the block

```
{ x = x + 1;
  y = x * y;
}
```

The expressions in the block are only executed if x is equal to 5.

else

```
if (logical_expression)
    statement1
else
    statement2
```

is interpreted as

If logical_expression is true then execute statement1.

If logical_expression is false then execute statement2.

else if

```
if (logical_expression1)
    statement1
else if (logical_expression2)
    statement2
else
    statement3
```

is interpreted as

If logical_expression1 is true then execute statement1.

If logical_expression1 is false then

(if logical_expression2 is true then execute statement2 otherwise execute statement3)

Conditional Expression

12dPL supports the standard C++ **conditional** expression:

logical_expression ? expression : expression2

is interpreted as

```
if (logical_expression) then
    expression1
else
    expression2
```

For example,

```
y = (x >= 0) ?  x : -x;
```

means that y is set to x if x is greater than or equal to zero, otherwise it is set to -x. Hence y is set to the absolute value of x.

Switch

12dPL supports a **switch** statement.

The **switch** statement is a multi-way decision that tests a value against a set of constants and branches accordingly.

In its general form, the switch structure is:

```
switch (expression) {
    case constant_expression : { statements }
    case constant_expression : { statements }
    default  : { statements }
}
```

Each case is labelled by one of more constants.

When **expression** is evaluated, control passes to the case that matches the expression value.

The case labelled **default** is executed if the expression matches none of the cases. A default is optional; if it isn't there and none of the cases match, no action takes place.

Once the code for one case is executed, execution falls through to the next case unless explicit action is taken to escape using **break**, **return** or **goto** statements.

A **break** statement transfers control to the end of the switch statement (see Break).

Warning

Unlike C++, in 12dPL the statements after the **case constant_expression**: *must be enclosed in curly brackets* ({}).

Example

An example of a switch statement is:

```
switch (a) {
        case 1 : {
          x = y;
          break;
        }
        case 2: {
          x = y + 1;
          z = x * y;
        }
        case 3: case 4: {
          x = z + 1;
          break;
        }
        default : {
          y = z + 2;
          break;
        }
}
```

Note

If control goes to case 2, it will execute the two statements after the case 2 label and then continue onto the statements following the case 3 label.

Restrictions

1. Currently the switch statement only supports an **Integer**, **Real** or **Text** expression. All other expression types are not supported.
2. Statements after the **case** constant_expression: must be enclosed in curly brackets ({}).

While Loop

12dPL supports the standard C++ **while** statement.

```
while (logical_expression)
    statement
```

is interpreted as:

(a) If **logical_expression** is true, execute **statement** and then test the **logical_expression** again.

(b) repeat (a) until the **logical_expression** is false.

For example, in

```
x = 10.0;
product = 1.0;
while (x > 0) {
  product = product * x;
  x = x - 1;
}
```

the block

```
{  product = product * x;
  x = x - 1;
}
```

will be repeated until x is not greater than zero (i.e. until x is less than or to equal zero).

For Loop

12dPL supports the standard C++ **for** statement.

```
for (expression1;logical_expression;expression2)
statement
```

is interpreted as:

```
expression1;
while (logical_expression) {
    statement;
    expression2;
}
```

In long hand, this means:

(a) first execute expression1.

(b) if logical_expression is true, execute statement and expression2 and then test logical_expression again.

(c) repeat (b) until the logical_expression is false.

For example

```
j = 0;
for (i = 1; i <= 10; i++)
   j = j + i;
```

would sum the numbers 1 through to 10.

Notes

1. Any of the three parts **expression1**, **logical_expression** and **expression2** can be omitted from the **for** statement but the semi-colons must remain.
2. If **expression1** or **expression2** is omitted, it is simply dropped from the expansion.
3. If the test, **logical_expression** is missing, it is taken as permanently true.

Restrictions

1. At this stage for(;;) is not allowed
2. At this stage, please avoid having more than one statement for expression2.

For example, avoid

for(expression1;logical_expression;i++,j++)

because j++ will not be evaluated correctly.

Do While Loop

12dPL supports the standard C++ **do while** statement:

```
do
        statement
while (logical_expressions);
```

is interpreted as:

Execute **statement** and then evaluate **logical_expression**.

If **logical_expression** is true, execute **statement** and then test **logical_expression** again.

This cycle continues until **logical_expression** is false.

For example

```
i = 0;
        do {
            x = x + 1;
            i++;
        } while (i < 10);
```

Continue

The **continue** statement causes the next iteration of the enclosing **for**, **while** or **do while** loop to begin.

In the **while** and **do while**, this means that the test part is executed immediately.

In the **for**, control passes to the evaluation of expression2, normally an increment step.

Important Note
The **continue** statement applies only to loops. A **continue** inside a **switch** inside a loop causes the next loop iteration.

Break

break is used to exit from a **do**, **for**, or **while** loop, bypassing the normal loop condition. It is also used to exit from a **switch** statement.

In a **switch** statement, **break** keeps program execution from "falling through" to the next **case**. A **break** statement transfers control to the **end** of the **switch** statement.

A break only terminates the **for**, **do**, **while** or **switch** statement that contains it. It will not break out of any nested loops or **switch** statements.

Goto and Labels

12dPL supports the standard C++ **goto** and **labels**.

A **label** has the same form as a variable name and is followed by a colon. It can be attached to any statement in a function. A label name must be unique within the function.

A **goto** is always followed by a **label** and then a semi-colon.

When a **goto** is executed in a macro, control is immediately transferred to the statement with the appropriate **label** attached to it. There may be many gotos with the same label in the function.

An example of a **label** and a **goto** is:

```
for ( ... ) {
        ...
        goto error;
        ...
      }
      ...
      error:
      statements
```

When the **goto** is executed, control is transferred to the **label error**.

Note

A **goto** cannot be used to jump over any variables defined at the same nested level as the **goto**. Extra curly bracket ({}) may need to be placed around the offending code to increase its level of nesting.

Precedence of Operators

12dPL has the same precedence and associativity rules as C++. For convenience, the order is summarized in the table below.

In the table,

operators on the same line have the **same precedence**;

rows are in **order of decreasing precedence**.

For example, *, / and % all have the same precedence which is higher than that of binary + and -.

The "operator" () refers to function call.

Operators	**Associativity**
() []	left to right
! ~ ++ -- + - * &	**right to left**
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?	**right to left**
= += -= *= /= %= &= ^= |=	**right to left**

Unary + and - have higher precedence than the binary forms.

Preprocessing

You can include other files by the command

#include "filename"

The example below shows how to include file "a.h" into "b.4dm.

// file a.h

```
Point Coord(Real x,Real y,Real z)
{
  Point p;  Set_point(p,x)   Set_point(p,y);  Set_point(p,z);
  return(p);
}
```

// file b.4dm

```
#include "a.h"
void main()
  {
  Point p = Coord(10.0,20.0,2.34);           // create a point
}
```

The above example is **equivalent** to the following one file:

```
Point Coord(Real x,Real y,Real z)
{
  Point p;  Set_point(p,x);  Set_point(p,y);  Set_point(p,z);
  return(p);
}
void main()
{
  Point p = Coord(10.0,20.0,2.34);            // create a point
}
```

3 Functions

See Functions
See Main Function
See User Defined Functions
See Function Prototypes
See Automatic Promotions
See Passing by Value or by Reference
See Overloading of Function Names
See Recursion
See Assignments Within Function Arguments
See Blocks and Scopes

Functions

Functions can be used to break large computing tasks into smaller ones and allow users to build on software that already exists.

Basically a program is just a set of definitions of variables and functions. Communication between the functions is by function arguments, by values returned by the functions, and through global variables (see the section Blocks and Scopes).

The 12dPL program file must contain a starting function called **main** as well as zero or more ***user defined*** functions.

User defined functions must occur in the file before they are used in the program file unless a ***Function Prototype*** is included before the function is used. If this occurs then the user defined function can be defined anywhere in the file. *See* Function Prototypes .

The syntax for user defined functions will be described in the following sections. *See* User Defined Functions .

A large number of functions are supplied with 12dPL to make tasks easier for the program writer. These 12dPL supplied functions are predefined and nothing special is needed to use them. The 12dPL supplied functions will all be defined later in the manual.

In 12dPL, function names must start with an alphabetic character and can consist of upper and/ or lower case alphabetic characters, numbers and underscores (_).

There is no restriction on the length of function names. Function names cannot be the same as any of the 12dPL keywords or variable names in the program.

12dPL function names are case sensitive.

Note

All 12dPL supplied functions begin with a capital letter to help avoid clashes with any user variable names.

Main Function

A 12dPL program must contain a special function called **main**. This function is the designated start of the program.

The main function is simply a header **void main ()** followed by the actual program code enclosed between a start brace { and an end brace }.

Hence the function called **main** is a header followed by a block of code:

```
void  main ()
      {
         declarations and statements
         i.e. program code
      }
```

When a program is run, the **entry point** to the program file is at the beginning of the function called **main**.

Hence every program file must have one and only one function called **main**.

The function **main** is terminated when either

(a) the last line of code in the function is run

or

(b) a return statement
```
return;
```
is executed in the function **main**.

The function **main** is usually referred to as the **main function**.

User Defined Functions

As well as the main function, a program file can also contain ***user defined*** functions.

Like the main function, *user defined functions* consist of a header followed by the program code enclosed in braces.

However the header for a user defined function must include a ***return type*** for the function and the ***order*** and ***variable types*** for each of the ***parameters of the function.***

Hence each user defined function definition has the form

```
return-type  function-name(argument declarations)
        {
            declarations and statements
        }
```

For example, a function called "user_function" which has a return type of Integer and parameters of type Integer, Real and Element could be:

```
Integer user_function (Integer fred, Real joe, Element tom)
        {
            program code
        }
```

Return Statement

The **return** statement in a function is the mechanism for returning a value from the called function to its caller using the *return-type* of the function.

The general definition of the return statement is:

```
return expression;
```

For a function with a *void* return-type (a void function), the expression must be empty. That is, for a void return-type you can only have return and no expression since no value can be returned.

Thus for a void function the return statement is

```
return;
```

Also for a void function, the function will implicitly return if it reaches the end of the function without executing a return statement.

The function *main* is an example of a void function.

For a function with a non-void return-type (a non-void function), the expression after the return must be of the same type as the return type of the function. Hence any function with a non-void return-type must have a return statement with the correct expression type.

The calling function is free to ignore the returned value.

Restrictions

Unlike C++, in 12dPL the last statement for a function with a non-void return type must be a *return* statement.

Array Variables as Function Arguments

Arrays can be used a function arguments.

The declaration of an **array variable** as a function argument consists of the array variable type followed by the array name and an empty set of square brackets ([]).

For example, the function

```
Integer user_function (Integer fred, Real joe[])
        {
            program code
        }
```

has a Real array as the second argument.

Function Prototypes

Since all functions and variables must be defined before they are used, then for any user defined functions either

(a) the function must appear in the file before it is called by another function

or

(b) a **prototype** of the function must be declared before the function is called.

A function **prototype** is simply a declaration of a function which specifies:

1. the function name
2. the function return type

and

3. the order and type of all the function parameters.

A function prototype looks like the function header except that it is ***terminated by a semi-colon*** instead of being followed by braces and the function code. Also, the variable names need not be included in the function prototype.

For example, two prototypes for the function user_function are

```
Integer user_function (Integer fred, Real joe, Element tom);
Integer user_function (Integer, Real, Element);
```

Thus **prototypes** are simply a method for defining the return type and the arguments and the argument types of a function so that the function can be used in a program before the code for the function has been found in the file.

Notes

(a) The function *main* and any 12dPL supplied functions do not have to be defined or prototyped by the user.

(b) A function prototype can occur more than once in a file.

(c) The *main* function and all the user defined functions must exist in either the one file or be included from other files using the #include statement.

Automatic Promotions

If needed, the following promotions are automatically made in the language:

From	To
Integer	Real
Real	Integer
Model	Dynamic_Element
Element	Dynamic_Element
Tin	Element, Dynamic_Element
Point	Segment
Line	Segment
Arc	Segment
Vector2	Vector3
Vector3	Vector4
Vector3	Vector2
Vector4	Vector3

These automatic promotions can occur

(a) when looking for functions with matching argument types

or

(b) for converting expressions in a return statement to the correct return-type required for the function.

Hence in the following example, the variable x is automatically promoted to a Real for use by the function silly.

```
Real silly(Real x) { return(x+1); }
void main()
{
    Integer   x = 10;
    Real y = silly(x);
}
```

Passing by Value or by Reference

12dPL follows C++ in that a function argument can be passed "by value" or "by reference".

Passed by Value

If a function argument is ***passed by value,*** then calling function only passes a temporary copy of the variable to the called function. Any modification of this temporary variable inside the called function will not affect the value of the variable in the calling function.

Hence in ***passed by value*** transfer of the argument value is only in one direction - ***from the calling function into the called function***.

In 12dPL, the default for non-array arguments is ***passed by value.***

Passed by Reference

However it is also possible to ***pass down the actual variables from the calling function*** to the called function. This is termed ***passed by reference.***

If a function argument is *passed by reference* then any modification made to the passed variable within the called function will be ***modifying the original*** argument in the calling function.

Hence in ***passed by reference*** transfer of the argument value is in two directions and any modifications to the passed variable withing the called function will affect the variable in the calling function.

To denote that a variable is to be **passed by reference**, an ampersand (**&**) is placed after the type of the argument in the function definition and any function prototypes.

For example, in the function user_function1, the variables *fred* and *tom* are to be passed by value and the variable *joe* is to be passed by reference. The function code is:

```
Integer user_function1 (Integer fred, Real &joe, Element tom)
{
        program code
 }
```

Matching prototypes for user_function1:

```
Integer user_function1 (Integer fred, Real& joe, Element tom);
Integer user_function1 (Integer fred, Real &joe, Element tom);
Integer user_function1 (Integer fred, Real & joe, Element tom);
Integer user_function1 (Integer, Real&, Element);
Integer user_function1 (Integer, Real &, Element);
```

If a called function is to return a value to the calling function via one of its arguments, then the argument **must** be passed by reference.

To clarify the difference between *passed by value* and *passed by reference*, consider the following examples:

```
void bad_square(Integer x) { x = x*x;}// x is passed by value
void main()
{
    Integer x = 10;
    bad_square(x);
    // pass by value
    // x still equals 10
}
void square(Integer &x) { x = x*x;}   // x is passed by reference
```

```
void main ()
{
   Integer x = 10;
   square(x);
// pass by reference
      // x now equals 100
}
```

Notes

(a) Fixed arrays are always *passed by reference*.

(b) In Fortran and Basic, all arguments are "pass by reference"

(c) In C++ and Pascal, arguments can be passed by value or by reference

Overloading of Function Names

In 12dPL, if you have a number of functions that have the same name but with a different number of arguments and/or different argument types, there is no need to give each function a different name.

As long as the argument numbers or argument types differ in some way, 12dPL will determine the correct function to call.

For example, three functions called swap have been defined but they are all different because they have differing argument types.

```
void swap(Integer &x,Integer &y) { Integer  z = x; x = y; y = z;}
void swap(Real &x,Real &y) { Real z = x; x = y; y = z;}
void swap(Text &x,Text &y) { Text z = x; x = y; y = z;}
void main()
{
        Integer ix = 1   , iy = 2;
        Real    rx = 1.0  , ry = 2;          // automatic promotion of 2 to 2.0
        Text    tx = "1" , ty = "2";
        swap(ix,iy);
        swap(rx,ry);
        swap(tx,ty);
}
```

Note however that in some cases there may be more than one function that can be used. This is especially true when promotions are required to match the function.

If more than one match is found, the compiler will issue an error and display the functions that match. If no match is found, the compiler will display all functions which overload the specified function name.

```
void swap(Integer &x,Integer &y) { Integer   z = x; x = y;}
void swap(Real &x,Real &y) { Real z = x; x = y;}
void swap(Text &x,Text &y) { Text   z = x; x = y;}
void main()
{
   Integer   ix = 1   , iy = 2;
   Real      rx = 1    , ry = 2;
   Text      tx = "1"  , ty = "2";
   swap(ix,ry);       //    2 matches
                      //    swap(Integer &,Integer &)
                      //    swap(Real &,Real &)
   swap(tx,ry); // no match
}
```

An example of overloaded functions is *redraw_views* in Example 6 .

WARNING FOR C++ PROGRAMMERS

Since there is no explicit cast operator, the only way to cast is to introduce a temporary variable and use an assignment. For example, to fix the error in the above example where two matches occur, assign ry to an intermediate variable.

```
Integer iry = ry;
swap(ix,iry);          // ok, it uses swap(Integer &,Integer &)
Real rix = ix;
swap(rix,ry);          // ok, it uses swap(Real &,Real &)
```

Recursion

Recursion for functions is supported.

For example,

```
int fib(int n)
{
   return n < 2 ? 1 : fib(n - 1) + fib(n - 2);
}
```

Assignments Within Function Arguments

In 12dPL, assignments are not allowed within function arguments.

For example, in the following code fragment, y = 10.0 does not assign 10.0 to y.

```
Real silly(Real x) { return(x); }
void main()
{
   Real y;
   Real z = silly(y=10.0);
 }
```

To actually assign 10.0 to y, enclose the statement in round brackets (and). That is

```
Real z = silly((y=10.0));
```

assigns 10.0 to y and z.

Assignment within a call argument is being reserved for future use by 12dPL for functions with **named arguments**.

Blocks and Scopes

As noted earlier, a block is a code fragment contained within the characters { and } (braces).

Blocks can be nested. That is, a block may contain one or more sub-blocks. However, blocks cannot overlap.

Hence a closing brace } is always paired with the closest previous unpaired open brace {.

In the example below, block a is also the function body of **main**. Blocks **b** and **c** are sub-blocks of block **a**.

```
void main()
{
  Integer a = 1;
  {

    Integer x = 10;
    Print(x+a);  Print("\n");
  }

  {

    Real x = 10;
    Print(x+a);  Print("\n");
   }
}
```

block b

block c

block a

The **scope** of a name is the region of the program text within which the name's characteristics are understood.

In 12dPL, there are three kinds of scope: local, function, and global (file).

Local — A name declared in a block is local to that block and can be used in the block, and in any blocks enclosed by the block after the point of declaration of the name.

Function — Labels can be used anywhere in the function in which they are declared, Only labels have function scope.

Global — A name declared outside all functions has global (or file) scope and can be used anywhere after its point of declaration.

In 12dPL, variables with global (file) scope must be declared in an enclosing set of braces.

There can be more than one global section.

Hence, in the following example

```
{ Integer    an_integer;
  Real       a_real;
  Element    an_element;
}
void main()
{                                        --*
```

```
  fred:Integer a = 1;                          |
  {                                --*         |
    Integer x = 10;                  |         |
    an_integer = 20;                 | block b |
    Print(x+a+an_integer);>          |         |
    Print("\n");                     |         |
  }                                --*         |
                                               | block a
  {                                --*         |
    Real x = 10;                     | block c |
    Print(x+a);  Print("\n");        |         |
  }                                 --*         |
  goto fred;                                   |
}                                            --*
```

the variables an_integer, a_real and an_element have global scope and can be used anywhere in the file after their definition.

The Integer variable "a" has local scope and because of the position in the block, can be used inside blocks b and c.

The Integer variable "x" is defined in block b and has local scope. It is not usable outside that block.

The Real variable "x" is defined in block c and has local scope. It is not usable outside that block.

WARNING

A variable name may be hidden by an explicit declaration of that same name in an enclosed block.

Because of the potential for confusion, it is best to avoid variable names that are the same as a variables in an outer block.

4 Locks

Because ***12d Model*** allows operations to be queued, it is possible that an Element may be selected at the same time by more that one macro or ***12d Model*** operation.

To prevent data corruptions, **locks** are automatically used within ***12d Model***.

When an Element is selected, a lock is placed on the element and later removed when the element is released.

Any locks on an element will prevent the Element from being deleted or modified until the locks are removed by the other operations which automatically placed the locks.

If a macro tries to delete a locked Element, a **macro exception** panel is placed on the screen to alert the user that the operation is currently prevented because of a lock on the Element.

The panel gives the user the chance to

skip jump over the current macro instruction

retry retry the instruction to see if the Element is still locked

abort stop the macro.

The usual scenario is that when an Element is locked and an **exception panel** appears on the screen, the user simply completes the other operations that have locked the Element and then continue with the macro by selecting the retry button.

5 12dPL Library Calls

The 12dPL Library Calls section consists of descriptions of all the supplied 12dPL functions and a number of examples.

For each function, the full function ***prototype*** is given

return-type function-name (function-arguments)

followed by a description of the function.

Note that to be able to ***return*** a value for a function argument to the calling routine, the argument must be passed by reference and hence will have an ampersand (&) in the function prototype.

For example,

Integer test (Integer fred, Real &joe, Element tom)

specifies a function called **test** with return type **Integer**, two arguments, fred and tom, that are passed by value and one argument, joe, that is passed by reference and hence capable of ***returning*** a value from the function.

See Creating a List of Prototypes
See Function Argument Promotions
See Function Return Codes
See Command Line-Arguments
See Array Bound Checking
See Exit
See Angles
See Text
See Textstyle Data
See Maths
See Random Numbers
See Vectors and Matrices
See Triangles
See System
See Uid's
See Input/Output
See Menus
See Dynamic Arrays
See Points
See Lines
See Arcs
See Spirals and Transitions
See Parabolas
See Segments
See Segment Geometry
See Colours
See User Defined Attributes
See Folders
See 12d Model Program and Folders
See Project
See Models
See Views
See Elements
See Tin Element
See Super String Element
See Examples of Setting Up Super Strings
See Super Alignment String Element

See Arc String Element
See Circle String Element
See Text String Element
See Pipeline String Element
See Drainage String Element
See Feature String Element
See Interface String Element
See Face String Element
See Plot Frame Element
See Strings Replaced by Super Strings
See Alignment String Element
See General Element Operations
See Strings Replaced by Super Strings
See Alignment String Element
See General Element Operations
See Creating Valid Names
See XML
See Map File
See Panels and Widgets
See General
See Utilities
See 12d Model Macro_Functions
See Plot Parameters
See Undos
See ODBC Macro Calls
See Macro Console

Creating a List of Prototypes

The 12dPL compiler is a program called *cc4d* that is installed in nt.x64 and nt.x32 (see (b) Compiling from Outside 12d Model).

cc4d can also be used to generate a list of prototypes for all the supplied 12dPL Library calls as both a text list and as an XML version.

To generate the list of prototypes use:

cc4d -list *prototype_list_file_name*

For example, type in

(a) when running a 64-bit 12d.exe on a 64-bit Microsoft Windows Operating System

"C:\Program Files\12d\12dmodel\10.00\nt.x64\cc4d" -list prototypes.4d

(b) or when running a 32-bit *12d.exe* on a 32-bit Microsoft Windows OS.

"C:\Program Files\12d\12dmodel\10.00\nt.x86\cc4d" -list prototypes.4d

(c) or when running a 32-bit *12d.exe* on a 64-bit Microsoft Windows OS.

"C:\Program Files (x86)\12d\12dmodel\10.00\nt.x86\cc4d" -list prototypes.4d

Each function prototype has a unique number called a Library Identity (Library Id). The Library Id is an integer starting at 1 and is incremented by 1 whenever a new function call is added to the 12dPL Library. The function prototypes are written out in Library Id order so the newest function calls will be at the bottom of the list.

Function Argument Promotions

Because 12dPL has automatic variable type promotions and function overloading, many of the 12dPL functions apply to a wider range of cases than the function definition may at first imply.

For example, because Model will promote to a Dynamic_Element, the Triangulate function

```
Integer Triangulate(Dynamic_Element de,Text tin_name,Integer tin_colour,Integer preserve,
                    Integer bubbles,Tin &tin)
```

also covers the case where a Model is used in place of the Dynamic_Element **de**.

That is, the function definition automatically includes the case

```
Integer Triangulate(Model model,Text tin_name,Integer tin_colour,Integer preserve,
                    Integer bubbles,Tin &tin)
```

Automatic Promotions

The 12dPL automatic promotions are

From	**To**
Integer	Real
Real	Integer
Model	Dynamic_Element
Element	Dynamic_Element
Tin	Element, Dynamic_Element
Point	Segment
Line	Segment
Arc	Segment

Function Return Codes

Many of the 12dPL functions have an Integer function return code that is used as an error code.

For most functions, the function return code is

zero if there were no errors when executing the function

and

non-zero if an error occurs.

This choice is to allow for future reporting of different types of errors for the function.

The only exceptions to this rule are the existence routines such as:

File_exists, Colour_exists, Model_exists, Element_exists, Tin_exists, View_exists, Template_exists, Match_name and Is_null.

They return

a non-zero value if the object exists

and

a zero value if the object does not exist.

This is to allow the existence functions to be used as logical expressions that are true if the object exists. For example

```
if(File_exists("data.dat")) {
    ...
}
```

Command Line-Arguments

When a 12d Model program is invoked, command-line arguments (parameters) can be passed down and accessed from within the program.

The command-line information is simply typed into the **macro arguments** field of the **macro run** panel.

The command-line is automatically broken into space separated tokens which can be accessed from within the program.

For example, if the **macro arguments** panel field contained

three "space separated" tokens

then the three tokens

"**three**", "**spaced separated**" and "**tokens**"

would be accessible inside the program.

As an example of how to use the command line argument calls:

```
Integer argc = Get_number_of_command_arguments();
if(argc > 0) {
   Text arg;
   Get_command_argument(1,arg);
   if(arg == "-function_recalc") {
         . . .
```

Get_number_of_command_arguements()

Name

Integer Get_number_of_command_arguments()

Description

Get the number of tokens in the program command-line.

The number of tokens is returned as the function return value.

For some example code, see Command Line-Arguments .

ID = 432

Get_command_argument(Integer i,Text &argument)

Name

Integer Get_command_argument(Integer i,Text &argument)

Description

Get the **i**'th token from the command-line.

The token is returned by the Text **argument**.

The arguments start from 1.

A function return value of zero indicates the i'th argument was successfully returned.

For some example code, see Command Line-Arguments .

ID = 433

Array Bound Checking

A programming error that is often difficult to find is when an array is called with a index that is outside the defined range of the array indices.

For example, the Integer array i_array defined by:

```
Integer    i_array[100]
```

only exists for indices 1 to 100.

That is, only i_array[1], i_array[2], ..., i_array[99], i_array[100] are valid.

Using i_array[101] or i_array[0] will cause problems.

To help overcome this problem, the 12dPL compiler has full array checking. That is, passing in an invalid array index will result in the program terminating with an error message written to the Output Window giving the line number where the overrun occurs, the actual size of the array and the index that was passed into the array.

For example

```
line: 1234 : stack array bounds error - size=10 index=12 array_base=1
```

Exit

12dPL program functions are normally terminated by a return statement or by reaching the closing bracket of the function with void function return type.

In the case of the main function, the program simply terminates.

For other user defined functions, control passes back to the calling function which then continues to execute.

However, 12dPL also has special exit routines that will immediately stop the execution of the program and write a message to the macro console panel. The exit functions are

Exit(Integer exit_code)

Name

void Exit(Integer exit_code)

Description

Immediately exit the program and write the message

macro exited with code **exit_code**

to the **information/error message area** of the macro console panel.

ID = 417

Exit(Text msg)

Name

void Exit(Text msg)

Description

Immediately exit the program and write the message

macro exited with message **msg**

to the **information/error message area** of the macro console panel.

ID = 418

Destroy_on_exit( )

Name

void Destroy_on_exit( )

Description

Destroy current macro console panel when exit the program.

ID = 815

Retain_on_exit( )

Name

void Retain_on_exit( )

Description

Retain current macro console panel on the screen after exit the macro.

ID = 816

Angles

Pi

The value of **pi** is commonly used in geometric macros so functions are provided to return the value of pi, pi/2 and 2*pi.

The functions are

Real Pi() the value of pi

ID = 192

Real Half_pi() the value of half pi

ID = 193

Real Two_pi() the value of 2 * pi

ID = 194

Types of Angles

In **12dPL**, the following definitions for the measurement of angles are used:

angle angles are measured in an anti-clockwise direction from the horizontal axis. The units for angles are radians.

sweep angle used for arcs - measured in a clockwise direction from the line joining the centre to the arc start point. The units for sweep angles are radians.

bearing bearings are measured in a clockwise direction from the vertical axis (north). The units for bearings are radians.

degrees degrees refers to decimal degrees

dms refers to degrees, minutes and seconds.

hp_degrees refers to degrees, minutes and seconds but using the notation ddd.mmssfff where

ddd are the whole degrees

. separator between degrees and minutes

mm whole minutes

ss whole seconds

fff fractions of seconds (as many as needed)

In **12dPL**, functions are provided to convert between the different angle types.

The return type for each of the functions is **Integer** and the return value is an **error indicator**.

If the return value is zero, the function call was successful.

If the return value is non-zero, an error occurred.

Integer Radians_to_degrees(Real rad,Real &deg)

ID = 203

Integer Degrees_to_radians(Real deg,Real &rad)

ID = 204

Integer Radians_to_hp_degrees(Real rad,Real &hp_deg)

ID = 205

Integer Hp_degrees_to_radians(Real hp_deg,Real &rad)

ID = 206

Integer Degrees_to_hp_degrees(Real deg,Real &hp_deg)

ID = 207

Integer Hp_degrees_to_degrees(Real hp_deg,Real &deg)

ID = 208

Integer Degrees_to_dms(Real deg,Integer &dd,Integer &mm,Real &ss)

ID = 209

Integer Dms_to_degrees(Integer dd,Integer mm,Real ss,Real &deg)

ID = 210

Integer Angle_to_bearing(Real angle,Real &bearing)

ID = 211

Integer Bearing_to_angle(Real bearing,Real &angle)

ID = 212

Text

A Text variable **text** consists of zero or more characters (spaces or blanks are valid characters).

The length of a Text is the total number of characters including any leading, trailing and embedded spaces. For example, the length of " fr ed " is seven.

Each character in the Text has a unique **position** or **index** which is defined to be the number of characters plus one that it is from the start of the Text. For example in " fr ed ", the index or position of "e" is five.

Hence parts of a Text (sub-Texts) can be easily referred to by giving the start and end positions of the part. For example, the sub-Text from start position three to end position five of " fr ed " is "r e".

12dPL provides functions to construct Texts and also work with parts of a Texts (sub-Text).

Text and Operators

The operators + += < > >= <= == != can be used with Text variables.

The + operator for Text variables means that the variables are concatenated. For example, after

```
Text        new = "fred" + "joe";
```

the value of new is "fredjoe".

When Text is used in equalities and inequalities such as <, <=, >, >= and ==, the ASCII sorting sequence value is used for the Text comparisons.

General Text

Text_length(Text text)

Name

Integer Text_length(Text text)

Description

The function return value is the length of the Text **text**.

ID = 381

Numchr(Text text)

Name

Integer Numchr(Text text)

Description

The function return value is the position of the last non-blank character in the Text **text**.

If there are no non-blank characters, the return value is zero.

ID = 478

Text_upper(Text text)

Name

Text Text_upper(Text text)

Description

Create a Text from the Text **text** that has all the alphabetic characters converted to upper

-case.

The function return value is the upper case Text.

ID = 383

Text_lower(Text text)

Name

Text Text_lower(Text text)

Description

Create a Text from the Text text that has all the alphabetic characters converted to lower-case.

The function return value is the lower case Text.

ID = 384

Text_justify(Text text)

Name

Text Text_justify(Text text)

Description

Create a Text from the Text **text** that has all the leading and trailing spaces removed.

The function return value is the justified Text.

ID = 382

Find_text(Text text,Text tofind)

Name

Integer Find_text(Text text,Text tofind)

Description

Find the first occurrence of the Text **tofind** within the Text **text**.

If **tofind** exists within **text**, the start position of **tofind** is returned as the function return value.

If **tofind** does not exist within **text**, a start position of zero is returned as the function return value.

Hence a function return value of zero indicates the Text **tofind does not** exist within the Text **text**.

ID = 380

Get_subtext(Text text,Integer start,Integer end)

Name

Text Get_subtext(Text text,Integer start,Integer end)

Description

From the Text **text**, create a new Text from character position **start** to character position **end** inclusive.

The function return value is the sub-Text.

ID = 379

Set_subtext(Text &text,Integer start,Text sub)

Name

void Set_subtext(Text &text,Integer start,Text sub)

Description

Set the Text **text** from character position **start** to be the Text **sub**. The existing characters of **text** are overwritten by sub.

If required, Text **text** will be automatically extended to fit **sub**.

If **start** is greater than the length of **text**, **text** will be extended with spaces and **sub** inserted at position **start**.

There is no function return value.

ID = 389

Insert_text(Text &text,Integer start,Text sub)

Name

void Insert_text(Text &text,Integer start,Text sub)

Description

Insert the Text **sub** into Text **text** starting at position **start**. The displaced characters of **text** are placed after **sub**.

The Text text is automatically extended to fit **sub** and no characters of **text** are lost.

There is no function return value.

ID = 390

Text Conversion

From_text(Text text, Integer &value)

Name

Integer From_text(Text text, Integer &value)

Description

Convert the Text **text** to an Integer **value**. The text should only include digits.

The function return value is zero if the conversion is successful.

ID = 30

From_text(Text text, Integer &value,Text format)

Name

Integer From_text(Text text, Integer &value,Text format)

Description

Convert the Text **text** to an Integer **value** using the Text **format** as a C++ format string.

The function return value is zero if the conversion is successful.

Warning

The user is responsible for ensuring that the format string is sensible.

ID = 387

From_text(Text text, Real &value)

Name

Integer From_text(Text text, Real &value)

Description

Convert the Text **text** to a Real **value**.

The function return value is zero if the conversion is successful.

ID = 31

From_text(Text text, Real &value,Text format)

Name

Integer From_text(Text text, Real &value,Text format)

Description

Convert the Text **text** to a Real **value** using the Text **format** as a C++ format string.

The function return value is zero if the conversion is successful.

Warning

The user is responsible for ensuring that the format string is sensible.

ID = 388

From_text(Text text,Text &value,Text format)

Name

Integer From_text(Text text,Text &value,Text format)

Description

Convert the Text **text** to a Text **value** using the Text **format** as a C++ format.

The function return value is zero if the conversion is successful.

Warning

The user is responsible for ensuring that the format string is sensible.

ID = 392

From_text(Text text,Dynamic_Text &dtext)

Name

Integer From_text(Text text,Dynamic_Text &dtext)

Description

Break the Text **text** into separate words (tokens) and add the individual words to the Dynamic_Text **dtext**.

Free format is used to break text up individual words EXCEPT for characters between matching double quotes ".

Hence any characters (including blanks) between matching double quotes are considered to be one word, and one or more spaces are the separators between individual words.

For example, in

This is "an example"

there are three words - "This", "is", and "an example".

Note that there is more than one space between "This" and "is" but they are ignored and taken to be only one space.

The function return value is the number of words returned in **dtext**.

ID = 377

From_text(Text text,Integer delimiter,Integer separator,Dynamic_Text &text)

Name

Integer From_text(Text text,Integer delimiter,Integer separator,Dynamic_Text &text)

Description

Break the Text **text** into separate words (tokens) and add the individual words to the Dynamic_Text **dtext**.

The character used to break up the text into individual words is given by the Integer **separator**.

Any characters between matching the character given by the Integer **delimiter** (including any characters equal to **separator**) are considered to be one word.

For example, if the delimiter is double quotes **"** and the separator is a semi-colon **;** then

This;is;"an;example"

has three words - "this", "is", and "an;example".

Note: **delimiter** and **separator** are Integers and can be specified by the actual number of a character or by giving the actual character between single quotes.

For example,

separator = 32 is the number for a space

separator = ' '	is the number for a space
separator = 'a'	will be the number for the letter **a**
separator = '\t'	will be the number for a tab

The function return value is the number of words returned in **dtext**.

ID = 2105

To_text(Integer value)

Name

Text To_text(Integer value)

Description

Convert the Integer **value** to text.

The function return value is the converted value.

ID = 32

To_text(Integer value,Text format)

Name

Text To_text(Integer value,Text format)

Description

Convert the Integer **value** to text using the Text **format** as a C++ format string.

The function return value is the converted value.

Warning

The user is responsible for ensuring that the format string is sensible.

ID = 385

To_text(Real value,Integer no_dec)

Name

Text To_text(Real value,Integer no_dec)

Description

Convert the Real **value** to text with **no_dec** decimal places.

If the Integer argument **no_dec** is missing, the number of decimal places defaults to zero.

The function return value is the converted value.

ID = 33

To_text(Real value,Text format)

Name

Text To_text(Real value,Text format)

Description

Convert the Real **value** to text using the Text **format** as a C++ format string.

The function return value is the converted value.

Warning

The user is responsible for ensuring that the format string is sensible.

ID = 386

To_text(Text text,Text format)

Name

Text To_text(Text text,Text format)

Description

Convert the Text **text** to text using the Text **format** as a C++ format string.

The function return value is the converted value.

Warning

The user is responsible for ensuring that the format string is sensible.

ID = 391

Get_char(Text t,Integer pos,Integer &c)

Name

Integer Get_char(Text t,Integer pos,Integer &c)

Description

Get a character from Text **t**. The position of the character is **pos**.

The character is returned in the Integer **c**.

The function return value of zero indicates the character returned successfully.

ID = 829

Set_char(Text &t,Integer n,Integer c)

Name

Integer Set_char(Text &t,Integer n,Integer c)

Description

Set the **n**th position (where position starts at 1 for the first character) in the Text **t** to the character given by integer **c**. Note that 'c' can be used to specify the number corresponding to the letter c.

A function return value of zero indicates the Text character is successfully set.

ID = 830

Textstyle Data

Text is part of many *12d Model* elements and there are a large number of properties for the text such as the text colour, size, angle, whiteout etc.

Instead of having separate variables for all of these, a Textstyle_Data has been introduced to hold all the Text variables.

One major benefit of the Textstyle_Data is that in the future, extra variables can be added to the Textstyle_Data structure and the variables are then immediately available everywhere a Textstyle_Data structure is used.

The current variables contained in the Textstyle_Data structure, which may or may not be used, are:

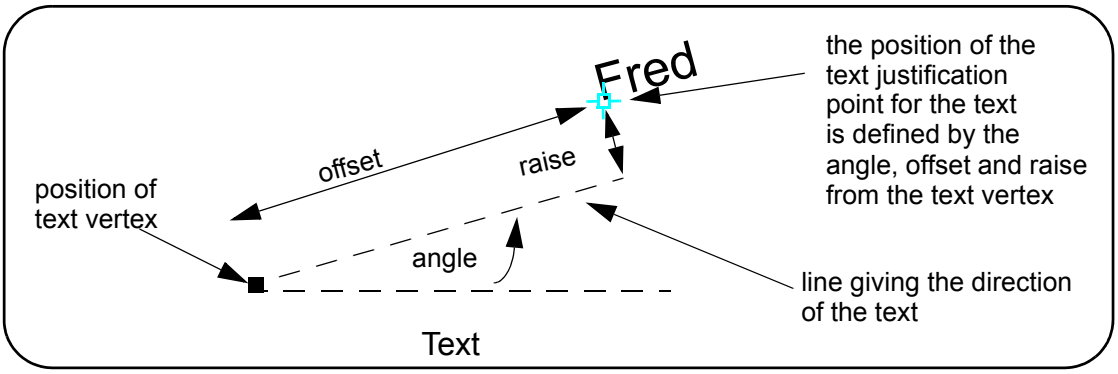
the text itself, text style, colour, height, offset, raise, justification, angle, slant, xfactor, italic, strikeout, underlines, weight, whiteout, border and a name.

Text strings have a height (size) which can be measured in either world units or pixels, a direction of the text (text angle), a justification point defined by an offset distance and a rise distance and a justification.

For text other than segment text, the ***justification point*** and the ***direction of the text*** are defined by:

(a) the *direction of the text* is given as a *counter clockwise* ***angle*** *of rotation* (measured from the x-axis) about the vert*ex (default 0) of the text. The units for* ***angle*** *is* ***radians****.*

(b) the *justification point* is given as an ***offset*** from the vertex *along the line through the vertex with the direction of the text*, and a perpendicular distance (called the ***raise***) from that offset point to the justification point (default 0).



The vertex and justification point only coincide if the offset and raise values are both zero.

The height (size) of the text, and the offset and raise are specified in either world units or pixels and the units are given by an Integer where

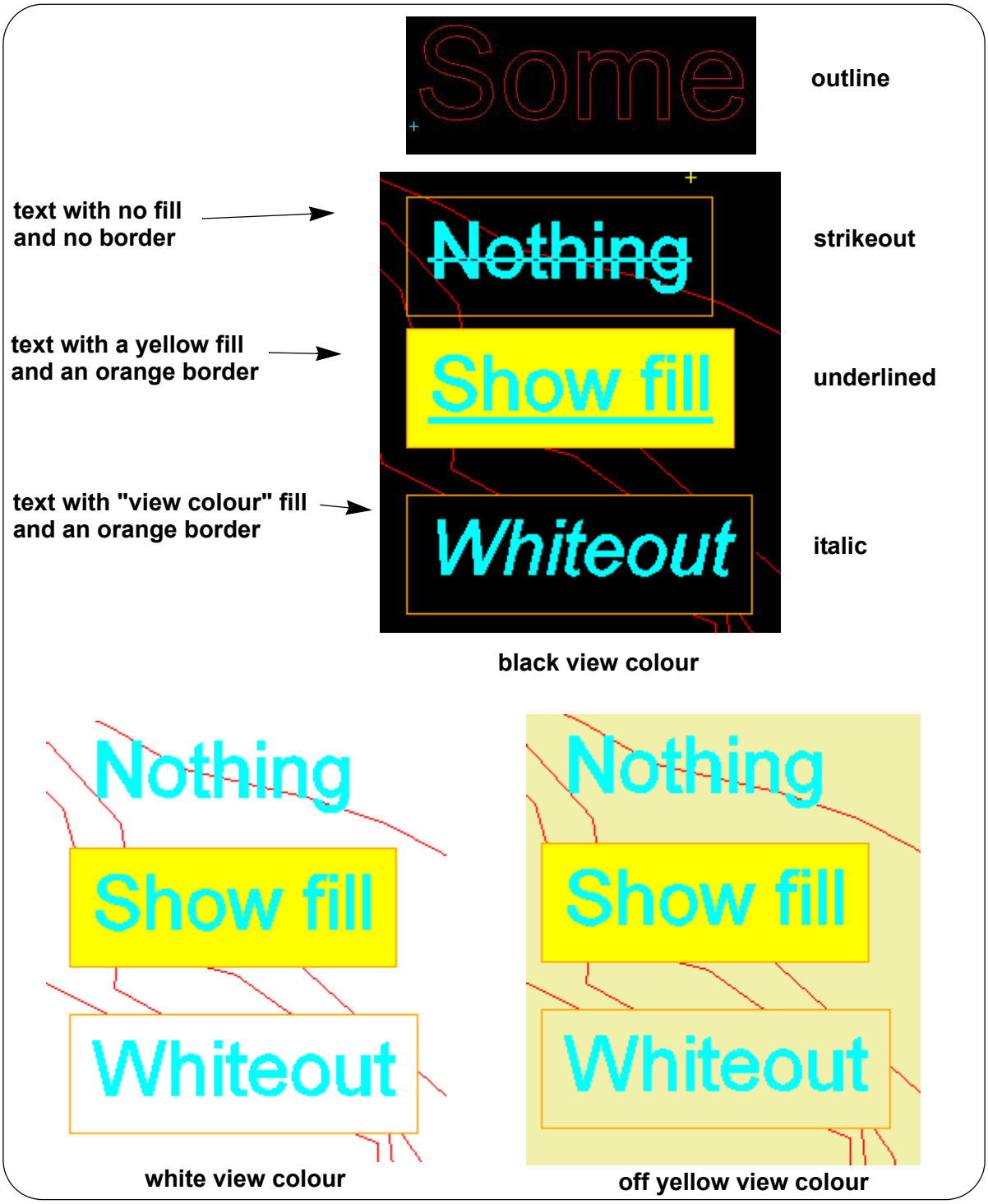
0	for pixel units (the default)
1	for world units
2	for paper units (millimetres)

The justification point (default 1) can be one of nine positions defined in *relation to the text* of the Text string:

		top		
	3	6	9	
left	2	5	8	**right**
	1	4	7	
		bottom		

The box that encloses the text can be coloured in (filled), and given a coloured border. If the colour to fill the box is VIEW_COLOUR, then the fill colour is what ever the view background colour for whatever view that the text is on.

Also true type fonts can have underlined, italic, strikeout and in outline only.



The following functions are used to get and set the variables of a Textstyle_Data.

Null(Textstyle_Data textdata)

Name

Integer Null(Textstyle_Data textdata)

Description

Set the Textstyle_Data **textdata** to null.

A function return value of zero indicates the **textdata** was successfully nulled.

ID = 1639

Null(Textstyle_Data textdata,Integer mode)

Name

Integer Null(Textstyle_Data textdata,Integer mode)

Description

Various fields of a Textstyle_Data can be turned of so they won't display (and so can't be set) in a Textstyle_Data pop-up.

To turn off the Textstyle_Data fields, the *Null(Textstyle_Data textdata,Integer mode)* call is made with **mode** giving what fields are to be turned off.

The values of **mode** and the Textstyle_Data field that they turn off are:

Textstyle_Data_Textstyle = 0x00001,

Textstyle_Data_Colour = 0x00002,

Textstyle_Data_Type = 0x00004,

Textstyle_Data_Size = 0x00008,

Textstyle_Data_Offset = 0x00010,

Textstyle_Data_Raise = 0x00020,

Textstyle_Data_Justify_X = 0x00040,

Textstyle_Data_Justify_Y = 0x00080,

Textstyle_Data_Angle = 0x00100,

Textstyle_Data_Slant = 0x00200,

Textstyle_Data_X_Factor = 0x00400,

Textstyle_Data_Name = 0x00800,

Textstyle_Data_Underline = 0x01000,

Textstyle_Data_Strikeout = 0x02000,

Textstyle_Data_Italic = 0x04000,

Textstyle_Data_Weight = 0x08000,

Textstyle_Data_Whiteout = 0x10000,

Textstyle_Data_Border = 0x20000,

Textstyle_Data_All = 0xfffff,

Note: the fields can be turned off one at a time by calling *Null(Textstyle_Data textdata,Integer mode)* a number of times, and/or more that one can be turned off at the one time by combining them with the logical OR operator "|".

For example,

Textstyle_Data_Offset | Textstyle_Data_Raise

will turn off both the fields Textstyle_Data_Offset and Textstyle_Data_Raise.

LJG? Please add to Set_up.h

A function return value of zero indicates the parts of the Textstyle_Data were successfully nulled.

ID = 1640

Set_data(Textstyle_Data textdata,Text text_data)

Name

Integer Set_data(Textstyle_Data textdata,Text text_data)

Description

Set the data of type Text for the Textstyle_Data **text** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 2163

Get_data(Textstyle_Data textstyle,Text &text_data)

Name

Integer Get_data(Textstyle_Data textstyle,Text &text_data)

Description

Get the data of type Text from the Textstyle_Data **textstyle** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 2162

Set_textstyle(Textstyle_Data textdata,Text style)

Name

Integer Set_textstyle(Textstyle_Data textdata,Text style)

Description

For the Textstyle_Data **textdata,** set the textstyle to **style**.

A function return value of zero indicates the textstyle was successfully set.

ID = 1652

Get_textstyle(Textstyle_Data textdata,Text &style)

Name

Integer Get_textstyle(Textstyle_Data textdata,Text &style)

Description

From the Textstyle_Data **textdata,** get the style and return it in **style**.

A function return value of zero indicates the style was successfully returned.

ID = 1641

Set_colour(Textstyle_Data textdata,Integer colour_num)

Name

Integer Set_colour(Textstyle_Data textdata,Integer colour_num)

Description

For the Textstyle_Data **textdata,** set the colour number to be **colour_num**.

A function return value of zero indicates the colour number was successfully set.

ID = 1653

Get_colour(Textstyle_Data textdata,Integer &colour_num)

Name

Integer Get_colour(Textstyle_Data textdata,Integer &colour_num)

Description

From the Textstyle_Data **textdata,** get the colour number and return it in **colour_num**.

A function return value of zero indicates the colour number was successfully returned.

ID = 1642

Set_text_type(Textstyle_Data textdata,Integer type)

Name

Integer Set_text_type(Textstyle_Data textdata,Integer type)

Description

For the Textstyle_Data **textdata,** set the units (pixel, world, paper) of the Textstyle_Data to be given by the Integer **type**.

For the value for each type of units, see Textstyle Data . The default units is pixel (type = 0).

A function return value of zero indicates the text units was successfully set.

ID = 1654

Get_text_type(Textstyle_Data textdata,Integer &type)

Name

Integer Get_text_type(Textstyle_Data textdata,Integer &type)

Description

For the Textstyle_Data **textdata,** get the units (pixel, world, paper) of the Textstyle_Data and return the value in **type**.

For the values of type, see Textstyle Data . The default units is pixel (type = 0).

If the field is not set then the function return value is 1.

A function return value of zero indicates the text units was successfully returned.

ID = 1643

Set_size(Textstyle_Data textdata,Real height)

Name

Integer Set_size(Textstyle_Data textdata,Real height)

Description

For the Textstyle_Data **textdata,** set the height to be **height**.

A function return value of zero indicates the height was successfully set.

ID = 1655

Get_size(Textstyle_Data textdata,Real &height)

Name

Integer Get_size(Textstyle_Data textdata,Real &height)

Description

From the Textstyle_Data **textdata,** get the height and return it in **height**.

A function return value of zero indicates the height was successfully returned.

ID = 1644

Set_offset(Textstyle_Data textdata,Real offset)

Name

Integer Set_offset(Textstyle_Data textdata,Real offset)

Description

For the Textstyle_Data **textdata,** set the offset to be **offset**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the offset was successfully set.

ID = 1656

Get_offset(Textstyle_Data textdata,Real &offset)

Name

Integer Get_offset(Textstyle_Data textdata,Real &offset)

Description

From the Textstyle_Data **textdata,** get the offset and return it in **offset**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the offset was successfully returned.

ID = 1645

Set_raise(Textstyle_Data textdata,Real raise)

Name

Integer Set_raise(Textstyle_Data textdata,Real raise)

Description

For the Textstyle_Data **textdata,** set the raise to be **raise**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the raise was successfully set.

ID = 1657

Get_raise(Textstyle_Data textdata,Real &raise)

Name

Integer Get_raise(Textstyle_Data textdata,Real &raise)

Description

From the Textstyle_Data **textdata,** get the raise and return it in **raise**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the raise was successfully returned.

ID = 1646

Set_justify(Textstyle_Data textdata,Integer justify)

Name

Integer Set_justify(Textstyle_Data textdata,Integer justify)

Description

For the Textstyle_Data **textdata,** set the justification number to be **justify**.

justify can have the value 1 to 9. *For the meaning of the values for* **justify**, *see* Textstyle Data .

A function return value of zero indicates the justification number was successfully set.

ID = 1658

Get_justify(Textstyle_Data textdata,Integer &justify)

Name

Integer Get_justify(Textstyle_Data textdata,Integer &justify)

Description

From the Textstyle_Data **textdata,** get the justification number and return it in **justify**.

justify can have the value 1 to 9. *For the meaning of the values for justify, see* Textstyle Data .

A function return value of zero indicates the justification number was successfully returned.

ID = 1647

Set_angle(Textstyle_Data textdata,Real angle)

Name

Integer Set_angle(Textstyle_Data textdata,Real angle)

Description

For the Textstyle_Data **textdata,** set the angle to be **angle**.

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

For a diagram, see Textstyle Data .

A function return value of zero indicates the angle was successfully set.

ID = 1659

Get_angle(Textstyle_Data textdata,Real &angle)

Name

Integer Get_angle(Textstyle_Data textdata,Real &angle)

Description

From the Textstyle_Data **textdata,** get the angle and return it in **angle**.

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

For a diagram, see Textstyle Data .

A function return value of zero indicates the angle was successfully returned.

ID = 1648

Set_slant(Textstyle_Data textdata,Real slant)

Name

Integer Set_slant(Textstyle_Data textdata,Real slant)

Description

For the Textstyle_Data **textdata,** set the slant to be **slant**.

A function return value of zero indicates the slant was successfully set.

ID = 1660

Get_slant(Textstyle_Data textdata,Real &slant)

Name

Integer Get_slant(Textstyle_Data textdata,Real &slant)

Description

From the Textstyle_Data **textdata,** get the slant of the textstyle and return it in **slant**.

A function return value of zero indicates the textstyle was successfully returned.

ID = 1649

Set_x_factor(Textstyle_Data textdata,Real xfactor)

Name

Integer Set_x_factor(Textstyle_Data textdata,Real xfactor)

Description

For the Textstyle_Data **textdata,** set the xfactor to be **xfactor**.

A function return value of zero indicates the xfactor was successfully set.

ID = 1661

Get_x_factor(Textstyle_Data textdata,Real &xfactor)

Name

Integer Get_x_factor(Textstyle_Data textdata,Real &xfactor)

Description

From the Textstyle_Data **textdata,** get the xfactor and return it in **xfactor**.

A function return value of zero indicates the xfactor was successfully returned.

ID = 1650

Set_name(Textstyle_Data textdata,Text name)

Name

Integer Set_name(Textstyle_Data textdata,Text name)

Description

For the Textstyle_Data **textdata,** set the name to be **name**.

A function return value of zero indicates the name was successfully set.

ID = 1662

Get_name(Textstyle_Data textdata,Text &name)

Name

Integer Get_name(Textstyle_Data textdata,Text &name)

Description

From the Textstyle_Data **textdata,** get the name of the Textstyle_Data and return it in **name**.

A function return value of zero indicates the name was successfully returned.

ID = 1651

Set_whiteout(Textstyle_Data textdata,Integer colour)

Name

Integer Set_whiteout(Textstyle_Data textdata,Integer colour)

Description

For the Textstyle_Data **textdata**, set the colour number of the colour used for the whiteout box around the text, to be **colour**.

If no text whiteout is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully set.

ID = 2753

Get_whiteout(Textstyle_Data textdata,Integer &colour)

Name

Integer Get_whiteout(Textstyle_Data textdata,Integer &colour)

Description

For the Textstyle_Data **textdata**, get the colour number that is used for the whiteout box around the text. The whiteout colour is returned as Integer **colour**.

NO_COLOUR is the returned as the colour number if whiteout is not being used.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully returned.

ID = 2754

Set_border(Textstyle_Data textdata,Integer colour)

Name

Integer Set_border(Textstyle_Data textdata,Integer colour)

Description

For the Textstyle_Data **textdata**, set the colour number of the colour used for the border of the whiteout box around the text, to be **colour**.

If no whiteout border is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully set.

ID = 2763

Get_border(Textstyle_Data textdata,Integer &colour)

Name

Integer Get_border(Textstyle_Data textdata,Integer &colour)

Description

For the Textstyle_Data **textdata**, get the colour number that is used for the border of the whiteout box around the text. The whiteout border colour is returned as Integer **colour**.

NO_COLOUR is the returned as the colour number if there is no whiteout border.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully returned.

ID = 2764

Set_ttf_underline(Textstyle_Data textdata,Integer underline)

Name

Integer Set_ttf_underline(Textstyle_Data textdata,Integer underline)

Description

For the Textstyle_Data **textdata,** set the underline state to **underline**.

If **underline** = 1, then for a true type font the text will be underlined.

If **underline** = 0, then text will not be underlined.

For a diagram, see Textstyle Data .

A function return value of zero indicates **underline** was successfully set.

ID = 2620

Get_ttf_underline(Textstyle_Data textdata,Integer &underline)

Name

Integer Get_ttf_underline(Textstyle_Data textdata,Integer &underline)

Description

For the Textstyle_Data **textdata,** get the underline state and return it in **underline**.

If **underline** = 1, then for a true type font, the text will be underlined.

If **underline** = 0, then text will not be underlined.

For a diagram, see Textstyle Data .

A function return value of zero indicates underlined was successfully returned.

ID = 2616

Set_ttf_strikeout(Textstyle_Data textdata,Integer strikeout)

Name

Integer Set_ttf_strikeout(Textstyle_Data textdata,Integer strikeout)

Description

For the Textstyle_Data **textdata,** set the strikeout state to **strikeout.**

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

For a diagram, see Textstyle Data .

A function return value of zero indicates **strikeout** was successfully set.

ID = 2621

Get_ttf_strikeout(Textstyle_Data textdata,Integer &strikeout)

Name

Integer Get_ttf_strikeout(Textstyle_Data textdata,Integer &strikeout)

Description

For the Textstyle_Data **textdata,** get the strikeout state and return it in **strikeout.**

If **strikeout** = 1, then for a true type font, the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

For a diagram, see Textstyle Data .

A function return value of zero indicates strikeout was successfully returned.

ID = 2617

Set_ttf_italic(Textstyle_Data textdata,Integer italic)

Name

Integer Set_ttf_italic(Textstyle_Data textdata,Integer italic)

Description

For the Textstyle_Data **textdata,** set the italic state to **italic.**

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

For a diagram, see Textstyle Data .

A function return value of zero indicates **italic** was successfully set.

ID = 2622

Get_ttf_italic(Textstyle_Data textdata,Integer &italic)

Name

Integer Get_ttf_italic(Textstyle_Data textdata,Integer &italic)

Description

For the Textstyle_Data **textdata,** get the italic state and return it in **italic.**

If **italic** = 1, then for a true type font, the text will be italic.
If **italic** = 0, then text will not be italic.

For a diagram, see Textstyle Data .

A function return value of zero indicates italic was successfully returned.

ID = 2618

Set_ttf_outline(Textstyle_Data textdata,Integer outline)

Name

Integer Set_ttf_outline(Textstyle_Data textdata,Integer outline)

Description

For the Textstyle_Data **textdata,** set the outline state to **outline.**

For the Element **elt** of type **Text,** set the outline state to **outline**.

If **outline** = 1, then for a true type font the text will be only shown in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

A function return value of zero indicates ***outline*** was successfully set.

ID = 2773

Get_ttf_outline(Textstyle_Data textdata,Integer &outline)

Name

Integer Get_ttf_outline(Textstyle_Data textdata,Integer &outline)

Description

For the Textstyle_Data **textdata,** get the outline state and return it in **outline.**

If **outline** = 1, then for a true type font the text will be shown only in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

A function return value of zero indicates ***outline*** was successfully returned.

ID = 2774

Set_ttf_weight(Textstyle_Data textdata,Integer weight)

Name

Integer Set_ttf_weight(Textstyle_Data textdata,Integer weight)

Description

For the Textstyle_Data **textdata,** set the font weight to **weight.**

For the list of allowable weights, go to Allowable Weights

A function return value of zero indicates weight was successfully set.

ID = 2623

Get_ttf_weight(Textstyle_Data textdata,Integer &weight)

Name

Integer Get_ttf_weight(Textstyle_Data textdata,Integer &weight)

Description

For the Textstyle_Data **textdata,** get the font weight and return it in **weight.**

For the list of allowable weights, go to Allowable Weights

A function return value of zero indicates weight was successfully returned.

ID = 2619

Maths

Most of the standard C++ mathematical functions are supported in 12dPL.

The angles for the trigonometric functions are expressed in radians

Real Sin(Real x) sine of x

ID = 1

Real Cos(Real x) cosine of x

ID = 2

Real Tan(Real x) tangent of x

ID = 3

Real Asin(Real x) arcsine(x) in range [-pi/2,pi/2], -1<= x <= 1

ID = 5

Real Acos(Real x) arccosine(x) in range [-pi/2,pi/2], -1<= value <= 1

ID = 4

Real Atan(Real x) arctan(x) in range [-pi/2,pi/2]

ID = 6

Real Atan2(Real y, Real x) Arctan(y/x) in range [-pi,pi]

ID = 7

Real Sinh(Real x) hyperbolic sine of x

ID = 8

Real Cosh(Real x) hyperbolic cosine of x

ID = 9

Real Tanh(Real x) hyperbolic tangent of x

ID = 10

Real Exp(Real x) exponential function

ID = 11

Real Log(Real x) natural logarithm ln(x), x > 0

ID = 12

Real Log10(Real x) base 10 logarithm log(x), x> 0

ID = 13

Real Pow(Real x, Real y) x raised to the power y.A domain error occurs if x=0 and y<=0, or if x<0 and y is not an integer.

ID = 14

Real Sqrt(Real x) square root of x, x >= 0

ID = 15

Real Ceil(Real x) smallest integer not less than x, as a Real

ID = 16

Real Floor(Real x) largest integer not greater than x, as a Real

ID = 17

Real Absolute(Real x) absolute value of x

ID = 18

Integer Absolute(Integer i) absolute value of x

ID = 330

Real Ldexp(Real x,Integer n) x*(2 to the power n)

ID = 19

Real Mod(Real x, Real y) Real remainder of x/y with the same sign as x.
If y is zero, the result is implementation defined

ID = 20

Random Numbers

Set_random_number(Integer seed,Integer method)

Name

void Set_random_number(Integer seed,Integer method)

Description

Set up the random number generator with the Integer seed, **seed** (the current time in seconds is a good seed).

If **method** is any value other than 1, the standard c library random number generator is used.

If **method** is 1, then a far more random seed generator than the standard c library one is used.

Once the random number generator is set with a seed, calling Get_Random_number will return a random number.

There is no function return value.

ID = 1900

Get_random_number()

Name

Integer Get_random_number()

Description

Generate the next random number as an Integer and return it as the function return value.

Note: the random number generator is initially set using Set_random_number.

ID = 1901

Get_random_number_closed()

Name

Real Get_random_number_closed()

Description

Generate the next random number as a number between 0 and 1 inclusive, and return it as the function return value.

Note: this function is only applicable is the random number generator is initially set using Set_random_number with method = 1.

ID = 1933

Get_random_number_open()

Name

Real Get_random_number_open()

Description

Generate the next random number as a number between 0 (included) and 1 (not included), and return it as the function return value.

Note: this function is only applicable is the random number generator is initially set using

Set_random_number with method = 1.

ID = 1934

Vectors and Matrices

Set_vector(Vector2 &vect,Real value)

Name

Integer Set_vector(Vector2 &vect,Real value)

Description

Set the two components of the two dimensional vector **vect** to the same Real value, **value**.

A function return value of zero indicates the values were successfully set.

ID = 2306

Set_vector(Vector3 &vect,Real value)

Name

Integer Set_vector(Vector3 &vect,Real value)

Description

Set the three components of the three dimensional vector **vect** to the same Real value, **value**.

A function return value of zero indicates the values were successfully set.

ID = 2307

Set_vector(Vector4 &vect,Real value)

Name

Integer Set_vector(Vector4 &vect,Real value)

Description

Set the four components of the four dimensional vector **vect** to the same Real value, **value**.

A function return value of zero indicates the values were successfully set.

ID = 2308

Set_vector(Vector2 &vect,Real x,Real y)

Name

Integer Set_vector(Vector2 &vect,Real x,Real y)

Description

Set the first component of the two dimensional vector **vect** to the value **x**.

Set the second component of the two dimensional vector **vect** to the value **y**.

A function return value of zero indicates the values were successfully set.

ID = 2309

Set_vector(Vector3 &vect,Real x,Real y,Real z)

Name

Integer Set_vector(Vector3 &vect,Real x,Real y,Real z)

Description

Set the first component of the three dimensional vector **vect** to the value **x**.

Set the second component of the three dimensional vector **vect** to the value **y**.

Set the third component of the three dimensional vector **vect** to the value **z**.

A function return value of zero indicates the values were successfully set.

ID = 2310

Set_vector(Vector4 &vect,Real x,Real y,Real z,Real w)

Name

Integer Set_vector(Vector4 &vect,Real x,Real y,Real z,Real w)

Description

Set the first component of the four dimensional vector **vect** to the value **x**.

Set the second component of the four dimensional vector **vect** to the value **y**.

Set the third component of the four dimensional vector **vect** to the value **z**.

Set the fourth component of the four dimensional vector **vect** to the value **w**.

A function return value of zero indicates the values were successfully set.

ID = 2311

Get_vector(Vector2 &vect,Real &x,Real &y)

Name

Integer Get_vector(Vector2 &vect,Real &x,Real &y)

Description

For the two dimensional vector **vect:**

return the first component of **vect** in **x**.

return the second component of **vect** in **y**

A function return value of zero indicates the components were successfully returned.

ID = 2312

Get_vector(Vector3 &vect,Real &x,Real &y,Real &z)

Name

Integer Get_vector(Vector3 &vect,Real &x,Real &y,Real &z)

Description

For the three dimensional vector **vect:**

return the first component of **vect** in **x**.

return the second component of **vect** in **y**

return the third component of **vect** in **z**

A function return value of zero indicates the components were successfully returned.

ID = 2313

Get_vector(Vector4 &vect,Real &x,Real &y,Real &z,Real &w)

Name

Integer Get_vector(Vector4 &vect,Real &x,Real &y,Real &z,Real &w)

Description

For the four dimensional vector **vect:**
return the first component of **vect** in **x**.

return the second component of **vect** in **y**

return the third component of **vect** in **z**

return the fourth component of **vect** in **w**

A function return value of zero indicates the components were successfully returned.

ID = 2314

Set_vector(Vector2 &vect,Integer index,Real value)

Name

Integer Set_vector(Vector2 &vect,Integer index,Real value)

Description

Set component number **index** of the two dimensional vector **vect** to the value **value**.

A function return value of zero indicates the component was successfully set.

ID = 2315

Set_vector(Vector3 &vect,Integer index,Real value)

Name

Integer Set_vector(Vector3 &vect,Integer index,Real value)

Description

Set component number **index** of the three dimensional vector **vect** to the value **value**.

A function return value of zero indicates the component was successfully set.

ID = 2316

Set_vector(Vector4 &vect,Integer index,Real value)

Name

Integer Set_vector(Vector4 &vect,Integer index,Real value)

Description

Set component number **index** of the four dimensional vector **vect** to the value **value**.

A function return value of zero indicates the component was successfully set.

ID = 2317

Get_vector(Vector2 &vect,Integer index,Real &value)

Name

Integer Get_vector(Vector2 &vect,Integer index,Real &value)

For the two dimensional vector **vect** return the component number **index** in **value.**

A function return value of zero indicates the component was successfully returned.

Description

ID = 2318

Get_vector(Vector3 &vect,Integer index,Real &value)

Name

Integer Get_vector(Vector3 &vect,Integer index,Real &value)

Description

For the three dimensional vector **vect** return the component number **index** in **value.**

A function return value of zero indicates the component was successfully returned.

ID = 2319

Get_vector(Vector4 &vect,Integer index,Real &value)

Name

Integer Get_vector(Vector4 &vect,Integer index,Real &value)

Description

For the four dimensional vector **vect** return the component number **index** in **value.**

A function return value of zero indicates the component was successfully returned.

ID = 2320

Get_vector(Vector2 &vect,Integer index)

Name

Real Get_vector(Vector2 &vect,Integer index)

Description

For the two dimensional vector **vect,** return the component number **index** as the return value of the function.

ID = 2321

Get_vector(Vector3 &vect,Integer index)

Name

Real Get_vector(Vector3 &vect,Integer index)

Description

For the three dimensional vector **vect,** return the component number **index** as the return value of the function.

ID = 2322

Get_vector(Vector4 &vect,Integer index)

Name

Real Get_vector(Vector4 &vect,Integer index)

Description

For the four dimensional vector **vect,** return the component number **index** as the return value of the function.

ID = 2323

Get_vector_length(Vector2 &vect,Real &value)

Name

Integer Get_vector_length(Vector2 &vect,Real &value)

Description

For the two dimensional vector **vect,** return the length of the vector in **value.**

Note: for V(x,y), length = square root of (x*x + y*y)

A function return value of zero indicates the length was successfully returned.

ID = 2324

Get_vector_length(Vector3 &vect,Real &value)

Name

Integer Get_vector_length(Vector3 &vect,Real &value)

Description

For the three dimensional vector **vect,** return the length of the vector in **value.**

Note: for V(x,y,z), length = square root of (x*x + y*y +z*z)

A function return value of zero indicates the length was successfully returned.

ID = 2325

Get_vector_length(Vector4 &vect,Real &value)

Name

Integer Get_vector_length(Vector4 &vect,Real &value)

Description

For the four dimensional vector **vect,** return the length of the vector in **value.**

Note: for V(x,y,z,w), length = square root of (x*x + y*y +z*z + w*w)

A function return value of zero indicates the length was successfully returned.

ID = 2326

Get_vector_length(Vector2 &vect)

Name

Real Get_vector_length(Vector2 &vect)

Description

Standard vector length and return it as return value

For the two dimensional vector **vect,** return the length of the vector as the return value of the function.

Note: for V(x,y), length = square root of (x*x + y*y)

ID = 2327

Get_vector_length(Vector3 &vect)

Name

Real Get_vector_length(Vector3 &vect)

Description

For the three dimensional vector **vect,** return the length of the vector as the return value of the function**.**

Note: for V(x,y,z), length = square root of (x*x + y*y +z*z)

ID = 2328

Get_vector_length(Vector4 &vect)

Name

Real Get_vector_length(Vector4 &vect)

Description

For the four dimensional vector **vect,** return the length of the vector as the return value of the function**.**

Note: for V(x,y,z,w), length = square root of (x*x + y*y +z*z + w*w)

ID = 2329

Get_vector_length_squared(Vector2 &vect,Real &value)

Name

Integer Get_vector_length_squared(Vector2 &vect,Real &value)

Description

For the two dimensional vector **vect,** return the square of the length of the vector in **value.**

Note: for V(x,y), length squared = x*x + y*y

A function return value of zero indicates the length squared was successfully returned.

ID = 2330

Get_vector_length_squared(Vector3 &vect,Real &value)

Name

Integer Get_vector_length_squared(Vector3 &vect,Real &value)

Description

For the three dimensional vector **vect,** return the square of the length of the vector in **value.**

Note: for V(x,y,z), length squared = x*x + y*y + z*z

A function return value of zero indicates the length squared was successfully returned.

ID = 2331

Get_vector_length_squared(Vector4 &vect,Real &value)

Name

Integer Get_vector_length_squared(Vector4 &vect,Real &value)

Description

For the four dimensional vector **vect,** return the square of the length of the vector in **value.**

Note: for V(x,y,z,w), length squared = x*x + y*y + z*z + w*w

A function return value of zero indicates the length squared was successfully returned.

ID = 2332

Get_vector_length_squared(Vector2 &vect)

Name

Real Get_vector_length_squared(Vector2 &vect)

Description

For the two dimensional vector **vect,** return the square of the length of the vector as the function return value**.**

Note: for V(x,y), length squared = x*x + y*y

ID = 2333

Get_vector_length_squared(Vector3 &vect)

Name

Real Get_vector_length_squared(Vector3 &vect)

Description

For the three dimensional vector **vect,** return the square of the length of the vector as the function return value**.**

Note: for V(x,y,z), length squared = x*x + y*y + z*z

ID = 2334

Get_vector_length_squared(Vector4 &vect)

Name

Real Get_vector_length_squared(Vector4 &vect)

Description

For the four dimensional vector **vect,** return the square of the length of the vector as the function return value**.**

Note: for V(x,y,z,w), length squared = x*x + y*y + z*z + w*w

ID = 2335

Get_vector_normalize(Vector2 &vect,Vector2 &normalised)

Name

Integer Get_vector_normalize(Vector2 &vect,Vector2 &normalised)

Description

For the two dimensional vector **vect,** return the normalised vector of **vect** in the Vector2 **normalised**.

Note: for a normalised vector, length = 1 and for the vector V(x,y), the normalised vector N(a,b) is:

N(a,b) = (x/length(V),y/length(V))

A function return value of zero indicates the normalised vector was successfully returned.

ID = 2336

Get_vector_normalize(Vector3 &vect,Vector3 &normalised)

Name

Integer Get_vector_normalize(Vector3 &vect,Vector3 &normalised)

Description

For the three dimensional vector **vect,** return the normalised vector of **vect** in the Vector3 **normalised**.

Note: for a normalised vector, length = 1 and for the vector V(x,y,z), the normalised vector N(a,b,c) is:

N(a,b,c) = (x/length(V),y/length(V),z/length(V))

A function return value of zero indicates the normalised vector was successfully returned.

ID = 2337

Get_vector_normalize(Vector4 &vect,Vector4 &normalised)

Name

Integer Get_vector_normalize(Vector4 &vect,Vector4 &normalised)

Description

For the four dimensional vector **vect,** return the normalised vector of **vect** in the Vector4 **normalised**.

Note: for a normalised vector, length = 1 and for the vector V(x,y,z,w), the normalised vector N(a,b,c,d) is:

N(a,b,c,d) = (x/length(V),y/length(V),z/length(V),w/length(V))

A function return value of zero indicates the normalised vector was successfully returned.

ID = 2338

Get_vector_normalize(Vector2 &vect)

Name

Vector2 Get_vector_normalize(Vector2 &vect)

Description

For the two dimensional vector **vect,** return the normalised vector of **vect** as the function return value.

Note: for a normalised vector, length = 1 and for the vector V(x,y), the normalised vector N(a,b) is:

N(a,b) = (x/length(V),y/length(V))

ID = 2339

Get_vector_normalize(Vector3 &vect)

Name

Vector3 Get_vector_normalize(Vector3 &vect)

Description

For the three dimensional vector **vect,** return the normalised vector as the function return value.

Note: for a normalised vector, length = 1 and for the vector V(x,y,z), the normalised vector N(a,b,c) is:

N(a,b,c) = (x/length(V),y/length(V),z/length(V))

ID = 2340

Get_vector_normalize(Vector4 &vect)

Name

Vector4 Get_vector_normalize(Vector4 &vect)

Description

For the four dimensional vector **vect,** return the normalised vector as the function return value.

Note: for a normalised vector, length = 1 and for the vector V(x,y,z,w), the normalised vector N(a,b,c,d) is:

N(a,b,c,d) = (x/length(V),y/length(V),z/length(V),w/length(V))

ID = 2341

Get_vector_homogenize(Vector3 &vect,Vector3 &homogenized)

Name

Integer Get_vector_homogenize(Vector3 &vect,Vector3 &homogenized)

Description

For the three dimensional vector **vect,** return the homogenized vector of **vect** in the Vector3 **homogenized**.

Note: for a homogenized vector, the third component = 1 and for the vector V(x,y,z), the homogenized vector H(a,b,c) is:

H(a,b,c) = (x/z,y/z,1)

A function return value of zero indicates the homogenized vector was successfully returned.

ID = 2342

Get_vector_homogenize(Vector4 &vect,Vector4 &homogenized)

Name

Integer Get_vector_homogenize(Vector4 &vect,Vector4 &homogenized)

Description

For the four dimensional vector **vect,** return the homogenized vector of **vect** in the Vector4 **homogenized**.

Note: for a homogenized vector, the fourth component = 1 and for the vector V(x,y,z,w), the homogenized vector H(a,b,c,d) is:

H(a,b,c,d) = (x/z,y/w,z/w,1)

A function return value of zero indicates the homogenized vector was successfully returned.

ID = 2343

Get_vector_homogenize(Vector3 &vect)

Name

Vector3 Get_vector_homogenize(Vector3 &vect)

Description

For the three dimensional vector **vect,** return the homogenized vector of **vect** as the function return value.

Note: for a homogenized vector, the third component = 1 and for the vector V(x,y,z), the homogenized vector H(a,b,c) is:

H(a,b,c) = (x/z,y/z,1)

ID = 2344

Get_vector_homogenize(Vector4 &vect)

Name

Vector4 Get_vector_homogenize(Vector4 &vect)

Description

For the four dimensional vector **vect,** return the homogenized vector of **vect** as the function return value.

Note: for a homogenized vector, the fourth component = 1 and for the vector V(x,y,z,w), the homogenized vector H(a,b,c,d) is:

H(a,b,c,d) = (x/z,y/w,z/w,1)

ID = 2345

Set_matrix_zero(Matrix3 &matrix)

Name

Integer Set_matrix_zero(Matrix3 &matrix)

Description

For the three by three Matrix3 **matrix**, set all the values in the matrix to zero.

A function return value of zero indicates the matrix was successfully zero'd.

ID = 2346

Set_matrix_zero(Matrix4 &matrix)

Name

Integer Set_matrix_zero(Matrix4 &matrix)

Description

For the four by four Matrix4 **matrix**, set all the values in the matrix to zero.

A function return value of zero indicates the matrix was successfully zero'd.

ID = 2347

Set_matrix_identity(Matrix3 &matrix)

Name

Integer Set_matrix_identity(Matrix3 &matrix)

Description

For the three by three Matrix3 **matrix**, set matrix to the identity matrix.

That is, for the matrix (row,column) values are:

matrix(1,1) = 1	matrix (1,2) = 0	matrix(1,3) = 0
matrix(2,1) = 0	matrix (2,2) = 1	matrix(2,3) = 0
matrix(3,1) = 0	matrix (3,2) = 0	matrix(3,3) = 1

A function return value of zero indicates the matrix was successfully set to the identity matrix.

ID = 2348

Set_matrix_identity(Matrix4 &matrix)

Name

Integer Set_matrix_identity(Matrix4 &matrix)

Description

For the four by four Matrix4 **matrix**, set matrix to the identity matrix.

That is, for the matrix (row,column) values are:

matrix(1,1) = 1	matrix (1,2) = 0	matrix(1,3) = 0	matrix(1,4) = 0
matrix(2,1) = 0	matrix (2,2) = 1	matrix(2,3) = 0	matrix(2,4) = 0
matrix(3,1) = 0	matrix (3,2) = 0	matrix(3,3) = 1	matrix(3,4) = 0
matrix(4,1) = 0	matrix (4,2) = 0	matrix(4,3) = 0	matrix(4,4) = 1

A function return value of zero indicates the matrix was successfully set to the identity matrix.

ID = 2349

Set_matrix(Matrix3 &matrix,Real value)

Name

Integer Set_matrix(Matrix3 &matrix,Real value)

Description

For the three by three Matrix4 **matrix**, set all the values in the rows and columns of **matrix** to **value**.

A function return value of zero indicates the matrix was successfully set to value.

ID = 2350

Set_matrix(Matrix4 &matrix,Real value)

Name

Integer Set_matrix(Matrix4 &matrix,Real value)

Description

For the four by four Matrix4 **matrix**, set all the values in the rows and columns of **matrix** to **value**.

A function return value of zero indicates the matrix was successfully set to value.

ID = 2351

Set_matrix(Matrix3 &matrix,Integer row,Integer col,Real value)

Name

Integer Set_matrix(Matrix3 &matrix,Integer row,Integer col,Real value)

Description

For the three by three Matrix3 **matrix**, set the value of matrix(**row**,**col**) to **value**.

A function return value of zero indicates the matrix(**row**,**col**) was successfully set to **value**.

ID = 2352

Set_matrix(Matrix4 &matrix,Integer row,Integer col,Real value)

Name

Integer Set_matrix(Matrix4 &matrix,Integer row,Integer col,Real value)

Description

For the four by four Matrix4 **matrix**, set the value of matrix(**row**,**col**) to **value**.

A function return value of zero indicates the matrix(**row**,**col**) was successfully set to **value**.

ID = 2353

Get_matrix(Matrix3 &matrix,Integer row,Integer col,Real &value)

Name

Integer Get_matrix(Matrix3 &matrix,Integer row,Integer col,Real &value)

Description

For the three by three Matrix3 **matrix**, get the value of matrix(**row**,**col**) and return it in **value**.

A function return value of zero indicates the matrix(**row**,**col**) was successfully returned.

ID = 2354

Get_matrix(Matrix4 &matrix,Integer row,Integer col,Real &value)

Name

Integer Get_matrix(Matrix4 &matrix,Integer row,Integer col,Real &value)

Description

For the four by four Matrix4 **matrix**, get the value of matrix(**row**,**col**) and return it in **value**.

A function return value of zero indicates the matrix(**row**,**col**) was successfully returned.

ID = 2355

Get_matrix(Matrix3 &matrix,Integer row,Integer col)

Name

Real Get_matrix(Matrix3 &matrix,Integer row,Integer col)

Description

For the three by three Matrix3 **matrix**, the value of matrix(**row**,**col**) is returned as the function return value.

ID = 2356

Get_matrix(Matrix4 &matrix,Integer row,Integer col)

Name

Real Get_matrix(Matrix4 &matrix,Integer row,Integer col)

Description

For the four by four Matrix3 **matrix**, the value of matrix(**row**,**col**) /.

ID = 2357

Set_matrix_row(Matrix3 &matrix,Integer row,Vector3 &vect)

Name

Integer Set_matrix_row(Matrix3 &matrix,Integer row,Vector3 &vect)

Description

For the three by three Matrix3 **matrix**, set the values of row **row** to the values of the components of the Vector3 **vect**. That is:

matrix(**row**,1) = vect(1) matrix(**row**,2) = vect(2) matrix(**row**,3) = vect(3).

A function return value of zero indicates that the row of **matrix** was successfully set.

ID = 2358

Set_matrix_row(Matrix4 &matrix,Integer row,Vector4 &vect)

Name

Integer Set_matrix_row(Matrix4 &matrix,Integer row,Vector4 &vect)

Description

For the four by four Matrix4 **matrix**, set the values of row **row** to the values of the components of the Vector4 **vect**. That is:

matrix(**row**,1)=vect(1) matrix(**row**,2)=vect(2) matrix(**row**,3)=vect(3) matrix(**row**,4)=vect(4).

A function return value of zero indicates the row of **matrix** was successfully set.

ID = 2359

Get_matrix_row(Matrix3 &matrix,Integer row,Vector3 &vect)

Name

Integer Get_matrix_row(Matrix3 &matrix,Integer row,Vector3 &vect)

Description

For the three dimensional vector **vect**, set the values of **vect** to the values of row **row** of the three by three Matrix3 **matrix**. That is:

vect(1) = matrix(**row**,1) vect(2) = matrix(**row**,2) vect(3) = matrix(**row**,3).

A function return value of zero indicates that the components of **vect** were successfully set.

ID = 2360

Get_matrix_row(Matrix4 &matrix,Integer row,Vector4 &vect)

Name

Integer Get_matrix_row(Matrix4 &matrix,Integer row,Vector4 &vect)

Description

For the four dimensional vector **vect**, set the values of **vect** to the values of row **row** of the four by four Matrix4 **matrix**. That is:

vect(1)=matrix(**row**,1) vect(2)=matrix(**row**,2) vect(3)=matrix(**row**,3) vect(4)=matrix(**row**,4).

A function return value of zero indicates that the components of **vect** were successfully set.

ID = 2361

Get_matrix_row(Matrix3 &matrix,Integer row)

Name

Vector3 Get_matrix_row(Matrix3 &matrix,Integer row)

Description

For the three by three Matrix3 **matrix**, the values of row **row** of matrix are returned as the Vector3 function return value.

ID = 2362

Get_matrix_row(Matrix4 &matrix,Integer row)

Name

Vector4 Get_matrix_row(Matrix4 &matrix,Integer row)

Description

For the four by four Matrix4 **matrix**, the values of row **row** of matrix are returned as the Vector4 function return value.

ID = 2363

Get_matrix_transpose(Matrix3 &source,Matrix3 &target)

Name

Integer Get_matrix_transpose(Matrix3 &source,Matrix3 &target)

Description

For the three by three Matrix3 **matrix**, return the transpose of matrix as Matrix3 **target**.

That is, target(row,column) = matrix(column,row).

A function return value of zero indicates the matrix transpose was successfully returned.

ID = 2364

Get_matrix_transpose(Matrix4 &source,Matrix4 &target)

Name

Integer Get_matrix_transpose(Matrix4 &source,Matrix4 &target)

Description

For the four by four Matrix3 **matrix**, return the transpose of matrix as Matrix4 **target**.

That is, target(row,column) = matrix(column,row).

A function return value of zero indicates the matrix transpose was successfully returned.

ID = 2365

Get_matrix_transpose(Matrix3 &source)

Name

Matrix3 Get_matrix_transpose(Matrix3 &source)

Description

For the three by three Matrix3 **source**, return the transpose of matrix as the function return value.

ID = 2366

Get_matrix_transpose(Matrix4 &source)

Name

Matrix4 Get_matrix_transpose(Matrix4 &source)

Description

For the four by four Matrix4 **source**, return the transpose of matrix as the function return value.

ID = 2367

Get_matrix_inverse(Matrix3 &source,Matrix3 &target)

Name

Integer Get_matrix_inverse(Matrix3 &source,Matrix3 &target)

Description

For the three by three Matrix3 **source**, return the inverse of the matrix as Matrix3 **target**.

A function return value of zero indicates the matrix inverse was successfully returned.

ID = 2368

Get_matrix_inverse(Matrix4 &source,Matrix4 &target)

Name

Integer Get_matrix_inverse(Matrix4 &source,Matrix4 &target)

Description

For the four by four Matrix4 **source**, return the inverse of the matrix as Matrix4 **target**.

A function return value of zero indicates the matrix inverse was successfully returned.

ID = 2369

Get_matrix_inverse(Matrix3 &source)

Name

Matrix3 Get_matrix_inverse(Matrix3 &source)

Description

For the three by three Matrix3 **source**, return the inverse of the matrix as the function return value.

ID = 2370

Get_matrix_inverse(Matrix4 &source)

Name

Matrix4 Get_matrix_inverse(Matrix4 &source)

Description

For the four by four Matrix4 **source**, return the inverse of the matrix as the function return value.

ID = 2371

Swap_matrix_rows(Matrix3 &matrix,Integer row1,Integer row2)

Name

Integer Swap_matrix_rows(Matrix3 &matrix,Integer row1,Integer row2)

Description

For the three by three Matrix3 **matrix**, swap row **row1** with row **row2**.

A function return value of zero indicates the swapped matrix was successfully returned.

ID = 2372

Swap_matrix_rows(Matrix4 &matrix,Integer row1,Integer row2)

Name

Integer Swap_matrix_cols(Matrix4 &matrix,Integer Swap_matrix_rows(Matrix4 &matrix,Integer row1,Integer row2)

Description

For the four by four Matrix4 **matrix**, swap row **row1** with row **row2**.

A function return value of zero indicates the swapped matrix was successfully returned.

ID = 2373

Swap_matrix_cols(Matrix3 &matrix,Integer col1,Integer col2)

Name

Integer Swap_matrix_cols(Matrix3 &matrix,Integer col1,Integer col2)

Description

For the three by three Matrix3 **matrix**, swap column **col1** with column **col2**.

A function return value of zero indicates the swapped matrix was successfully returned.

ID = 2374

Swap_matrix_cols(Matrix4 &matrix,Integer col1,Integer col2)

Name

Integer Swap_matrix_cols(Matrix4 &matrix,Integer col1,Integer col2)

Description

For the four by four Matrix4 **matrix**, swap column **col1** with column **col2**.

A function return value of zero indicates the swapped matrix was successfully returned.

ID = 2375

Get_translation_matrix(Vector2 &vect,Matrix3 &matrix)

Name

Integer Get_translation_matrix(Vector2 &vect,Matrix3 &matrix)

Description

From the two dimension vector **vect**, create the three by three matrix representing the vector as a translation and return it as **matrix**.

That is, for vect(x,y), the matrix(row,column) values are:

matrix(1,1) = 1	matrix(1,2) = 0	matrix(1,3) = **x**
matrix(2,1) = 0	matrix(2,2) = 1	matrix(2,3) = **y**
matrix(3,1) = 0	matrix(3,2) = 0	matrix(3,3) = 1

A function return value of zero indicates the translation matrix was successfully returned.

ID = 2376

Get_translation_matrix(Vector3 &vect,Matrix4 &matrix)

Name

Integer Get_translation_matrix(Vector3 &vect,Matrix4 &matrix)

Description

From the three dimension vector **vect**, create the four by four Matrix4 **matrix** representing the vector as a translation and return it as matrix.

That is, for vect(x,y,z), the matrix(row,column) values are:

matrix(1,1) = 1	matrix(1,2) = 0	matrix(1,3) = 0	matrix(1,4) = **x**
matrix(2,1) = 0	matrix(2,2) = 1	matrix(2,3) = 0	matrix(2,4) = **y**
matrix(3,1) = 0	matrix(3,2) = 0	matrix(3,3) = 1	matrix(3,4) = **z**
matrix(4,1) = 0	matrix(4,2) = 0	matrix(4,3) = 0	matrix(4,4) = 1

A function return value of zero indicates the translation matrix was successfully returned.

ID = 2377

Get_translation_matrix(Vector2 &vect)

Name

Matrix3 Get_translation_matrix(Vector2 &vect)

Description

For the two dimension vector **vect**, the three by three Matrix3 representing the vector as a translation is returned as the function return value.

ID = 2378

Get_translation_matrix(Vector3 &vect)

Name

Matrix4 Get_translation_matrix(Vector3 &vect)

Description

For the three dimension vector **vect**, the four by four Matrix4 representing the vector as a translation is returned as the function return value.

ID = 2379

Get_rotation_matrix(Vector2 &centre,Real angle,Matrix3 &matrix)

Name

Integer Get_rotation_matrix(Vector2 &centre,Real angle,Matrix3 &matrix)

Description

From the Vector2 **centre** and Real **angle**, construct the three by three Matrix3 **matrix** given below.

If **centre** is (x,y), C = cos(angle) and S = sin(angle).

the matrix(row,column) values are:

matrix(1,1) = C matrix(1,2) = -S matrix(1,3) = **x*(**1 - C**) + y***S

matrix(2,1) = S matrix(2,2) = C matrix(2,3) = **y*(**1 - C**) - x***S

matrix(3,1) = 0 matrix(3,2) = 0 matrix(3,3) = 1

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

A function return value of zero indicates the matrix was successfully returned.

ID = 2380

Get_rotation_matrix(Vector3 &axis,Real angle,Matrix4 &matrix)

Name

Integer Get_rotation_matrix(Vector3 &axis,Real angle,Matrix4 &matrix)

Description

From the Vector3 **axis** and Real **angle**, construct the four by four Matrix4 **matrix** given below.

If **Naxis** is **axis normalised** and Naxis = (X,Y,Z), C = cos(angle), S = sin(angle) and T = 1 - C

the matrix(row,column) values are:

matrix(1,1) = T*X*X+C matrix(1,2) = T*X*Y-SZ matrix(1,3) = T*X*Z+S*Y matrix(1,4) = 0

matrix(2,1) = T*X*Y+S*Z matrix(2,2) = T*Y*Y+C matrix(2,3) = T*Y*Z-S*X matrix(1,4) = 0

matrix(3,1) = T*X*Z-S*Y matrix(3,2) = T*Y*Z+S*X matrix(3,3) = T*Z*Z+C matrix(1,4) = 0

matrix(4,1) = 0 matrix(4,2) = 0 matrix(4,3) = 0 matrix(4,1) = 1

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

A function return value of zero indicates the matrix was successfully returned.

ID = 2381

Get_rotation_matrix(Vector2 &centre,Real angle)

Name

Matrix3 Get_rotation_matrix(Vector2 &centre,Real angle)

Description

From the Vector2 **centre** and Real **angle**, construct the three by three Matrix3 **matrix** given below and return it as the function return value.

If **centre** is (X,Y), C = cos(angle) and S = sin(angle) and Matrix3 matrix.

the matrix(row,column) values are:

matrix(1,1) = C matrix(1,2) = -S matrix(1,3) = X*(1 - C) + Y*S

matrix(2,1) = S matrix(2,2) = C matrix(2,3) = Y*(1 - C) - X*S

matrix(3,1) = 0 matrix(3,2) = 0 matrix(3,3) = 1

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

ID = 2382

Get_rotation_matrix(Vector3 &axis,Real angle)

Name

Matrix4 Get_rotation_matrix(Vector3 &axis,Real angle)

Description

From the Vector3 **axis** and Real **angle**, construct the four by four Matrix4 **matrix** given below and return it as the function return value.

If **Naxis** is **axis normalised** and Naxis = (X,Y,Z), C = cos(angle), S = sin(angle), T = 1 - C and Matrix4 **matrix**

the matrix(row,column) values are:

matrix(1,1) = T*X*X+C matrix(1,2) = T*X*Y-SZ matrix(1,3) = T*X*Z+S*Y matrix(1,4) = 0

matrix(2,1) = T*X*Y+S*Z matrix(2,2) = T*Y*Y+C matrix(2,3) = T*Y*Z-S*X matrix(1,4) = 0

matrix(3,1) = T*X*Z-S*Y matrix(3,2) = T*Y*Z+S*X matrix(3,3) = T*Z*Z+C matrix(1,4) = 0

matrix(4,1) = 0 matrix(4,2) = 0 matrix(4,3) = 0 matrix(4,1) = 1

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

ID = 2383

Get_scaling_matrix(Vector2 &scale,Matrix3 &matrix)

Name

Integer Get_scaling_matrix(Vector2 &scale,Matrix3 &matrix)

Description

From the two dimension vector **scale**, create the three by three Matrix3 representing the vector as a scaling matrix and return it as **matrix**.

That is, for scale(S,T), the matrix(row,column) values are:

matrix(1,1) = S matrix(1,2) = 0 matrix(1,3) = 0

matrix(2,1) = 0 matrix(2,2) = T matrix(2,3) = 0

matrix(3,1) = 0 matrix(3,2) = 0 matrix(3,3) = 1

A function return value of zero indicates the translation matrix was successfully returned.

ID = 2384

Get_scaling_matrix(Vector3 &scale,Matrix4 &matrix)

Name

Integer Get_scaling_matrix(Vector3 &scale,Matrix4 &matrix)

Description

From the three dimension vector **scale**, create the four by four Matrix4 representing the vector as a scaling matrix and return it as **matrix**.

That is, for scale(S,T,U), the matrix(row,column) values are:

matrix(1,1) = S matrix(1,2) = 0 matrix(1,3) = 0 matrix(1,4) = 0
matrix(2,1) = 0 matrix(2,2) = T matrix(2,3) = 0 matrix(2,4) = 0
matrix(3,1) = 0 matrix(3,2) = 0 matrix(3,3) = U matrix(3,4) = 0
matrix(4,1) = 0 matrix(4,2) = 0 matrix(4,3) = 0 matrix(4,4) = 1

A function return value of zero indicates the scaling matrix was successfully returned.

ID = 2385

Get_scaling_matrix(Vector2 &scale)

Name

Matrix3 Get_scaling_matrix(Vector2 &scale)

Description

From the two dimension vector **scale**, create the three by three Matrix3 **matrix** as given below. The matrix represents the vector as a scaling and it is return as the function return value.

That is, for scale(S,T), the returned matrix(row,column) values are:

matrix(1,1) = S matrix(1,2) = 0 matrix(1,3) = 0
matrix(2,1) = 0 matrix(2,2) = T matrix(2,3) = 0
matrix(3,1) = 0 matrix(3,2) = 0 matrix(3,3) = 1

ID = 2386

Get_scaling_matrix(Vector3 &scale)

Name

Matrix4 Get_scaling_matrix(Vector3 &scale)

Description

From the three dimension vector **scale**, create the four by four Matrix4 **matrix** as given below. The matrix represents the vector as a scaling and it is return as the function return value.

That is, for scale(S,T,U), the returned matrix(row,column) values are:

matrix(1,1) = S matrix(1,2) = 0 matrix(1,3) = 0 matrix(1,4) = 0
matrix(2,1) = 0 matrix(2,2) = T matrix(2,3) = 0 matrix(2,4) = 0
matrix(3,1) = 0 matrix(3,2) = 0 matrix(3,3) = U matrix(3,4) = 0
matrix(4,1) = 0 matrix(4,2) = 0 matrix(4,3) = 0 matrix(4,4) = 1

ID = 2387

Get_perspective_matrix(Real d,Matrix4 &matrix)

Name

Integer Get_perspective_matrix(Real d,Matrix4 &matrix)

Description

For the distance **d**, create the four by four Matrix4 and return it as **matrix**.

That is, for Real **d**, the matrix(row,column) values are:

matrix(1,1) = 1 matrix(1,2) = 0 matrix(1,3) = 0 matrix(1,4) = 0

matrix(2,1) = 0 matrix(2,2) = 1 matrix(2,3) = 0 matrix(2,4) = 0
matrix(3,1) = 0 matrix(3,2) = 0 matrix(3,3) = 1 matrix(3,4) = 0
matrix(4,1) = 0 matrix(4,2) = 0 matrix(4,3) = 1/d matrix(4,4) = 0

A function return value of zero indicates the matrix was successfully returned.

ID = 2388

Get_perspective_matrix(Real d)

Name

Matrix4 Get_perspective_matrix(Real d)

Description

For the distance **d**, create the four by four Matrix4 and return it as the function return value.

That is, for Real **d**, the matrix(row,column) values are:

matrix(1,1) = 1 matrix(1,2) = 0 matrix(1,3) = 0 matrix(1,4) = 0
matrix(2,1) = 0 matrix(2,2) = 1 matrix(2,3) = 0 matrix(2,4) = 0
matrix(3,1) = 0 matrix(3,2) = 0 matrix(3,3) = 1 matrix(3,4) = 0
matrix(4,1) = 0 matrix(4,2) = 0 matrix(4,3) = 1/d matrix(4,4) = 0

matrix is returned as the function return value.

ID = 2389

Triangles

Triangle_normal(Real xarray[],Real yarray[],Real zarray[],Real Normal[])

Name

Integer Triangle_normal(Real xarray[],Real yarray[],Real zarray[],Real Normal[])

Description

Calculate the normal vector to the triangle given by the coordinates in the arrays xarray[], yarray[], zarray[] (the arrays are of dimension 3).

The normal vector is returned in Normal[1], Normal [2] and Normal[3].

A function return value of zero indicates the function was successful.

ID = 1737

Triangle_normal(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3,Real &xn,Real &yn,Real &zn)

Name

Integer Triangle_normal(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3,Real &xn,Real &yn,Real &zn)

Description

Calculate the normal vector to the triangle given by the coordinates (x1,y1,z1), (x2,y2,z2) and (x3,y3,z3).

The normal vector is returned in (xn,yx,zn).

A function return value of zero indicates the function was successful.

ID = 1738

Triangle_slope(Real xarray[],Real yarray[],Real zarray[],Real &slope)

Name

Integer Triangle_slope(Real xarray[],Real yarray[],Real zarray[],Real &slope)

Description

Calculate the slope of the triangle given by the coordinates in the arrays xarray[], yarray[], zarray[] (the arrays are of dimension 3), and return the value as ***slope***.

The units for slope is an angle in radians measured from the horizontal plane.

A function return value of zero indicates the function was successful.

ID = 1739

Triangle_slope(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3,Real &slope)

Name

Integer Triangle_slope(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3,Real &slope)

Description

Calculate the slope of the triangle given by the coordinates (x1,y1,z1), (x2,y2,z2) and (x3,y3,z3), and return the value as ***slope***.

The units for slope is an angle in radians measured from the horizontal plane.

A function return value of zero indicates the function was successful.

ID = 1740

Triangle_aspect(Real xarray[],Real yarray[],Real zarray[],Real &aspect)

Name

Integer Triangle_aspect(Real xarray[],Real yarray[],Real zarray[],Real &aspect)

Description

Calculate the aspect of the triangle given by the coordinates in the arrays xarray[], yarray[], zarray[] (the arrays are of dimension 3), and return the value as ***aspect***.

The units for aspect is a bearing in radians. That is, aspect is given as a clockwise angle measured from the positive y-axis (North).

A function return value of zero indicates the function was successful.

ID = 1741

Triangle_aspect(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3,Real &aspect)

Name

Integer Triangle_aspect(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3,Real &aspect)

Description

Calculate the aspect of the triangle given by the coordinates (x1,y1,z1), (x2,y2,z2) and (x3,y3,z3), and return the value as ***aspect***.

The units for aspect is a bearing in radians. That is, aspect is given as a clockwise angle measured from the positive y-axis (North).

A function return value of zero indicates the function was successful.

ID = 1742

System

System(Text msg)

Name

Integer System(Text msg)

Description

Make a system call.

The message passed to the system call is given by Text **msg**.

For example,

```
system ("ls *.tmp>fred")
```

A function return value of zero indicates success.

Note

The types of system calls that can be made is operating system dependant.

ID = 21

Date(Text &date)

Name

Integer Date(Text &date)

Description

Get the current date.

The date is returned in Text **date** with the format

```
DDD MMM dd yyyy
```

where DDD is three characters for the day, MMM is three characters for the month

dd is two numbers for the day of the month and yyyy is four numbers for the year, and each is separated by one space.

For example,

```
Sun Mar 17 1996
```

A function return value of zero indicates the date was returned successfully.

ID = 658

Date(Integer &d,Integer &m,Integer &y)

Name

Integer Date(Integer &d,Integer &m,Integer &y)

Description

Get the current date as the day of the month, month & year.

The day of the month value is returned in Integer **d**.

The month value is returned in Integer **m**.

The year value is returned in Integer **y** (fours digits).

A function return value of zero indicates the date was returned successfully.

ID = 659

Time(Integer &time)

Name

Integer Time(Integer &time)

Description

Get the current time as seconds since January 1 1970.

The time value is returned in Integer **time**.

A function return value of zero indicates the time was returned successfully.

ID = 660

Time(Real &time)

Name

Integer Time(Real &time)

Description

Get the current time as the number of seconds since January 1st 1601 down to precision of 10-7 (100 nanoseconds) and return it as **time**.

A function return value of zero indicates the time was returned successfully.

ID = 661

Time(Text &time)

Name

Integer Time(Text &time)

Description

Get the current time.

The time is returned in Text **time** with the format (known as the **ctime** format)

DDD MMM dd hh:mm:ss yyyy where

where **DDD** is three characters for the day, **MMM** is three characters for the month

dd two digits for the day of the month, **hh** two digits for the hour, **mm** two digits for the hour (in twenty four hour format), **ss** two digits for seconds and **yyyy** is four digits for the year.

For example,

Sun Mar 17 23:19:24 1996

A function return value of zero indicates the time was returned successfully.

ID = 662

Time(Integer &h,Integer &m,Real &sec)

Name

Integer Time(Integer &h,Integer &m,Real &sec)

Description

Get the current time in hours, minutes & seconds.

The hours value is returned in Integer **h**.

The minutes value is returned in Integer **m**.

The seconds value is returned in Real **s**.

A function return value of zero indicates the time was returned successfully.

ID = 663

Convert_time(Integer t1,Text &t2)

Name

Integer Convert_time(Integer t1,Text &t2)

Description

Convert the time in seconds since January 1 1970, to the standard ctime format given in an earlier Time function.

The time in seconds is given by Integer **t1** and the Text **t2** returns the time in **ctime** format.

ID = 671

Convert_time(Text &t1,Integer t2)

Name

Integer Convert_time(Text &t1,Integer t2)

Description

Convert the time in ctime format to the time in seconds since January, 1 1970.

The time in ctime format is given by Text **t1** and the time in seconds is returned as Integer **t2**.

Note

Not yet implemented.

LJG?

ID = 672

Convert_time(Integer t1,Text format,Text &t2)

Name

Integer Convert_time(Integer t1,Text format,Text &t2)

Description

Convert the time in seconds since January 1 1970, to the Text **format** (as defined in the section on Title Blocks in the *12d Model* Reference Manual).

The time in seconds is given by Integer **t1** and the Text **t2** returns the time in the specified format.

ID = 683

Get_macro_name()

Name

Text Get_macro_name()

Description

Get the name of the macro file.

The function return value is the macro file name.

ID = 1093

Get_user_name(Text &name)

Name

Integer Get_user_name(Text &name)

Description

Get user's name, the name currently logged onto the system.

The name is returned in Text **name**.

A function return value of zero indicates the name was returned successfully.

ID = 814

Get_host_id()

Name

Text Get_host_id()

Description

For the current *12d Model* session, get the 12d dongle number of the 12d dongle being used to provide the *12d Model* license for the session.

The dongle number, which is alphanumeric, is returned as Text as the function return value.

ID = 2678

Get_module_license(Text module_name)

Name

Integer Get_module_license(Text module_name)

Description

Get the status of each module license.

If the **module_name** is:

points_limit
tins_limit
remaining_days
warned

the function returns number of available units.

If the **module_name** is:

ok	lite
drainage	digitizer
pipeline	
sewer	survey
tin_analysis	volumes
volumesII	trarr
vehicle_path	sight_distance
cartographic	dxf
genio	keays
geocomp	dgn
civilcad	mapinfo
arcview	alignment

The function returns **1** if the module is licensed, **0** if it is not licensed.

ID = 1094

Getenv(Text env)

Name

Text Getenv(Text env)

Description

Get the temporary directory for Windows.

LJG? what is env?

<no description>

ID = 1087

Find_system_file(Text new_file_name,Text old_file_name,Text env)

Name

Text Find_system_file(Text new_file_name,Text old_file_name,Text env)

Description

Returns the path to the setup file **new_file_name** as the function return value.

If old_file_name is not blank, it also looks for the old file names for the set ups files that were used in the Unix version of *12d Model.*

So if you want to support the legacy file names then you pass in new_file_name and old_file_name. If you are only looking for the post Unix names for the set up files, pass old_file_name = "".

env is the name of the environment variable that can also point to the set up file.

The search order is

1. If not blank, search for the file given by the environment variable **env**
2. If new_file_name is not blank, next search for a file with the name **new_file_name** in the normal Set Ups files search order.
3. Finally if the no file has yet been found, if old_file_name is not blank, search for old_file_name in the normal Set Ups files search order.

If no file is found then the function return value is a blank Text (i.e. "").

For example,

```
Find_system_file("colours.4d", "colour_map.def", "COLOURS_4D)
```

will find the colours set up file which may be pointed to by the environment variable COLOURS_4D (if non zero), or may have the name "colours.4d", or finally may have the name "colour_map.def".

ID = 1088

Get_4dmodel_version(Integer &major,Integer &minor,Text &patch)

Name

void Get_4dmodel_version(Integer &major,Integer &minor,Text &patch)

Description

Get information about the 12d Model build.

The function return value is a special patch version description for pre-release versions and it is written after the 12d Model version information. It is blank for release versions.

major - is the major number for *12d Model.* The is, the number before the ".".
For example 9 for 12d Model 9.00

minor - is the minor number for *12d Model*. That is, the number after the ".".
For example 00 for 12d Model 9.00

patch - special patch description for pre-release versions. It is written after the 12d Model version information. It is blank for release versions.

For example "Alpha 274 SLF,SLX,Image Dump - Not For Production"

A function return value of zero indicates the function was successful.

ID = 1089

Is_practise_version()

Name

Integer Is_practise_version()

Description

Check if the current *12d Model* is a practise version.

A non-zero function return value indicates that *12d Model* is a practise version.

A zero function return value indicates that *12d Model* is not a practise version.

Warning this is the opposite of most 12dPL function return values

ID = 1090

Create_process(Text program_name,Text command_line,Text start_directory, Integer flags,Integer wait,Integer inherit)

Name

Integer Create_process(Text program_name,Text command_line,Text start_directory,Integer flags,Integer wait,Integer inherit)

Description

This function basically calls the Microsoft *CreateProcess* function as defined in

http://msdn.microsoft.com/en-us/library/ms682425%28v=vs.85%29.aspx.

The 12d function gives access to the Microsoft *CreateProcess* arguments that are in bold (and also do not have a // in front of them):

```
 BOOL WINAPI CreateProcess(
    __in_opt      LPCTSTR lpApplicationName,
    __inout_opt   LPTSTR lpCommandLine,
//  __in_opt      LPSECURITY_ATTRIBUTES lpProcessAttributes,
//  __in_opt      LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in          BOOL bInheritHandles,
    __in          DWORD dwCreationFlags,
//  __in_opt      LPVOID lpEnvironment,
    __in_opt      LPCTSTR lpCurrentDirectory,
//  __in          LPSTARTUPINFO lpStartupInfo,
//  __out         LPPROCESS_INFORMATION lpProcessInformation
```

);

where **program_name** is passed as *lpApplicationName*, **command_line** is passed as dwCreationFlags *lpCommandLine*, **start_directory** is passed as *lpCurrentDirectory*, **flags** is passed as *dwCreationFlags and* **inherit** *is passed as bInheritHandles*.

If **wait** = 1, the macro will wait until the process finishes before continuing.
If **wait** = 0, the macro won't wait until the process finishes before continuing.

A function return value of zero indicates the function was successful.

Note: *Create_process* can not be called from the *12d Model* the Practise version.

ID = 1620

Create_process(Text program_name,Text command_line,Text start_directory,Integer flags,Integer inherit,Unknown &handle)

Name

Integer Create_process(Text program_name,Text command_line,Text start_directory,Integer flags,Integer inherit,Unknown &handle)

Description

This function calls the Microsoft *CreateProcess* function as defined in

http://msdn.microsoft.com/en-us/library/ms682425%28v=vs.85%29.aspx.

The 12d function gives access to the Microsoft *CreateProcess* arguments that are in bold (and also not have a // in front of them):

```
 BOOL WINAPI CreateProcess(
   __in_opt      LPCTSTR lpApplicationName,
   __inout_opt   LPTSTR lpCommandLine,
//  __in_opt     LPSECURITY_ATTRIBUTES lpProcessAttributes,
//  __in_opt     LPSECURITY_ATTRIBUTES lpThreadAttributes,
   __in          BOOL bInheritHandles,
   __in          DWORD dwCreationFlags,
//  __in_opt     LPVOID lpEnvironment,
   __in_opt      LPCTSTR lpCurrentDirectory,
//  __in         LPSTARTUPINFO lpStartupInfo,
//  __out        LPPROCESS_INFORMATION lpProcessInformation
 );
```

where **program_name** is passed as *lpApplicationName*, **command_line** is passed as dwCreationFlags *lpCommandLine*, **start_directory** is passed as *lpCurrentDirectory*, **flags** is passed as *dwCreationFlags and* **inherit** *is passed as bInheritHandles*.

The handle to the created process is returned in Unknown **handle**.

The macro can check if the process is still running by calling *Process_exists*.

A function return value of zero indicates the function was successful.

Note: The difference between this function and *Create_process(Text program_name,Text command_line,Text start_directory,Integer flags,Integer wait,Integer inherit)* is that a handle to the process is created and returned as **handle** and this can be checked to see if the process is still running. So there is no *wait* flag but there is more flexibility since the macro can check with *Process_exists* and decide when, and when not to wait.

Note: *Create_process* can not be called from *12d Model* the Practise version.

ID = 2635

Process_exists(Unknown handle)

Name

Integer Process_exists(Unknown handle)

Description

Check to see if the process given by **handle** exists. That is, check that the process created by *Create_process(Text program_name,Text command_line,Text start_directory,Integer flags,Integer inherit,Unknown &handle)* is still running.

A non-zero function return value indicates that the process handle is still running (i.e. the process exists).

A zero function return value indicates that the process does not exist.

Warning this is the opposite of most 12dPL function return values

ID = 2636

Shell_execute(Widget widget,Text operation,Text file,Text parameters, Text directory,Integer showcmd)

Name

Integer Shell_execute(Widget widget,Text operation,Text file,Text parameters,Text directory,Integer showcmd)

Description

This function calls the Microsoft *ShellExecute* function as defined in

http://msdn.microsoft.com/en-us/library/bb762153%28v=vs.85%29.aspx

This Microsoft call executes an operation on a file.

The 12d function gives access to the Microsoft *ShellExecute* arguments that are in bold (and also not have a // in front of them):

HINSTANCE ShellExecute(
__in_opt HWND hwnd,
__in_opt LPCTSTR **lpOperation**,
__in LPCTSTR **lpFile**,
__in_opt LPCTSTR **lpParameters**,
__in_opt LPCTSTR **lpDirectory**,
__in INT **nShowCmd**);

where **operation** is passed as *lpOperation*, **file** is passed as lp, **parameters** is passed as lpParameters, **directory** is passed as lpDirectory and **showcmd** is passed as ShowCmd.

The handle to the created process is returned in Unknown **handle**.

The macro can check if the process is still running by calling *Process_exists*.

A function return value of zero indicates the function was successful.

LJG? what is **widget**? Is it a message box?

Note: *Create_process* can not be called from *12d Model* the Practise version.

ID = 1623

Uid's

Elements and Models created within 12d Model are given a **u**nique **id**entifier called a Uid.

When a new element or model is created, it is given the next available Uid. Uid's are never reused so when an element or model is deleted, its Uid is not available for any other element or model.

A Uid is made up of two parts:

(a) a Global Unique Identifier (Guid)

and a

(b) 12d Model generated Id.

Guid's

A **Global Unique Identifier** (Guid) is a unique number which encodes space and time (see Guid in Wikipedia). Whenever a 12d Model project is created, a Guid is generated at the time of creation and this Guid is permanently stored as part of the 12d Model project. The Guid takes 128 bits of storage. If a 12d Model copy is made of a project, then the new project is given a new unique Guid.

Id's

When a 12d Model project is created, the project Id counter, which is a 64-bit Integer, is set to zero and every time a new element is created, the Id counter is incremented and the new element given the current Id value.

The Id counter only ever increases and if an element in a project is deleted, its Id is never reused.

Uid

For a 12d Model Element, the Uid consists of both the Guid of its parent project and its unique Id within that project.

To make things easier, if an element is created in a project, then for the Uid of that element, the *Print* and *To_text* calls for the Uid just print out the local Id of the Uid.

Note - the call *Is_Global* checks to see if the Uid is a local Uid (that is, from the project that the macro is running in), or a Global Uid (that is, from a shared project). *See* Is_global(Uid uid) .

For documentation on Uid Arithmetic, go to the section Uid Arithmetic
For documentation on Uid calls, go to the section Uid Functions

Uid Arithmetic

Because a Uid's consist of a Guid and an Integer Id, a Uid Arithmetic has been included in the 12dPL where for an Uid uid,

uid + n

is defined to be that n is added to the Id part of the Uid where n is a positive or negative integer (whole number). This works for either a local or a global Uid.

The increment and decrement operators also work for local and global Uids. That is,

```
uid++
++uid
uid--
--uid
```

are all defined for both local and global uids.

If two Uids are both local Uids, then they can be subtracted and the value is the subtraction of the two Ids of the Uids.

That is, if the Uids uid1 and uid2 are both local Uids, then

```
Integer   diff = uid1 - uid2
```

is defined and is the difference between the Id of uid1 and the Id of uid2.

If either uid1 or uid1 are global Uids then the difference of them is not defined.

Note - the call *Is_Global* checks to see if the Uid is a local Uid (that is, from the project that the macro is running in), or a Global Uid (that is, from a shared project). *See* Is_global(Uid uid) .

Uid Functions

Get_next_uid()

Name

Uid Get_next_uid()

Description

Get the next available Uid and return it as the function return value.

This is often used in Undo's.

ID = 1920

Get_next_id()

Name

Integer Get_next_id()

Description

Get the next available Id and return it as the function return value.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Uid Get_next_uid()* instead.

ID = 1892

Get_last_uid()

Name

Uid Get_last_uid()

Description

Get the last used Uid (that is the one from the last created Element) and return it as the function return value.

ID = 2072

Get_last_id()

Name

Integer Get_last_id()

Description

Get the last used Id (that is the one from the last created Element) and return it as the function return value.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Get_last_uid* instead (*see* Get_last_uid() .

ID = 2071

void Print(Uid uid)

Name

void Print(Uid uid)

Description

Prints a text conversion of the UID **uid** to the Output Window.

Three is no function return value.

ID = 2052

Convert_uid(Uid uid,Text &txt)

Name

Integer Convert_uid(Uid uid,Text &txt)

Description

Convert the UID **uid** to a Text. The Text is returned in **txt**.

A function return value of zero indicates the Uid was successfully converted to text.

ID = 2053

Convert_uid(Uid uid,Integer &id)

Name

Integer Convert_uid(Uid uid,Integer &id)

Description

Convert the UID **uid** to an Integer The Integer is returned in **id**.

Note - this in only possible if the uid can be expressed as an Integer,

A function return value of zero indicates the Uid was successfully converted. to an Integer.

ID = 2054

Convert_uid(Text txt,Uid &uid)

Name

Integer Convert_uid(Text txt,Uid &uid)

Description

Convert the Text **txt** to an UID. The Uid is returned in **uid**.

Note - this in only possible if **txt** is in the correct form of an Uid.

A function return value of zero indicates the Text was successfully converted to a Uid.

ID = 2055

Convert_uid(Integer id,Uid &uid)

Name

Integer Convert_uid(Integer id,Uid &uid)

Description

Convert the Integer id to an UID. The Uid is returned in **uid**.

Note - this in only possible if the Integer **id** can be expressed as an Uid.

A function return value of zero indicates the Integer was successfully converted to a Uid.

ID = 2056

To_text(Uid uid)

Name

Text To_text(Uid uid)

Description

Convert the UID **uid** to a Text.

The Text is returned as the function return value.

ID = 2057

From_text(Text txt,Uid &uid)

Name

Integer From_text(Text txt,Uid &uid)

Description

Convert the Text **txt** to a Uid and the Uid is returned in **uid**.

A function return value of zero indicates the txt was successfully converted to a Uid.

ID = 2063

Null(Uid &uid)

Name

void Null(Uid &uid)

Description

Set the UID uid to be a **null** Uid.

There is no function return value.

ID = 2058

Is_null(Uid uid)

Name

*Integer Is_null(Uid uid) *

Description

Check to see if the UID **uid** is a **null** Uid.

A non-zero function return value indicates that **uid** is null.

A zero function return value indicates that **uid** is **not** null.

Warning this is the opposite of most 12dPL function return values

ID = 2059

Is_contour(Uid uid)

Name

Integer Is_contour(Uid uid)

Description

Check to see if the UID **uid** is the Uid of a string created by a 12d Model Contour option.

Note - such strings are ignored in 12d Model number counts for Base size.

A non-zero function return value indicates that the uid is of a string created by a 12d Model Contour option.

A zero function return value indicates that the uid is not the uid of a string created by a 12d Model Contour option.

Warning this is the opposite of most 12dPL function return values

ID = 2064

Is_plot(Uid uid)

Name

Integer Is_plot(Uid uid)

Description

Check to see if the UID **uid** is the Uid of a string created by a 12d Model Plot option.

Note - such strings are ignored in 12d Model number counts for Base size.

A non-zero function return value indicates that the uid is of a string created by a 12d Model Plot option.

A zero function return value indicates that the uid is not the uid of a string created by a 12d Model Plot option.

Warning this is the opposite of most 12dPL function return values

ID = 2065

Is_function(Uid uid)

Name

Integer Is_function(Uid uid)

Description

Check to see if the UID 12d Model is the Uid of a 12d Model Function/Macro_Function.

A non-zero function return value indicates that the uid is of a 12d Model Function/ Macro_Function

A zero function return value indicates that the uid is not the uid of a 12d Model Function/ Macro_Function.

Warning this is the opposite of most 12dPL function return values

ID = 2066

Function_exists(Integer id)

Name

Integer Function_exists(Integer id)

Description

Check to see if *id* is the Id of a 12d Function.

1 for yes

A non-zero function return value indicates that *id* is the Id of a 12d Model Function/ Macro_Function

A zero function return value indicates that *id* is not the Id of a 12d Model Function/ Macro_Function.

Warning this is the opposite of most 12dPL function return values

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Integer Is_function(Uid uid)* instead.

ID = 1187

Is_valid(Uid uid)

Name

Integer Is_valid(Uid uid)

Description

Check to see if the UID **uid** is a valid Uid.

A non-zero function return value indicates that **uid** is a valid Uid.

Warning this is the opposite of most 12dPL function return values

ID = 2060

Is_unknown(Uid uid)

Name

Integer Is_unknown(Uid uid)

Description

Check to see if the UID **uid** is a valid Uid.

A non-zero function return value indicates that **uid** is not a valid Uid.

Warning this is the opposite of most 12dPL function return values

ID = 2061

Is_global(Uid uid)

Name

Integer Is_global(Uid uid)

Description

Check to see if the UID **uid** is of a shared element. That is, the element has not been created in this project but has been shared in from another project.

A non-zero function return value indicates that **uid** is of a shared element.

Warning this is the opposite of most 12dPL function return values

ID = 2062

Input/Output

Output Window

Information can be written out to the 12d Model Output Window.

Print(Text msg)

Name

void Print(Text msg)

Description

Print the Text **msg** to the Output Window.

ID = 24

Print(Integer value)

Name

void Print(Integer value)

Description

Print the Integer **value** out in text to the Output Window.

ID = 22

Print(Real value)

Name

void Print(Real value)

Description

Print the Real **value** out in text to the Output Window.

ID = 23

Print()

Name

void Print()

Description

Print the text "\n" (a new line) to the Output Window.

ID = 25

Clear_console()

Name

void Clear_console()

Description

Clear the Output Window of any previous information.

Warning: This function work on the Output Window, **not** the Macro Console.

ID = 1295

Show_console(Integer show)

Name

Integer Show_console(Integer show)

Description

If **show** = 0, the Output Window is hidden.
If **show** = 1, the Output Window is shown.

Warning: This function works on the *Output Window*, **not** the Macro Console.

A function return value of zero indicates the function was successful.

Note: the Output Window can also be turned on/off with the ***12d Mode***l toggle option

Window =>Output Window.

ID = 1728

Is_console_visible()

Name

Integer Is_console_visible()

Description

The function return value indicates if the Output Window is visible or hidden.

If the Integer return value is 0 then the Output Window is hidden.
If the Integer return value is 1 then the Output Window is visible (not hidden).

Warning: This function works on the *Output Window*, **not** the Macro Console.

ID = 1729

Is_console_floating()

Name

Integer Is_console_floating()

Description

The function return value indicates if the Output Window is floating or not floating.

If the Integer return value is 1 then the Output Window is floating.

If the Integer return value is 0 then the Output Window is either not floating or not visible.
Warning: This function works on the *Output Window*, **not** the Macro Console.

ID = 1731

Clipboard

Data can be written to, and read from the Clipboard.

Console_to_clipboard()

Name

Integer Console_to_clipboard()

Description

Copy the ***highlighted*** contents of the Output Window to the clip board.

Warning: This function works on the *Output Window*, **not** the Macro Console.

A function return value of zero indicates the copy was successful.

ID = 1736

Set_clipboard_text(Text txt)

Name

Integer Set_clipboard_text(Text txt)

Description

Write the Text **txt** to the clip board.

A function return value of zero indicates the write was successful.

ID = 1521

Get_clipboard_text(Text &txt)

Name

Integer Get_clipboard_text(Text &txt)

Description

<no description>

A function return value of zero indicates the read was successful.

ID = 1522

Files

Disk files are used extensively in computing for reasons such as passing data between programs, writing out permanent records and reading in bulk input data.

12dPL provides a wide range of functions to allow the user to easily read and write files within macros.

For reading in text data, 12dPL provides the *File_read_line* function which reads one line of text. The powerful 12dPL Text functions are then be used on the line of text line to "pull the line apart" and extract the relevant information.

Similarly, the *File_write_line* function outputs one text line and the powerful Text functions are used to build up the line of text before it is written out.

For binary files, there are functions to read and write out *Real*, *Integer* and *Text* variables and *Real* and *Integer* arrays.

File_exists(Text file_name)

Name

Integer File_exists(Text file_name)

Description

Checks to see if a file of name **file_name** exists.

A non-zero function return value indicates the file exists.

A zero function return value indicates the file does not exist.

Warning - this is the opposite to most 12dPL function return values

ID = 202

File_open(Text file_name,Text mode,Text ccs_text,File &file)

Name

Integer File_open(Text file_name,Text mode,Text ccs_text,File &file)

Description

Opens a file of name **file_name** with open type **mode**. The file unit is returned as File **file**.

The file can be opened as a Unicode file with a specified encoding or as an ANSI file by using a non-blank value for the **ccs_text** parameter.

The available **modes** are

r	open for reading. If the file does not exist then it fails.
r+	open for update, that is for reading and writing. The file must exist.
rb	read binary
w	opens a file for writing. If the files exists, its current contents are destroyed.
w+	opens a file for reading and writing. If the files exists, its current contents are destroyed
wb	write binary
a	open for writing at the end of file (before the end of file marker). If the file does not exist then it is created.
a+	opens for reading and writing to the end of the file (before the end of file marker). If the file does not exist then it is created.

When a file is open for append (i.e. **a** or **a+**), it is impossible to overwrite information that is already in the file. Any writes are automatically added to the end of the file.

ccs_text specifies the *coded character set* to use and can have the values:

ccs_text = "ccs = UTF-8"

ccs_text = "ccs = UTF-16LE"
ccs_text = "ccs = UNICODE"

or ccs_text = "" (leave it blank) if ANSI encoding is required.

For example

```
File_open("test file", "w","ccs=UNICODE",file_handle);
```

Note: BOM detection only applies to files that are opened in Unicode mode (that is, by passing a non blank **ccs** parameter).

If the file already exists and is opened for reading or appending, the Byte Order Mark (BOM), if it present in the file, determines the encoding. The BOM encoding takes precedence over the encoding that is specified by the **ccs** flag. The **ccs** encoding is only used when no BOM is present or the file is a new file.

The following table summarises the use of Byte Order Marks (BOM's) for the various **ccs** flags given to **File_open** and what happens when there is a BOM in an existing file.

Encodings Used When Opening a File Based on non blank ccs Flag and BOM

ccs flag	No BOM (or new file)	BOM: UTF-8	BOM: UTF-16
UNICODE	UTF-16LE	UTF-8	UTF-16LE
UTF-8	UTF-8	UTF-8	UTF-16LE
UTF-16LE	UTF-16LE	UTF-8	UTF-16LE

Files opened for writing in Unicode mode (non-blank **ccs**) automatically have a BOM written to them.

When a file that begins with a Byte Order Mark (BOM) is opened, the file pointer is positioned after the BOM (that is, at the start of the file's actual content).

For more information on ANSI, ASCII, Unicode, UTF's and BOM's, please see Set Ups.h which is a copy of the information from the ***12d Model*** Reference manual.

A function return value of zero indicates the file was opened successfully.

ID = 2076

File_open(Text file_name,Text mode,File &file)

Name

Integer File_open(Text file_name,Text mode,File &file)

Description

Note: this option now only creates UNICODE files. To open a ANSI file, use File_open(Text file_name,Text mode,Text ccs_text,File &file) with ccs_text = "" instead.

Opens a file of name **file_name** with open type **mode**. The file unit is returned as File file.

The available **modes** are

r	open for reading
r+	open for update, reading and writing
rb	read binary
w	truncate or create for writing
w+	truncate or create for update
wb	write binary

a	append open for writing at the end of file or create for writing
a+	open for update at end of file or create for update

When a file is open for append (i.e. **a** or **a+**), it is impossible to overwrite information that is already in the file.

A function return value of zero indicates the file was opened successfully.

ID = 335

File_read_line(File file,Text &text_in)

Name

Integer File_read_line(File file,Text &text_in)

Description

Read a line of text from the File **file**. The text is read into the Text **text_in**.

A function return value of ***-1*** indicates the end of the file.

A function return value of zero indicates the text was successfully read in.

ID = 337

File_write_line(File file,Text text_out)

Name

Integer File_write_line(File file,Text text_out)

Description

Write a line of text to the File **file**. The text to write out is Text **text_out**.

A function return value of zero indicates the text was successfully written out.

ID = 338

File_tell(File file,Integer &pos)

Name

Integer File_tell(File file,Integer &pos)

Description

Get the current position in the File **file**.

A function return value of zero indicates the file position was successfully found.

ID = 341

File_seek(File file,Integer pos)

Name

Integer File_seek(File file,Integer pos)

Description

Go to the position **pos** in the File **file**.

Position pos has normally been found by a previous File_tell call.

If the file open type was **a** or **a+**, then a File_seek cannot be used to position for a write in any part of the file that existed when the file was opened.

If you have to **File_seek** to the beginning of the file, use **File_tell** to get the initial position and **File_seek** to it rather than to position 0.

So for a Unicode file, if you have to **File_seek** to the beginning of the file but after the BOM you need to first have used a **File_tell** to get and record the position of the initial start of the file when it is opened (for a Unicode file, **File_open** positions after the BOM) and then to **File_seek** to that recorded beginning of the file rather than to **File_seek** to position 0.

For more information on ANSI, ASCII, Unicode, UTF's and BOM's, please see Set Ups.h which is a copy of the information from the ***12d Model*** Reference manual.

A function return value of zero indicates the file position was successfully found.

ID = 342

File_flush(File file)

Name

Integer File_flush(File file)

Description

Make sure the File **file** is up to date with what has been written out.

A function return value of zero indicates the file was successfully flushed.

ID = 340

File_rewind(File file)

Name

Integer File_rewind(File file)

Description

Rewind the File **file** to its beginning.

WARNING: This function is not to be used with a Unicode file.

If the file is a Unicode file then **File_rewind** will rewind to BEFORE the BOM. Then writing out any information will overwrite the BOM.

So for a Unicode file, to correctly position to the beginning of the file but after the BOM you need to first have used a **File_tell** when opening the file to get and record position of the initial start of the file (for a Unicode file, **File_open** positions after the BOM) and then to **File_seek** to that recorded beginning of the file rather than to **File_seek** to position 0.

For more information on ANSI, ASCII, Unicode, UTF's and BOM's, please see Set Ups.h which is a copy of the information from the ***12d Model*** Reference manual.

A function return value of zero indicates the file was successfully rewound.

ID = 339

File_read(File file,Integer &value)

Name

Integer File_read(File file,Integer &value)

Description

Read four bytes from the binary file **file** and return it as an Integer in **value**.

A function return value of zero indicates the Integer was successfully returned.

ID = 1710

File_write(File file,Integer value)

Name

Integer File_write(File file,Integer value)

Description

Write out **value** as a four byte integer to the binary file **file**.

A function return value of zero indicates the Integer was successfully written.

ID = 1713

File_read(File file,Real &value)

Name

Integer File_read(File file,Real &value)

Description

Read eight bytes from the binary file **file** and return it as a Real in **value**.

A function return value of zero indicates the Real was successfully returned.

ID = 1711

File_write(File file,Real value)

Name

Integer File_write(File file,Real value)

Description

Write out **value** as an eight byte real to the binary file **file**.

A function return value of zero indicates the Real was successfully written.

ID = 1714

File_read_unicode(File file,Integer length,Text &value)

Name

Integer File_read_unicode(File file,Integer length,Text &value)

Description

Read **length** bytes from the binary file **file** and return it as Text in **value**.

Note - this works for UNICODE files.

For more information on ANSI, ASCII, Unicode, UTF's and BOM's, please see Set Ups.h which is a copy of the information from the ***12d Model*** Reference manual.

A function return value of zero indicates the Text was successfully returned.

ID = 2676

File_write_unicode(File file,Integer length,Text value)

Name

Integer File_write_unicode(File file,Integer length,Text value)

Description

Write out **value** as **length** lots of two byte Unicode characters to the binary file **file**.

If there is less than **length** characters in Text then the number of characters is brought up to **length** by writing out null padding.

For more information on ANSI, ASCII, Unicode, UTF's and BOM's, please see Set Ups.h which is a copy of the information from the ***12d Model*** Reference manual.

A function return value of zero indicates the Text was successfully written.

ID = 2677

File_read(File file,Integer length,Text &value)

Name

Integer File_read(File file,Integer length,Text &value)

Description

Read **length** bytes from the binary file **file** and return it as Text in **value**.

Note - this only works for ANSI Text.

If any of the characters of Text is not ANSI, then a non-zero function return value is returned.

WARNING: This function is not to be used for Unicode files. For Unicode files, use File_read_unicode(File file,Integer length,Text &value) instead.

For more information on ANSI, ASCII, Unicode, UTF's and BOM's, please see Set Ups.h which is a copy of the information from the ***12d Model*** Reference manual.

A function return value of zero indicates the Text was successfully returned.

ID = 1712

File_write(File file,Integer length,Text value)

Name

Integer File_write(File file,Integer length,Text value)

Description

Write out **value** as **length** lots of one byte ANSI characters to the binary file **file**.

If any of the characters of Text is not ANSI, then no data is written out and a non-zero function return value is returned.

If there is less than **length** characters in Text then the number of characters is brought up to **length** by writing out null padding.

WARNING: This function is not to be used for Unicode files. For Unicode files, use File_write_unicode(File file,Integer length,Text value) instead.

For more information on ANSI, ASCII, Unicode, UTF's and BOM's, please see Set Ups.h which is a copy of the information from the ***12d Model*** Reference manual.

A function return value of zero indicates the Text was successfully written.

ID = 1715

File_read(File file,Integer length,Integer array[])

Name

Integer File_read(File file,Integer length,Integer array[])

Description

Read the next **length** lots of four bytes from the binary file **file** and return them as an Integer array in **array[].**

A function return value of zero indicates the Integer array was successfully returned.

ID = 1716

File_write(File file,Integer length,Integer array[])

Name

Integer File_write(File file,Integer length,Integer array[])

Description

Write out the Integer array **array[]** as **length** lots of four byte integers to the binary file **file**.

A function return value of zero indicates the Integer array was successfully written.

ID = 1718

File_read(File file,Integer length,Real array[])

Name

Integer File_read(File file,Integer length,Real array[])

Description

Read the next **length** lots of eight bytes from the binary file **file** and return them as a Real array in **array[].**

A function return value of zero indicates the Real array was successfully returned.

ID = 1717

File_write(File file,Integer length,Real array[])

Name

Integer File_write(File file,Integer length,Real array[])

Description

Write out the Integer array **array[]** as **length** lots of eight byte reals to the binary file **file**.

A function return value of zero indicates the Real array was successfully written.

ID = 1719

File_read_short(File file,Integer &value)

Name

Integer File_read_short(File file,Integer &value)

Description

Read two bytes from the binary file **file** and return it as an Integer in **value**.

A function return value of zero indicates the Integer was successfully returned.

ID = 1720

File_write_short(File file,Integer value)

Name

Integer File_write_short(File file,Integer value)

Description

Write out **value** as a two byte integer to the binary file **file**.

Because it is only a two byte integer, **value** must be between -2 to the power of 32, and +2 to the power 32.

A function return value of zero indicates the Integer was successfully written.

ID = 1722

File_read_short(File file,Real &value)

Name

Integer File_read_short(File file,Real &value)

Description

Read four bytes from the binary file **file** and return it as a Real in **value**.

Note - **value** can only be in the range -32,768 and 32,767.

A function return value of zero indicates the Real was successfully returned.

ID = 1721

File_write_short(File file,Real value)

Name

Integer File_write_short(File file,Real value)

Description

Write out **value** as a four byte real to the binary file **file**.

Because it is only a four byte real, only seven significant figures can be written out.

A function return value of zero indicates the Real was successfully written.

ID = 1723

File_close(File file)

Name

Integer File_close(File file)

Description

Close the File **file**.

A function return value of zero indicates **file** was closed successfully.

ID = 336

File_delete(Text file_name)

Name

Integer File_delete(Text file_name)

Description

Delete a file from the disk

A function return value of zero indicates the file was deleted.

ID = 213

File_set_endian(File file,Integer big)

Name

Integer File_set_endian(File file,Integer big)

Description

<not implemented>

ID = 1708

File_get_endian(File file,Integer &big)

Name

Integer File_get_endian(File file,Integer &big)

Description

<not implemented>

ID = 1709

12d Ascii

Read_4d_ascii(Text filename,Text prefix)

Name

Integer Read_4d_ascii(Text filename,Text prefix)

Description

Read in and process the file called **filename** as a 12d Ascii file. The post-prefix for models is given in **prefix**.

A function return value of zero indicates the file was successfully read.

ID = 1166

Read_4d_ascii(Text filename,Dynamic_Element &list)

Name

Integer Read_4d_ascii(Text filename,Dynamic_Element &list)

Description

Read the data from the 12d Ascii file called **filename** and load all the created Elements into the Dynamic_Element list.

A function return value of zero indicates the file was successfully read.

ID = 2073

Write_4d_ascii(Element elt,Text filename,Integer precision,Integer output_model_name)

Name

Integer Write_4d_ascii(Element elt,Text filename,Integer precision,Integer output_model_name)

Description

Open the file called **filename**, and append the 12d Ascii of the Element **elt** to the file. Any coordinates and Reals are written out to **precision** decimal places.

If **output_model_name** = 1 then write the name of the Model containing **elt** to the file before writing out **elt**.

If **output_model_name** = 0 then don't write out the Model name.

A function return value of zero indicates the data was successfully written.

ID = 1630

Write_4d_ascii(Dynamic_Element list,Text filename,Integer precision,Integer output_model_name)

Name

Integer Write_4d_ascii(Dynamic_Element list,Text filename,Integer precision,Integer output_model_name)

Description

Open the file called **filename**, and append the 12d Ascii of all the Elements in the Dynamic_Element **list** to the file. Any coordinates and Reals are written out to **precision** decimal

places.

If **output_model_name** = 1 then if write the name of the Model containing each Element to the file before writing out the Element. The Model name is not repeated if is the same as the previous Element).

If **output_model_name** = 0 then don't write out the Model names.

A function return value of zero indicates the data was successfully written.

ID = 1631

Write_4d_ascii(Model model,Text filename,Integer precision,Integer output_model_name)

Name

Integer Write_4d_ascii(Model model,Text filename,Integer precision,Integer output_model_name)

Description

Open the file called **filename**, and append the 12d Ascii of all the Elements in the Model **model** to the file. Any coordinates and Reals are written out to **precision** decimal places.

If **output_model_name** = 1 then write the name of **model** out to the file before the Elements.

If **output_model_name** = 0 then don't write out the Model name.

A function return value of zero indicates the data was successfully written.

ID = 1632

Write_4d_ascii(Element elt,File file,Integer precision,Integer indent_level)

Name

Integer Write_4d_ascii(Element elt,File file,Integer precision,Integer indent_level)

Description

Write the 12d Ascii of the Element **elt** to the File **file**. Any coordinates and Reals are written out to **precision** decimal places. The information written to the file is indented by **indent_level** spaces.

A function return value of zero indicates the data was successfully written.

ID = 1928

Write_4d_ascii(Element elt,File file,Integer precision,Integer indent_level,Text header)

Name

Integer Write_4d_ascii(Element elt,File file,Integer precision,Integer indent_level,Text header)

Description

Write the Text **header** to the File file and then write the 12d Ascii of the Element **elt** to the File **file**. Any coordinates and Reals are written out to **precision** decimal places. The information written to the file is indented by **indent_level** spaces.

A function return value of zero indicates the data was successfully written.

ID = 1929

Menus

Menus with the same look and feel as 12d Model menus can be easily created within 12dPL.

A 12dPL menu consists of a title and any number of menu options (called buttons) that are displayed one per line down the screen.

When the menu is displayed on the screen, the menu buttons will highlight as the cursor passes over them. If a menu button is selected (by pressing the LB whilst the button is highlighted), the menu will be removed from the screen and the user-defined code for the selected button returned to the macro.

To represent menus, 12dPL has a special variable type called **Menu**.

Screen Co-Ordinates

When placing Menus, screen positions are given as co-ordinates (across_pos,down_pos) where **across_pos** and **down_pos** are measured from the top left-hand corner of the 12d Model window.

The units for screen co-ordinates are pixels.

A full computer screen is approximately 1000 pixels across by 800 pixels down.

Create_menu(Text menu_title)

Name

Menu Create_menu(Text menu_title)

Description

A Menu is created which is used when referring to this particular menu. The menu title is defined when the menu variable is created and is the **Text menu_title**.

The function return value is the required Menu variable.

(To represent menus, 12dPL has this special variable type called **Menu**.)

ID = 171

Menu_delete(Menu menu)

Name

Integer Menu_delete(Menu menu)

Description

Delete the menu defined by Menu **menu**.

A function return value of zero indicates the menu was deleted successfully.

ID = 588

Create_button(Menu menu,Text button_text,Text button_reply)

Name

Integer Create_button(Menu menu,Text button_text,Text button_reply)

Description

This function adds *buttons* to the menu with **button_text** as the text for the button.

The button is also supplied with a Text **button_reply** which is returned to the macro through the function Display or Display_relative when the button is selected.

The menu buttons will appear in the Menu in the order that they are added to the menu structure by the Create_button function.

A function return value of zero indicates that the button was created successfully.

ID = 172

Display(Menu menu,Integer &across_pos,Integer &down_pos,Text &reply)

Name

Integer Display(Menu menu,Integer &across_pos,Integer &down_pos,Text &reply)

Description

When called, the Menu **menu** is displayed on the screen with screen co-ordinates (across_pos,down_pos).

The menu remains displayed on the screen until a menu button is selected by the user.

When a menu button is selected, the menu is removed from the screen and the appropriate button return code returned in the Text variable **reply**.

Whilst displayed on the screen, the menu can be moved around the 12d Model window by using the mouse. When a menu selection is finally made, the actual position of the menu at selection time is returned as (across_pos,down_pos).

A function return value of zero indicates that a successful menu selection was made.

Note

An (across_pos,down_pos) of (-1,-1) indicates the current cursor position.

ID = 173

Display_relative(Menu menu,Integer &across_rel,Integer &down_rel,Text &reply)

Name

Integer Display_relative(Menu menu,Integer &across_rel,Integer &down_rel,Text &reply)

Description

When called, the Menu **menu** is displayed on the screen with screen co-ordinates of (across_rel,down_rel) **relative** to the cursor position.

The menu remains displayed until a menu button is selected.

When a menu button is selected, the menu is removed from the screen and the appropriate button return code returned in the Text variable **reply**.

Whilst displayed, the menu can be moved in 12d Model by using the mouse. When the selection is made, the final **absolute** position of the menu is returned as (across_rel,down_rel).

A function return value of zero indicates that a successful menu selection was made.

Thus the sequence used to define and display a menu and the relevant functions used are:

(a) a Menu variable is created which is used when referring to this particular menu. The menu title is defined when the menu variable is created. Use:

```
Create_menu(Text menu_title)
```

For example

```
Menu menu = Create_menu("Test");
```

(b) the menu buttons are added to the menu structure in the order that they will appear in the menu. The button text and the text that will be returned to the macro if the button is selected are both supplied. Use:

```
Create_button(Menu menu,Text button_text,Text reply)
```

For example

```
Create_button(menu,"First options","Op1");
Create_button(menu,"Second options","Op2");
Create_button(menu,"Finish","Fin");
```

(c) the menu is displayed on the screen. The menu will continued to be displayed until a menu button is selected. When the menu button is selected, the menu is removed from the screen and the appropriate button return code returned to the macro.

Use:

```
Display(Menu menu,Integer row_pos,Integer col_pos,
        Text &reply)
```

```
Display_relative(Menu menu,Integer row_pos,Integer col_pos,
        Text &reply)
```

For example

```
Display(menu,5,10,reply);
```

A more complete example of defining and using a menu is:

```
void main()
{
 // create a menu with title "Silly Menu"
  Menu menu = Create_menu("Silly Menu");

  /* add menu button with titles "Read", "Write", "Draw"
    and "Quit". The returns codes for the buttons are
    the same as the button titles
  */

  Create_button(menu,"Read","Read");
  Create_button(menu,"Write","Write");
  Create_button(menu,"Draw","Draw");
  Create_button(menu,"Quit","Quit");

  /* display the menu on the screen at the current cursor
     position and wait for a button to selected.
     When a button is selected, print out its return code
     If the return code isn't "Quit", redisplay the menu.
  */

  Text reply;

  do {
     Display(menu,-1,-1,reply);
     Print(reply);  Print("\n");
  } while(reply != "Quit");
}
```

ID = 364

Dynamic Arrays

The 12dPL Dynamic Arrays are used to hold one or more items. That is, a Dynamic Arrays contains an arbitrary number of items.

The items in a Dynamic Array are accessed by their unique number position number in the Dynamic Array.

As for fixed arrays, the Dynamic Array positions go from one to the number of items in the Dynamic Array. However, unlike fixed arrays, extra items can be added to a Dynamic Array at any time.

Hence a 12dPL Dynamic Array can be thought of as a dynamic array of items.

The types of Dynamic Arrays are Dynamic_Element, Dynamic_Text, Dynamic_Real and Dynamic_Integer

For more information on Dynamic_Element, go to Dynamic Element Arrays .
Dynamic_Text, go to Dynamic Text Arrays .
Dynamic_Real, go to Dynamic Real Arrays .
Dynamic_Integer, go to Dynamic Integer Arrays .

Dynamic Element Arrays

The 12dPL variable type **Dynamic_Element** is used to hold one or more Elements. That is, a Dynamic_Element contains an arbitrary number of Elements.

The Elements in a Dynamic_Element are accessed by their unique number position number in the Dynamic_Element.

As for fixed arrays, the Dynamic_Element positions go from one to the number of Elements in the Dynamic_Element. However, unlike fixed arrays, extra Elements can be added to a Dynamic_Element at any time.

Hence a 12dPL Dynamic_Element can be thought of as a dynamic array of Elements.

The following functions are used to access and modify Elements in a Dynamic_Element.

Append(Dynamic_Element from_de,Dynamic_Element &to_de)

Name

Integer Append(Dynamic_Element from_de,Dynamic_Element &to_de)

Description

Append the contents of the Dynamic_Element **from_de** to the Dynamic_Element **to_de**.

A function return value of zero indicates the append was successful.

ID = 220

Null(Dynamic_Element &delt)

Name

Integer Null(Dynamic_Element &delt)

Description

Removes and nulls all the Elements from the Dynamic_Element **delt** and sets the number of items to zero.

A function return value of zero indicates that **delt** was successfully nulled.

ID = 127

Get_number_of_items(Dynamic_Element &delt,Integer &no_items)

Name

Integer Get_number_of_items(Dynamic_Element &delt,Integer &no_items)

Description

Get the number of Elements currently in the Dynamic_Element **delt**.

The number of Elements is returned in Integer **no_items**.

A function return value of zero indicates the number of Elements was returned successfully.

ID = 128

Get_item(Dynamic_Element &delt,Integer i,Element &elt)

Name

Integer Get_item(Dynamic_Element &delt,Integer i,Element &elt)

Description

Get the **i**th Element from the Dynamic_Element **delt**.

The Element is returned in **elt**.

A function return value of zero indicates the ith Element was returned successfully.

ID = 129

Set_item(Dynamic_Element &delt,Integer i,Element elt)

Name

Integer Set_item(Dynamic_Element &delt,Integer i,Element elt)

Description

Set the **i**th Element in the Dynamic_Element **delt** to the Element **elt**.

If the position **i** is greater or equal to the total number of Elements in the Dynamic_Element, then the Dynamic_Element will automatically be extended so that the number of Elements is i. Any extra Elements that are added will be set to null.

A function return value of zero indicates the Element was successfully set.

ID = 130

Null_item(Dynamic_Element &delt,Integer i)

Name

Integer Null_item(Dynamic_Element &delt,Integer i)

Description

Set the **i**th Element to null.

A function return value of zero indicates the Element was successfully set to null.

ID = 131

Dynamic Text Arrays

The 12dPL variable type Dynamic_Text is used to hold one or more Texts. That is, a Dynamic_Text contains an arbitrary number of Texts.

The Texts in a **Dynamic_Text** are accessed by their unique number position number in the Dynamic_Text.

As for fixed arrays, the Dynamic_Text positions go from one to the total number of items in the Dynamic_Text. However, unlike fixed arrays, extra Text can be added to a Dynamic_Text at any time.

Hence a 12dPL Dynamic_Text can be thought of as a dynamic array of Texts.

The following functions are used to access and modify Dynamic_Text's.

Append(Text text,Dynamic_Text &dt)

Name

Integer Append(Text text,Dynamic_Text &dt)

Description

Append the Text **text** to the end of the contents of the Dynamic_Text **dt**. This will increase the size of the Dynamic_Text by one.

A function return value of zero indicates the append was successful.

ID = 434

Append(Dynamic_Text from_dt,Dynamic_Text &to_dt)

Name

Integer Append(Dynamic_Text from_dt,Dynamic_Text &to_dt)

Description

Append the contents of the Dynamic_Text **from_dt** to the Dynamic_Text **to_dt**.

A function return value of zero indicates the append was successful.

ID = 230

Null(Dynamic_Text &dt)

Name

Integer Null(Dynamic_Text &dt)

Description

Removes and deletes all the Texts from the Dynamic_Text **dt** and sets the number of items to zero.

A function return value of zero indicates that **dt** was successfully nulled.

ID = 226

Get_number_of_items(Dynamic_Text &dt,Integer &no_items)

Name

Integer Get_number_of_items(Dynamic_Text &dt,Integer &no_items)

Description

Get the number of Texts currently in the Dynamic_Text **dt**.

The number of Texts is returned by Integer **no_items**.

A function return value of zero indicates the number of Texts was successfully returned.

ID = 227

Get_item(Dynamic_Text &dt,Integer i,Text &text)

Name

Integer Get_item(Dynamic_Text &dt,Integer i,Text &text)

Description

Get the **i**th Text from the Dynamic_Text **dt**.

The Text is returned by **text**.

A function return value of zero indicates the ith Text was returned successfully.

ID = 228

Set_item(Dynamic_Text &dt,Integer i,Text text)

Name

Integer Set_item(Dynamic_Text &dt,Integer i,Text text)

Description

Set the ith Text in the Dynamic_Text **dt** to the Text **text**.

A function return value of zero indicates success.

ID = 229

Get_all_linestyles(Dynamic_Text &linestyles)

Name

Integer Get_all_linestyles(Dynamic_Text &linestyles)

Description

Get all linestyle names defined in the Linestyles pop-up for the current project,

and return the list in the Dynamic_Text **linestyles**.

A function return value of zero indicates the linestyle names were returned successfully.

ID = 688

Get_all_textstyles(Dynamic_Text &textstyles)

Name

Integer Get_all_textstyles(Dynamic_Text &textstyles)

Description

Get all textstyle names defined in the Textstyles pop-up for the current project,

and return the list in the Dynamic_Text **textstyles**.

A function return value of zero indicates the textstyle names are returned successfully.

ID = 689

Get_all_symbols(Dynamic_Text &symbols)

Name

Integer Get_all_symbols(Dynamic_Text &symbols)

Description

Get all symbol names defined in the *Symbols* pop-up for the current project, and return the list in the Dynamic_Text **symbols**.

A function return value of zero indicates the symbol names were returned successfully.

ID = 1724

Get_all_patterns(Dynamic_Text &patterns)

Name

Integer Get_all_patterns(Dynamic_Text &patterns)

Description

Get all pattern names defined in the *Patterns* pop-up for the current project, and return the list in the Dynamic_Text **patterns**.

A function return value of zero indicates the function was successful.

ID = 1725

Dynamic Real Arrays

The 12dPL variable type Dynamic_Real is used to hold one or more Reals. That is, a Dynamic_Real contains an arbitrary number of Reals.

The Reals in a **Dynamic_Real** are accessed by their unique number position number in the Dynamic_Real.

As for fixed arrays, the Dynamic_Real positions go from one to the total number of items in the Dynamic_Real. However, unlike fixed arrays, extra Reals can be added to a Dynamic_Real at any time.

Hence a 12dPL Dynamic_Real can be thought of as a dynamic array of Reals.

The following functions are used to access and modify Dynamic_Real's.

Append(Real value,Dynamic_Real &real_list)

Name

Integer Append(Real value,Dynamic_Real &real_list)

Description

Append the Real **value** to the end of the contents of the Dynamic_Real **real_list**. This will increase the size of the Dynamic_Real by one.

A function return value of zero indicates the append was successful.

ID = 1795

Append(Dynamic_Real from_dr,Dynamic_Real &to_dr)

Name

Integer Append(Dynamic_Real from_dr,Dynamic_Real &to_dr)

Description

Append the contents of the Dynamic_Real **from_dr** to the Dynamic_Real **to_dr**.

A function return value of zero indicates the append was successful.

ID = 1794

Null(Dynamic_Real &real_list)

Name

Integer Null(Dynamic_Real &real_list)

Description

Removes all the Reals from the Dynamic_Real **real_list** and sets the number of items to zero.

A function return value of zero indicates that **real_list** was successfully nulled.

ID = 1790

Get_number_of_items(Dynamic_Real &real_list,Integer &no_items)

Name

Integer Get_number_of_items(Dynamic_Real &real_list,Integer &no_items)

Description

Get the number of Reals currently in the Dynamic_Real **real_list**.

The number of Reals is returned in Integer **no_items**.

A function return value of zero indicates the number of Reals was returned successfully.

ID = 1791

Set_item(Dynamic_Real &real_list,Integer index,Real value)

Name

Integer Set_item(Dynamic_Real &real_list,Integer i,Real value)

Description

Set the **i**th Real in the Dynamic_Real **real_list** to the Real **value**.

If the position **i** is greater or equal to the total number of Real in the Dynamic_Real, then the Dynamic_Real will automatically be extended so that the number of Reals is i. Any extra Real values that are added will be set to null (LJG? or zero?).

A function return value of zero indicates the Real was successfully set.

ID = 1793

Get_item(Dynamic_Real &real_list,Integer i,Real &value)

Name

Integer Get_item(Dynamic_Real &real_list,Integer index,Real &value)

Description

Get the **i**'th Real from the Dynamic_Real **real_list**.

The Real is returned in **value**.

A function return value of zero indicates the **i**'th Real was returned successfully.

ID = 1792

Dynamic Integer Arrays

The 12dPL variable type Dynamic_Integer is used to hold one or more Integers. That is, a Dynamic_Integer contains an arbitrary number of Integers.

The Integers in a **Dynamic_Integer** are accessed by their unique number position number in the Dynamic_Integer.

As for fixed arrays, the Dynamic_Integer positions go from one to the total number of items in the Dynamic_Integer. However, unlike fixed arrays, extra Integers can be added to a Dynamic_Integer at any time.

Hence a 12dPL Dynamic_Integer can be thought of as a dynamic array of Integers.

The following functions are used to access and modify Dynamic_Integer's.

Append(Integer value,Dynamic_Integer &integer_list)

Name

Integer Append(Integer value,Dynamic_Integer &integer_list)

Description

Append the Integer **value** to the end of the contents of the Dynamic_Integer **integer_list**. This will increase the size of the Dynamic_Integer by one.

A function return value of zero indicates the append was successful.

ID = 1785

Append(Dynamic_Integer from_di,Dynamic_Integer &to_di)

Name

Integer Append(Dynamic_Integer from_di,Dynamic_Integer &to_di)

Description

Append the contents of the Dynamic_Integer **from_di** to the Dynamic_Integer **to_di**.

A function return value of zero indicates the append was successful.

ID = 1784

Null(Dynamic_Integer &integer_list)

Name

Integer Null(Dynamic_Integer &integer_list)

Description

Removes all the Integers from the Dynamic_Integer **integer_list** and sets the number of items to zero.

A function return value of zero indicates that **integer_list** was successfully nulled.

ID = 1780

Get_number_of_items(Dynamic_Integer &integer_list,Integer &count)

Name

Integer Get_number_of_items(Dynamic_Integer &integer_list,Integer &count)

Description

Get the number of Integers currently in the Dynamic_Integer **integer_list**.

The number of Integers is returned in Integer **no_items**.

A function return value of zero indicates the number of Integers was returned successfully.

ID = 1781

Set_item(Dynamic_Integer &integer_list,Integer i,Integer value)

Name

Integer Set_item(Dynamic_Integer &integer_list,Integer i,Integer value)

Description

Set the **i**th Integer in the Dynamic_Integer **integer_list** to the Integer **value**.

If the position **i** is greater or equal the total number of Integer in the Dynamic_Integer, then the Dynamic_Integer will automatically be extended so that the number of Integers is i. Any extra Integer values that are added will be set to zero (LJG? or zero?).

A function return value of zero indicates the Integer was successfully set.

ID = 1783

Get_item(Dynamic_Integer &integer_list,Integer i,Integer &value)

Name

Integer Get_item(Dynamic_Integer &integer_list,Integer i,Integer &value)

Description

Get the **i**'th Integer from the Dynamic_Integer **integer_list**.

The Integer is returned in **value**.

A function return value of zero indicates the **i**'th Integer was returned successfully.

ID = 1782

Points

A variable of type Point in created in the same way as Integers and Reals. That is, the Point variable name is given after the Point declaration.

For example, a Point of name **pt** is created by:

Point pt;

When the Point **pt** is created, it has the default co-ordinates of (0,0,0).

The co-ordinates for **pt** can then be set to new values using Set commands.

Get_x(Point pt)

Name

Real Get_x(Point pt)

Description

Get the x co-ordinate of the Point **pt**.

The function return value is the x co-ordinate value of **pt**.

ID = 241

Get_y(Point pt)

Name

Real Get_y(Point pt)

Description

Get the y co-ordinate of the Point **pt**.

The function return value is the y co-ordinate value of **pt**.

ID = 242

Get_z(Point pt)

Name

Real Get_z(Point pt)

Description

Get the z co-ordinate of the Point **pt**.

The function return value is the z co-ordinate value of **pt**.

ID = 243

Set_x(Point &pt,Real x)

Name

Real Set_x(Point &pt,Real x)

Description

Set the x co-ordinate of the Point **pt** to the value **x**.

The function return value is the x co-ordinate value of **pt**.

ID = 244

Set_y(Point &pt,Real y)

Name

Real Set_y(Point &pt,Real y)

Description

Set the y co-ordinate of the Point pt to the value y.

The function return value is the y co-ordinate value of pt.

ID = 245

Set_z(Point &pt,Real z)

Name

Real Set_z(Point &pt,Real z)

Description

Set the z co-ordinate of the Point **pt** to the value **z**.

The function return value is the z co-ordinate value of **pt**.

ID = 246

Lines

A **Line** is three dimensional line joining two **Points**.

A variable of type Line is created in the same way as Points. That is, the Line variable name is given after the Line declaration.

For example, a Line of name line created by:

Line line;

When the Line **line** is created, it has default start and end Points with co-ordinates of (0,0,0).

The co-ordinates for the start and end Points of the Line line can then be set to new values using Set commands.

The direction of the Line is from the start point to the end point.

Get_start(Line line)

Name

Point Get_start(Line line)

Description

Get the start Point of the Line **line**.

The function return value is the start Point of **line**.

ID = 251

Get_end(Line line)

Name

Point Get_end(Line line)

Description

Get the end Point of the Line **line**.

The function return value is the start Point of **line**.

ID = 252

Set_start(Line &line, Point pt)

Name

Point Set_start(Line &line, Point pt)

Description

Set the start Point of the Line **line** to be the Point **pt**.

The function return value is also the start Point of **line**.

ID = 253

Set_end(Line &line, Point pt)

Name

Point Set_end(Line &line, Point pt)

Description

Set the end Point of the Line **line** to be the Point **pt**.

The function return value is also the end Point of **line**.

ID = 254

Reverse(Line line)

Name

Line Reverse(Line line)

Description

Reverse the direction of the Line **line**.

That is, Reverse swaps the start and end Points of the Line **line**.

The unary operator "-" will also reverse a Line.

The function return value is the reversed Line.

ID = 255

Arcs

A 12dPL Arc is a helix which projects onto a circle in the (x,y) plane.

An Arc has a radius and Points for its centre, start and end. The radius can be positive or negative (but not zero).

A positive radius indicates that the direction of travel between the start and end points is in the clockwise directions (*to the right*).

A negative radius indicates that the direction of travel between the start and end points is in the anti-clockwise direction (*to the left*).

A variable of type Arc is created in the same way as Points and Lines. That is, the Arc variable name is given after the Arc declaration.

For example, an Arc of name arc created by:

```
Arc arc;
```

When the Arc **arc** is created, it has default centre (0,0,0), start, end Points with co-ordinates of (1,0,0) and a radius of one.

The radius and co-ordinates for centre, start and end points of the Arc can then be set to new values using Set commands.

Creating an Arc

A 12dPL Arc can be created by first setting the radius and the (x,y) co-ordinates of the centre point to define a plan circle.

This defines the unique plan circle that the 12dPL Arc projects onto.

Next the (x,y) part of the start and end points are dropped perpendicularly onto the plan circle to define the start and the end points of the plan projection of the arc. Thus the start and end points used to define the arc may not lie on the created arc but stored projected points will.

Finally, the arc is given the start and end heights of the start and end points respectively.

WARNING

For a new Arc, the radius and centre point **must** be defined before the start and end points.

Get_centre(Arc arc)

Name

Point Get_centre(Arc arc)

Description

Get the centre point of the Arc **arc**.

The function return value is the centre point of the arc.

ID = 260

Get_radius(Arc arc)

Name

Real Get_radius(Arc arc)

Description

Get the radius of the Arc **arc**.

The function return value is the radius of the arc.

ID = 261

Get_start(Arc arc)

Name

Point Get_start(Arc arc)

Description

Get the start point of the Arc **arc**.

The function return value is the start point of the arc.

ID = 262

Get_end(Arc arc)

Name

Point Get_end(Arc arc)

Description

Get the end point of the Arc **arc**.

The function return value is the end point of the arc.

ID = 263

Set_centre(Arc &arc,Point pt)

Name

Point Set_centre(Arc &arc,Point pt)

Description

Set the centre point of the Arc arc to be the Point pt. The start and end points are also translated by the vector between the new and old arc centres.

The function return value is the centre point of the arc.

ID = 264

Set_radius(Arc &arc,Real rad)

Name

Real Set_radius(Arc &arc,Real rad)

Description

Set the radius of the Arc arc to the value **rad**. The start and end points are projected radially onto the new arc.

The function return value is the radius of the arc.

ID = 265

Set_start(Arc &arc,Point start)

Name

Point Set_start(Arc &arc,Point start)

Description

Set the start point of the Arc arc to be the Point start. If the start point is not on the Arc, the point is dropped perpendicularly onto the Arc to define the actual start point that lies on the Arc.

The function return value is the actual start point on the arc.

ID = 266

Set_end(Arc &arc,Point end)

Name

Point Set_end(Arc &arc,Point end)

Description

Set the end point of the Arc **arc** to be the Point **end**. If the end point is not on the Arc, the point is dropped perpendicularly onto the Arc to define the actual end point that lies on the Arc.

The function return value is the actual end point on the arc.

ID = 267

Reverse(Arc arc)

Name

Arc Reverse(Arc arc)

Description

Reverse the sign of the radius and swap the start and end points of the Arc arc. Hence the direction of travel for the Arc is reversed.

The unary operator "-" will also reverse an Arc.

The function return value is the Arc **arc**.

ID = 268

Spirals and Transitions

There is often confusion between the words spirals and transitions.

Basically a **transition** is a curve which starts with a **radius** of curvature of infinity, and the **radius** of curvature then **continuously decreases** along the transition until it reaches a **final value** of **R**.

The purpose of a transition is to have a curve to join straights and arcs so that the radius of curvature varies continuously between the infinite radius on the straight and the radius of curvature on the arc (the radius of curvature of an arc is the arc radius). So a transition is used to makes a smooth transition from a straight to an arc.

A **spiral** (also known as Euler spiral, or natural or a clothoid) is a special curve defined for each point on the curve by:

r x **len** = a constant = K

where **r** is the radius of curvature at a point and **len** is the length of the curve to that point.

This spiral is the most common theoretical transition used in road design (and some rail design) however because the definition was difficult to use with hand calculations, various approximations to the real spiral have been used.

For example, what is normally called a clothoid by most road authorities is only an approximation to the full spiral. The Westrail Cubic used by Westrail in Western Australia is a different approximation. The Cubic Spiral is another very simple approximation used in early textbooks.

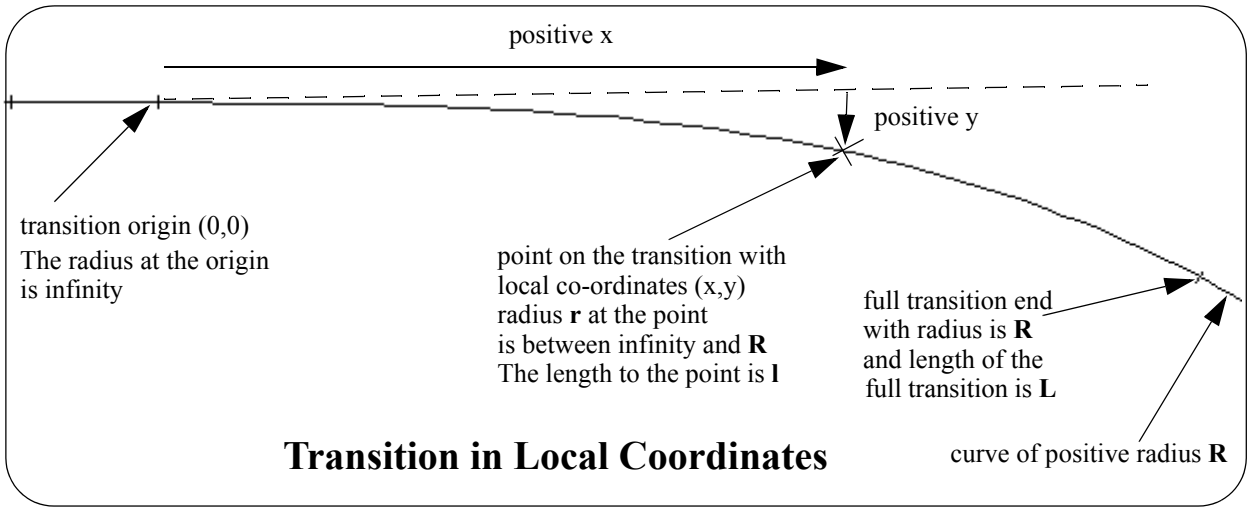
Examples of a common transitions used (mainly for rail) are:

Cubic Parabola - used by NSW Railways. This is NOT a spiral.
Bloss
Sinusoidal
Cosinusoidal

So in its basic form, a transition starts with an infinite radius of curvature, and ends with a radius of curvature of **R** and a total transition length of **L**.

R can be:
positive. The transition will then curve to the **right**
or
or **negative**. The transition will curve to the **left**. The start radius of curvature would then be considered to be negative infinity.

The transition can be drawn in local co-ordinates with the origin (0,0) at the point where the radius of curvature is infinity.



Transition in Local Coordinates

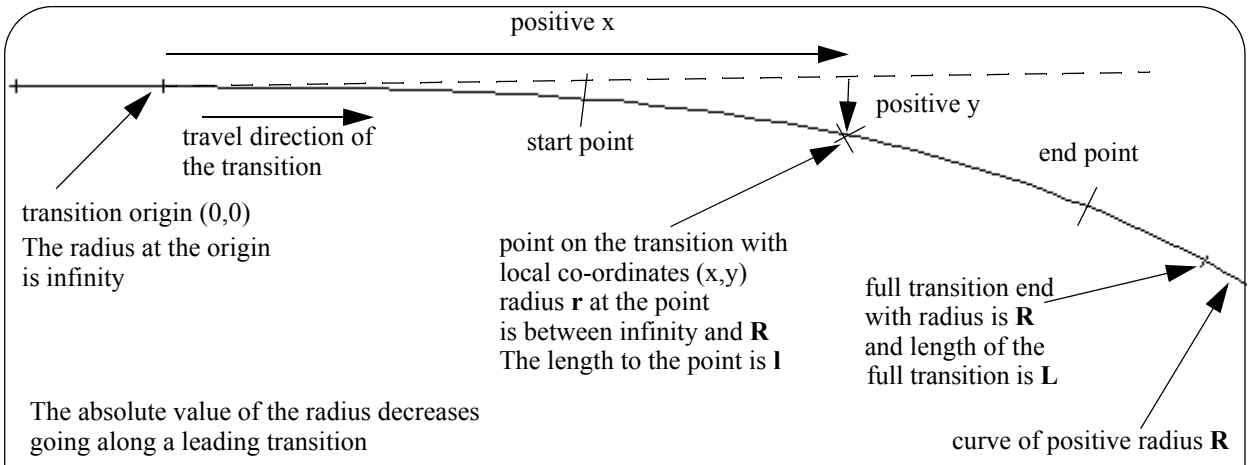
Sometimes the full transition curve is not required and only a part of the transition is used. The transition is only used from a **start point** (at transition length **start length** from the beginning of the full transition), to and **end point** (at transition length **end length** from the beginning of the full transition).

In practise transitions are required to be used in both directions. That is, starting on a straight and ending on a curve, or starting on a curve and ending on a straight.

So a

leading transition starts on a straight and ends on an arc of absolute value R. The absolute value of the radius of curvature goes from infinity to a value R.

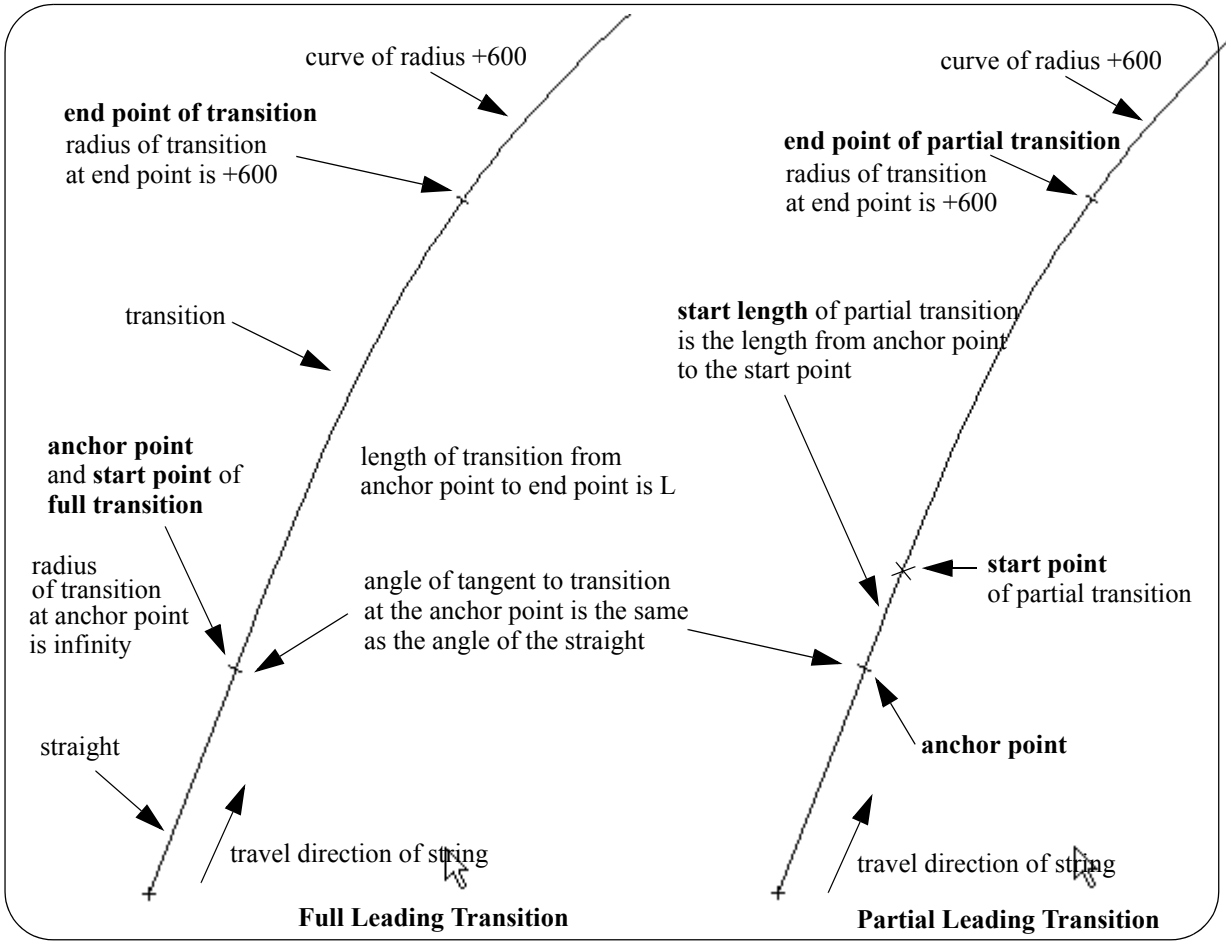
trailing transition starts on a curve of absolute radius R and ends on a straight. The absolute value of the radius of curvature goes from infinity to a value R

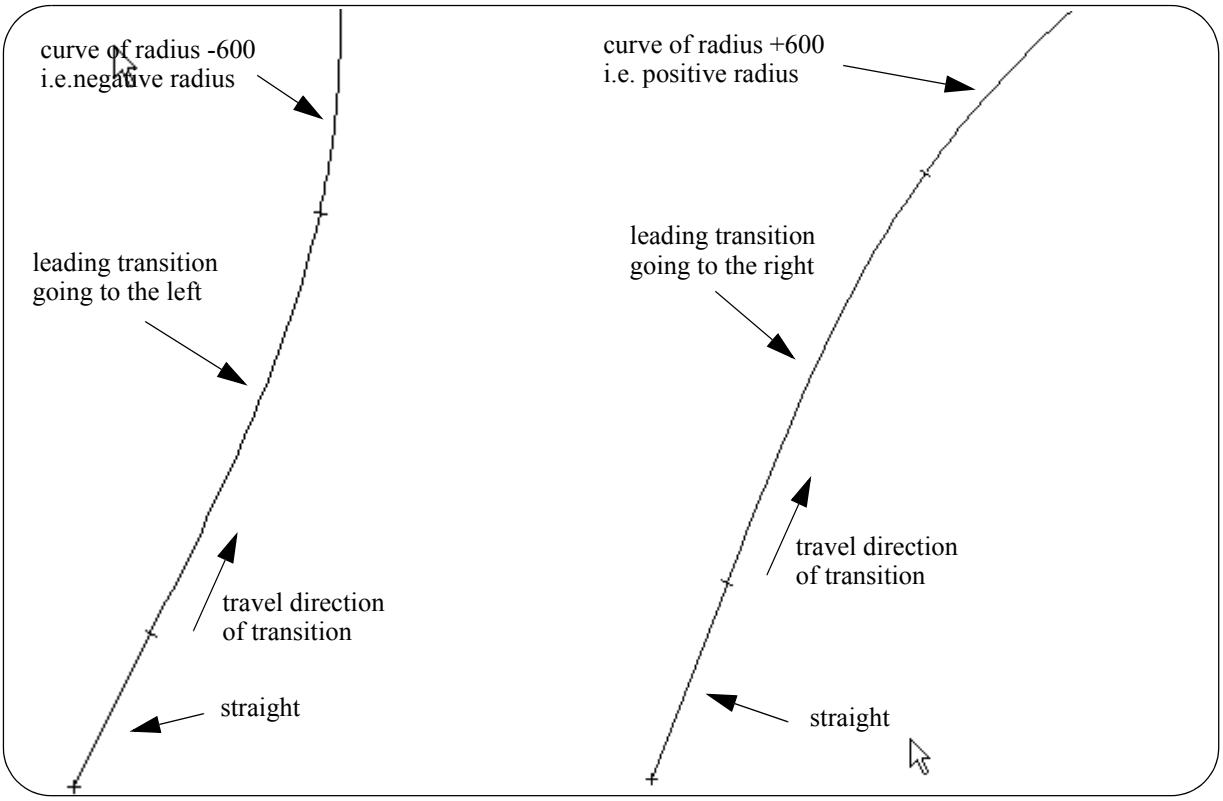


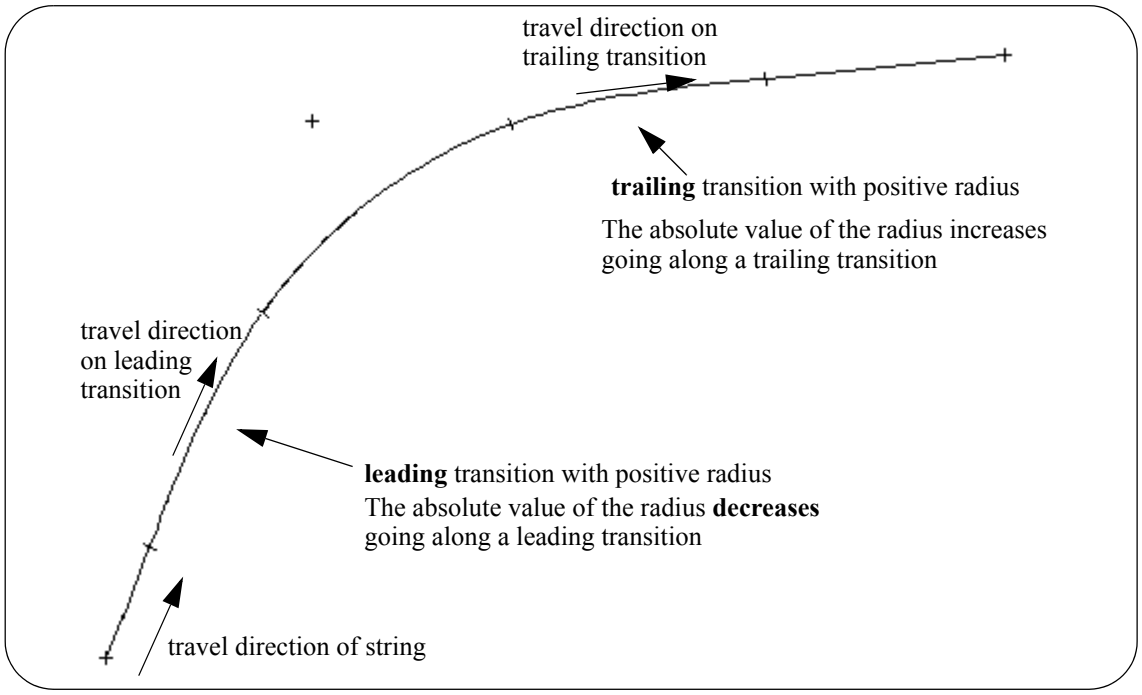
A Leading Transition in Local Coordinates

Finally the transition needs to be placed in world coordinates.

So to position the transition in world coordinates, the local transition origin (0,0) is translated to the position (x,y) (called the **anchor point** of the transition) and the transition is rotated about the anchor point though the angle **direction** (the angle is measure in a counterclockwise direction from the positive x axis). So the at the anchor point will be at the angle **direction**.


curve of radius +600
end point of transition
radius of transition
at end point is +600
transition
anchor point
and **start point** of
full transition
length of transition from
anchor point to end point is L
radius
of transition
at anchor point
is infinity
angle of tangent to transition
at the anchor point is the same
as the angle of the straight
straight
travel direction of string
Full Leading Transition
curve of radius +600
end point of partial transition
radius of transition
at end point is +600
start length of partial transition
is the length from anchor point
to the start point
start point
of partial transition
anchor point
travel direction of string
Partial Leading Transition


curve of radius -600
i.e.negative radius
leading transition
going to the left
travel direction
of transition
straight
curve of radius +600
i.e. positive radius
leading transition
going to the right
travel direction
of transition
straight



In *12d Model*, a variable of type **Spiral** exists to define and manipulate transitions and it is used in the same way as variable types Points, Lines and Arcs. That is, a Spiral variable name is given after the Spiral declaration.

Note: the radius of curvature at a point on a transition is simply referred to as the **radius** at that point.

Defining a Transition

A 12dPL transition (Spiral) is defined by giving:

(a) the transition type

(b) the length of the full transition **L**

(c) the radius **R** at length L That is, the radius at the end of the full transition. This is a signed radius.

(d) the **start length** for the part of the full transition that is actually going to be used. - the transition length from the start of the

This is enough to define the full transition in Local Transition Coordinates with origin at (0,0).

(e) the (x,y) position of the **anchor point**. That is the real world co-ordinates (x,y) of what is the origin in local transition coordinates. It if the real world coordinates of the point on the full transition where the radius is infinity.

(f) the angle of the tangent of the transition at the anchor point (the **direction**).

This defines where the full transition is in world coordinates.

(g) the start length - the length of transition from the anchor point (the position on the full transition where the radius in infinity) to what is the first position used on the transition

(h) the end length - the length of transition from the anchor point (the position on the transition where the radius in infinity) to what is final position used on the transition

This finally defines what part of the full transition is actually used.

Set_type(Spiral spiral,Integer type)

Name

Integer Set_type(Spiral spiral,Integer type)

Description

LJG - this could have problems with changes. This is broken for V8, V9, V10

V7? depends on file Spirals.4d; type = 0 clothoid, 1 westrail cubic, 2 cubic spiral 3 natural clothoid (LandXML) 4 NSW cubic parabola

V9? type = 1 clothoid, 2 westrail cubic, 3 clothoid LandXML 4 Cubic spiral 5 Natural clothoid 6 Cubic parabola

ID = 1805

Set_leading(Spiral transition,Integer leading)

Name

Integer Set_leading(Spiral transition,Integer leading)

Description

Set whether **transition** is a leading transition (radius decreases along the transition) or a trailing transition (radius increases along the transition).

If **leading** is non-zero then it is a leading transition.
If **leading** is zero then it is a trailing transition.

A function return value of zero indicates that the function call was successful.

ID = 1806

Set_length(Spiral transition,Real length)

Name

Integer Set_length(Spiral transition,Real length)

Description

Set the length of the full length **transition** to **length**.

A function return value of zero indicates that the function call was successful.

Note - the length of the transition is defined from the position on the transition where the radius is infinity (i.e. is a straight) to the other end of the transition.

For a *leading* transition, the radius is infinity at the start of the transition.
For a *trailing* transition, the radius is infinity at the end of the transition.

ID = 1807

Set_radius(Spiral trans,Real radius)

Name

Integer Set_radius(Spiral trans,Real radius)

Description

Sign of radius.
For a *leading* transition, set the end radius of the transition **trans** to **radius**.
For a *trailing* transition, set the start radius of the transition **trans** to **radius**.

Note - the radius is a signed value.
If radius > 0 the transition curves to the right.
If radius <0, the transition curves to the left.

A function return value of zero indicates that the function call was successful.

ID = 1808

Set_direction(Spiral trans,Real angle)

Name

Integer Set_direction(Spiral trans,Real angle)

Description

For the end of the transition **trans** where the radius is infinity, set the angle of the tangent at that position to **angle**. **angle** is in radians and is measured in a counterclockwise direction from the positive x-axis.

For a *leading* transition, set the angle of the tangent at the start of **trans** to **angle**.
For a *trailing* transition, set the angle of the tangent at the end of **trans** to **angle**.

A function return value of zero indicates that the function call was successful.

ID = 1809

Set_anchor(Spiral trans,Real point)

Name

Integer Set_anchor(Spiral trans,Real point)

Description

For the end of the transition **trans** where the radius is infinity, set the co-ordinates of that position to **point**.

For a *leading* transition, the anchor point is the start of **trans**.
For a *trailing* transition, the anchor point is the end of **trans**.

A function return value of zero indicates that the function call was successful.

ID = 1810

Set_start_length(Spiral trans,Real start_length)

Name

Integer Set_start_length(Spiral trans,Real start_length)

Description

Set the start length of the transition **trans** to **start_length**.

A function return value of zero indicates that the function call was successful.

Note - the start length is the distance from the position on the full transition where the radius is infinity (anchor point) to the start of the transition. If the start_length is non-zero then it is not a full transition but a partial transition.

ID = 1811

Set_end_length(Spiral trans,Real length)

Name

Integer Set_end_length(Spiral trans,Real end_length)

Description

Set the end length of the transition **trans** to **end_length**.

The end length is the distance from the position on the full transition where the radius is infinity to the point on the transition where no more of the transition is used.

A function return value of zero indicates that the function call was successful.

Note: even through the full transition has a length of L say, the part of the transition that is actually used is only from the **start length** to the **end length**.

ID = 1812

Set_start_height(Spiral trans,Real height)

Name

Integer Set_start_height(Spiral trans,Real height)

Description

For the transition **trans**, set the z-value at the position **start length** along the transition to **height**.

A function return value of zero indicates that the function call was successful.

ID = 1813

Set_end_height(Spiral trans,Real height)

Name

Integer Set_end_height(Spiral trans,Real height)

Description

For the transition **trans**, set the z-value at the position **end length** along the transition to **height**.

A function return value of zero indicates that the function call was successful.

ID = 1814

Get_valid(Spiral trans)

Name

Integer Get_valid(Spiral trans)

Description

If **trans** is a valid transition, then the function return value is zero.

If **trans** is not a valid transition, then the function return value is non-zero.

Note - the parameters given to define the transition may be inconsistent and not be able to define an actual transition.

ID = 1815

Get_type(Spiral trans)

Name

Integer Get_type(Spiral trans)

Description

LJG? yes what are they?

ID = 1816

Get_leading(Spiral trans)

Name

Integer Get_leading(Spiral trans)

Description

A transition is a leading transition if the radius decreases along the transition, or a trailing transition if the radius increases along the transition.

If **trans** is a leading transition then return a non-zero function return value.
If **trans** is a trailing transition then return zero as the function return value.

ID = 1817

Get_length(Spiral trans)

Name

Real Get_length(Spiral trans)

Description

For the full transition of **trans**, return the length to the end of the full transition as the function return value.

ID = 1818

Get_radius(Spiral trans)

Name

Real Get_radius(Spiral trans)

Description

For a *leading* transition **trans**, get the radius at the end of the full transition and return it as the function return value.

For a *trailing* transition **trans**, get the radius at the start of the full transition and return it as the function return value.

ID = 1819

Get_direction(Spiral trans)

Name

Real Get_direction(Spiral trans)

Description

Get the *angle* of the tangent at the anchor point (the end of the transition **trans** where the radius is infinity), and return it as the function return value.

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

For a *leading* transition **trans**, it is the angle of the tangent at the start of the full transition.
For a *trailing* transition **trans**, it is the angle of the tangent at the end of the full transition.

ID = 1820

Get_anchor(Spiral trans)

Name

Point Get_anchor(Spiral trans)

Description

Get the co-ordinates of the anchor point (the end of the full transition where the radius is infinity), and return them as the function return value.

For a *leading* transition **trans**, the anchor point is the start of the full transition.
For a *trailing* transition **trans**, the anchor point is the end of the full transition.

ID = 1821

Get_start_length(Spiral trans)

Name

Real Get_start_length(Spiral trans)

Description

Get the start length of the transition **trans** and return it as the function return value.

ID = 1822

Get_end_length(Spiral trans)

Name

Real Get_end_length(Spiral trans)

Description

Get the end length of the transition **trans** and return it as the function return value.

ID = 1823

Get_start_height(Spiral trans)

Name

Real Get_start_height(Spiral trans)

Description

For the transition **trans**, get the height at the position **start length** along the transition and return it as the function return value.

ID = 1824

Get_end_height(Spiral trans)

Name

Real Get_end_height(Spiral trans)

Description

For the transition **trans**, get the height at the position **end length** along the transition and return it as the function return value.

ID = 1825

Get_start_point(Spiral trans)

Name

Point Get_start_point(Spiral trans)

Description

For the transition **trans**, get the Point at the position **start length** along the transition and return it as the function return value.

ID = 1826

Get_end_point(Spiral trans)

Name

Point Get_end_point(Spiral trans)

Description

For the transition **trans**, get the Point at the position **end length** along the transition and return it as the function return value.

ID = 1827

Get_local_point(Spiral trans,Real len)

Name

Point Get_local_point(Spiral trans,Real len)

Description

For the transition **trans**, get the *local* co-ordinates (as a Point) of the position at length **len** from the start of the ***full transition*** and return it as the function return value.

Note - the transition is in world coordinates and needs to be translated and rotated before getting the local coordinates of the position at length **len** along the transition.

ID = 1828

Get_point(Spiral trans,Real len)

Name

Point Get_point(Spiral trans,Real len)

Description

For the transition **trans**, get the co-ordinates of the position (as a Point) at length **len** from the start of the ***full transition,*** and return it as the function return value.

ID = 1829

Get_local_angle(Spiral trans,Real len)

Name

Real Get_local_angle(Spiral trans,Real len)

Description

For the transition **trans**, get the *local* angle of the tangent at the position at length **len** from the start of the ***full transition,*** and return it as the function return value.

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

Note - the transition is in world coordinates and needs to be translated and rotated before getting the angle of the tangent of the position at length **len** along the transition.

ID = 1830

Get_angle(Spiral trans,Real len)

Name

Real Get_angle(Spiral trans,Real len)

Description

For the transition **trans**, get the angle of the tangent of the position at length **len** from the start of the ***full transition,*** and return it as the function return value.

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

ID = 1831

Get_radius(Spiral trans,Real len)

Name

Real Get_radius(Spiral trans,Real len)

Description

For the transition **trans**, get the radius at the position at length **len** from the start of the ***full transition,*** and return it as the function return value.

ID = 1832

Get_shift_x(Spiral trans)

Name

Real Get_shift_x(Spiral trans)

Description

shift at end point of transition **trans** (what is x/y which is offset, which is along tangent)

ID = 1833

Get_shift_y(Spiral trans)

Name

Real Get_shift_y(Spiral trans)

Description

shift at end point of transition **trans**

ID = 1834

Get_shift(Spiral trans)

Name

Real Get_shift(Spiral trans)

Description

shift

ID = 1835

Reverse(Spiral trans)

Name

Spiral Reverse(Spiral trans)

Description

Create a Spiral that is the same as transition **trans** but has the reverse travel direction. The created transition is returned as the function return value.

So a leading transition becomes a trailing transition and a trailing transition becomes a leading transition.

The unary operator "-" will also reverse a Spiral.

The function return value is the reversed Spiral.

ID = 1803

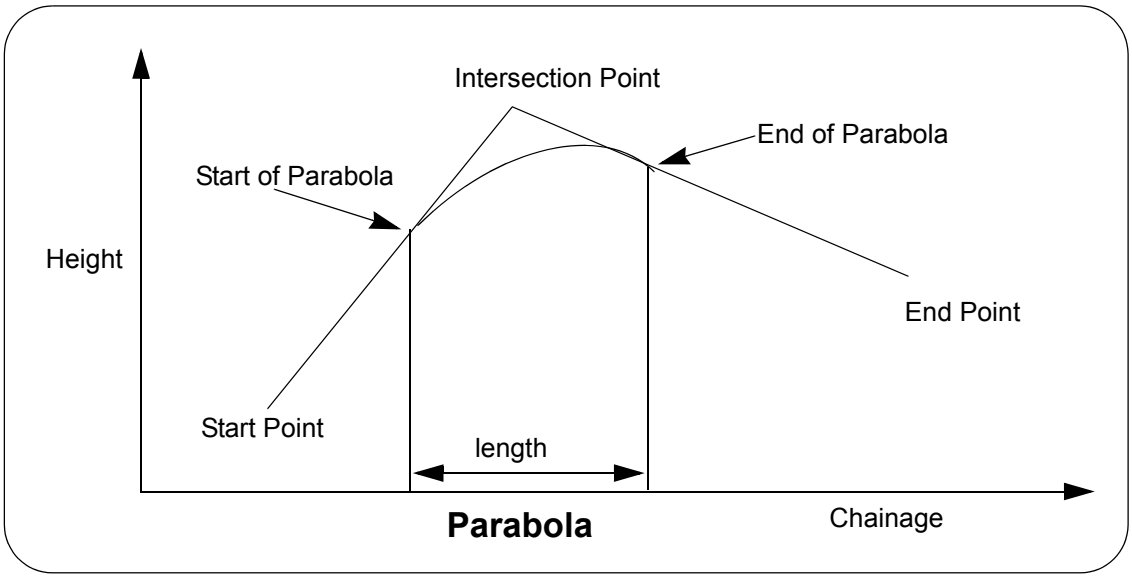
Parabolas

Parabolas are used in the vertical geometry of an Alignment or Super Alignment. The vertical geometry is defined in the (chainage, height) plane and are placed on vertical intersection points. So the parabola is defined in the (chainage, height) plane.

In 12dPL, a Parabola is a construction entity and is not stored in **12d Model** models.

A Parabola is defined by a start point, an intersection point and end point. The start point to the intersection point, and the intersection point to the end point define the start grade and the end grade of the parabola.

The parabola is then finally defined by giving the chainage distance between the beginning of the parabola and the end of the parabola. This is called the **length** of the parabola.



Segments

A **Segment** is either a **Point**, **Line, Arc** or a **Spiral**.

A **Segment** has a unique type that specifies whether it is a **Point**, **Line, Arc** or a **Spiral**.

Note: a **Spiral** is a general transition, not just a clothoid spiral.

Get_type(Segment segment)

Name

Integer Get_type(Segment segment)

Description

Get the type of the Segment segment.

A Segment type of

1	denotes a Point
2	denotes a Line
3	denotes an Arc
4	denotes a Spiral

The function return value is the Segment type.

ID = 273

Get_point(Segment segment,Point &point)

Name

Integer Get_point(Segment segment,Point &point)

Description

If the Segment is of type 1, the Point of the Segment is returned as **point**, otherwise it is an error.

A function return value of zero indicates the Segment was a Point Segment and that the Point was returned successfully.

ID = 274

Get_line(Segment segment,Line &line)

Name

Integer Get_line(Segment segment,Line &line)

Description

If the Segment is of type 2, the Line of the Segment is returned as **line**, otherwise it is an error.

A function return value of zero indicates the Segment was a Line Segment and that the Line was returned successfully.

ID = 275

Get_arc(Segment segment,Arc &arc)

Name

Integer Get_arc(Segment segment,Arc &arc)

Description

If the Segment is of type 3, the Arc of the Segment is returned as **arc**, otherwise it is an error.

A function return value of zero indicates the Segment was an Arc Segment and that the Arc was returned successfully.

ID = 276

Get_spiral(Segment segment,Spiral &trans)

Name

Integer Get_spiral(Segment segment,Spiral &trans)

Description

If the Segment is of type 4, the Spiral of the Segment is returned as transition **trans**, otherwise it is an error.

A function return value of zero indicates the Segment was an Spiral Segment and that the Spiral was returned successfully.

ID = 1837

Get_start(Segment segment,Point &point)

Name

Integer Get_start(Segment segment,Point &point)

Description

Get the start Point of the Segment **segment**.

The start value is returned by Point **point**.

A function return value of zero indicates the start point was successfully returned.

ID = 550

Get_end(Segment segment,Point &point)

Name

Integer Get_end(Segment segment,Point &point)

Description

Get the end Point of the Segment segment.

The end value is returned by Point point.

A function return value of zero indicates the end point was successfully returned.

ID = 551

Set_point(Segment &segment,Point point)

Name

Integer Set_point(Segment &segment,Point point)

Description

Sets the Segment type to 1 and the Point of the Segment to **point**.

A function return value of zero indicates the Segment was successfully set.

ID = 277

Set_line(Segment &segment,Line line)

Name

Integer Set_line(Segment &segment,Line line)

Description

Sets the Segment type to 2 and the Line of the Segment to **line**.

A function return value of zero indicates the Segment was successfully set.

ID = 278

Set_arc(Segment &segment,Arc arc)

Name

Integer Set_arc(Segment &segment,Arc arc)

Description

Sets the Segment type to 3 and the Arc of the Segment to **arc**.

A function return value of zero indicates the Segment was successfully set.

ID = 279

Set_spiral(Segment &segment,Spiral trans)

Name

Integer Set_spiral(Segment &segment,Spiral trans)

Description

Sets the Segment type to 4 and the Spiral of the Segment to transition **trans**.

A function return value of zero indicates the Segment was successfully set.

ID = 1836

Set_start(Segment &segment,Point point)

Name

Integer Set_start(Segment &segment,Point point)

Description

Set the start Point of the Segment **segment**.

The start value is defined by Point **point**.

A function return value of zero indicates the start point was successfully set.

ID = 552

Set_end(Segment &segment,Point point)

Name

Integer Set_end(Segment &segment,Point point)

Description

Set the end Point of the Segment **segment**.

The end value is defined by Point **point**.

A function return value of zero indicates the end point was successfully set.

ID = 553

Reverse(Segment segment)

Name

Segment Reverse(Segment segment)

Description

Reverse the direction of the Segment **segment**.

Note that the reverse of a segment of type 1 (a Point segment) is simply a point of exactly the same co-ordinates.

The unary operator "-" will also reverse a Segment.

The function return value is the reversed Segment.

ID = 280

Get_segments(Element elt,Integer &nsegs)

Name

Integer Get_segments(Element elt,Integer &nsegs)

Description

Get the number of segments for a string Element **elt**.

The number of segments is returned as **nsegs**

A function return value of zero indicates the data was successfully returned.

Note

If a string has n points, then it has n-1 segments.

For example, a seven point string consists of six segments.

ID = 545

Get_segment(Element elt,Integer i,Segment &seg)

Name

Integer Get_segment(Element elt,Integer i,Segment &seg)

Description

Get the segment for the ith segment on the string.

The segment is returned as **seg**.

The types of segments returned are Line, or Arc.

A function return value of zero indicates the data was successfully returned.

ID = 546

Segment Geometry

Length and Area

Get_length(Segment segment,Real &length)

Name

Integer Get_length(Segment segment,Real &length)

Description

Get the plan length of the Segment segment.

A function return value of zero indicates the plan length was successfully returned.

ID = 361

Get_length_3d(Segment segment,Real &length)

Name

Integer Get_length_3d(Segment segment,Real &length)

Description

Get the 3d length of the Segment **segment**.

A function return value of zero indicates the 3d length was successfully returned.

ID = 362

Plan_area(Segment segment,Real &plan_area)

Name

Integer Plan_area(Segment segment,Real &plan_area)

Description

Calculate the plan area of the Segment segment. For an Arc, the plan area of the sector is returned. For a Line and a Point, zero area is returned.

The area is returned in the Real plan_area.

A function return value of zero indicates the plan area was successfully returned.

ID = 360

Parallel

The parallel command is a plan parallel and is used for Lines, Arcs and Segments.

The sign of the distance to parallel the object is used to indicate whether the object is parallelled to the left or to the right.

A **positive** distance means to parallel the object to the **right**.

A **negative** distance means to parallel the object to the **left**.

Parallel(Line line,Real distance,Line &parallelled)

Name

Integer Parallel(Line line,Real distance,Line &parallelled)

Description

Plan parallel the Line **line** by the distance distance.

The parallelled Line is returned as the Line **parallelled**. The z-values are not modified, i.e. they are the same as for **line**.

A function return value of zero indicates the parallel was successful.

ID = 284

Parallel(Arc arc,Real distance,Arc &parallelled)

Name

Integer Parallel(Arc arc,Real distance,Arc &parallelled)

Description

Plan parallel the Arc **arc** by the distance **distance**.

The parallelled Arc is returned as the Arc **parallelled**. The z-values are not modified, i.e. they are the same as for arc.

A function return value of zero indicates the parallel was successful.

ID = 285

Parallel(Segment segment,Real dist,Segment &parallelled)

Name

Integer Parallel(Segment segment,Real dist,Segment &parallelled)

Description

Plan parallel the Segment **segment** by the distance **dist**.

The parallelled Segment is returned as the Segment **parallelled**. The z-values are not modified, i.e. they are the same as for **segment**.

If the Segment is of type Point, a Segment is not returned and the function return value is set to non-zero.

A function return value of zero indicates the parallel was successful.

ID = 286

Fit Arcs (fillets)

Fitarc(Point pt_1,Point pt_2,Point pt_3,Arc &fillet)

Name

Integer Fitarc(Point pt_1,Point pt_2,Point pt_3,Arc &fillet)

Description

Fit a plan arc through the (x,y) co-ordinates of the three Points **pt_1, pt_2** and **pt_3**.

The arc is returned as Arc **fillet** and the z-values of its start and end points are zero.

A function return value of zero indicates success.

A non-zero return value indicates no arc exists.

ID = 289

Fitarc(Segment seg_1,Segment seg_2,Real rad,Point cpt,Arc &fillet)

Name

Integer Fitarc(Segment seg_1,Segment seg_2,Real rad,Point cpt,Arc &fillet)

Description

Create an plan arc from Segment seg_1 to Segment **seg_2** with radius rad.

The arc start point is on the extended Segment **seg_1** with start direction the same as the direction of **seg_1**.

The arc end point is on the extended Segment **seg_2** with end direction the same as the direction of **seg_1**.

If more than one arc satisfies the above conditions, then the arc with centre closest to the Point **cpt** will be selected.

The arc is returned as Arc **fillet** and the z-values of its start and end points are zero.

A function return value of zero indicates an arc exists.

A non-zero return value indicates no arc exists.

ID = 287

Fitarc(Segment seg_1,Segment seg_2,Point start_tp,Arc &fillet)

Name

Integer Fitarc(Segment seg_1,Segment seg_2,Point start_tp,Arc &fillet)

Description

Create a plan arc from Segment **seg_1** to Segment **seg_2**.

The arc start point is the perpendicular projection of the Point **start_tp** onto the extended Segment **seg_1**. The start direction of the arc is the same as the direction of **seg_1**.

The arc end point is be on the extended Segment **seg_2** with end direction the same as the direction of **seg_1**.

There is at most one arc that satisfies the above conditions.

The arc is returned as Arc **fillet** and the z-values of its start and end points are zero.

A function return value of zero indicates success.

A non-zero return value indicates no arc exists.

ID = 288

Tangents

Tangent(Segment seg_1,Segment seg_2,Line &line)

Name

Integer Tangent(Segment seg_1,Segment seg_2,Line &line)

Description

Create the plan tangent line from the extended Segment **seg_1** to the extended Segment **set_2**.

The direction of the Segments **seg_1** and **seg_2** is used to select a unique tangent line.

The tangent **line** is returned as the Line line with z-values of zero.

A function return value of zero indicates there were no errors in the calculations.

ID = 290

Intersections

Intersect(Segment seg_1,Segment seg_2,Integer &no_intersects,Point &p1,Point &p2)

Name

Integer Intersect(Segment seg_1,Segment seg_2,Integer &no_intersects,Point &p1,Point &p2)

Description

Find the **internal** intersection between the Segments **seg_1** and **seg_2**. That is, only find the intersections of the two Segments that occur between the start and end points of the Segments.

The number of intersections is given by **no_intersects** and the possible intersections are given in Points **p1** and **p2**. The z-values of **p1** and **p2** are set to zero.

There may be zero, one or two intersection points.

A function return value of zero indicates there were no errors in the calculations.

ID = 291

Intersect_extended(Segment seg_1,Segment seg_2,Integer &no_intersects,Point &p1,Point &p2)

Name

Integer Intersect_extended(Segment seg_1,Segment seg_2,Integer &no_intersects,Point &p1,Point &p2)

Description

Find the intersection between the extended Segments **seg_1** and **seg_2**.

The number of intersections is given by **no_intersects** and the possible intersections are given in Points **p1** and **p2**. The z-values of **p1** and **p2** are set to zero.

There may be zero, one or two intersection points.

A function return value of zero indicates there were no errors in the calculations.

ID = 303

Offset Intersections

Intersect_extended(Segment seg_1,Segment seg_2,Integer &no_intersects,Point &p1,Point &p2)

Name

Integer Offset_intersect(Segment seg_1,Real off_1,Segment seg_2,Real off_2,Integer &no_intersects,Point &p1,Point &p2)

Description

Find the **internal** intersection between the Segments **seg_1** and **seg_2** that have been perpendicularly offset by the amounts **off_1** and **off_2** respectively.

The number of intersections is given by **no_intersects** and the possible intersections are given in Points **p1** and **p2**.

The z-values of **p1** and **p2** are set to zero.

There may be zero, one or two intersection points.

A function return value of zero indicates there were no errors in the calculations.

ID = 292

Offset_intersect_extended(Segment seg_1,Real off_1,Segment seg_2,Real off_2,Integer &no_intersects,Point &p1,Point &p2)

Name

Integer Offset_intersect_extended(Segment seg_1,Real off_1,Segment seg_2,Real off_2,Integer &no_intersects,Point &p1,Point &p2)

Description

Find the intersection between the extended Segments **seg_1** and **seg_2** that have been perpendicularly offset by the amounts **off_1** and **off_2** respectively.

The number of intersections is given by **no_intersects** and the possible intersections are given in Points **p1** and **p2**. The z-values of **p1** and **p2** are set to zero.

There may be zero, one or two intersection points.

A function return value of zero indicates there were no errors in the calculations.

ID = 304

Angle Intersect

Angle_intersect(Point pt_1,Real ang_1,Point pt_2, Real ang_2,Point &p)

Name

Integer Angle_intersect(Point pt_1,Real ang_1,Point pt_2,Real ang_2,Point &p)

Description

Find the point of intersection of the line going through the Point **pt_1** with angle **ang_1** and the line going through the Point **pt_2** with angle **ang_2**.

The intersection point is returned as Point **p**. The z-values of **p1** and **p2** are set to zero.

A function return value of zero indicates that the two lines intersect.

A function return value of zero indicates there were no errors in the calculations.

ID = 293

Distance

Get_distance(Point p1,Point p2)

Name

Real Get_distance(Point p1,Point p2)

Description

Calculate the **plan distance** between the Points **p1** and **p2**.

The function return value is the plan distance.

ID = 297

Get_distance_3d(Point p1,Point p2)

Name

Real Get_distance_3d(Point p1,Point p2)

Description

Calculate the **3d distance** between the Points **p1** and **p2**.

The function return value is the 3d distance.

ID = 363

Locate Point

Locate_point(Point from,Real ang,Real dist,Point &to)

Name

Integer Locate_point(Point from,Real ang,Real dist,Point &to)

Description

Create the Point **to** which is a plan distance **dist** along the line of angle **ang** which goes through the Point **from**. The z-value of to is the same as the z-value of **from**.

A function return value of zero indicates there were no errors in the calculations.

ID = 298

Drop Point

Drop_point(Segment segment,Point pt_to_drop,Point &dropped_pt)

Name

Integer Drop_point(Segment segment,Point pt_to_drop,Point &dropped_pt)

Description

Drop a Point **pt_to_drop** perpendicularly in plan onto the Segment segment.

The position of the dropped point on the Segment in returned in the Point **dropped_pt**.

If the point cannot be dropped perpendicularly onto the Segment, then the point is dropped onto the closest end point of the Segment. A z-value for **dropped_pt** is created by interpolation.

A function return value of zero indicates the point was dropped successfully.

ID = 299

Drop_point(Segment segment,Point pt_to_drop,Point &dropped_pt,Real &dist)

Name

Integer Drop_point(Segment segment,Point pt_to_drop,Point &dropped_pt,Real &dist)

Description

Drop a Point **pt_to_drop** onto the Segment **segment**.

The position of the dropped point on the Segment in returned in the Point **dropped_pt**.

The plan distance from **pt_to_drop** to **dropped_pt** is returned as **dist**.

If the point cannot be dropped perpendicularly onto the Segment, then the point is dropped onto the closest end point of the Segment. A z-value for **dropped_pt** is created by interpolation.

A function return value of zero indicates the point was dropped successfully.

ID = 306

Projection

Projection(Segment segment,Real dist,Point &projected_pt)

Name

Integer Projection(Segment segment,Real dist,Point &projected_pt)

Description

Create the Point projected_pt that is a plan distance of dist along from the start of the extended Segment segment.

The z-value for projected_pt is calculated by linear interpolation. Note that for an Arc, the z-value is interpolated for one full circuit of the arc beginning at the start point and the one circuit is used for z-values for distances greater than the length of one circuit.

A function return value of zero indicates the projection was successful.

ID = 300

Projection(Segment segment,Point start_point, Real dist,Point &projected_pt)

Name

Integer Projection(Segment segment,Point start_point,Real dist,Point &projected_pt)

Description

Create the Point **projected_pt** that is a plan distance of **dist** along the extended Segment **segment** where distance is measured from the Point **start_point**.

If **start_point** does not lie on the extended Segment, then **start_point** is automatically dropped onto the extended Segment to create the start point for distance measurement.

The z-value for **projected_pt** is calculated by linear interpolation. Note that for an Arc, the z-value is interpolated for one full circuit of the arc beginning at the start point and the one circuit is used for z-values for distances greater than the length of one circuit.

A function return value of zero indicates the projection was successful.

ID = 301

Change Of Angles

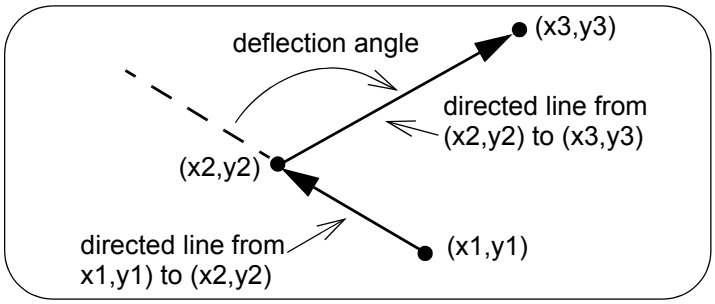
Change_of_angle(Real x1,Real y1,Real x2,Real y2,Real x3,Real y3,Real &angle)

Name

Integer Change_of_angle(Real x1,Real y1,Real x2,Real y2,Real x3,Real y3,Real &angle)

Description

Calculate the deflection angle between the directed line going from (x1,y1) to (x2,y2) and the directed line going from (x2,x2) and (x3,y3) where the deflection angle is measured in radians and in a CLOCKWISE direction. The deflection angle is returned in **angle**.



The use of clockwise fits in with the definition of travelling along a road where going to the right is considered positive and going to the left is considered negative.

WARNING: This is **not** the normal mathematical angle between the two vectors which is measured in the counter clockwise direction and would be the negative of this value.

A function return value of zero indicates the angle was returned successfully.

ID = 656

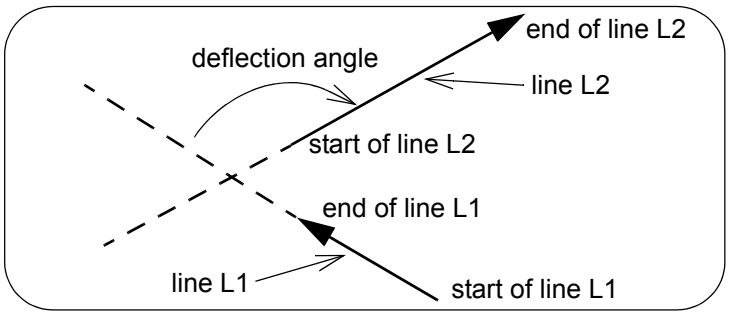
Change_of_angle(Line L1,Line L2,Real &angle)

Name

Integer Change_of_angle(Line L1,Line L2,Real &angle)

Description

Calculate the deflection angle between the line **L1** and the line **L2** where the deflection angle is measured in radians and in a CLOCKWISE direction. The deflection angle is returned in **angle**.



The use of clockwise fits in with the definition of travelling along a road where going to the right is considered positive and going to the left is considered negative.

WARNING: This is **not** the normal mathematical angle between the two vectors which is measured in the counter clockwise direction and would be the negative of this value.

A function return value of zero indicates the angle was returned successfully.
ID = 657

Colours

Colours are stored in 12d Model as a number between 0 and 15, or if defined by the user, between 0 and anything up to 255.

Colour numbers from 0 to 15 always exist.

The actual (red,green,blue) intensities and colour names used for each colour number can be user defined.

Hence it is necessary that 12dPL provides functions to check if colours of given names or numbers exist and to convert between colour numbers and colour names.

Colour_exists(Text col_name)

Name

Integer Colour_exists(Text col_name)

Description

Checks if a colour of name col_name exists in 12dPL.

The colour name to check for is given by Text **col_name**.

A non-zero function return value indicates the colour exist.

A zero function return value indicates the colour does not exist.

Warning - this is the opposite to most 12dPL function return values

ID = 66

Colour_exists(Integer col_number)

Name

Integer Colour_exists(Integer col_number)

Description

Checks if a number is a valid colour number.

The number to check for is given by Integer **col_number**.

A non-zero function return value indicates the number is a valid colour number.

A zero function return value indicates the number is not a valid colour number.

Warning - this is the opposite of most 12dPL function return values

ID = 65

Convert_colour(Text col_name,Integer &col_number)

Name

Integer Convert_colour(Text col_name,Integer &col_number)

Description

Tries to convert the Text **col_name** to a colour number.

If successful, the colour number is returned in Integer **col_number**.

A function return value of zero indicates the conversion was successful.

ID = 67

Convert_colour(Integer col_number,Text &col_name)

Name

Integer Convert_colour(Integer col_number,Text &col_name)

Description

Tries to convert the Integer **col_number** to a colour name.

If successful, the colour name is returned in Text **col_name**.

A function return value of zero indicates the conversion was successful.

ID = 68

Convert_colour(Integer value,Integer &red,Integer &green,Integer &blue)

Name

Integer Convert_colour(Integer value,Integer &red,Integer &green,Integer &blue)

Description

Convert the colour number ***value*** to its red, green and blue components (0-255) and return them in ***red***, ***green*** and ***blue*** respectively.

A function return value of zero indicates the colour was successfully converted.

ID = 2138

Get_project_colours(Dynamic_Text &colours)

Name

Integer Get_project_colours(Dynamic_Text &colours)

Description

Get a Dynamic_Text of all the colour names defined for the project.

The colour names are returned in the Dynamic_Text **colours**.

A function return value of zero indicates the colours were returned successfully.

ID = 235

User Defined Attributes

Extra data can be attached to the Project, Models and Elements as **user defined attributes**.

The *user defined attributes* are contained in a variable of type **Attributes**.

Any number of bits of data of type **Real**, **Integer**, **Text, Binary (blobs), 64-bit Integer** and **Attributes** can be attached to Attributes and when a bit of data is attached, it is given a unique name which is used to retrieved the data at a later date.

The ***attribute type*** used for each data type is:

Data Type	Attribute Type
Integer	1
Real	2
Text	3
Binary (blob)	4
Attributes	5
Uid	6
64-bit integer	7

Note that an **Attributes att** can contain zero or more user defined attributes, and zero or more **Attributes**, so the **Attributes** definition allows **Attributes** inside **Attributes**, inside **Attributes** and so on. So the data inside an **Attributes** forms a tree structure just like a Windows folder system (that is, Windows folders can not only contain files and links, but also Windows folders).

For an **Attributes att**, all the data attached to it (called attributes) is said to be of the first level and all the attributes must have a unique name (attribute names are case sensitive). So the **Attributes att** may have zero or more attributes attached to it, each with a unique case sensitive name, and each with an attribute type.

Attributes are added to **att** in a sequential order so each attribute of **att** will have a unique *attribute number.*

If **bb** is an attribute of **att** and **bb** is of type **Attributes**, then **bb** is also an **Attributes** and can contain its own attributes of various attribute types. The first level of **bb** is considered to be the second level of **att**.

Attribute_exists(Attributes attr,Text att_name)

Name

Integer Attribute_exists(Attributes attr,Text att_name)

Description

Checks to see if an attribute with the name **att_name** exists in the Attributes **attr**.

att_name can have a full path name of the attribute. Attribute names are case sensitive.

A non-zero function return value indicates that the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 1939

Attribute_exists(Attributes attr,Text name,Integer &no)

Name

Integer Attribute_exists(Attributes attr,Text name,Integer &no)

Description

Checks to see if an attribute with the name **att_name** exists in the Attributes **attr**.

att_name can have a full path name of the attribute. Attribute names are case sensitive.

If the attribute exists, its position is returned in Integer **no**.

This position can be used in other Attribute functions.

A non-zero function return value indicates the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 1940

Attribute_delete(Attributes attr,Text att_name)

Name

Integer Attribute_delete(Attributes attr,Text att_name)

Description

Deletes the attribute with the name **att_name** from the Attributes **attr**.

A function return value of zero indicates the attribute was deleted.

ID = 1941

Attribute_delete(Attributes attr,Integer att_no)

Name

Integer Attribute_delete(Attributes attr,Integer att_no)

Description

Delete the attribute with the attribute number **att_no** from the Attributes **attr**.

A function return value of zero indicates the attribute was deleted.

ID = 1942

Attribute_delete_all(Attributes attr)

Name

Integer Attribute_delete_all(Attributes attr)

Description

Delete all attributes from the Attributes **attr**.

A function return value of zero indicates all the attribute were deleted.

ID = 1943

Get_number_of_attributes(Attributes attr,Integer &no_atts)

Name

Integer Get_number_of_attributes(Attributes attr,Integer &no_atts)

Description

Get the number of top level attributes in the Attributes **attr**. The number is returned in **no_atts.**

A function return value of zero indicates the number is successfully returned.

ID = 1944

Get_attribute(Attributes attr,Text att_name,Text &att)

Name

Integer Get_attribute(Attributes attr,Text att_name,Text &att)

Description

From the Attributes **attr**, get the attribute called **att_name** and return the attribute value in **att**. The attribute must be of type Text.

If the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1945

Get_attribute(Attributes attr,Text att_name,Integer &att)

Name

Integer Get_attribute(Attributes attr,Text att_name,Integer &att)

Description

From the Attributes **attr**, get the attribute called **att_name** and return the attribute value in **att**. The attribute must be of type Integer.

If the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1946

Get_attribute(Attributes attr,Text att_name,Real &att)

Name

Integer Get_attribute(Attributes attr,Text att_name,Real &att)

Description

From the Attributes **attr**, get the attribute called **att_name** and return the attribute value in **att**. The attribute must be of type Real.

If the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1947

Get_attribute(Attributes attr,Text att_name,Uid &att)

Name

Integer Get_attribute(Attributes attr,Text att_name,Uid &att)

Description

From the Attributes **attr**, get the attribute called **att_name** and return the attribute value in **att**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1948

Get_attribute(Attributes attr,Text att_name,Attributes &att)

Name

Integer Get_attribute(Attributes attr,Text att_name,Attributes &att)

Description

From the Attributes **attr**, get the attribute called **att_name** and return the attribute value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attributes value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1949

Get_attribute(Attributes attr,Integer att_no,Text &att)

Name

Integer Get_attribute(Attributes attr,Integer att_no,Text &att)

Description

From the Attributes **attr**, get the attribute with number **att_no** and return the attribute value in **att**. The attribute must be of type Text.

If the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1950

Get_attribute(Attributes attr,Integer att_no,Integer &att)

Name

Integer Get_attribute(Attributes attr,Integer att_no,Integer &att)

Description

From the Attributes **attr**, get the attribute with number **att_no** and return the attribute value in **att**. The attribute must be of type Integer.

If the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1951

Get_attribute(Attributes attr,Integer att_no,Real &att)

Name

Integer Get_attribute(Attributes attr,Integer att_no,Real &att)

Description

From the Attributes **attr**, get the attribute with number **att_no** and return the attribute value in **att**. The attribute must be of type Real.

If the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1952

Get_attribute(Attributes attr,Integer att_no,Uid &att)

Name

Integer Get_attribute(Attributes attr,Integer att_no,Uid &att)

Description

From the Attributes **attr**, get the attribute with number **att_no** and return the attribute value in **att**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1953

Get_attribute(Attributes attr,Integer att_no,Attributes &att)

Name

Integer Get_attribute(Attributes attr,Integer att_no,Attributes &att)

Description

From the Attributes **attr**, get the Attribute with number **att_no** and return the Attributes value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number att_no.

ID = 1954

Get_attribute_name(Attributes attr,Integer att_no,Text &name)

Name

Integer Get_attribute_name(Attributes attr,Integer att_no,Text &name)

Description

From the Attributes **attr**, get the attribute with number **att_no** and return the Text value in **name**. The attribute must be of type Text.

If the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1955

Get_attribute_type(Attributes attr,Text att_name,Integer &att_type)

Name

Integer Get_attribute_type(Attributes attr,Text att_name,Integer &att_type)

Description

Get the type of the attribute with the name **att_name** from the Attribute **attr**. The type is returned in **att_type**.

For the list of attribute types, go to Data Type Attribute Type .

A function return value of zero indicates the attribute type was successfully returned.

ID = 1956

Get_attribute_type(Attributes attr,Integer att_num,Integer &att_type)

Name

Integer Get_attribute_type(Attributes attr,Integer att_num,Integer &att_type)

Description

Get the type of the attribute with the number **att_num** from the Attribute **attr**. The type is returned in **att_type**.

For the list of attribute types, go to Data Type Attribute Type .

A function return value of zero indicates the attribute type is successfully returned.

ID = 1957

Get_attribute_length(Attributes attr,Text att_name,Integer &att_len)

Name

Integer Get_attribute_length(Attributes attr,Text att_name,Integer &att_len)

Description

For the Attributes **attr**, get the length in bytes of the attribute of name **att_name**. The number of bytes is returned in **att_len**.

This is mainly for use with attributes of types Text and Binary (blobs)

A function return value of zero indicates the attribute length is successfully returned.

ID = 1958

Get_attribute_length(Attributes attr,Integer att_no,Integer &att_len)

Name

Integer Get_attribute_length(Attributes attr,Integer att_no,Integer &att_len)

Description

For the Attributes **attr**, get the length in bytes of the attribute with number **att_no**. The number of bytes is returned in **att_len**.

This is mainly for use with attributes of types Text and Binary (blobs)

A function return value of zero indicates the attribute length is successfully returned.

ID = 1959

Set_attribute(Attributes attr,Text att_name,Text att)

Name

Integer Set_attribute(Attributes attr,Text att_name,Text att)

Description

For the Attributes **attr**,
if the attribute called **att_name** does not exist then create it as type Text and give it the value **att**.
if the attribute called **att_name** does exist and it is type Text, then set its value to **att**.

If the attribute exists and is not of type Text, then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1960

Set_attribute(Attributes attr,Text att_name,Integer att)

Name

Integer Set_attribute(Attributes attr,Text att_name,Integer att)

Description

For the Attributes **attr**,
if the attribute called **att_name** does not exist then create it as type Integer and give it the value

att.
if the attribute called **att_name** does exist and it is type Integer, then set its value to **att**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1961

Set_attribute(Attributes attr,Text att_name,Real att)

Name

Integer Set_attribute(Attributes attr,Text att_name,Real att)

Description

For the Attributes **attr**,
if the attribute called **att_name** does not exist then create it as type Real and give it the value **att**.
if the attribute called **att_name** does exist and it is type Real, then set its value to **att**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1962

Set_attribute(Attributes attr,Text att_name,Uid att)

Name

Integer Set_attribute(Attributes attr,Text att_name,Uid att)

Description

For the Attributes **attr**,
if the attribute called **att_name** does not exist then create it as type Uid and give it the value **att**.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **att**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1963

Set_attribute(Attributes attr,Text att_name,Attributes att)

Name

Integer Set_attribute(Attributes attr,Text att_name,Attributes att)

Description

For the Attributes **attr**,
if the attribute called **att_name** does not exist then create it as type Attributes and give it the value **att**.
if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1964

Set_attribute(Attributes attr,Integer att_no,Text att)

Name

Integer Set_attribute(Attributes attr,Integer att_no,Text att)

Description

For the Attributes **attr**, if the attribute number **att_no** exists and it is of type Text, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1965

Set_attribute(Attributes attr,Integer att_no,Integer att)

Name

Integer Set_attribute(Attributes attr,Integer att_no,Integer att)

Description

For the Attributes **attr**, if the attribute number **att_no** exists and it is of type Integer, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1966

Set_attribute(Attributes attr,Integer att_no,Real att)

Name

Integer Set_attribute(Attributes attr,Integer att_no,Real att)

Description

For the Attributes **attr**, if the attribute number **att_no** exists and it is of type Real, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1967

Set_attribute(Attributes attr,Integer att_no,Uid att)

Name

Integer Set_attribute(Attributes attr,Integer att_no,Uid att)

Description

For the Attributes **attr**, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1968

Set_attribute(Attributes attr,Integer att_no,Attributes att)

Name

Integer Set_attribute(Attributes attr,Integer att_no,Attributes att)

Description

For the Attributes **attr**, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no Attributes with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1969

Attribute_debug(Attributes attr)

Name

Integer Attribute_debug(Attributes attr)

Description

For internal *12d Solutions* use only.

Write out even more information about the Attributes **attr** to the Output Window.

A function return value of zero indicates the function was successful.

ID = 1971

Folders

Directory_exists(Text folder_name)

Name

Integer Directory_exists(Text folder_name)

Description

Check if a folder of name *folder_name* exists.

If *folder_name* is a relative path, the folder is created in the current working folder of the project.

If *folder_name* is an absolute (starts with say C:, \\, //), then the folder is created in the absolute path.

A non-zero function return value indicates that the folder was created.

A zero function return value indicates that there is an error and the folder was not created.

Warning - this is the opposite of most 12dPL function return values

ID = 2468

Get_file_size(Text file_name,Integer &size)

Name

Integer Get_file_size(Text file_name,Integer &size)

Description

Get the size in bytes of the file named *file_name* and returns the number of bytes in Integer size. Note that the file needs to be a file of size less than 2 Gigabytes.

A function return value of zero indicates the function was successful.

ID = 2407

Directory_create(Text folder_name)

Name

Integer Directory_create(Text folder_name)

Description

Create the folder *folder_name* in the current working folder (the folder name can not contain any paths)

Note - *Directory_create_recursive* will create a folder tree.

A function return value of zero indicates the function was successful.

ID = 2470

Directory_create_recursive(Text folder_name)

Name

Integer Directory_create_recursive(Text folder_name)

Description

Create the folder *folder_name*. The folder name can contain paths and if any of the folders along the path do not exist, then they will also be created.

If folder_name does not contain any path then the folder is created in the current working folder.

A function return value of zero indicates the function was successful.

ID = 2471

Directory_delete(Text folder_name)

Name

Integer Directory_delete(Text folder_name)

Description

If the folder named *folder_name* is empty, delete the folder *folder_name*.

Note - *Directory_delete_recursive* will delete a non-empty folder and all of its sub-folders.

A function return value of zero indicates the function was successful.

ID = 2469

Directory_delete_recursive(Text folder_name)

Name

Integer Directory_delete_recursive(Text folder_name)

Description

Delete the folder named folder_name, and all the sub-folders of *folder_name*.

A function return value of zero indicates the function was successful.

WARNING Using a folder name of d: will delete the entire d drive.

You have been warned.

ID = 2472

12d Model Program and Folders

Get_program_version_number()

Name

Integer Get_program_version_number()

Description

The function return value is the *12d Model* version number.

For example, 10 for *12d Model* ***10****C1c*

ID = 2291

Get_program_major_version_number()

Name

Integer Get_program_major_version_number()

Description

The function return value is the *12d Model* major version number. That is 1 for C1, 2 for C2 etc, 0 for Alpha or Beta.

For example, 1 for *12d Model 10****C1****c*

ID = 2292

Get_program_minor_version_number()

Name

Integer Get_program_minor_version_number()

Description

The function return value is the *12d Model* minor version number. That is 1 for a, 2 for b, 3 of c etc.

For example, 3 for *12d Model 10C1****c***

ID = 2293

Get_program_folder_version_number()

Name

Integer Get_program_folder_version_number()

Description

The function return value is the *12d Model* folder version number.

For example, 00 in "Program Files\12dModel\10.**00**

ID = 2294

Get_program_build_number()

Name

Integer Get_program_build_number()

Description

The function return value is the *12d Model* build number.

This is for internal use only and for minidumps.

ID = 2295

Get_program_special_build_name()

Name

Text Get_program_special_build_name()

<no description>

ID = 2296

Get_program_patch_version_name()

Name

Text Get_program_patch_version_name()

Description

The function return value is a special patch version description for pre-release versions and it is written after the 12d Model version information. It is blank for release versions.

For example "Alpha 274 SLF,SLX,Image Dump - Not For Production"

ID = 2297

Get_program_full_title_name()

Name

Text Get_program_full_title_name()

Description

The function return value is the full name that is written out after 12d Model on the top of the 12d Model Window.

For example "10.0 Alpha 274 SLF,SLX,Image Dump - Not For Production"

ID = 2298

Get_program()

Name

Text Get_program()

Description

The function return value is the full path to where the 12d.exe is on disk. It includes the "12d.exe".

For example "C:\Program Files\12d\12dmodel\10.00\nt.x86\12d.exe"

ID = 2299

Get_program_name()

Name

Text Get_program_name()

Description

The function return value is the name of the 12d Model executable without the ".exe".

That is, "12d".

ID = 2300

Get_program_folder()

Name

Text Get_program_folder()

Description

The function return value is the full path to the folder where the 12d Model executable (12d.exe) is on disk.

For example "C:\Program Files\12d\12dmodel\10.00\nt.x86"

ID = 2301

Get_program_parent_folder()

Name

Text Get_program_parent_folder()

Description

The function return value is the full path to the folder ***above*** where the 12d Model executable (12d.exe) is on disk.

For example "C:\Program Files\12d\12dmodel\10.00"

ID = 2302

Get_project_folder(Text &name)

Name

Integer Get_project_folder(Text &name)

Description

Get the path to the working folder (the folder containing the current project) and return it in *name*.

A function return value of zero indicates the function was successful.

ID = 1891

Get_temporary_directory(Text &folder_name)

Name

Integer Get_temporary_directory(Text &folder_name)

Description

Get the name of the Windows temporary folder %TEMP% and return it as *folder_name*.

A function return value of zero indicates the function was successful.

ID = 2473

Get_temporary_12d_directory(Text &folder_name)

Name

Integer Get_temporary_12d_directory(Text &folder_name)

Description

Get the name of the *12d Model* temporary folder "%TEMP%\12d", and return it as *folder_name*.

A function return value of zero indicates the function was successful.

ID = 2474

Get_temporary_project_directory(Text &folder_name)

Name

Integer Get_temporary_project_directory(Text &folder_name)

Description

Get the name of the current *12d Model* Project temporary folder "%TEMP%\12d\process-id" (where *process-id* is the process id of the current running 12d.exe), and return it as *folder_name*

A function return value of zero indicates the function was successful.

Note - Every 12d project has a independent temporary folder.

ID = 2475

Project

All the 12d Model information is saved in a ***Project***.

Projects are made up of data in the form of elements in models, and tins, and views to look at selected data sets from the project.

Projects also have information such as functions, linestyles, textstyles, fonts and colours.

Get_project_name(Text &name)

Name

Integer Get_project_name(Text &name)

Description

Get the names of the current project.

The names is returned in the Text **name**.

A function return value of zero indicates the function names were successfully returned.

ID = 813

Project_save()

Name

Integer Project_save()

Description

Save the Project to the disk.

A function return value of zero indicates the Project was successfully saved.

ID = 1570

Program_exit(Integer ignore_save)

Name

Integer Program_exit(Integer ignore_save)

Description

Exit the 12d Model program.

If *ignore_save* is non-zero then the project is closed without saving and 12d Model then stops.

If *ignore_save* is zero then a save of the project is done and 12d Model then stops.

ID = 1571

Get_project_functions(Dynamic_Text &function_names)

Name

Integer Get_project_functions(Dynamic_Text &function_names)

Description

Get the names of all the functions in the project.

The dynamic array of function names is returned in the Dynamic_Text **function_names**.

A function return value of zero indicates the function names were successfully returned.

ID = 236

Sleep(Integer milli)

Name

Integer Sleep(Integer milli)

Description

Send *12d Model* to sleep for *milli* milliseconds

A function return value of zero indicates the function was successful.

ID = 2476

Set_project_attributes(Attributes att)

Name

Integer Set_project_attributes(Attributes att)

Description

For the Project, set the Attributes to **att**.

A function return value of zero indicates the Attributes was successfully set.

ID = 1982

Get_project_attributes(Attributes &att)

Name

Integer Get_project_attributes(Attributes &att)

Description

For the Project, return the Attributes for the Project as **att**.

If the Project has no attribute then a non-zero return value is returned.

A function return value of zero indicates the attribute is successfully returned.

ID = 1983

Get_project_attribute(Text att_name,Uid &att)

Name

Integer Get_project_attribute(Text att_name,Uid &att)

Description

For the Project, get the attribute called **att_name** and return the attribute value in **uid**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1984

Get_project_attribute(Text att_name,Attributes &att)

Name

Integer Get_project_attribute(Text att_name,Attributes &att)

Description

For the Project, get the attribute called **att_name** and return the attribute value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1985

Get_project_attribute(Integer att_no,Uid &uid)

Name

Integer Get_project_attribute(Integer att_no,Uid &att)

Description

For the Project, get the attribute with number **att_no** and return the attribute value in **uid**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1986

Get_project_attribute(Integer att_no,Attributes &att)

Name

Integer Get_project_attribute(Integer att_no,Attributes &att)

Description

For the Project, get the attribute with number att_no and return the attribute value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1987

Set_project_attribute(Text att_name,Uid uid)

Name

Integer Set_project_attribute(Text att_name,Uid uid)

Description

For the Project,
if the attribute called **att_name** does not exist then create it as type Uid and give it the value **uid**.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **uid**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1988

Set_project_attribute(Text att_name,Attributes att)

Name

Integer Set_project_attribute(Text att_name,Attributes att)

Description

For the Project,
if the attribute called **att_name** does not exist then create it as type Attributes and give it the value **att**.
if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1989

Set_project_attribute(Integer att_no,Uid uid)

Name

Integer Set_project_attribute(Integer att_no,Uid uid)

Description

For Project, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **uid**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1990

Set_project_attribute(Integer att_no,Attributes att)

Name

Integer Set_project_attribute(Integer att_no,Attributes att)

Description

For Project, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1991

Project_attribute_exists(Text att_name)

Name

Integer Project_attribute_exists(Text att_name)

Description

Checks to see if a Project attribute with the name **att_name** exists in current project.

A non-zero function return value indicates that the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 1378

Project_attribute_exists(Text name,Integer &no)

Name

Integer Project_attribute_exists(Text name,Integer &no)

Description

Checks to see if a project attribute with the name **name** exists in current project.

If the attribute exists, its position is returned in Integer **no**.

This position can be used in other Attribute functions described below.

A non-zero function return value indicates the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 1379

Project_attribute_delete(Text att_name)

Name

Integer Project_attribute_delete(Text att_name)

Description

Delete the project attribute with the name **att_name** in current project.

A function return value of zero indicates the attribute was deleted.

ID = 1380

Project_attribute_delete(Integer att_no)

Name

Integer Project_attribute_delete(Integer att_no)

Description

Delete the project attribute with the Integer **att_no** in current project.

A function return value of zero indicates the attribute was deleted.

ID = 1381

Project_attribute_delete_all(Element elt)

Name

Integer Project_attribute_delete_all(Element elt)

Description

Delete all the attributes for Project.

Element **elt** has nothing to do with this call and is ignored.

A function return value of zero indicates all the attributes were deleted.

ID = 1382

Project_attribute_dump()

Name

Integer Project_attribute_dump()

Description

Write out information about the Project attributes to the Output Window.

A function return value of zero indicates the function was successful.

ID = 1383

Project_attribute_debug()

Integer Project_attribute_debug()

Description

Write out even more information about the Project attributes to the Output Window.

A function return value of zero indicates the function was successful.

ID = 1384

Get_project_number_of_attributes(Integer &no_atts)

Name

Integer Get_project_number_of_attributes(Integer &no_atts)

Description

Get number of attributes Integer **no_atts** in current project.

A function return value of zero indicates the number is successfully returned.

ID = 1385

Get_project_attribute_name(Integer att_no,Text &name)

Name

Integer Get_project_attribute_name(Integer att_no,Text &name)

Description

Get project attribute name Text **name** with attribute number Integer **att_no** in current project.

A function return value of zero indicates the name is successfully returned.

ID = 1392

Get_project_attribute_length(Integer att_no,Integer &att_len)

Name

Integer Get_project_attribute_length(Integer att_no,Integer &att_len)

Description

Get the length of the project attribute at position **att_no**.

The project attribute length is returned in **att_len**.

A function return value of zero indicates the attribute type was successfully returned.

Note

The length is useful for user attributes of type **Text** and **Binary (Blobs)**.

ID = 1396

Get_project_attribute_length(Text att_name,Integer &att_len)

Name

Integer Get_project_attribute_length(Text att_name,Integer &att_len)

Description

Get the length of the project attribute with the name **att_name** for the current project.

The project attribute length is returned in **att_len**.

A function return value of zero indicates the attribute type was successfully returned.

Note

The length is useful for user attributes of type **Text** and **Binary (Blobs)**.

ID = 1395

Get_project_attribute_type(Text att_name,Integer &att_type)

Name

Integer Get_project_attribute_type(Text att_name,Integer &att_type)

Description

Get the type of the project attribute with the name **att_name** from the current project.

The project attribute type is returned in Integer **att_type**.

For the list of attribute types, go to Data Type Attribute Type .

A function return value of zero indicates the attribute type was successfully returned.

ID = 1393

Get_project_attribute_type(Integer att_no,Integer &att_type)

Name

Integer Get_project_attribute_type(Integer att_no,Integer &att_type)

Description

Get the type of the project attribute at position **att_no** for the current project.

The project attribute type is returned in att_type.

For the list of attribute types, go to Data Type Attribute Type .

A function return value of zero indicates the attribute type was successfully returned.

ID = 1394

Get_project_attribute(Text att_name,Real &att)

Name

Integer Get_project_attribute(Text att_name,Real &att)

Description

Get project attribute Real **att** with attribute name Text **att_name** in current project.

A function return value of zero indicates the name is successfully returned.

ID = 1388

Set_project_attribute(Text att_name,Real att)

Name

Integer Set_project_attribute(Text att_name,Real att)

Description

Set the project attribute with name att_name to the Real **att**.

The project attribute **must** be of type **Real**

A function return value of zero indicates the attribute was successfully set.

ID = 1399

Get_project_attribute(Text att_name,Integer &att)

Name

Integer Get_project_attribute(Text att_name,Integer &att)

Description

Get project attribute Integer **att** with attribute name Text **att_name** in current project.

A function return value of zero indicates the name is successfully returned.

ID = 1387

Set_project_attribute(Text att_name,Integer att)

Name

Integer Set_project_attribute(Text att_name,Integer att)

Description

Set the project attribute with name **att_name** to the Integer **att**.

The project attribute **must** be of type **Integer**

A function return value of zero indicates the attribute was successfully set.

ID = 1398

Get_project_attribute(Integer att_no,Text &att)

Name

Integer Get_project_attribute(Integer att_no,Text &att)

Description

Get project attribute Text **att** with attribute number Integer **att_no** in current project.

A function return value of zero indicates the name is successfully returned.

ID = 1389

Set_project_attribute(Integer att_no,Text att)

Name

Integer Set_project_attribute(Integer att_no,Text att)

Description

Set the project attribute at position **att_no** to the Text att.

The project attribute **must** be of type **Text**

A function return value of zero indicates the attribute was successfully set.

ID = 1400

Get_project_attribute(Integer att_no,Integer &att)

Name

Integer Get_project_attribute(Integer att_no,Integer &att)

Description

Get project attribute Integer **att** with attribute number Integer **att_no** in current project.

A function return value of zero indicates the name is successfully returned.

ID = 1390

Set_project_attribute(Integer att_no,Integer att)

Name

Integer Set_project_attribute(Integer att_no,Integer att)

Description

Set the project attribute at position **att_no** to the Integer **att**.

The project attribute **must** be of type **Integer**

A function return value of zero indicates the attribute was successfully set.

ID = 1401

Get_project_attribute(Integer att_no,Real &att)

Name

Integer Get_project_attribute(Integer att_no,Real &att)

Description

Get project attribute Real **att** with attribute number Integer **att_no** in current project.

A function return value of zero indicates the name is successfully returned.

ID = 1391

Set_project_attribute(Integer att_no,Real att)

Name

Integer Set_project_attribute(Integer att_no,Real att)

Description

Set the project attribute at position **att_no** to the Real att.

The project attribute **must** be of type **Real**

A function return value of zero indicates the attribute was successfully set.

ID = 1402

Get_project_attribute(Text att_name,Text &att)

Name

Integer Get_project_attribute(Text att_name,Text &att)

Description

Get project attribute Text **att** with attribute name Text **att_name** in current project.

A function return value of zero indicates the name is successfully returned.

ID = 1386

Set_project_attribute(Text att_name,Text att)

Name

Integer Set_project_attribute(Text att_name,Text att)

Description

Set the project attribute with name **att_name** to the Text att.

The project attribute **must** be of type **Text**

A function return value of zero indicates the attribute was successfully set.

ID = 1397

Project_attribute_delete_all()

Name

Integer Project_attribute_delete_all()

Description

Delete all the project attributes.

A function return value of zero indicates all the attribute were successfully deleted.

ID = 2679

Models

The variable type **Model** is used to refer to *12d Model* models.

Model variables act as ***handles*** to the actual *12d Model* model so that the model can be easily referred to and manipulated within a macro (*see* 12d Model Database Handles).

The items that can be stored in Models are known as **Elements** (strings, tins, plot frames etc - *see* Elements).

The list of Elements in a model can be obtained as a Dynamic_Element (see and this allows you to "walk" through all the Elements in a Model (*see* Dynamic Element Arrays):

```
Element elt;
Dynamic_Element de;                // a list of Elements
Integer  number_of_elts;
Text   elt_type;
Get_elements(model,de,number_of_elts);
for (Integer i;i<=number_of_elements;i++) {
  Get_item(de,i,elt);              // get the next Element from the Model model.
//   the Element elt can now be processed
    ...
```

Important Note:

To add an Element **elt** to a Model **model**, use the call Set_model(Element elt,Model model) .

Create_model(Text model_name)

Name

Model Create_model(Text model_name)

Description

Create a Model with the name **model_name**.

If the model is created, its handle is returned as the function return value.

If no model can be created, a null Model is returned as the function return value.

ID = 59

Get_model_create(Text model_name)

Name

Model Get_model_create(Text model_name)

Description

Get a handle to the model with name **model_name**.

If the model exists, its handle is returned as the function return value.

If no such model exists, then a new model with the name **model_name** is created, and its handle returned as the function return value.

If no model exists and the creation fails, a null Model is returned as the function return value.

ID = 60

Get_number_of_items(Model model,Integer &num)

Name

Integer Get_number_of_items(Model model,Integer &num)

Description

Get the number of items (Elements) in the Model **model**.

The number of Elements is returned as the Integer **num**.

A function return value of zero indicates success.

ID = 452

Get_elements(Model model,Dynamic_Element &de,Integer &total_no)

Name

Integer Get_elements(Model model,Dynamic_Element &de,Integer &total_no)

Description

Get all the Elements from the Model model and add them to the Dynamic_Element array, **de**.

The total number of Elements in **de** is returned by **total_no**.

Note: whilst this Dynamic_Element exists, all of the elements with handles in the Dynamic_Element are locked.

A function return value of zero indicates success.

ID = 132

Model_exists(Text model_name)

Name

Integer Model_exists(Text model_name)

Description

Checks to see if a model with the name **model_name** exists.

A non-zero function return value indicates a model does exist.

A zero function return value indicates that no model of name **model_name** exists.

Warning - this is the opposite of most 12dPL function return values

ID = 63

Model_exists(Model model)

Name

Integer Model_exists(Model model)

Description

Checks if the Model **model** is valid (that is, not null).

A non-zero function return value indicates model is not null.

A zero function return value indicates that **model** is null.

Warning - this is the opposite of most 12dPL function return values

ID = 62

Get_project_models(Dynamic_Text &model_names)

Name

Integer Get_project_models(Dynamic_Text &model_names)

Description

Get the names of all the models in the project.

The dynamic array of model names is returned in the Dynamic_Text **model_names**.

A function return value of zero indicates the model names are returned successfully.

ID = 231

Get_model(Text model_name)

Name

Model Get_model(Text model_name)

Description

Get the Model model with the name **model_name**.

If the model exists, its handle is returned as the function return value.

If no model of name **model_name** exists, a null Model is returned as the function return value.

ID = 58

Get_name(Model model,Text &model_name)

Name

Integer Get_name(Model model,Text &model_name)

Description

Get the name of the Model **model**.

The model name is returned in the Text **model_name**.

A function return value of zero indicates the model name was successfully returned.

If **model** is null, the function return value is non-zero.

ID = 57

Get_time_created(Model model,Integer &time)

Name

Integer Get_time_created(Model model,Integer &time)

Description

Get the time that the Model **model** was created and return the time in **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully returned.

ID = 2111

Get_time_updated(Model model,Integer &time)

Name

Integer Get_time_updated(Model model,Integer &time)

Description

Get the time that the Model **model** was last updated and return the time in **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully returned.

ID = 2112

Set_time_updated(Model model,Integer time)

Name

Integer Set_time_updated(Model model,Integer time)

Description

Set the update time for the Model **model** to **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully set.

ID = 2113

Get_id(Model model,Uid &id)

Name

Integer Get_id(Model model,Uid &id)

Description

Get the Uid of the Model **model** and return it in **id**.

A function return value of zero indicates the Uid was successfully returned.

ID = 1914

Get_id(Model model,Integer &id)

Name

Integer Get_id(Model model,Integer &id)

Description

Get the id of the Model **model** and return it in **id**.

A function return value of zero indicates the id was successfully returned.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Get_id(Model model,Uid &id)* instead.

ID = 1182

Get_model(Uid model_id,Model &model)

Name

Integer Get_model(Uid model_id,Model &model)

Description

Get the model in the Project that has the Uid **model_id** and return it in **model**.

If the model does not exist then a non-zero function return value is returned.

A function return value of zero indicates the model was successfully returned.

ID = 1912

Get_model(Integer model_id,Model &model)

Name

Integer Get_model(Integer model_id,Model &model)

Description

Get the model in the Project that has the id **model_id** and return it in **model**.

If the model does not exist then a non-zero function return value is returned.

A function return value of zero indicates the model was successfully returned.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Get_model(Uid model_id,Model &model)* instead.

ID = 1180

Get_element(Uid model_id,Uid element_id,Element &elt)

Name

Integer Get_element(Uid model_id,Uid element_id,Element &elt)

Description

Get the Element with Uid **element_id** from the model that has the Uid **model_id** and return it in **elt**.

If the Element does not exist in the model with Uid **model_id** then a non-zero function return value is returned.

A function return value of zero indicates the Element was successfully returned.

ID = 1913

Get_element(Integer model_id,Integer element_id,Element &elt)

Name

Integer Get_element(Integer model_id,Integer element_id,Element &elt)

Description

Get the Element with id **element_id** from the model that has the id **model_id** and return it in **elt**.

If the Element does not exist in the model with **model_id** then a non-zero function return value is returned.

A function return value of zero indicates the Element was successfully returned.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Get_element(Uid model_id,Uid element_id,Element &elt)* instead.

ID = 1181

Get_extent_x(Model model,Real &xmin,Real &xmax)

Name

Integer Get_extent_x(Model model,Real &xmin,Real &xmax)

Description

Gets the x-extents of the Model **model**.

The minimum x extent is returned by the Real **xmin**.

The maximum x extent is returned by the Real **xmax**.

A function return value of zero indicates the x-extents were returned successfully.

ID = 163

Get_extent_y(Model model,Real &ymin,Real &ymax)

Name

Integer Get_extent_y(Model model,Real &ymin,Real &ymax)

Description

Gets the y-extents of the Model **model**.

The minimum y extent is returned by the Real **ymin**.

The maximum y extent is returned by the Real **ymax**.

A function return value of zero indicates the y-extents were returned successfully.

ID = 164

Get_extent_z(Model model,Real &zmin,Real &zmax)

Name

Integer Get_extent_z(Model model,Real &zmin,Real &zmax)

Description

Gets the z-extents of the Model **model**.

The minimum z extent is returned by the Real **zmin**.

The maximum z extent is returned by the Real **zmax**.

A function return value of zero indicates the z-extents were returned successfully.

ID = 165

Calc_extent(Model model)

Name

Integer Calc_extent(Model model)

Description

Calculate the extents of the Model **model**. This is necessary when Elements have been deleted from a model.

A function return value of zero indicates the extent calculation was successful.

ID = 166

Model_duplicate(Model model,Text dup_name)

Name

Integer Model_duplicate(Model model,Text dup_name)

Description

Create a new Model with the name dup_name and add duplicates of all the elements in **model** to it.

It is an error if a Model called **dup_name** already exists.

A function return value of zero indicates the duplication was successful.

ID = 428

Model_rename(Text original_name,Text new_name)

Name

Integer Model_rename(Text original_name,Text new_name)

Description

Change the name of the Model **original_name** to the new name **new_name**.

A function return value of zero indicates the rename was successful.

ID = 423

Model_draw(Model model)

Name

Integer Model_draw(Model model)

Description

Draw each element in the Model **model** for each view that the model is on. The elements are drawn in their own colour.

A function return value of zero indicates the draw was successful.

ID = 415

Model_draw(Model model,Integer col_num)

Name

Integer Model_draw(Model model,Integer col_num)

Description

Draw, in the colour number **col_num**, each element in the Model **model** for each view that the model is on.

A function return value of zero indicates the draw was successful.

ID = 416

Null(Model model)

Name

Integer Null(Model model)

Description

Set the Model handle **model** to null. This does not affect the 12d Model model that the handle pointed to.

A function return value of zero indicates model was successfully nulled.

ID = 134

Model_delete(Model model)

Name

Integer Model_delete(Model model)

Description

Delete from the project and the disk, the 12d Model model pointed to by the Model **model**. The handle **model** is then set to null.

A function return value of zero indicates the model was successfully deleted.

ID = 61

Get_model_attributes(Model model,Attributes &att)

Name

Integer Get_model_attributes(Model model,Attributes &att)

Description

For the Model **model**, return the Attributes for the Model as **att**.

If the Model has no Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute is successfully returned.

ID = 2042

Set_model_attributes(Model model,Attributes att)

Name

Integer Set_model_attributes(Model model,Attributes att)

Description

For the Model **model**, set the Attributes for the Model to **att**.

A function return value of zero indicates the attribute is successfully set.

ID = 2043

Get_model_attribute(Model model,Text att_name,Uid &uid)

Name

Integer Get_model_attribute(Model model,Text att_name,Uid &uid)

Description

From the Model **model**, get the attribute called **att_name** and return the attribute value in **uid**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2044

Get_model_attribute(Model model,Text att_name,Attributes &att)

Name

Integer Get_model_attribute(Model model,Text att_name,Attributes &att)

Description

From the Model **model**, get the attribute called **att_name** from **model** and return the attribute value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - this function is more efficient than getting the Attributes from the Model and then getting the data from that Attributes.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2045

Get_model_attribute(Model model,Integer att_no,Uid &uid)

Name

Integer Get_model_attribute(Model model,Integer att_no,Uid &uid)

Description

From the Model **model**, get the attribute with number **att_no** and return the attribute value in **uid**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 2046

Get_model_attribute(Model model,Integer att_no,Attributes &att)

Name

Integer Get_model_attribute(Model model,Integer att_no,Attributes &att)

Description

From the Model **model**, get the attribute with number att_no and return the Attribute value in att. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number att_no.

ID = 2047

Set_model_attribute(Model model,Text att_name,Uid att)

Name

Integer Set_model_attribute(Model model,Text att_name,Uid att)

Description

For the Model **model**,
if the attribute called **att_name** does not exist then create it as type Uid and give it the value

att.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **att**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2048

Set_model_attribute(Model model,Text att_name,Attributes att)

Name

Integer Set_model_attribute(Model model,Text att_name,Attributes att)

Description

For the Model **model**,
if the attribute called **att_name** does not exist then create it as type Attributes and give it the value **att**.
if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2049

Set_model_attribute(Model model,Integer att_no,Uid uid)

Name

Integer Set_model_attribute(Model model,Integer att_no,Uid uid)

Description

For the Model **model**, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **uid**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 2050

Set_model_attribute(Model model,Integer att_no,Attributes att)

Name

Integer Set_model_attribute(Model model,Integer att_no,Attributes att)

Description

For the Model **model**, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value

is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 2051

Model_attribute_exists(Model model,Text att_name)

Name

Integer Model_attribute_exists(Model model,Text att_name)

Description

Checks to see if a model attribute with the name **att_name** exists in the Model **model**.

A non-zero function return value indicates that the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 1403

Model_attribute_exists(Model model,Text name,Integer &no)

Name

Integer Model_attribute_exists(Model model,Text name,Integer &no)

Description

Checks to see if a model attribute with the name **name** exists in the Model **model**.

If the attribute exists, its position is returned in Integer no.

This position can be used in other Attribute functions described below.

A non-zero function return value indicates the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 1404

Model_attribute_delete(Model model,Text att_name)

Name

Integer Model_attribute_delete(Model model,Text att_name)

Description

Delete the model attribute with the name **att_name** for Model **model**.

A function return value of zero indicates the attribute was deleted.

ID = 1405

Model_attribute_delete(Model model,Integer att_no)

Name

Integer Model_attribute_delete(Model model,Integer att_no)

Description

Delete the model attribute at the position **att_no** for Model **model**.

A function return value of zero indicates the attribute was deleted.

ID = 1406

Model_attribute_delete_all(Model model,Element elt)

Name

Integer Model_attribute_delete_all(Model model,Element elt)

Description

Delete all the model attributes for Model **model**.

A function return value of zero indicates all the attributes were deleted.

ID = 1407

Model_attribute_dump(Model model)

Name

Integer Model_attribute_dump(Model model)

Description

Write out information about the Model attributes to the Output Window.

A function return value of zero indicates the function was successful.

ID = 1408

Model_attribute_debug(Model model)

Name

Integer Model_attribute_debug(Model model)

Description

Write out even more information about the Model attributes to the Output Window.

A function return value of zero indicates the function was successful.

ID = 1409

Get_model_attribute(Model model,Text att_name,Text &att)

Name

Integer Get_model_attribute(Model model,Text att_name,Text &att)

Description

Get the data for the model attribute with the name **att_name** for Model **model**.

The model attribute must be of type **Text** and is returned in Text **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 1411

Get_model_attribute(Model model,Text att_name,Integer &att)

Name

Integer Get_model_attribute(Model model,Text att_name,Integer &att)

Description

Get the data for the model attribute with the name **att_name** for Model **model**.

The model attribute must be of type Integer and is returned in **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 1412

Get_model_attribute(Model model,Text att_name,Real &att)

Name

Integer Get_model_attribute(Model model,Text att_name,Real &att)

Description

Get the data for the model attribute with the name **att_name** for Model **model**.

The model attribute must be of type **Real** and is returned in **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 1413

Get_model_attribute(Model model,Integer att_no,Text &att)

Name

Integer Get_model_attribute(Model model,Integer att_no,Text &att)

Description

Get the data for the model attribute at the position **att_no** for Model **model**.

The model attribute must be of type **Text** and is returned in **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 1414

Get_model_attribute(Model model,Integer att_no,Integer &att)

Name

Integer Get_model_attribute(Model model,Integer att_no,Integer &att)

Description

Get the data for the model attribute at the position **att_no** for Model **model**.

The model attribute must be of type **Integer** and is returned in Integer **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 1415

Get_model_attribute(Model model,Integer att_no,Real &att)

Name

Integer Get_model_attribute(Model model,Integer att_no,Real &att)

Description

Get the data for the model attribute at the position **att_no** for Model **model**.

The model attribute must be of type **Real** and is returned in Real **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 1416

Set_model_attribute(Model model,Integer att_no,Real att)

Name

Integer Set_model_attribute(Model model,Integer att_no,Real att)

Description

For the Model **model**, set the model attribute at position **att_no** to the Real **att**.

The model attribute **must** be of type **Real**

A function return value of zero indicates the attribute was successfully set.

ID = 1427

Set_model_attribute(Model model,Integer att_no,Integer att)

Name

Integer Set_model_attribute(Model model,Integer att_no,Integer att)

Description

For the Model **model**, set the model attribute at position **att_no** to the Integer **att**.

The model attribute **must** be of type **Integer**

A function return value of zero indicates the attribute was successfully set.

ID = 1426

Set_model_attribute(Model model,Integer att_no,Text att)

Name

Integer Set_model_attribute(Model model,Integer att_no,Text att)

Description

For the Model **model**, set the model attribute at position **att_no** to the Text **att**.

The model attribute **must** be of type **Text**

A function return value of zero indicates the attribute was successfully set.

ID = 1425

Set_model_attribute(Model model,Text att_name,Real att)

Name

Integer Set_model_attribute(Model model,Text att_name,Real att)

Description

For the Model **model**, set the model attribute with name **att_name** to the Real **att**.

The model attribute **must** be of type **Real**

A function return value of zero indicates the attribute was successfully set.

ID = 1424

Set_model_attribute(Model model,Text att_name,Integer att)

Name

Integer Set_model_attribute(Model model,Text att_name,Integer att)

Description

For the Model **model**, set the model attribute with name **att_name** to the Integer **att**.

The model attribute **must** be of type **Integer**

A function return value of zero indicates the attribute was successfully set.

ID = 1423

Set_model_attribute(Model model,Text att_name,Text att)

Name

Integer Set_model_attribute(Model model,Text att_name,Text att)

Description

For the Model **model**, set the model attribute with name **att_name** to the Text **att**.

The model attribute **must** be of type **Text**

A function return value of zero indicates the attribute was successfully set.

ID = 1422

Get_model_attribute_name(Model model,Integer att_no,Text &name)

Name

Integer Get_model_attribute_name(Model model,Integer att_no,Text &name)

Description

Get the name for the model attribute at the position **att_no** for Model **model**.

The model attribute name found is returned in Text **name**.

A function return value of zero indicates the attribute name was successfully returned.

ID = 1417

Get_model_attribute_type(Model model,Text att_name,Integer &att_type)

Name

Integer Get_model_attribute_type(Model model,Text att_name,Integer &att_type)

Description

Get the type of the model attribute with the name **att_name** from the Model **model**.

The model attribute type is returned in Integer **att_type**.

For the list of attribute types, go to Data Type Attribute Type .

A function return value of zero indicates the attribute type was successfully returned.

ID = 1418

Get_model_attribute_type(Model model,Integer att_name,Integer &att_type)

Name

Integer Get_model_attribute_type(Model model,Integer att_name,Integer &att_type)

Description

Get the type of the model attribute at position **att_no** for the Model **model**.

The model attribute type is returned in **att_type**.

For the list of attribute types, go to Data Type Attribute Type .

A function return value of zero indicates the attribute type was successfully returned.

ID = 1419

Get_model_attribute_length(Model model,Text att_name,Integer &att_len)

Name

Integer Get_model_attribute_length(Model model,Text att_name,Integer &att_len)

Description

Get the length of the model attribute with the name **att_name** for Model **model**.

The model attribute length is returned in **att_len**.

A function return value of zero indicates the attribute type was successfully returned.

Note - the length is useful for user attributes of type **Text** and **Binary (Blobs)**.

ID = 1420

Get_model_attribute_length(Model model,Integer att_no,Integer &att_len)

Name

Integer Get_model_attribute_length(Model model,Integer att_no,Integer &att_len)

Description

Get the length of the model attribute at position **att_no** for Model **model**.

The model attribute length is returned in **att_len**.

A function return value of zero indicates the attribute type was successfully returned.

Note - the length is useful for user attributes of type **Text** and **Binary (Blobs)**.

ID = 1421

Get_model_number_of_attributes(Model model,Integer &no_atts)

Name

Integer Get_model_number_of_attributes(Model model,Integer &no_atts)

Description

Get the total number of model attributes for Model **model**.

The total number of attributes is returned in Integer **no_atts**.

A function return value of zero indicates the attribute was successfully returned.

ID = 1410

Views

The variable type **View** is used to refer to 12d Model views.

View variables act as ***handles*** to the actual view so that the view can be easily referred to and manipulated within a macro (see 12d Model Database Handles).

View_exists(Text view_name)

Name

Integer View_exists(Text view_name)

Description

Checks to see if a view with the name **view_name** exists.

A non-zero function return value indicates a view does exist.

A zero function return value indicates value that no view of that name exists.

Warning - this is the opposite of most 12dPL function return values

ID = 373

View_exists(View view)

Name

Integer View_exists(View view)

Description

Checks if the View **view** is valid (that is, not null).

A non-zero function return value indicates **view** is not null.

A zero function return value indicates that **view** is null.

Warning - this is the opposite of most 12dPL function return values

ID = 374

Get_name(View view,Text &view_name)

Name

Integer Get_name(View view,Text &view_name)

Description

Get the name of the View **view**.

The view name is returned in the Text **view_name**.

If **view** is null, the function return value is non-zero.

A function return value of zero indicates the view name was returned successfully.

ID = 435

Null(View view)

Name

Integer Null(View view)

Description

Set the View handle **view** to null. This does not affect the 12d Model view that the handle pointed

to.

A function return value of zero indicates **view** was successfully nulled.

ID = 375

Get_project_views(Dynamic_Text &view_names)

Name

Integer Get_project_views(Dynamic_Text &view_names)

Description

Get the names of all the views in the project.

The dynamic array of view names is returned in the Dynamic_Text **view_names**.

A function return value of zero indicates the view names were returned successfully.

ID = 234

Get_view(Text view_name)

Name

View Get_view(Text view_name)

Description

Get the View with the name **view_name**.

If the view exists, its handle is returned as the function return value.

If no view of name **view_name**, a null View is returned as the function return value.

ID = 347

Get_type(View view,Text &type)

Name

Integer Get_type(View view,Text &type)

Description

Get the type of the View **view** as the Text **type**.

The type is

Plan	if the view is a plan view
Section	section view
Perspective	perspective view or Opengl perspective view
Hidden_perspective	hidden perspective view.

A function return value of zero indicates that the view type was returned successfully.

ID = 358

Get_type(View view,Integer &view_num)

Name

Integer Get_type(View view,Integer &view_num)

Description

For the view **view**, **view_num** returns the type of the view.

view_num = 2010 if view is a PLAN VIEW
view_num = 2011 if view is a SECTION VIEW

view_num = 2012 if view is a PERSP VIEW and OPEN GL 2012
view_num = 2030 if view is a HIDDEN PERSPECTIVE
A function return value of zero indicates the successfully.

ID = 357

Model_get_views(Model model,Dynamic_Text &view_names)

Name

Integer Model_get_views(Model model,Dynamic_Text &view_names)

Description

Get the names of all the views that the Model **model** is on.

The view names are returned in the Dynamic_Text **view_names**.

A function return value of zero indicates that the view names were returned successfully.

ID = 354

View_get_models(View view,Dynamic_Text &model_names)

Name

Integer View_get_models(View view,Dynamic_Text &model_names)

Description

Get the names of all the Models on the View **view**.

The model names are returned in the Dynamic_Text **model_names**.

A function return value of zero indicates that the model names were returned successfully.

ID = 350

View_add_model(View view,Model model)

Name

Integer View_add_model(View view,Model model)

Description

Add the Model **model** to the View view.

A function return value of zero indicates that **model** was successfully added to the view.

ID = 348

View_remove_model(View view,Model model)

Name

Integer View_remove_model(View view,Model model)

Description

Remove the Model **model** from the View view.

A function return value of zero indicates that **model** was successfully removed from the view.

ID = 349

View_redraw(View view)

Name

Integer View_redraw(View view)

Description

Redraw the 12d Model View **view**.

A function return value of zero indicates that the view was successfully redrawn.

ID = 351

View_fit(View view)

Name

Integer View_fit(View view)

Description

Perform a fit on the 12d Model View **view**.

A function return value of zero indicates that the view was successfully fitted.

ID = 353

Section_view_profile(View view,Element string,Integer fit_view)

Name

Integer Section_view_profile(View view,Element string,Integer fit_view)

Description

Profile the Element **string** on the View **view**.

If **fit_view** = 1 then a fit is also done on the view.

If **view** is **not** a Section view, then a non-zero function return value is returned.

A function return value of zero indicates the profile was successful.

ID = 2110

View_get_size(View view,Integer &width,Integer &height)

Name

Integer View_get_size(View view,Integer &width,Integer &height)

Description

Find the size in screen units (pixels) of the View **view**.

The width and height of the view are **width** and **height** pixels respectively.

A function return value of zero indicates that the view size was successfully returned.

ID = 352

Calc_extent(View view)

Name

Integer Calc_extent(View view)

Description

Calculate the extents of the View **view**. This is necessary when Elements have been deleted from a model on a view.

A function return value of zero indicates the extent calculation was successful.

ID = 477

Elements

The variable type **Element** is used as a *handle* to all the data types that can be stored in a 12d Model ***model***. That is, it is used to refer to 12d Model strings, tins, super tins and plot frames (see 12d Model Database Handles).

This allows you to "walk" through a model getting access to each of the Elements stored in the model without having to know what type it is. Once the Element is retrieved, it can then be processed within the macro.

For example, for a given Model *model*, you access all the Elements in *model* by loading them into a dynamic array of Elements (Dynamic_Element) and then stepping through the dynamic array:

```
 Element elt;
 Dynamic_Element de;                // a list of Elements
 Integer  number_of_elts;
 Text   elt_type;
 Get_elements(model,de,number_of_elts);
 for (Integer i;i<=number_of_elements;i++) {
   Get_item(de,i,elt);              // get the next Element from the Model model.
//   the Element elt can now be processed
   Get_type(elt,elt_type);          // find out if elt is a super string, arc, tin, plot frame etc
   if (elt_type == "Super") {
   . . .
```

See Types of Elements
See Parts of 12d Elements
See Element Header Functions
See Element Attributes Functions

See Tin Element
See Super String Element
See Interface String Element
See Super Alignment String Element
See Arc String Element
See Circle String Element
See Text String Element
See Drainage String Element
See Pipeline String Element
See Face String Element
See Plot Frame Element
See Feature String Element

From 12d Model 9, some strings types are being phased out (superseded) and replaced by the *Super String* or the *Super Alignment.*

See Alignment String Element
See 2d Strings
See 3d Strings
See 4d Strings
See Polyline Strings
See Pipe Strings

Types of Elements

There are different types of elements and the type is found by the call Get_type(Element elt,Text &elt_type) .

The different types of Elements are

Element Type **Descriptions**

Super for a super string - a general string with (x,y,z,radius,text,attributes) at each point, plus the possibility of many other dimensions of information. See Super String Element

In earlier versions of ***12d Model***, there were a large number of string types but from ***12d Model 9*** onwards, the *Super String* was introduced which with its possible dimensions, replaces *2d*, *3d*, *4d*, *polyline* and *pipe* strings.

However, for some applications it was important to know if the super string was like one of the original strings. For example, some options required a string to be a contour string, the original 2d string. That is, the string has the one z-value (or height) for the entire string. To make it easier than checking on the various dimensions, there is a call that returns a **Type Like** value. For example, a Super String that has a constant dimension for height, behaves like a 2d string and in that case will return the **Type Like** of **2d**.

Over time, all the *12d Model* options that create strings that can be replaced by a *Super String* are being modified to only create Super Strings, and with the correct **Type Like** if it is required in some circumstances.

The **Type Like**'s an be referred to by a number or by a text.

Type Like Number	Type Like Text
11	**2d** string - a constant height for the entire string
12	**3d** string - a different height allowed for each vertex.
13	**interface** string
29	**4d** string - variable vertex text
36	**pipe** string - a constant diameter for the entire string
62	**polyline** string - a different radius allowed for each segment
40	**face** string
71	none of the above - just a normal super string

For a Super String, the **Type Like** is found by the calls Get_type_like(Element super,Integer &type) and Get_type_like(Element elt,Text &type) .

Super_Alignment for a Super Alignment string - a string with separate horizontal and vertical geometry

In earlier versions of ***12d Model*** there was only the Alignment string whose geometry could only contain horizontal ips and vertical ip. In later versions of ***12d Model***, the Super Alignment was introduced which allowed not only hips and vips but also fixed and floating methods, computators etc.

Over time, all the options inside ***12d Model*** that create strings with a a separate horizontal and vertical geometry are being modified so that they only create *Super Alignments*.

Arc for an Arc string - a string of an arc in plan and with a linearly varying z value. Note that this is a helix in three dimensional space. See Arc String Element .

Circle for a Circle string - a string of a circle in plan with a constant z value. Note that this is a circle in a plane parallel to the (x,y) plane. See Circle String Element .

Feature a circle with a z-value at the centre but only null values on the circumference. See Feature String Element .

Drainage string for drainage and sewer elements. See Drainage String Element .

Interface string with (x,y,z,cut-fill flag) at each point. See Interface String Element .

Text string with text at a point.See Text String Element .

Tin triangulated irregular network - a triangulation. See Tin Element .

SuperTin a SuperTin of tins.

Plot Frame for a plot frame - an element used for production of plan plots.
See Plot Frame Element .

Pipeline a string with separate horizontal and vertical geometry defined by Intersection points only, and one diameter for the entire string. See Pipeline String Element .

Strings being replaced by *Super Strings*:

2d for a 2d string - a string with (x,y) at each pt but constant z value.
An old string type being replaced by a *Super String* with ***Type Like*** 11.

3d for a 3d string - a string with (x,y,z) at each point
An old string type being replaced by a *Super String* with ***Type Like*** 12.

4d for a 4d string - a string with (x,y,z,text) at each point
An old string type being replaced by a *Super String* with ***Type Like*** 29.

Pipe for a pipe string - a string with (x,y,z) at each point and a diameter
An old string type replaced by a *Super String* with ***Type Like*** 36.

Polyline for a polyline string - a string with (x,y,z,radius) at each point
An old string type replaced by a *Super String* with ***Type Like*** 62.

String being replaced by *Super Alignment*:

Alignment for an Alignment string - a string with separate horizontal and vertical geometry defined by Intersection Points only.
An old string type replaced by the *Super Alignment* string. See Alignment String Element

Note

The Element of type tin is provided because tins (triangulations) can be part of a model. Tins are normally created using the Triangulation functions and there are special Tin functions for modifying tin information.

Parts of 12d Elements

All 12d Elements consists of three parts -

(a) **Header Information** which exists for all Elements. The header information includes the Element type, name, colour, style, number of points, start chainage, model and extents.

The functions for manipulating the header information are in the section Element Header Functions

(b) **Element Attributes** for the entire Element

The functions for manipulating the Element attributes are in the section Element Attributes Functions

Note that for some types of Elements, there are additional attributes as part of the element-type body of the Element. For example super strings have attributes for vertices and segments, and drainage strings have attributes for maintenance holes/pits and pipes.

The functions for manipulating the header information and attributes are documented first, followed by the specific functions for each type of Element (e.g. tins, super strings).

(c) **Element Body** - element-type specific information (the body of the Element) such as the (x,y,z) values for an vertex.

Super strings, interface strings and the old 2d, 3d, 4d and polyline strings consist of data values given at one or more points in the string.

For the above types, the associated Element body is created by giving fixed arrays containing the required information at each point, and extra data for optional super string dimensions.

Text, Plot Frames and strings of type Super Alignment, Alignment, Arc, Circle do not have simple arrays to define them.

Tins consist of vertices for the triangles and all the triangle edges that make up the tin. See Tin Element for functions for working with Tins.

The Element-type specific functions for each type of Element (e.g. tins, super strings) are given in:

Tin Element
Super String Element
Examples of Setting Up Super Strings
Super Alignment String Element
Arc String Element
Circle String Element
Text String Element
Pipeline String Element
Drainage String Element
Feature String Element
Interface String Element
Face String Element
Plot Frame Element
Strings Replaced by Super Strings

Other general and miscellaneous Element functions are collected in the section General Element Operations .

Element Header Functions

When an Element is created, its type is given by the Element creation function.

All new Elements are given the default header information:

Uid	unique Uid for the Element
model	none
colour	magenta
name	none
chainage	0
style	1
weight	0

For all Element types, inquiries and modifications to the Element header information can be made by the following 12dPL functions.

Element_exists(Element elt)

Name

Integer Element_exists(Element elt)

Description

Checks the validity of an Element **elt**. That is, it checks that **elt** has not been set to null.

A non-zero function return value indicates **elt** is not null.

A zero function return value indicates that **elt** is null.

ID = 56

Get_points(Element elt,Integer &num_verts)

Name

Integer Get_points(Element elt,Integer &num_verts)

Description

Get the number of vertices in the Element **elt**.

The number of vertices is returned as the Integer **num_verts**.

For Elements of type Alignment, Arc and Circle, Get_points gives the number of vertices when the Element is approximated using the 12d Model cord-to-arc tolerance.

A function return value of zero indicates the number of vertices was successfully returned.

ID = 43

Get_data(Element elt,Integer i,Real &x,Real &y,Real &z)

Name

Integer Get_data(Element elt,Integer i,Real &x,Real &y,Real &z)

Description

Get the (x,y,z) data for the **i**th vertex of the string Element **elt**.

The x value is returned in Real **x**.

The y value is returned in Real **y**.

The z value is returned in Real **z**.

A function return value of zero indicates the data was successfully returned.

NOTE: The functions to set the data arrays are given in the sections of each string type. For

example Super String Create Functions .

ID = 653

Set_name(Element elt,Text elt_name)

Name

Integer Set_name(Element elt,Text elt_name)

Description

Set the name of the Element **elt** to the Text **elt_name**.

A function return value of zero indicates the Element name was successfully set.

Note

This will not set the name of an Element of type Tin.

ID = 45

Get_name(Element elt,Text &elt_name)

Name

Integer Get_name(Element elt,Text &elt_name)

Description

Get the name of the Element **elt**.

The name is returned by the Text **elt_name**.

A function return value of zero indicates the name was returned successfully.

If **elt** is null, the function return value is non-zero.

ID = 44

Set_colour(Element elt,Integer colour)

Name

Integer Set_colour(Element elt,Integer colour)

Description

Set the colour of the Element **elt**. The colour is given by the Integer **colour**.

A function return value of zero indicates that the colour was successfully set.

Notes

(a) For an Interface string, the colour is only used when the string is converted to a different string type.

(b) There are supplied functions to convert the colour number to a colour name and vice-versa.

ID = 47

Get_colour(Element elt,Integer &colour)

Name

Integer Get_colour(Element elt,Integer &colour)

Description

Get the colour of the Element **elt**.

The colour (as a number) is returned as the Integer colour.

A function return value of zero indicates the Element colour was successfully returned.

Note

There are 12dPL functions to convert the colour number to a colour name and vice-versa.

ID = 46

Set_model(Element elt,Model model)

Name

Integer Set_model(Element elt,Model model)

Description

Sets the 12d Model model of the Element **elt** to be Model **model**.

If **elt** is already in a model, then it is moved to the Model **model**.

If elt is not in a model, then elt is added to the Model **model**.

A function return value of zero indicates the model was successfully set.

ID = 55

Set_model(Dynamic_Element de,Model model)

Name

Integer Set_model(Dynamic_Element de,Model model)

Description

Sets the Model of all the Elements in the Dynamic_Element **de** to **model**.

For each Element **elt** in the Dynamic_Element, **de** if **elt** is already in a model, then it is moved to the Model **model**. If elt is not in a model, **elt** is added to the Model **model**.

A function return value of zero indicates the models were successfully set.

ID = 141

Get_model(Element elt,Model &model)

Name

Integer Get_model(Element elt,Model &model)

Description

Get the model handle of the model containing the Element **elt**. The model is returned by the Model **model**.

A function return value of zero indicates the handle was returned successfully.

ID = 54

Set_breakline(Element elt,Integer break_type)

Name

Integer Set_breakline(Element elt,Integer break_type)

Description

Sets the breakline type for triangulation purposes for the Element **elt**.

The breakline type is given as the Integer **break_type**.

The break_type is

0 if **elt** is to be used as a point string
1 if **elt** is to be used as a breakline string

A function return value of zero indicates the breakline type was successfully set.

LJG? what about arcs, circles

ID = 53

Get_breakline(Element elt,Integer &break_type)

Name

Integer Get_breakline(Element elt,Integer &break_type)

Description

Gets the breakline type of the Element **elt**. The breakline type is used for triangulation purposes and is returned as the Integer break_type.

The **break_type** is

0 if **elt** is used as a point string

1 breakline string

A function return value of zero indicates the breakline type was returned successfully.

ID = 52

Get_type(Element elt,Text &elt_type)

Name

Integer Get_type(Element elt,Text &elt_type)

Description

Get the Element type of the Element **elt**.

The Element type is returned by the Text **elt_type**.

For the types of elements, go to Types of Elements .

A function return value of zero indicates the type was returned successfully.

ID = 64

Set_style(Element elt,Text elt_style)

Name

Integer Set_style(Element elt,Text elt_style)

Description

Set the line style of the Element **elt**.

The name of the line style is given by the Text **elt_style**.

A function return value of zero indicates the style was successfully set.

ID = 49

Get_style(Element elt,Text &elt_style)

Name

Integer Get_style(Element elt,Text &elt_style)

Description

Get the line style of the Element **elt**.

The name of the line style is returned by the Text **elt_style**.

The style is not used for Elements of type Tin or Text.

A function return value of zero indicates the style was returned successfully.

ID = 48

Set_chainage(Element elt,Real start_chain)

Name

Integer Set_chainage(Element elt,Real start_chain)

Description

Set the start chainage of the Element **elt**.

The start chainage is given by the Real **start_chain**.

A function return value of zero indicates the start chainage was successfully set.

ID = 51

Get_chainage(Element elt,Real &start_chain)

Name

Integer Get_chainage(Element elt,Real &start_chain)

Description

Get the start chainage of the Element **elt**.

The start chainage is returned by the Real **start_chain**.

A function return value of zero indicates the chainage was returned successfully.

ID = 50

Get_end_chainage(Element elt,Real &chainage)

Name

Integer Get_end_chainage(Element elt,Real &chainage)

Description

Get the end chainage of the Element **elt**.

The end chainage is returned by the Real **chainage**.

A function return value of zero indicates the chainage was returned successfully.

ID = 654

Get_id(Element elt,Uid &uid)

Name

Integer Get_id(Element elt,Uid &uid)

Description

Get the unique Uid of the Element **elt** and return it in **uid**.

If **elt** is null or an error occurs, **uid** is set to zero.

A function return value of zero indicates the Element Uid was successfully returned.

ID = 1908

Get_id(Element elt,Integer &id)

Name

Integer Get_id(Element elt,Integer &id)

Description

Get the unique id of the Element **elt** and return it in **id**.

If **elt** is null or an error occurs, **id** is set to zero.

A function return value of zero indicates the Element id was successfully returned.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Get_id(Element elt,Uid &id)* instead.

ID = 378

Get_time_created(Element elt,Integer &time)

Name

Integer Get_time_created(Element elt,Integer &time)

Description

Get the time of creation of the Element **elt**.

The time value is returned in Integer **time** (seconds since January 1 1970).

A function return value of zero indicates the data was returned successfully.

ID = 673

Get_time_updated(Element elt,Integer &time)

Name

Integer Get_time_updated(Element elt,Integer &time)

Description

Get the time of the last update of the Element **elt**.

The time value is returned in Integer **time** (seconds since January 1 1970).

A function return value of zero indicates the data was returned successfully.

ID = 674

Set_time_updated(Element elt,Integer time)

Name

Integer Set_time_updated(Element elt,Integer time)

Description

Set the time of the last update of the Element **elt**.

The time value is defined in Integer **time**.

A function return value of zero indicates the time was updated successfully.

ID = 675

Integer Null(Element elt)

Name

Integer Null(Element elt)

Description

Set the Element **elt** to null.

A function return value of zero indicates the Element **elt** was successfully set to null.

Note

The database item pointed to by the Element **elt** is not affected in any way.

ID = 133

Get_extent_x(Element elt,Real &xmin,Real &xmax)

Name

Integer Get_extent_x(Element elt,Real &xmin,Real &xmax)

Description

Gets the x-extents of the Element **elt**.

The minimum x extent is returned by the Real **xmin**.

The maximum x extent is returned by the Real **xmax**.

A function return value of zero indicates the x extents were successfully returned.

ID = 159

Get_extent_y(Element elt,Real &ymin,Real &ymax)

Name

Integer Get_extent_y(Element elt,Real &ymin,Real &ymax)

Description

Gets the y-extents of the Element **elt**.

The minimum y extent is returned by the Real **ymin**.

The maximum y extent is returned by the Real **ymax**.

A function return value of zero indicates the y extents were successfully returned.

ID = 160

Get_extent_z(Element elt,Real &zmin,Real &zmax)

Name

Integer Get_extent_z(Element elt,Real &zmin,Real &zmax)

Description

Gets the z-extents of the Element elt.

The minimum z extent is returned by the Real zmin.

The maximum z extent is returned by the Real zmax.

A function return value of zero indicates the z extents were successfully returned.

ID = 161

Calc_extent(Element elt)

Name

Integer Calc_extent(Element elt)

Description

Calculate the extents of the Element **elt**.

This is necessary after an Element's body data has been modified.

A function return value of zero indicates the extent calculation was successful.

ID = 162

Element_duplicate(Element elt,Element &dup_elt)

Name

Integer Element_duplicate(Element elt,Element &dup_elt)

Description

Create a duplicate of the Element **elt** and return it as the Element **dup_elt**.

A function return value of zero indicates the duplication was successful.

ID = 430

Element_delete(Element elt)

Name

Integer Element_delete(Element elt)

Description

Delete from the 12d Model database the item that the Element **elt** points to. The Element **elt** is then set to null.

A function return value of zero indicates the data base item was deleted successfully.

ID = 41

Get_type(Element elt,Integer &elt_type)

Name

Integer Get_type(Element elt,Integer &elt_type)

Description

NOT IMPLEMENTED.

Get the Element type of the Element elt.

The Element type is returned as the Integer **elt_type**.

A function return value of zero indicates the type was returned successfully.

ID = 42

Element Attributes Functions

Get_attributes(Element elt,Attributes &att)

Name

Integer Get_attributes(Element elt,Attributes &att)

Description

For the Element **elt**, return the Attributes for the Element as **att**.

If the Element has no attribute then a non-zero return value is returned.

A function return value of zero indicates the attribute is successfully returned.

ID = 1972

Set_attributes(Element elt,Attributes att)

Name

Integer Set_attributes(Element elt,Attributes att)

Description

For the Element **elt**, set the Attributes for the Element to **att**.

A function return value of zero indicates the attribute is successfully set.

ID = 1973

Get_attribute(Element elt,Text att_name,Uid &uid)

Name

Integer Get_attribute(Element elt,Text att_name,Uid &uid)

Description

From the Element **elt**, get the attribute called **att_name** from **elt** and return the attribute value in **uid**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - this function is more efficient than getting the Attributes from the Element and then getting the data from that Attributes.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1974

Get_attribute(Element elt,Text att_name,Attributes &att)

Name

Integer Get_attribute(Element elt,Text att_name,Attributes &att)

Description

From the Element **elt**, get the attribute called **att_name** from **elt** and return the attribute value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - this function is more efficient than getting the Attributes from the Element and then getting

the data from that Attributes.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1975

Get_attribute(Element elt,Integer att_no,Uid &uid)

Name

Integer Get_attribute(Element elt,Integer att_no,Uid &uid)

Description

From the Element **elt**, get the attribute with number **att_no** and return the attribute value in **uid**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1976

Get_attribute(Element elt,Integer att_no,Attributes &att)

Name

Integer Get_attribute(Element elt,Integer att_no,Attributes &att)

Description

From the Element **elt**, get the attribute with number **att_no** and return the attribute value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1977

Set_attribute(Element elt,Text att_name,Uid uid)

Name

Integer Set_attribute(Element elt,Text att_name,Uid uid)

Description

For the Element **elt**,
if the attribute called **att_name** does not exist in the element then create it as type Uid and give it the value **uid**.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **att**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1978

Set_attribute(Element elt,Text att_name,Attributes att)

Name

Integer Set_attribute(Element elt,Text att_name,Attributes att)

Description

For the Element **elt**,
if the attribute called **att_name** does not exist in the element then create it as type Attributes and give it the value **att**.
if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1979

Set_attribute(Element elt,Integer att_no,Uid uid)

Name

Integer Set_attribute(Element elt,Integer att_no,Uid uid)

Description

For the Element **elt**, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **uid**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1980

Set_attribute(Element elt,Integer att_no,Attributes att)

Name

Integer Set_attribute(Element elt,Integer att_no,Attributes att)

Description

For the Element **elt**, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1981

Attribute_exists(Element elt,Text att_name)

Name

Integer Attribute_exists(Element elt,Text att_name)

Description

Checks to see if a user attribute with the name **att_name** exists in the Element **elt**.

A non-zero function return value indicates that the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values.

ID = 555

Attribute_exists(Element elt,Text att_name,Integer &att_no)

Name

Integer Attribute_exists(Element elt,Text att_name,Integer &att_no)

Description

Checks to see if a user attribute with the name **att_name** exists in the Element **elt**.

If the attribute exists, its position is returned in Integer **att_no**.

This position can be used in other Attribute functions described below.

A non-zero function return value indicates the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 556

Attribute_delete(Element elt,Text att_name)

Name

Integer Attribute_delete(Element elt,Text att_name)

Description

Delete the user attribute with the name **att_name** for Element **elt**.

A function return value of zero indicates the attribute was deleted.

ID = 557

Attribute_delete(Element elt,Integer att_no)

Name

Integer Attribute_delete(Element elt,Integer att_no)

Description

Delete the user attribute at the position **att_no** for Element **elt**.

A function return value of zero indicates the attribute was deleted.

ID = 558

Attribute_delete_all(Element elt)

Name

Integer Attribute_delete_all(Element elt)

Description

Delete all the user attributes for Element **elt**.

A function return value of zero indicates all the attributes were deleted.

ID = 559

Get_number_of_attributes(Element elt,Integer &no_atts)

Name

Integer Get_number_of_attributes(Element elt,Integer &no_atts)

Description

Get the total number of user attributes for Element **elt**.

The total number of attributes is returned in Integer **no_atts**.

A function return value of zero indicates the attribute was successfully returned.

ID = 560

Get_attribute(Element elt,Text att_name,Text &att)

Name

Integer Get_attribute(Element elt,Text att_name,Text &att)

Description

Get the data for the user attribute with the name **att_name** for Element **elt**.

The user attribute must be of type **Text** and is returned in Text **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 561

Get_attribute(Element elt,Text att_name,Integer &att)

Name

Integer Get_attribute(Element elt,Text att_name,Integer &att)

Description

Get the data for the user attribute with the name att_name for Element elt.

The user attribute must be of type Integer and is returned in att.

A function return value of zero indicates the attribute was successfully returned.

ID = 562

Get_attribute(Element elt,Text att_name,Real &att)

Name

Integer Get_attribute(Element elt,Text att_name,Real &att)

Description

Get the data for the user attribute with the name **att_name** for Element **elt**.

The user attribute must be of type **Real** and is returned in **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 563

Get_attribute(Element elt,Integer att_no,Text &att)

Name

Integer Get_attribute(Element elt,Integer att_no,Text &att)

Description

Get the data for the user attribute at the position **att_no** for Element **elt**.

The user attribute must be of type **Text** and is returned in **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 564

Get_attribute(Element elt,Integer att_no,Integer &att)

Name

Integer Get_attribute(Element elt,Integer att_no,Integer &att)

Description

Get the data for the user attribute at the position **att_no** for Element **elt**.

The user attribute must be of type **Integer** and is returned in Integer **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 565

Get_attribute(Element elt,Integer att_no,Real &att)

Name

Integer Get_attribute(Element elt,Integer att_no,Real &att)

Description

Get the data for the user attribute at the position **att_no** for Element **elt**.

The user attribute must be of type Real and is returned in Real **att**.

A function return value of zero indicates the attribute was successfully returned.

ID = 566

Get_attribute_name(Element elt,Integer att_no,Text &name)

Name

Integer Get_attribute_name(Element elt,Integer att_no,Text &name)

Description

Get the name for the user attribute at the position **att_no** for Element **elt**.

The user attribute name found is returned in Text **name**.

A function return value of zero indicates the attribute name was successfully returned.

ID = 567

Get_attribute_type(Element elt,Text att_name,Integer &att_type)

Name

Integer Get_attribute_type(Element elt,Text att_name,Integer &att_type)

Description

Get the type of the user attribute with the name **att_name** from the Element **elt**.

The user attribute type is returned in Integer **att_type**.

For the list of attribute types, go to Data Type Attribute Type .

A function return value of zero indicates the attribute type was successfully returned.

ID = 568

Get_attribute_type(Element elt,Integer att_no,Integer &att_type)

Name

Integer Get_attribute_type(Element elt,Integer att_no,Integer &att_type)

Description

Get the type of the user attribute at position **att_no** for the Element **elt**.

The user attribute type is returned in **att_type**.

For the list of attribute types, go to Data Type Attribute Type .

A function return value of zero indicates the attribute type was successfully returned.

ID = 569

Get_attribute_length(Element elt,Text att_name,Integer &att_len)

Name

Integer Get_attribute_length(Element elt,Text att_name,Integer &att_len)

Description

Get the length of the user attribute with the name **att_name** for Element **elt**.

The user attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for user attributes of type **Text** and **Binary.**

ID = 570

Get_attribute_length(Element elt,Integer att_no,Integer &att_len)

Name

Integer Get_attribute_length(Element elt,Integer att_no,Integer &att_len)

Description

Get the length of the user attribute at position **att_no** for Element **elt**.

The user attribute length is returned in **att_len**.

A function return value of zero indicates the attribute type was successfully returned.

Note - the length is useful for user attributes of type **Text** and **Binary.**

ID = 571

Set_attribute(Element elt,Text att_name,Text att)

Name

Integer Set_attribute(Element elt,Text att_name,Text att)

Description

For the Element **elt**, set the user attribute with name **att_name** to the Text **att**.

The user attribute **must** be of type **Text**

A function return value of zero indicates the attribute was successfully set.

ID = 572

Set_attribute(Element elt,Text att_name,Integer att)

Name

Integer Set_attribute(Element elt,Text att_name,Integer att)

Description

For the Element **elt**, set the user attribute with name **att_name** to the Integer att.

The user attribute **must** be of type **Integer**

A function return value of zero indicates the attribute was successfully set.

ID = 573

Set_attribute(Element elt,Text att_name,Real att)

Name

Integer Set_attribute(Element elt,Text att_name,Real att)

Description

For the Element **elt**, set the user attribute with name **att_name** to the Real **att**.

The user attribute **must** be of type **Real**

A function return value of zero indicates the attribute was successfully set.

ID = 574

Set_attribute(Element elt,Integer att_no,Text att)

Name

Integer Set_attribute(Element elt,Integer att_no,Text att)

Description

For the Element **elt**, set the user attribute at position **att_no** to the Text **att**.

The user attribute **must** be of type **Text**

A function return value of zero indicates the attribute was successfully set.

ID = 575

Set_attribute(Element elt,Integer att_no,Integer att)

Name

Integer Set_attribute(Element elt,Integer att_no,Integer att)

Description

For the Element **elt**, set the user attribute at position **att_no** to the Integer **att**.

The user attribute **must** be of type **Integer**

A function return value of zero indicates the attribute was successfully set.

ID = 576

Set_attribute(Element elt,Integer att_no,Real att)

Name

Integer Set_attribute(Element elt,Integer att_no,Real att)

Description

For the Element **elt**, set the user attribute at position **att_no** to the Real **att**.

The user attribute **must** be of type **Real**

A function return value of zero indicates the attribute was successfully set.

ID = 577

Attribute_dump(Element elt)

Name

Integer Attribute_dump(Element elt)

Description

Write out information about the Element attributes to the Output Window.

A function return value of zero indicates the function was successful.

ID = 578

Attribute_debug(Element elt)

Name

Integer Attribute_debug(Element elt)

Description

Write out even more information about the Element attributes to the Output Window.

A function return value of zero indicates the function was successful.

ID = 589

Tin Element

The variable type **Tin** is used to refer to the standard 12d Model tins or triangulations.

Tin variables act as ***handles*** to the actual tin so that the tin can be easily referred to and manipulated within a macro.

See Triangulate Data
See Tin Functions
See Null Triangles
See Colour Triangles

Triangulate Data

Triangulate(Dynamic_Element de,Text tin_name,Integer tin_colour,Integer preserve,Integer bubbles,Tin &tin)

Name

Integer Triangulate(Dynamic_Element de,Text tin_name,Integer tin_colour,Integer preserve,Integer bubbles,Tin &tin)

Description

The elements from the Dynamic_Element **de** are triangulated and a tin named **tin_name** created with colour **tin_colour**.

A non zero value for **preserve** allows break lines to be preserved.

A non zero value for **bubbles** removes bubbles from the triangulation.

A created tin is returned by Tin **tin**.

A function return value of zero indicates the triangulation was successful.

ID = 142

Triangulate(Dynamic_Text list,Text tin_name,Integer colour,Integer preserve,Integer bubbles,Tin &tin)

Name

Integer Triangulate(Dynamic_Text list,Text tin_name,Integer colour,Integer preserve,Integer bubbles,Tin &tin)

Description

Triangulate the data from a list of models Dynamic_Text **list**.

The tin name is given as Text **tin_name**, the tin colour is given as Integer **colour**, the preserve string option is given by Integer **preserve**, and the remove bubbles option is given by Integer **bubbles**, 1 is on, 0 is off.

A function return value of zero indicates the Tin **tin** was successfully returned.

ID = 1428

Tin Functions

Tin_exists(Text tin_name)

Name

Integer Tin_exists(Text tin_name)

Description

Checks to see if a tin with the name **tin_name** exists.

A non-zero function return value indicates a tin does exist.

A zero function return value indicates that no tin of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 355

Tin_exists(Tin tin)

Name

Integer Tin_exists(Tin tin)

Description

Checks if the Tin **tin** is valid (that is, not null).

A non-zero function return value indicates that **tin** is not null.

A zero function return value indicates that tin is null.

Warning this is the opposite of most 12dPL function return values

ID = 356

Get_project_tins(Dynamic_Text &tins)

Name

Integer Get_project_tins(Dynamic_Text &tins)

Description

Get the names of all the tins in the project. The names are returned in the Dynamic_Text, **tins**.

A function return value of zero indicates the tin names were returned successfully.

ID = 232

Get_tin(Text tin_name)

Name

Tin Get_tin(Text tin_name)

Description

Get a Tin handle for the tin with name **tin_name**.

If the tin exists, the handle to it is returned as the function return value.

If the tin does not exist, a null Tin is returned as the function return value.

ID = 146

Get_tin(Element elt)

Name

Tin Get_tin(Element elt)

Description

If the Element **elt** is of type **Tin** and the tin exists, a Tin handle to the tin is returned as the function return value.

If the tin does not exist or the Element is not of type Tin, a null Tin is returned as the function return value.

ID = 370

Get_name(Tin tin,Text &tin_name)

Name

Integer Get_name(Tin tin,Text &tin_name)

Description

Get the name of the Tin **tin**.

The tin name is returned in the Text **tin_name**.

A function return value of zero indicates success.

If **tin** is null, the function return value is non-zero.

Tin_models(Tin tin, Dynamic_Text &models_used)

Name

Integer Tin_models(Tin tin, Dynamic_Text &models_used)

Description

Get the names of all the models that were used to create the Tin **tin**.

The model names are returned in the Dynamic_Text **models_used**.

A function return value of zero indicates that the view names were returned successfully.

ID = 431

Get_time_created(Tin tin,Integer &time)

Name

Integer Get_time_created(Tin tin,Integer &time)

Description

Get the time that the Tin **tin** was created and return the time in **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully returned.

ID = 2114

Get_time_updated(Tin tin,Integer &time)

Name

Integer Get_time_updated(Tin tin,Integer &time)

Description

Get the time that the Tin **tin** was last updated and return the time in **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully returned.

ID = 2115

Set_time_updated(Tin tin,Integer time)

Name

Integer Set_time_updated(Tin tin,Integer time)

Description

Set the update time for the Tin **tin** to **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully set.

ID = 2116

Tin_number_of_points(Tin tin,Integer &notri)

Name

Integer Tin_number_of_points(Tin tin,Integer &notri)

Description

Get the total number of points used in creating the Tin **tin**.

This value includes duplicate points.

The number of triangles is returned in the Integer **notri**.

A function return value of zero indicates success.

If **tin** is null, the function return value is non-zero.

ID = 472

Tin_number_of_triangles(Tin tin,Integer &notri)

Name

Integer Tin_number_of_triangles(Tin tin,Integer &notri)

Description

Get the number of triangles in the Tin **tin**.

The number of triangles is returned in the Integer **notri**.

A function return value of zero indicates success.

If **tin** is null, the function return value is non-zero.

ID = 473

Tin_number_of_duplicate_points(Tin tin,Integer &notri)

Name

Integer Tin_number_of_duplicate_points(Tin tin,Integer &notri)

Description

Get the number of duplicate points found whilst creating the Tin **tin**.

The number of duplicate points is returned in the Integer **notri**.

A function return value of zero indicates success.

If **tin** is null, the function return value is non-zero.

ID = 474

Tin_number_of_items(Tin tin,Integer &num_items)

Name

Integer Tin_number_of_items(Tin tin,Integer &num_items)

Description

The number of strings in the tin **tin** is returned as **num_items**. Note that if the original string in the data set to be triangulated had invisible segments (discontinuities) then that string is broken into two or more strings in the tin.

A function return value of zero indicates that **num_items** was successfully returned.

ID = 475

Tin_colour(Tin tin,Real x,Real y,Integer &colour)

Name

Integer Tin_colour(Tin tin,Real x,Real y,Integer &colour)

Description

Get the colour of the tin at the point (x,y)

A function return value of zero indicates success.

ID = 218

Tin_height(Tin tin,Real x,Real y,Real &height)

Name

Integer Tin_height(Tin tin,Real x,Real y,Real &height)

Description

Get the height of the tin at the point (x,y).

If (x,y) is outside the tin, then an error has occurred and a non-zero function return value is set.

A function return value of zero indicates the height was successfully returned.

ID = 215

Tin_slope(Tin tin,Real x,Real y,Real &slope)

Name

Integer Tin_slope(Tin tin,Real x,Real y,Real &slope)

Description

Get the slope of the tin at the point (x,y).

The units for slope is an angle in radians measured from the horizontal plane.

If (x,y) is outside the tin, then an error has occurred and a non-zero function return value is set.

A function return value of zero indicates the slope was successfully returned.

ID = 216

Tin_aspect(Tin tin,Real x,Real y,Real &aspect)

Name

Integer Tin_aspect(Tin tin,Real x,Real y,Real &aspect)

Description

Get the aspect of the tin at the point (x,y).

The units for aspect is a bearing in radians. That is, aspect is given as a clockwise angle measured from the positive y-axis (North).

If (x,y) is outside the tin, then an error has occurred and a non-zero function return value is set.

A function return value of zero indicates the aspect was successfully returned.

ID = 217

Tin_duplicate(Tin tin,Text dup_name)

Name

Integer Tin_duplicate(Tin tin,Text dup_name)

Description

Create a new Tin with name dup_name which is a duplicate the Tin **tin**.

IT is an error if a Tin called **dup_name** already exists.

A function return value of zero indicates the duplication was successful.

ID = 429

Tin_rename(Text original_name,Text new_name)

Name

Integer Tin_rename(Text original_name,Text new_name)

Description

Change the name of the Tin **original_name** to the new name **new_name**.

A function return value of zero indicates the rename was successful.

ID = 422

Tin_boundary(Tin tin,Integer colour_for_strings,Dynamic_Element &de)

Name

Integer Tin_boundary(Tin tin,Integer colour_for_strings,Dynamic_Element &de)

Description

Get the boundary polygons for the Tin **tin**. The polygons are returned in the Dynamic_Element **de** with colour **colour_for_strings**.

A function return value of zero indicates the data was successfully returned.

ID = 476

Tin_delete(Tin tin)

Name

Integer Tin_delete(Tin tin)

Description

Delete the Tin **tin** from the project and the disk.

A function return value of zero indicates the tin was deleted successfully.

ID = 219

Tin_get_point(Tin tin,Integer np,Real &x,Real &y,Real &z)

Name

Integer Tin_get_point(Tin tin,Integer np,Real &x,Real &y,Real &z)

Description

Get the (x,y,z) coordinate of **np**'th point of the **tin**.

The x value is returned in Real **x**.

The y value is returned in Real **y**.

The z value is returned in Real **z**.

A function return value of zero indicates the coordinate of the point was successfully returned.

ID = 831

Tin_get_triangle_points(Tin tin,Integer nt,Integer &p1,Integer &p2,Integer &p3)

Name

Integer Tin_get_triangle_points(Tin tin,Integer nt,Integer &p1,Integer &p2,Integer &p3)

Description

Get the three points of **nt**'th triangle of the **tin**.

The first point value is returned in Integer **p1**.

The second point value is returned in Integer **p2**.

The third point value is returned in Integer **p3**.

A function return value of zero indicates the points were successfully returned.

ID = 832

Tin_get_triangle_neighbours(Tin tin,Integer nt,Integer &n1,Integer &n2, Integer &n3)

Name

Integer Tin_get_triangle_neighbours(Tin tin,Integer nt,Integer &n1,Integer &n2,Integer &n3)

Description

Get the three neighbour triangles of the **nt**'th triangle of the **tin**.

The first triangle neighbour is returned in Integer **n1**.

The second triangle neighbour is returned in Integer **n2**.

The third triangle neighbour is returned in Integer **n3**.

A function return value of zero indicates the triangles were successfully returned.

ID = 833

Tin_get_point_from_point(Tin tin,Real x,Real y,Integer &np)

Name

Integer Tin_get_point_from_point(Tin tin,Real x,Real y,Integer &np)

Description

For the Tin ***tin*** and the coordinate (**x**,**y**), get the tin point number of the vertex of the triangle closest to (**x,y**), and returned it in ***np***.

A function return value of zero indicates the function was successful.

ID = 1436

Tin_get_triangles_about_point(Tin tin,Integer n,Integer &no_triangles)

Name

Integer Tin_get_triangles_about_point(Tin tin,Integer n,Integer &no_triangles)

Description

For the Tin ***tin*** and the **n**th point of tin, get the number of triangles surrounding the point and return the number in *no_triangles*.

A function return value of zero indicates the function was successful.

ID = 1628

Tin_get_triangles_about_point(Tin tin,Integer n,Integer max_triangles,Integer &no_triangles,Integer triangles[],Integer points[],Integer status[])

Name

Integer Tin_get_triangles_about_point(Tin tin,Integer n,Integer max_triangles,Integer &no_triangles,Integer triangles[],Integer points[],Integer status[])

Description

For the Tin ***tin*** and the **n**th point of tin,

get the number of triangles surrounding the point and return it as **no_triangles**
return the list of triangle numbers in **triangles[]**
return the list of all the point numbers of vertices of the triangles that surround the point in **points[]** (the number of these is the same as the number of triangle around the point)
LJG? return the *status* of each triangle in **triangles[].** *status* is 0 for a null triangle, 1 for other triangles.

Note: *max_triangles* is the size of the arrays triangles[], points[] and status[]. The number of triangles surrounding the **n**th point of a tin is given by *Tin_get_triangles_about_point*.

A function return value of zero indicates the function was successful.

ID = 1629

Tin_get_triangle_inside(Tin tin,Integer triangle,Integer &Inside)

Name

Integer Tin_get_triangle_inside(Tin tin,Integer triangle,Integer &Inside)

Description

Get the condition of the nth **triangle** of the **tin**.

If the value of the flag **Inside** is

0	not valid triangle.
1	not valid triangle.
2	the triangle is a non-null triangle.

So for a valid triangle, **inside** = **2**.

A function return value of zero indicates the flag was successfully returned.

ID = 835

Tin_get_triangle*(Tin tin,Integer triangle,Integer &p1,Integer &p2,Integer &p3,Integer &n1,Integer &n2,Integer &n3,Real &x1,Real &y1,Real &z1,Real &x2,Real &y2,Real &z2,Real &x3,Real &y3,Real &z3)*

Name

Integer Tin_get_triangle(Tin tin,Integer triangle,Integer &p1,Integer &p2,Integer &p3,Integer &n1,Integer &n2,Integer &n3,Real &x1,Real &y1,Real &z1,Real &x2,Real &y2,Real &z2,Real &x3,Real &y3,Real &z3)

Description

Get the three points and their (x,y,z) data and three neighbour triangles of nth **triangle** of the **tin**.

The first point is returned in Integer **p1**, the (x, y, z) value is returned in **x1**,**y1**,**z1**.

The second point is returned in Integer **p2**, the (x, y, z) value is returned in **x2**,**y2**,**z2**.

The third point is returned in Integer **p3**, the x, y, z values are returned in **x3**,**y3**,**z3**.

The first triangle neighbour is returned in Integer **n1**.

The second triangle neighbour is returned in Integer **n2**.

The third triangle neighbour is returned in Integer **n3**.

A function return value of zero indicates the data was successfully returned.

ID = 836

Tin_get_triangle_from_point(Tin tin,Real x,Real y,Integer &triangle)

Name

Integer Tin_get_triangle_from_point(Tin tin,Real x,Real y,Integer &triangle)

Description

Get the triangle of the Tin **tin** that contains the given coordinate (**x**,**y**).

The triangle number is returned in Integer **triangle**.

A function return value of zero indicates the triangle was successfully returned.

ID = 837

Draw_triangle(Tin tin,Integer tri,Integer c)

Name

Integer Draw_triangle(Tin tin,Integer tri,Integer c)

Description

Draw the triangle **tri** with colour **c** inside the Tin **tin**.

A function return value of zero indicates the triangle was successfully drawn.

ID = 1433

Draw_triangles_about_point(Tin tin,Integer pt,Integer c)

Name

Integer Draw_triangles_about_point(Tin tin,Integer pt,Integer c)

Description

Draw the triangles about a point **pt** with colour **c** inside Tin **tin**.

A function return value of zero indicates the triangles were successfully drawn.

ID = 1434

Triangles_clip(Real x1,Real y1,Real x2,Real y2,Real x3,Real y3,Real x4,Real y4,Real z4,Real x5,Real y5,Real z5,Real x6,Real y6,Real z6, Integer &npts_out,Real xarray_out[],Real yarray_out[],Real zarray_out[])

Name

Integer Triangles_clip(Real x1,Real y1,Real x2,Real y2,Real x3,Real y3,Real x4,Real y4,Real z4,Real x5,Real y5,Real z5,Real x6,Real y6,Real z6,Integer &npts_out,Real xarray_out[],Real yarray_out[],Real zarray_out[])

Description

The vertices of a 2d triangle is defined by the coordinates (**x1**,**y1**), (**x2**,**y2**) and (**x3**,**y3**).

The vertices of a 3d triangle is defined by the coordinates (**x4**,**y4**,**z4**), (**x5**,**y5**,**z5**) and (**x6**,**y6**,**z6**).

The Real arrays **xarray_out[]**, **yarray_out[]**, **zarrary_out[]** must exist and have dimensions at least 9.

The function uses the 2d triangle to clip the 3d triangle and return the polygon of 3d clips points in the arrays xarray_out[], yarray_out[], zarrar_out[]. The number of clips points is returned in **npts_out**.

A function return value of zero indicates the function was successful.

ID = 1439

Tin_models(Tin tin,Dynamic_Text &models)

Name

Integer Tin_models(Tin tin,Dynamic_Text &models)

Description

WARNING - this does not appear to be correct. There is another Tin_models documented.

LJG ERROR

Get the model names **models** that contains Tin **tin**.

Type of models must be **Dynamic_Text**.

A function return value of zero indicates the models were successfully returned.

Retriangulate(Tin tin)

Name

Integer Retriangulate(Tin tin)

Description

Retriangulate the Tin **tin**.

A function return value of zero indicates the Tin **tin** was successfully returned.

ID = 1429

Breakline(Tin tin,Integer p1,Integer p2)

Name

Integer Breakline(Tin tin,Integer p1,Integer p2)

Description

Add breakline in Tin **tin** from point 1 **p1** to point 2 **p2**.

A function return value of zero indicates the breakline was successfully added.

ID = 1430

Flip_triangles(Tin tin,Integer t1,Integer t2)

Name

Integer Flip_triangles(Tin tin,Integer t1,Integer t2)

Description

From the triangles **t1** and **t2** in Tin **tin**.

A function return value of zero indicates the triangles were successfully flipped.

ID = 1431

Set_height(Tin tin,Integer pt,Real ht)

Name

Integer Set_height(Tin tin,Integer pt,Real ht)

Description

Set the height Real ht for the point pt on the Tin **tin**.

A function return value of zero indicates the height was successfully set.

ID = 1432

Set_supertin(Tin_Box box,Integer mode)

Name

Integer Set_supertin(Tin_Box box,Integer mode)

Description

ID = 1311

Null Triangles

Null(Tin tin)

Name

Integer Null(Tin tin)

Description

Set the Tin handle **tin** to null. This does not affect the 12d Model tin that the handle pointed to.

A function return value of zero indicates **tin** was successfully nulled.

ID = 376

Null_triangles(Tin tin,Element poly,Integer mode)

Name

Integer Null_triangles(Tin tin,Element poly,Integer mode)

Description

Set any triangle whose centroid is inside or outside a given polygon to null.

tin is the tin to null and **poly** is the polygon which restricts the nulling.

If **mode** is

0	the inside of the polygon is nulled.
1	the outside is nulled.

A function return value of zero indicates there were no errors in the nulling calculations.

ID = 153

Reset_null_triangles(Tin tin,Element poly,Integer mode)

Name

Integer Reset_null_triangles(Tin tin,Element poly,Integer mode)

Description

Set any null triangle whose centroid is inside or outside a given polygon to be a valid triangle.

tin is the tin to reset and **poly** is the polygon which determines which triangles are to be reset

If **mode** is

0	the inside of the polygon is reset.
1	the outside is reset.

A function return value of zero indicates there were no errors in the reset calculations.

ID = 154

Reset_null_triangles(Tin tin)

Name

Integer Reset_null_triangles(Tin tin)

Description

Set all the triangles of the tin **tin** to be valid triangles.

A function return value of zero indicates there were no errors in the reset calculations.

ID = 155

Null_by_angle_length(Tin tin,Real l1,Real a1,Real l2,Real a2)

Name

Integer Null_by_angle_length(Tin tin,Real l1,Real a1,Real l2,Real a2)

Description

Refer to reference manual Page 444 “Null by Angle and Length”.

A function return value of zero indicates the triangle was nulled successfully.

ID = 1435

Colour Triangles

Get_colour(Tin tin,Integer &colour)

Name

Integer Get_colour(Tin tin,Integer &colour)

Description

Get the colour of the Tin **tin**.

The colour (as a number) is returned as the Integer **colour**.

A function return value of zero indicates the colour was returned successfully.

Note

There are 12dPL functions to convert the colour number to a colour name and vice-versa.

Set_colour(Tin tin,Integer colour)

Name

Integer Set_colour(Tin tin,Integer colour)

Description

Set the colour of the Tin **tin**. The colour is given by the Integer **colour**.

A function return value of zero indicates that the colour was successfully set.

Tin_get_triangle_colour(Tin tin,Integer triangle,Integer &colour)

Name

Integer Tin_get_triangle_colour(Tin tin,Integer triangle,Integer &colour)

Description

Get the **colour** of the nth **triangle** of the tin.

The colour value is returned in Integer **colour**.

A function return value of zero indicates the colour were successfully returned.

ID = 834

Colour_triangles(Tin tin,Integer col_num,Element poly,Integer mode)

Name

Integer Colour_triangles(Tin tin,Integer colour,Element poly,Integer mode)

Description

Colour all the triangles in the Tin **tin** whose centroids are inside or outside a given polygon to a specified colour.

The triangulation is **tin**, the polygon **poly** and the colour number **col_num**.

The value of **mode** determines whether the triangles whose centroids are inside or outside the polygon are coloured.

If mode equals 0, the triangles inside the polygon are coloured.

If mode equals 1, the triangles outside the polygon are coloured.

A function return value of zero indicates there were no errors in the colour calculations.

ID = 156

Reset_colour_triangles(Tin tin,Element poly,Integer mode)

Name

Integer Reset_colour_triangles(Tin tin,Element poly,Integer mode)

Description

Set any triangle in the Tin **tin** whose centroid is inside or outside a given polygon back to the base tin colour.

The value of **mode** determines whether the triangles whose centroids are inside or outside the polygon are set back to the base colour.

If mode equals 0, the triangles inside the polygon are set

If mode equals 1, the triangles outside the polygon are set

A function return value of zero indicates there were no errors in the colour reset calculations.

ID = 157

Reset_colour_triangles(Tin tin)

Name

Integer Reset_colour_triangles(Tin tin)

Description

Set all the triangles in the Tin **tin** back to the base tin colour.

A function return value of zero indicates success.

ID = 158

Super String Element

The Super String is a very general string which was introduced to not only replace the string types 2d, 3d, 4d, interface, face, pipe and polyline, but also to allow for combinations that were never allowed in the old strings. For example, to have a polyline string but with a pipe diameter, or a 2d string with text at each vertex.

Different strings to cover every possible combination would have required hundreds of different string types. A better solution was to have one string type that has information to cover all of the properties of the other strings, and the ability to more easily add other properties now and in the future. This flexible string is the ***Super String***.

Having all possible combinations defined for every Super String would be very inefficient for computer storage and processing speed, so the Super String uses the concept of ***dimensions*** to refer to the different types of information that ***could*** be stored in the Super String.

Each **dimension** is well defined and is also ***optional*** so that no unnecessary information is required to be stored.

A Super String **always** has an (x,y) value for each vertex but what other information exists for a particular Super String depends on what optional dimensions are defined for that Super String.

For example, there are two **Height** dimensions called Att_ZCoord_Value and Att_ZCoord_Array. If Att_ZCoord_Value is set then the super string has a constant height value for the entire string (2d super string), and if Att_ZCoord_Array is set, then there is a z value for each vertex (3d super string). If **both** are set then Att_ZCoord_Array takes precedence

So the two Height dimensions cover the functionality of both the old 2d string (one height for the entire string) and the old 3d string (different z value at each vertex). Plus the 2d super string only requires the storage of one height like the old 2d string and not the additional storage required for a z value at every vertex that the 3d string needs.

Please continue to Super String Dimensions

Super String Dimensions

The super string supports over 50 different dimensions.

Each *dimension has a* ***unique number*** and also a **unique name** and either the unique name or the dimension number can be used in calls requiring a super string dimension.

When **creating** a super string, the super string must be told that a particular dimension is to exist (by setting the dimension on or off) and there are function calls to set each dimension (Set_super_*use* calls) on or off.

For an **existing** super string, there are inquiry calls to check if a particular dimension is on or off (Get_super_*use* calls). The Set_super_use and Get_super_use function calls are documented after the documentation on dimensions.

Some dimensions are mutually exclusive (that is, only one of them can exist) and others can exist together but one may take precedence over others.

In the definitions of the dimensions, where two dimensions are listed on the one line with an ***or*** between them, then if **both** exist, ***the array dimension takes precedence over the value dimension***, and the super string may compress or remove the value dimension.

Although there are calls to set each of the dimensions individually, it is also possible to set more than one dimension at once using flags that combine dimension values (see Dimension Combinations and Super String Flags)

The dimension definitions and the user function calls are not given in dimension number order

but for convenience are grouped together by common functionality.

Finally there are also general super string creation and data setting calls documented in the sections Basic Super String Functions and General Element Operations .

For information on each of the Super String Dimensions:

See Height Dimensions
See Segment Radius Dimension
See Interval Dimensions
See Pipe/Culvert Dimensions
See Vertex Text Dimensions
See Vertex Text Annotation Dimensions
See Segment Text Dimensions
See Segment Text Annotation Dimensions
See Point Id Dimension
See Vertex Symbol Dimensions
See Tinability Dimensions
See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions
See Hole Dimension
See User Defined Vertex Attributes Dimensions
See User Defined Segment Attributes Dimensions
See Colour Dimension
See Vertex Image Dimensions
See Segment Geometry Dimension
See Visibility Dimensions
See Matrix Dimension
See UID Dimensions
See Database Point Dimensions
See Extrude Dimensions
See Null Levels Dimensions

For information on setting more than one dimension at once, see Dimension Combinations and Super String Flags

For information on the functions for creating super strings (with flags to set dimension) and for loading and inquiring on the standard (x,y,z,radius,bulge) data, see Basic Super String Functions

For information on the Super String function calls for setting and inquiring on each particular dimension, and calls for loading and inquiring on the particular data for that dimension:

See Super String Height Functions
See Super String Tinability Functions
See Super String Segment Radius Functions
See Super String Point Id Functions
See Super String Vertex Symbol Functions
See Super String Pipe/Culvert Functions
See Super String Vertex Text and Annotation Functions
See Super String Segment Text and Annotation Functions
See Super String Fills - Hatch/Solid/Bitmap/Pattern/ACAD Pattern Functions
See Super String Hole Functions
See Super String Segment Colour Functions
See Super String Segment Geometry Functions
See Super String Extrude Functions
See Super String Vertex Attributes Functions
See Super String Segment Attributes Functions
See Super String Uid Functions

See Super String Vertex Image Functions
See Super String Visibility Functions

Height Dimensions

Att_ZCoord_Array 2 **or only** Att_ZCoord_Value 1

If Att_ZCoord_Array is set, then the super string has a z-value for each vertex.

If Att_ZCoord_Value is set and Att_ZCoord_Array not set, then the super string has one z-value for the entire string.

If neither dimension exists, then the string with no height. That is, it is a string with null height.

See Super String Height Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Segment Radius Dimension

Att_Radius_Array 3

Att_Major_Array 4

If Att_Radius_Array is set, then the super string segments can be arcs, and there is an array to record the radius of the arc for each segment.

If Att_Major_Array is set, then there is an array to record for each segment if the arc is a major or minor arc. That is, the bulge value (bulge of segment b = 1 for major arc > 180 degrees, b = 0 for minor arc < 180 degrees).

If neither dimension is set, then all the string segments are straight lines.

NOTE: In the current implementation, the Att_Major_Array is automatically set when Att_Radius_Array is set.

See Super String Segment Radius Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Interval Dimensions

Att_Interval_Value 50

If Att_Interval_Value is set, then for triangulation purposes there is a Real *interval_distance* used to add extra temporary vertices into the super string, and a *chord_arc_distance* which is also used as a chord to arc tolerance for adding additional temporary vertices into the super string.

See Super String Interval Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Point Id Dimension

Att_Point_Array 11 For a Point id at each vertex

If Att_Point_Array is set, then the super string can have a Point Id at each vertex.

See Super String Point Id Functions *for calls to set/inquire on this dimension, and to load/retrieve data for this dimension.*

Vertex Symbol Dimensions

Att_Symbol_Array 18 **or only** Att_Symbol_Value 17

If Att_Symbol_Array is set, then the super string can have symbols at each vertex.

If Att_Symbol_Value is set and Att_Symbol_Array not set, then the super string has the one symbol for each vertex of the string.

See Super String Vertex Symbol Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Tinability Dimensions

Att_Contour_Array 3 This dimension applies for both vertex and segment tinability.

Att_Vertex_Tinable_Array 38 **or only** Att_Vertex_Tinable_Value 37

If Att_Vertex_Tinable_Array is set, then the super string can have a different tinability at each vertex.

If Att_Vertex_Tinable_Value is set and Att_Vertex_Tinable_Array not set, then the super string has the one tinability value to be used for all vertices of the string.

Att_Segment_Tinable_Value 39 **or** Att_Segment_Tinable_Array 40

If Att_Segment_Tinable_Array is set, then the super string can have a different tinability for each segment.

If Att_Segment_Tinable_Value is set and Att_Segment_Tinable_Array not set, then the super string has the one tinability value to be used for all segments of the string.

See Super String Tinability Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Pipe/Culvert Dimensions

Att_Pipe_Justify 23

If Att_Pipe_Justify is set, then the super string has a justification for the pipe or culvert.

Att_Diameter_Value 5 **or** Att_Diameter_Array 6

If Att_Diameter_Array is set, then the super string is a round pipe has a diameter and wall thickness for each segment.

If Att_Diameter_Value is set and Att_Diameter_Array not set, then the super string is a round pipe has one diameter and one wall thickness value for the entire string.

Att_Culvert_Value 24 **or** Att_Culvert_Array 25

If Att_Culvert_Array is set, then the super string is a rectangular pipe (culvert) and has a width, height and top, bottom, left and right wall thicknesses for each segment.

If Att_Att_Culvert_Value is set and Att_Att_Culvert_Array not set, then the super string has one width, height, and top, bottom, left and right wall thicknesses for the entire string.

If none of the Pipe/Culvert dimensions exist, then the string is infinitesimally thin. Note that you **cannot** have both diameter dimensions and culvert dimensions.

Also having the Att_Pipe_Justify dimension by itself will do nothing. If Att_Pipe_Justify does not exist, the pipe/culvert are centreline based.

See Super String Pipe/Culvert Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Vertex Text Dimensions

Att_Vertex_Text_Value 10 **or** Att_Vertex_Text_Array 7

If Att_Vertex_Text_Array is set, then the super string can have different text at each vertex.

If Att_Vertex_Text_Value is set and Att_Vertex_Array not set, then the super string has the same text for each vertex of the string.

Note that it is possible to have text associated with a vertex but it is not visible on a plan view. To be able to draw the text on a plan view, see Vertex Text Annotation Dimensions .

See Super String Vertex Text and Annotation Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Vertex Text Annotation Dimensions

Att_Vertex_World_Annotate 30

Att_Vertex_Paper_Annotate 45

Att_Vertex_Annotate_Value 14 **or** Att_Vertex_Annotate_Array 15

If Att_Vertex_Annotate_Array is set, then the super string can have a different annotation for the text at each vertex.

If Att_Vertex_Annotate_Value is set and Att_Vertex_Annotate_Array not set, then the super string has the one annotation to be used for all text on all the vertices of the string.

If *Att_Vertex_World_Annotate* and Att_Vertex_Paper_Annotate do not exist, then the annotated text is device.

See Super String Vertex Text and Annotation Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Segment Text Dimensions

Att_Segment_Text_Value 22 **or** Att_Segment_Text_Array 8

If Att_Segment_Array is set, then the super string can have text for each segment.

If Att_Segment_Value is set and Att_Segment_Array not set, then the super string has the same text for each segment of the string.

Note that it is possible to have text associated with a segment but it is not visible. To be able to draw the text, see Segment Text Annotation Dimensions .

See Super String Segment Text and Annotation Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Segment Text Annotation Dimensions

Att_Segment_World_Annotate 31

Att_Segment_Paper_Annotate 46

Att_Segment_Annotate_Value 20 **or** Att_Segment_Annotate_Array 21

If Att_Segment_Annotate_Array is set, then the super string can have a different annotation for the text on each segment.

If Att_Segment_Annotate_Value is set and Att_Segment_Annotate_Array not set, then the super string has the one annotation to be used for all text on all the segments of the string.

If *Att_Segment_World_Annotate* and Att_Segment_Paper_Annotate do not exist, then the annotated text is device.

See Super String Segment Text and Annotation Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions

Att_Solid_Value 28

If Att_Solid_Value is set, then the super string can be filled with a solid colour.

Att_Bitmap_Value 29

If Att_Bitmap_Value is set, then the super string can be filled with a bitmap.

Att_Hatch_Value 27

If Att_Hatch_Value is set, then the super string can be filled with a hatch.

Att_Pattern_Value 33

If Att_Pattern_Value is set, then the super string can be filled with a 12d pattern.

Att_Autocad_Pattern_Value 54

If Att_Autocad_Pattern_Value is set, then the super string can be filled with an AutoCad pattern.

Note that all the Solid/Bitmap/Hatch/Pattern/Autocad_Pattern dimensions can exist. They are drawn in the order solid, bitmap, pattern, hatch and then Autocad pattern. Note that because the bitmap allows for transparency, it is possible to use one bitmap with a variety of different background colours.

See Super String Fills - Hatch/Solid/Bitmap/Pattern/ACAD Pattern Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Hole Dimension

Att_Hole_Value 26

If Att_Hole_Value is set, then the super string can have zero or more super strings as internal holes.

So it is possible to have a solid object like a horse shoe where the holes for the nails exist so that no filling occurs in the nail holes.

Note that the holes themselves may have their own solid/bitmap/hatch dimensions.

Warning, holes may not contain their own holes in the current implementation (that is, only one level of holes is allowed).

See Super String Hole Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

User Defined Vertex Attributes Dimensions

Att_Vertex_Attribute_Array 16

If Att_Vertex_Attribute_Array is set, then the super string can have a different Attributes at each vertex.

See Super String Vertex Attributes Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

User Defined Segment Attributes Dimensions

Att_Segment_Attribute_Array 19

If Att_Segment_Attribute_Array is set, then the super string can have a different Attributes on each segment

See Super String Segment Attributes Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Colour Dimension

Att_Colour_Array 9 LJG? For a colour for each segment (what about vertex?)

See Super String Segment Colour Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Vertex Image Dimensions

Att_Vertex_Image_Value 51 For an image at each vertex

Att_Vertex_Image_Array 52 For many images at each vertex

See Super String Vertex Image Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Segment Geometry Dimension

Att_Geom_Array 32 allow transitions for segments

If Att_Geom_Array is set, then each super string segment can be a line, arc, transition or offset transition.

See Super String Segment Geometry Functions *for calls to set/inquire on this dimension, and to load/retrieve data for this dimension.*

Visibility Dimensions

Att_Visible_Array 12 This dimension applies for both vertex and segment visibility.

Att_Vertex_Visible_Value 41 **or** Att_Vertex_Visible_Array 42

Att_Segment_Visible_Value 43 **or** Att_Segment_Visible_Array 44

See Super String Visibility Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Matrix Dimension

Att_Matrix_Value 53 ?

UID Dimensions

Att_Vertex_UID_Array 35

If Att_Vertex_Array is set, then the super string can have an Integer (referred to as a uid) stored at each vertex. This is mainly used by programmers to store a number on each vertex.

Att_Segment_UID_Array 36

If Att_Segment_UID_Array is set, then the super string can have an Integer (referred to as a uid) stored on each segment. This is mainly used by programmers to store a number on each segment.

See Super String Uid Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Database Point Dimensions

Att_Database_Point_Array 47

Extrude Dimensions

Att_Extrude_Value 48

If Att_Extrude_Value is set, then the super string can have zero or more extrudes on the string.

See Super String Extrude Functions *for calls to set/inquire on these dimensions, and to load/retrieve data for these dimensions.*

Null Levels Dimensions

// only used internally - not a normal dimension

Att_Null_Levels_Value 55

For information on setting flags to set more than one dimension at see, see Dimension Combinations and Super String Flags .

For information on creating super string using the dimension flags, see Basic Super String Functions .

Dimension Combinations and Super String Flags

There is a function call for each dimension to tell the super string to use that particular dimension and if more than one dimension is required, then simply call each function to set each of the required dimensions.

It is also possible to set one or many dimensions at once through one call by using a call with Integer **flags**.

An Integer is actually made up of 32-bits and each bit can be taken to mean that if the bit is 1 then a particular dimension is to be set (that is used) and 0 if it is not to be set.

So for example, 0 = binary 0 would mean no dimensions are to be used.
1 = binary 1 would mean only the first dimension is to be used
2 = binary 10 would mean only the second dimension is used
3 = binary 11 would mean the first and second dimensions only are used
4 = binary 100 would mean that only the third dimensions is used

So for the nth dimension to be set, you simply add 2 raised to the power n-1 to the Integer flag.

Because an Integer is only 32-bits, one Integer can only be used for thirty two (32) dimensions.

A second Integer is required to specify the dimensions 33 to a maximum of 64.

Since there is currently under 64 dimensions, then two Integer flags (flag1, flag2) can be used to set all the required dimensions on/off in the one call.

The following macros to help create the flags are defined in the include file “Setups.H”, as are all the Att_ dimension values.

```
#define concat(a,b) a##b
#define String_Super_Bit(n) (1 << concat(Att_,n))          // for dimensions 1 to 32
#define String_Super_Bit_Ex(n) (1 << concat(Att_,n) - 32)  // for dimensions 32 to 64
```

// So if **flag1** holds dimensions 1 to 32 (i.e. from Att_ZCoord_Value to Att_Geom_Array)

then the definition

```
Integer flag1 = String_Super_Bit(ZCoord_Value) | String_Super_Bit(Radius_Array);
```

means that **flag1** represents having the two dimensions Att_ZCoord_Value and Att_Radius_Array

// If **flag2** holds dimensions 33 to 64 (i.e. from Att_Pattern_Value to last current dimension)

then the definition

```
Integer flags2 = String_Super_Bit_Ex(Pattern_Value)
                 |String_Super_Bit_Ex(Vertex_Tinable_Array);
```

means that **flag2** represents having the two dimensions Att_Pattern_Value and Att_Vertex_Tinable_Array

Note that when using the String_Super_Bit and String_Super_Bit_Ex that you leave off the Att_ before the dimension names. The Att_ is automatically added by the #define.

As a code example, the code below defines a super string with independent heights at each vertex and the ability for arcs on each segment. This is the equivalent of the polyline string.

```
Integer flag1 = String_Super_Bit(ZCoord_Array) | String_Super_Bit(Radius_Array);
Integer flag2 = 0;      // no dimensions greater than 32
Integer npts = 100;
Element super = Create_super(flag1,flag2,npts);
```

For information on creating super string using the dimension flags, see Basic Super String Functions .

Basic Super String Functions

The super string can have a variable number of dimensions but it must have at least (x,y) values for every vertex.

There are functions to create a new super strings.

The create functions use dimension flags (or a seed super string) to specify how many vertices and what dimensions are created (if any).

Some of the super string create functions will also load (x,y,z,radius,bulge) data into the super string at creation time.

Once a super string is created, the other dimensions can be added using the *use* calls for that dimension, and the extra data for that dimension can then be loaded in. These calls are grouped together by super string dimension.

Also for an existing super string, there are calls to insert new vertices into the super string and to delete existing vertices.

See Super String Create Functions
See Inserting and Deleting Vertices
See Loading and Retrieving X, Y, Z, Radius and Bulge Data
See Getting Forward and Backward Vertex Direction
See Getting Super String Type and Type Like

For the calls for setting/inquiring for each dimension and for loading/retrieving data for each dimension:

See Super String Height Functions
See Super String Segment Colour Functions
See Super String Segment Radius Functions
See Super String Pipe/Culvert Functions
See Super String Pipe/Culvert Functions
See Super String Vertex Symbol Functions
See Super String Vertex Text and Annotation Functions
See Super String Segment Text and Annotation Functions
See Super String Tinability Functions
See Super String Point Id Functions
See Super String Fills - Hatch/Solid/Bitmap/Pattern/ACAD Pattern Functions
See Super String Hole Functions
See Super String Segment Geometry Functions
See Super String Extrude Functions
See Super String Vertex Attributes Functions
See Super String Segment Attributes Functions
See Super String Uid Functions
See Super String Vertex Image Functions
See Super String Visibility Functions

Super String Create Functions

Create_super(Integer flag1,Integer num_pts)

Name

Element Create_super(Integer flag1,Integer num_pts)

Description

Create an Element of type **Super** with room for **num_pts** vertices and **num_pts-1** segments if the string is not closed or **num_pts** segments if the string is closed.

flag1 is used to specify which of the dimensions from 1 to 32 are used/not used. See Super String Dimensions for the values that **flag1** may take.

The actual values of the arrays are set by other function calls after the string is created.

The return value is an Element handle to the created super string.

If the Super string could not be created, then the returned Element will be null.

Note - if dimensions greater than 32 are required, then calls with two flags must be used.

For example *Integer Create_super(Integer flag1, Integer flag2,Integer num_pts).*

ID = 691

Create_super(Integer flag1,Integer flag2,Integer npts)

Name

Element Create_super(Integer flag1,Integer flag2,Integer npts)

Description

create super string with arrays set aside following flag1 and flag 2 (extended dimensions).

Create an Element of type **Super** with room for **num_pts** vertices and **num_pts-1** segments if the string is not closed or **num_pts** segments if the string is closed.

flag1 is used to specify which of the dimensions from 1 to 32 are used/not used.
flag2 is used to specify which of the dimensions from 33 to 64 are used/not used.

See Super String Dimensions for the values that **flag1** and **flag2** may take.

The actual values of the arrays are set by other function calls after the string is created.

The return value is an Element handle to the created super string.

If the Super string could not be created, then the returned Element will be null.

ID = 1499

Create_super(Integer num_pts,Element seed)

Name

Element Create_super(Integer num_pts,Element seed)

Description

Create an Element of type **Super** with room for **num_pts** vertices and **num_pts-1** segments if the string is not closed or **num_pts** segments if the string is closed.

Set the colour, name, style, flags etc. of the new string to be the same as those from the Element **seed.** Note that the seed string must also be a super string.

The actual values of the arrays are set after the string is created.

The return value is an Element handle to the created super string.

If the Super string could not be created, then the returned Element will be null.

ID = 692

Create_super(Integer flag1,Segment seg)

Name

Element Create_super(Integer flag1,Segment seg)

Description

Create an Element of type **Super** with two vertices if **seg** is a Line, Arc or Spiral, or one vertex if **seg** is a Point. The co-ordinates for the one or two vertices are taken from **seg**.

flag1 is used to specify which of the dimensions from 1 to 32 are used/not used. See Super String Dimensions for the values that **flag1** may take.

LJG? if seg is an Arc or a Spiral, then what dimensions are set and what values are they given?

The return value is an Element handle to the created super string.

If the Super string could not be created, then the returned Element will be null.

Note - if dimensions greater than 32 are required, then calls with two flags must be used.

For example *Integer Create_super(Integer flag1, Integer flag2,Segment seg).*

ID = 693

Create_super(Integer flag1,Integer flag2,Segment seg)

Name

Element Create_super(Integer flag1,Integer flag2,Segment seg)

Description

Create an Element of type **Super** with two vertices if **seg** is a Line, Arc or Spiral, or one vertex if **seg** is a Point. The co-ordinates for the one or two vertices are taken from **seg**.

flag1 is used to specify which of the dimensions from 1 to 32 are used/not used.
flag2 is used to specify which of the dimensions from 33 to 64 are used/not used.

See Super String Dimensions for the values that **flag1** and **flag2** may take.

LJG? if seg is an Arc or a Spiral, then what dimensions are set and what values are they given?

The return value is an Element handle to the created super string.

If the Super string could not be created, then the returned Element will be null.

ID = 1500

Create_super(Integer flag1,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_pts)

Name

Element Create_super(Integer flag1,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_pts)

Description

Create an Element of type **Super** with **num_pts** vertices.

The basic geometry for the super string is supplied by the arrays **x** (x values)**, y** (y values)**, z** (z values)**, r** (radius of segments)**, b** (bulge of segment b = 1 for major arc > 180 degrees, b = 0 for minor arc < 180 degrees).

flag1 is used to specify which of the dimensions from 1 to 32 are used/not used.

Note that depending on the **flag1** value, the **z, r, b** arrays may or may not be used, but the arrays must still be supplied. See Super String Dimensions for the values that **flag1** may take.

The arrays must be of length **num_pts** or greater.

The function return value is an Element handle to the created super string.

If the Super string could not be created, then the returned Element will be null.

Note - if dimensions greater than 32 are required, then calls with two flags must be used.

For example *Integer Create_super(Integer flag1, Integer flag2,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_pts).*

ID = 690

Create_super(Integer flag1,Integer flag2,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_pts)

Name

Element Create_super(Integer flag1,Integer flag2,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_pts)

Description

Create an Element of type **Super** with **num_pts** vertices.

The basic geometry for the super string is supplied by the arrays **x** (x values)**, y** (y values)**, z** (z values)**, r** (radius of segments)**, b** (bulge of segment b = 1 for major arc > 180 degrees, b = 0 for minor arc < 180 degrees).

flag1 is used to specify which of the dimensions from 1 to 32 are used/not used.
flag2 is used to specify which of the dimensions from 33 to 64 are used/not used.

Note that depending on the **flag1** value, the **z, r, b** arrays may or may not be used, but the arrays must still be supplied. See Super String Dimensions for the values that **flag1** and **flag2** may take.

The arrays must be of length **num_pts** or greater.

The function return value is an Element handle to the created super string.

If the Super string could not be created, then the returned Element will be null.

ID = 1498

Inserting and Deleting Vertices

Super_insert_vertex(Element super,Integer where,Integer count)

Name

Integer Super_insert_vertex(Element super,Integer where,Integer count)

Description

For the super string **super**, insert **count** new vertices BEFORE vertex index **where**.

All the existing vertices from index position **where** onwards are move to after the new **count** inserted vertices.

For example, Super_insert_vertex(super,1,10) will insert 10 new vertices before vertex index 1, and all the existing vertices will be moved to after vertex index 10.

Note that if the string is a closed string then the closure applies to the new last vertex.

If the Element **super** is not of type **Super**, then the function return value is set to a non zero value.

A return value of 0 indicates the function call was successful.

ID = 2168

Super_remove_vertex(Element super,Integer where,Integer count)

Name

Integer Super_remove_vertex(Element super,Integer where,Integer count)

Description

For the super string **super**, delete **count** existing vertices starting at vertex index **where**.

If there are not enough vertices to delete then the delete stops at the last vertex of the super string.

Note that if the string is closed then the closure applies to the new last vertex.

If the Element **super** is not of type **Super**, then the function return value is set to a non zero value.

A return value of 0 indicates the function call was successful.

ID = 2169

Loading and Retrieving X, Y, Z, Radius and Bulge Data

Set_super_vertex_coord(Element super,Integer i,Real x,Real y,Real z)

Name

Integer Set_super_vertex_coord(Element super,Integer i,Real x,Real y,Real z)

Description

Set the coordinate data (x,y,z) for the **i**'th vertex (the vertex with index number **i**) of the super Element **super** where

the x value to set is in Real **x**.
the y value to set is in Real **y**.
the z value to set is in Real **z**.

If the Element **super** is not of type **Super**, then the function return value is set to a non zero value.

A function return value of zero indicates the data was successfully set.

ID = 732

Get_super_vertex_coord(Element super,Integer i,Real &x,Real &y,Real &z)

Name

Integer Get_super_vertex_coord(Element super,Integer i,Real &x,Real &y,Real &z)

Description

Get the coordinate data (x,y,z) for the **i**'th vertex (the vertex with index number **i**) of the super Element **super**.

The x coordinate is returned in Real **x**.
The y coordinate is returned in Real **y**.
The z coordinate is returned in Real **z**.

If the Element **super** is not of type **Super**, then the function return value is set to a non zero value.

A return value of 0 indicates the function call was successful.

ID = 733

Set_super_data(Element super,Integer i,Real x,Real y,Real z,Real r,Integer b)

Name

Integer Set_super_data(Element super,Integer i,Real x,Real y,Real z,Real r,Integer b)

Description

Set the (x,y,z,r,f) data for the **i**'th vertex of the super Element **super** where

the x value to set is the Real **x**.

the y value to set is the Real **y**.
the z value to set is the Real **z**.
the radius value to set is the Real **r**.
the major/minor arc bulge value to set is the Integer **b (**0 for minor arc < 180 degrees, non zero for major arc > 180 degrees).

If the Element **super** is not of type **Super**, then the function return value is set to a non zero value.

A function return value of zero indicates the data was successfully set.

ID = 699

Get_super_data(Element super,Integer i,Real &x,Real &y,Real &z,Real &r, Integer &b)

Name

Integer Get_super_data(Element super,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &b

Description

Get the (x,y,z,r,b data for the **i'**th vertex of the super string **super**.

The x value is returned in Real **x**.
The y value is returned in Real **y**.
The z value is returned in Real **z**.
The radius value is returned in Real **r**.
The major/minor arc bulge value is returned in Integer **b**.(bulge of segment b = 1 for major arc > 180 degrees, b = 0 for minor arc < 180 degrees).

If the Element **super** is not of type **Super**, then the function return value is set to a non zero value.

A function return value of zero indicates the data was successfully returned.

ID = 696

Set_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer b[], Integer num_pts)

Name

Integer Set_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer b[], Integer num_pts)

Description

Set the (x,y,z,r,b) data for the first **num_pts** vertices of the string Element **super**.

This function allows the user to modify a large number of vertices of the string in one call.

The maximum number of vertices that can be set is given by the number of vertices in the string.

The (x,y,z,r,f) values for each string vertex are given in the Real arrays **x[]**, **y[],z[],r[]** and Integer array **b[]** where the (x,y,z) are coordinate, r the radius of the arc on the following segment and b is the bulge to say whether the arc is a major or minor arc (bulge of segment b = 1 for major arc > 180 degrees, b = 0 for minor arc < 180 degrees).

The number of vertices to be set is given by Integer **num_pts**

If the Element **super** is not of type **Super**, then nothing is modified and the function return value is set to a non zero value.

Note: this function can not create new super Elements but only modify existing super Elements.

A function return value of zero indicates the data was set successfully.

ID = 697

Get_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer max_pts,Integer &num_pts)

Name

Integer Get_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z,r,f) data for the first **max_pts** vertices of the super string Element **super**.

The (x,y,z,r,f) values at each string vertex are returned in the Real arrays **x[]**, **y[],z[],r[]** and Integer array **b[]** (the arrays are x values**,** y values**,** z values**,** radius of segments**, b** is bulge to denote if the segment is major or minor arc (bulge of segment b = 1 for major arc > 180 degrees, b = 0 for minor arc < 180 degrees).

The maximum number of vertices that can be returned is given by max_pts (usually the size of the arrays).

The vertex data returned starts at the first vertex and goes up to the minimum of max_pts and the number of vertices in the string.

The actual number of vertices returned is returned by Integer **num_pts**

num_pts <= max_pts

If the Element **super** is not of type **Super**, then num_pts is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

ID = 694

Set_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_pts,Integer start_pt)

Name

Integer Set_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_pts,Integer start_pt)

Description

For the super Element **super**, set the (x,y,z,r,b) data for num_pts vertices, starting at vertex number **start_pt**.

This function allows the user to modify a large number of vertices of the string in one call starting at vertex

number **start_pt** rather than vertex one.

The maximum number of vertices that can be set is given by the difference between the number of vertices in the string and the value of **start_pt**.

The (x,y,z,r,f) values for the string vertices are given in the Real arrays **x[]**, **y[],z[],r[]** and **b[]** where the (x,y,z) are coordinate, r the radius of the arc on the following segment and b is the bulge to say whether the arc is a major or minor arc (bulge of segment b = 1 for major arc > 180 degrees, b = 0 for minor arc < 180 degrees).

The number of the first string vertex to be modified is **start_pt**.

The total number of vertices to be set is given by Integer **num_pts**

If the Element **super** is not of type **Super**, then nothing is modified and the function return value is set to a non zero value.

A function return value of zero indicates the data was set successfully.

Notes

(a) A start_pt of one gives the same result as the previous function.

(b) This function **can not** create new super strings but only modify existing super strings.

ID = 698

Get_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer b[], Integer max_pts,Integer &num_pts,Integer start_pt)

Name

Integer Get_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer b[], Integer max_pts,Integer &num_pts,Integer start_pt)

Description

For a super string Element **super**, get the (x,y,z,r,b) data for **max_pts** vertices starting at vertex number **start_pt** (the arrays are x values**,** y values**,** z values**,** radius of segments**, b** is if segment is major or minor arc).

This routine allows the user to return the data from a super string in user specified chunks. This is necessary if the number of vertices in the string is greater than the size of the arrays available to contain the information.

As in the previous function, the maximum number of vertices that can be returned is given by **max_pts** (usually the size of the arrays).

However, for this function, the vertex data returned starts at vertex number **start_pt** rather than vertex one.

The (x,y,z,r,b) values at each string vertex are returned in the Real arrays **x[]**, **y[],z[],r[]** and Integer array **b[]**.

The actual number of vertices returned is given by Integer **num_pts**

num_pts <= max_pts

If the Element **super** is not of type **Super**, then **num_pts** is set to zero and the function return value is set to a non zero value.

Note A start_pt of one gives the same result as for the previous function.

A function return value of zero indicates the data was successfully returned.

ID = 695

Getting Forward and Backward Vertex Direction

Get_super_vertex_forward_direction(Element super,Integer vert,Real &ang)

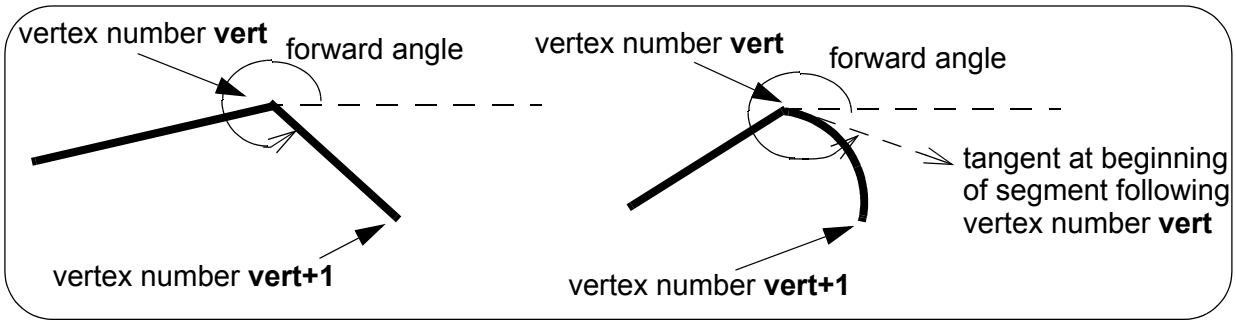
Name

Integer Get_super_vertex_forward_direction(Element super,Integer vert,Real &ang)

Description

For the Element **super** of type **Super**, get the angle of the tangent at the *beginning* of the segment *leaving* vertex number **vert**. That is, the segment going from vertex **vert** to vertex **vert**+1. Return the angle in **ang**.

ang is in radians and is measured in a counterclockwise direction from the positive x-axis.



If the super string is closed, the angle will still be valid for the last vertex of the super string and it is the angle of the closing segment between the last vertex and the first vertex.

If super string is open, the call fails for the last vertex and a non-zero return code is returned.

If the Element **super** is not of type **Super,** then a non-zero return code is returned

A function return value of zero indicates the angle was successfully returned.

ID = 1501

Get_super_vertex_backward_direction(Element super,Integer vert,Real &ang)

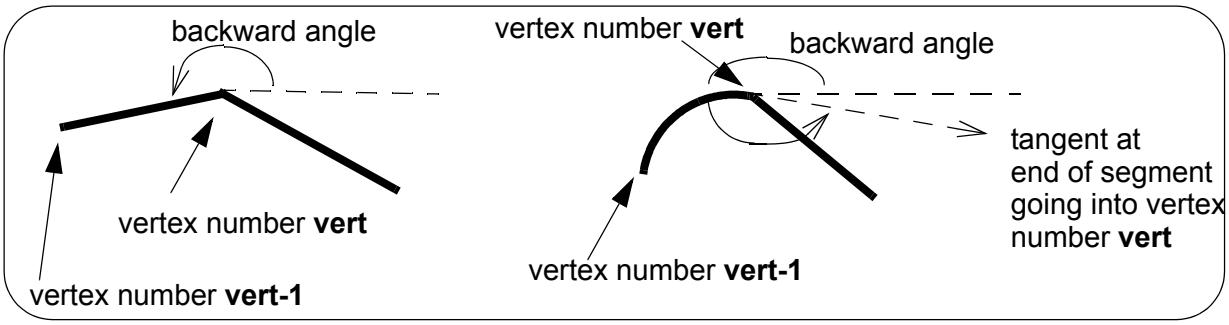
Name

Integer Get_super_vertex_backward_direction(Element super,Integer vert,Real &ang)

Description

For the Element **super** of type **Super**, get the angle of the tangent at the *end* of the segment *entering* vertex number **vert**. That is, the segment going from vertex **vert-1** to vertex **vert.** Return the angle in **ang.**

ang is in radians and is measured in a counterclockwise direction from the positive x-axis.



If the super string is closed, the angle will still be valid for the first vertex of the super string and it is the angle of the closing segment between the first vertex and the last vertex.

If super string is open, the call fails for the first vertex and a non-zero return code is returned.

If the Element **super** is not of type **Super,** then a non-zero return code is returned

A function return value of zero indicates the angle was successfully returned.

ID = 1502

Getting Super String Type and Type Like

Get_type_like(Element super,Integer &type)

Name

Integer Get_type_like(Element super,Integer &type)

Description

In earlier versions of 12d Model, there were a large number of string types but in later versions of 12d Model, the super string was introduced which with its possible dimensions, could replace many of the other strings.

However, for some applications it was important to know if the super string was like one of the original strings. For example, some options required a string to be a contours string, the original 2d string. That is, the string has the one z-value (or height) for the entire string. So a super string that has a constant dimension for height, behaves like a 2d string and in that case will return the **Type Like** of **2d**.

The **Type Like's** can be referred to by a number (*Integer)* or by text (*Text)*.

See Types of Elements for the values of the Type Like numbers and Type Like text.

The Type Like for the super string is returned in **type**.

If the Element **string** is not a super string, then a non zero function return value is returned.

A function return value of zero indicates the Type Like was returned successfully.

ID = 2074

Get_type_like(Element elt,Text &type)

Name

Integer Get_type_like(Element elt,Text &type)

Description

In earlier versions of 12d Model, there were a large number of string types but in later versions of 12d Model, the super string was introduced which with its possible dimensions, could replace many of the other strings.

However, for some applications it was important to know if the super string was like one of the original strings. For example, some options required a string to be a contours string, the original 2d string. That is, the string has the one z-value (or height) for the entire string. So a super string that has a constant dimension for height, behaves like a 2d string and in that case will return the **Type Like** of **2d**.

The **Type Like's** can be referred to by a number (*Integer)* or by text (*Text)*.

See Types of Elements for the values of the Type Like numbers and Type Like text.

The Text Type Like for the super string is returned in **type**.

If the Element **string** is not a super string, then a non zero function return value is returned.

A function return value of zero indicates the Type Like was returned successfully.

ID = 2075

Super String Height Functions

For definitions of the height dimensions, see Height Dimensions

See Super String Use Height Functions
See Setting Super String Height Values

Super String Use Height Functions

Set_super_use_2d_level(Element super,Integer use)

Name

Integer Set_super_use_2d_level(Element super,Integer use)

Description

For the super string Element **super**, define whether the height dimension Att_ZCoord_Value is used or removed.

See Height Dimensions for information on Height dimensions or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. If **use** is 0, the dimension Att_ZCoord_Value is removed.

Note that if the height dimension Att_ZCoord_Array exists, this call is ignored.

If the Element **super** is not a super string, then a non zero function return value is returned.

A return value of 0 indicates the function call was successful.

ID = 700

Get_super_use_2d_level(Element super,Integer &use)

Name

Integer Get_super_use_2d_level(Element super,Integer &use)

Description

Query whether the dimension height dimension Att_ZCoord_Value exists for the super string **super**.

See Height Dimensions for information on Height dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists, or 0 if the dimension doesn't exist.

If the Element **super** is not a super string, then a non zero function return value is returned.

A return value of 0 indicates the function call was successful.

ID = 701

Set_super_use_3d_level(Element super,Integer use)

Name

Integer Set_super_use_3d_level(Element super,Integer use)

Description

For the super string Element **super**, define whether the height dimension Att_ZCoord_Array is used or removed.

See Height Dimensions for information on Height dimensions or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. If **use** is 0, the dimension Att_ZCoord_Array is removed.

If the Element **super** is not a super string, then a non zero function return value is returned.

A return value of 0 indicates the function call was successful.

ID = 730

Get_super_use_3d_level(Element super,Integer &use)

Name

Integer Get_super_use_3d_level(Element super,Integer &use)

Description

Query whether the height dimension Att_ZCoord_Array exists for the super string **super**.

See Height Dimensions for information on Height dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists, or 0 if the dimension doesn't exist.

If the Element **super** is not a super string, then a non zero function return value is returned.

A return value of 0 indicates the function call was successful.

ID = 731

Super_vertex_level_value_to_array(Element super)

Name

Integer Super_vertex_level_value_to_array(Element super)

Description

If for the super string **super** the dimension Att_ZCoord_Value exists and the dimension Att_ZCoord_Array does not exist then there will be one z value **zval** (height or level) for the entire string.

In this case (when the dimension Att_ZCoord_Value exists and the dimension Att_ZCoord_Array does not exist) this function sets the Att_ZCoord_Array dimension and creates a new z-value for each vertex of **super** and it is given the value **zval**.

See Height Dimensions for information on the Height (ZCoord) dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 2174

Setting Super String Height Values

Get_super_2d_level(Element elt,Real &level)

Name

Integer Get_super_2d_level(Element elt,Real &level)

Description

For the Element **elt**, if the height dimension Att_ZCoord_Value is set and Att_ZCoord_Array is not set, then the z-value for the entire string is returned in **level**.

See Height Dimensions for information on Height dimensions or Super String Dimensions for information on all dimensions.

If the Element **elt** is not of type **Super**, or the dimension Att_ZCoord_Value is not set, this call fails and a non zero return value is returned.

A return value of zero indicates the function call was successful.

ID = 703

Set_super_2d_level(Element elt,Real level)

Name

Integer Set_super_2d_level(Element elt,Real level)

Description

For the Element **elt** of type **Super**, if the dimension Att_ZCoord_Value is set and Att_ZCoord_Array is not set, then the z-value for the entire string is set to **level**.

See Height Dimensions for information on Height dimensions or Super String Dimensions for information on all dimensions.

If the Element **elt** is not of type **Super**, or the dimension Att_ZCoord_Value is not set, this call fails and a non zero return value is returned.

A return value of zero indicates the function call was successful.

ID = 702

Super String Tinability Functions

For definitions of the Tinability dimension, see Tinability Dimensions

See Super String Combined Tinability
See Super String Vertex Tinability
See Super String Segment Tinability

Super String Combined Tinability

Set_super_use_tinability(Element super,Integer use)

Name

Integer Set_super_use_tinability(Element super,Integer use)

Description

Tell the super string whether to use the dimension Att_Contour_Array.

LJG?

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

A return value of 0 indicates the function call was successful.

ID = 722

Get_super_use_tinability(Element super,Integer &use)

Name

Integer Get_super_use_tinability(Element super,Integer &use)

Description

Query whether the dimension Att_Contour_Array exists for the super string.

LJG?

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 723

Super String Vertex Tinability

Set_super_use_vertex_tinability_value(Element super,Integer use)

Name

Integer Set_super_use_vertex_tinability_value(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Vertex_Tinable_Value is used or removed.

If Att_Vertex_Tinable_Value is set and Att_Vertex_Tinability_Array is not set then the tinability is the same for all vertices of **super**.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and the tinability is the same for **all** vertices.
If **use** is 0, the dimension is removed.

Note that if the dimension Att_Vertex_Tinable_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

ID = 1584

Get_super_use_vertex_tinability_value(Element super,Integer &use)

Name

Integer Get_super_use_vertex_tinability_value(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Tinable_Value exists for the super string **super**.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1585

Set_super_use_vertex_tinability_array(Element super,Integer use)

Name

Integer Set_super_use_vertex_tinability_array(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Vertex_Tinable_Array is used.

If Att_Vertex_Tinable_Array is set then there can be a different tinability defined for each vertex of **super**.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and the tinability is different for each vertex.
If **use** is 0, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 1586

Get_super_use_vertex_tinability_array(Element super,Integer &use)

Name

Integer Get_super_use_vertex_tinability_array(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Tinable_Array exists for the super string **super**.

See Tinability Dimensions for information on the Tinability dimensions or Super String

Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1587

Set_super_vertex_tinability(Element super,Integer vert,Integer tinability)

Name

Integer Set_super_vertex_tinability(Element super,Integer vert,Integer tinability)

Description

For the Element **super** (which must be of type **Super**), set the tinability value for vertex number **vert** to the value **tinability**.

If **tinability** is 1, the vertex is tinable.
If **tinability** is 0, the vertex is not tinable.

If the Element **super** is not of type **Super,** or Att_Vertex_Tinable_Array is not set for **super**, then a non-zero return code is returned.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 736

Get_super_vertex_tinability(Element super,Integer vert,Integer &tinability)

Name

Integer Get_super_vertex_tinability(Element super,Integer vert,Integer &tinability)

Description

For the Element **super** (which must be of type **Super**), get the tinability value for vertex number **vert** and return it in the Integer **tinability**.

If **tinability** is 1, the vertex is tinable.
If **tinability** is 0, the vertex is not tinable.

If the Element **super** is not of type **Super,** or Att_Vertex_Tinable_Array is not set for **super**, then a non-zero return code is returned.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 737

Super String Segment Tinability

Set_super_use_segment_tinability_value(Element super,Integer use)

Name

Integer Set_super_use_segment_tinability_value(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Segment_Tinable_Value is

used or removed.

If Att_Segment_Tinable_Value is set and Att_Segment_Tinability_Array is not set then the tinability is the same for all segments of **super**.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and the tinability is the same for **all** segments.
If **use** is 0, the dimension is removed.

Note that if the dimension Att_Segment_Tinable_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

ID = 1592

Get_super_use_segment_tinability_value(Element super,Integer &use)

Name

Integer Get_super_use_segment_tinability_value(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Tinable_Value exists for the super string **super**.

If Att_Segment_Tinable_Value is set and Att_Segment_Tinability_Array is not set then the tinability is the same for all segments of **super**.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1593

Set_super_use_segment_tinability_array(Element super,Integer use)

Name

Integer Set_super_use_segment_tinability_array(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Segment_Tinable_Array is set or removed.

If Att_Segment_Tinable_Array is set then there can be a different tinability defined for each segment in **super**.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and the tinability is different for each segment.
If **use** is 0, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 1594

Get_super_use_segment_tinability_array(Element super,Integer &use)

Name

Integer Get_super_use_segment_tinability_array(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Tinable_Array exists for the super string **super**.

If Att_Segment_Tinable_Array is set then there can be a different tinability defined for each segment in **super**.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1595

Set_super_segment_tinability(Element super,Integer seg,Integer tinability)

Name

Integer Set_super_segment_tinability(Element super,Integer seg,Integer tinability)

Description

For the Element **super** (which must be of type **Super**), set the tinability value for segment number **seg** to the value **tinability**.

If **tinability** is 1, the segment is tinable.
If **tinability** is 0, the segment is not tinable.

If the Element **super** is not of type **Super,** or Att_Segment_Tinable_Array is not set for **super**, then a non-zero return code is returned.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 724

Get_super_segment_tinability(Element super,Integer seg,Integer &tinability)

Name

Integer Get_super_segment_tinability(Element super,Integer seg,Integer &tinability)

Description

For the Element **super** (which must be of type **Super**), get the tinability value for segment number **seg** and return it in the Integer **tinability**.

If **tinability** is 1, the segment is tinable.
If **tinability** is 0, the segment is not tinable.

If the Element **super** is not of type **Super,** or Att_Segment_Tinable_Array is not set for **super**, then a non-zero return code is returned.

See Tinability Dimensions for information on the Tinability dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 725

Super String Segment Radius Functions

For definitions of the Segment Radius dimensions, see Segment Radius Dimension

Set_super_use_segment_radius(Element super,Integer use)

Name

Integer Set_super_use_segment_radius(Element super,Integer use)

Description

For the super string Element **super**, define whether the segment radius dimension Att_Radius_Array is to be used or removed.

See Segment Radius Dimension for information on the Segment Radius dimensions or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the segments between vertices of the **super** can be straights or arcs.

If **use** is 0, the dimension is removed. That is, the segments between vertices of the **super** can only be straights.

Note that if the dimension Att_Radius_Array is set then the Att_Major_Array is also automatically set.

A return value of 0 indicates the function call was successful.

ID = 708

Get_super_use_segment_radius(Element super,Integer &use)

Name

Integer Get_super_use_segment_radius(Element super,Integer &use)

Description

Query whether the segment radius dimension Att_Radius_Array exists for the super string.

use is returned as 1 if the dimension Att_Radius_Array exists, or 0 if the dimension doesn't exist.

See Segment Radius Dimension for information on the Segment Radius dimensions or Super String Dimensions for information on all dimensions.

A return value of 0 indicates the function call was successful.

ID = 709

Set_super_segment_radius(Element super,Integer seg,Real rad)

Name

Integer Set_super_segment_radius(Element super,Integer seg,Real rad)

Description

For the super string **super**, set the radius of segment number **seg** to the value **rad**.

See Segment Radius Dimension for information on the Segment Radius dimensions or Super String Dimensions for information on all dimensions.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Radius_Array set.

A return value of 0 indicates the function call was successful.

ID = 710

Get_super_segment_radius(Element super,Integer seg,Real &rad)

Name

Integer Get_super_segment_radius(Element super,Integer seg,Real &rad)

Description

For the super string **super**, get the radius of segment number **seg** and return the radius in **rad**.

See Segment Radius Dimension for information on the Segment Radius dimensions or Super String Dimensions for information on all dimensions.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Radius_Array set.

A return value of 0 indicates the function call was successful.

ID = 711

Set_super_segment_major(Element super,Integer seg,Integer bulge)

Name

Integer Set_super_segment_major(Element super,Integer seg,Integer bulge)

Description

For the super string **super**, set the major/minor arc value of segment number **seg** to the value **bulge**. (bulge of segment b = 1 for major arc > 180 degrees, b = 0 for minor arc < 180 degrees)

See Segment Radius Dimension for information on the Segment Radius dimensions or Super String Dimensions for information on all dimensions.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Major_Array set.

A return value of 0 indicates the function call was successful.

ID = 712

Get_super_segment_major(Element super,Integer seg,Integer &bulge)

Name

Integer Get_super_segment_major(Element super,Integer seg,Integer &major)

Description

For the super string **super**, get the major/minor arc bulge of segment number **seg** and return the value in **bulge** (bulge of segment bulge = 1 for major arc > 180 degrees, bulge = 0 for minor arc < 180 degrees).

See Segment Radius Dimension for information on the Segment Radius dimensions or Super String Dimensions for information on all dimensions.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Major_Array set.

A return value of 0 indicates the function call was successful.

ID = 713

Super String Point Id Functions

For definitions of the Point Id dimension, see Point Id Dimension

Set_super_use_vertex_point_number(Element super,Integer use)

Name

Integer Set_super_use_vertex_point_number(Element super,Integer use)

Description

Tell the super string whether to use, remove, the dimension Att_Point_Array.

If Att_Point_Array exists, the string can have a Point Id for each vertex.

If **use** is 1, the dimension is set and each vertex can have a Point Id.
If **use** is 0, the dimension is removed.

See Point Id Dimension for information on the Point Id dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 738

Get_super_use_vertex_point_number(Element super,Integer &use)

Name

Integer Get_super_use_vertex_point_number(Element super,Integer &use)

Description

Query whether the dimension Att_Point_Array exists for the super string.

If Att_Point_Array exists, the string can have a Point Id for each vertex.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

See Point Id Dimension for information on the Point Id dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 739

Set_super_vertex_point_number(Element super,Integer vert,Integer point_number)

Name

Integer Set_super_vertex_point_number(Element super,Integer vert,Integer point_number)

Description

For the Element **super** which must be of type **Super**, set the Point Id for vertex number **vert** to the have the text value of the integer **point_number**.

If the Element **super** is not of type **Super,** or the dimension Att_Point_Array is not set, then a non-zero return code is returned.

See Point Id Dimension for information on the Point Id dimensions or Super String Dimensions for information on all the dimensions.

Note - in earlier versions of 12d Model (pre v6), point id's were only integers. This was extended to being a text when surveying equipment allowed non-integer point ids.

A function return value of zero indicates the point id was successfully set.

ID = 740

Get_super_vertex_point_number(Element super,Integer vert,Integer &point_number)

Name

Integer Get_super_vertex_point_number(Element super,Integer vert,Integer &point_number)

Description

This function should no longer be used because now Point Id's do not have to be integers.

From the Element **super** which must be of type **Super**, get the Point Id for vertex number **vert** and return it in the Integer **point_number**.

If the Element **super** is not of type **Super,** or the dimension Att_Point_Array is not set for **super**, then a non-zero return code is returned.

See Point Id Dimension for information on the Point Id dimensions or Super String Dimensions for information on all the dimensions.

Note - in earlier versions of 12d Model (pre v6), Point Id's were only integers. This was extended to being a text when surveying equipment allowed non-integer Point Ids.

A function return value of zero indicates the point id was successfully returned.

ID = 741

Set_super_vertex_point_number(Element super,Integer vert,Text point_id

Name

Integer Set_super_vertex_point_number(Element super,Integer vert,Text point_id)

Description

For the Element **super** which must be of type **Super**, set the Point Id for vertex number **vert** to the text **point_id**.

If the Element **super** is not of type **Super,** or the dimension Att_Point_Array is not set, then a non-zero return code is returned.

See Point Id Dimension for information on the Point Id dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the point id was successfully set.

ID = 1625

Get_super_vertex_point_number(Element super,Integer vert,Text &point_id)

Name

Integer Get_super_vertex_point_number(Element super,Integer vert,Text &point_id)

Description

From the Element **super** which must be of type **Super**, get the Point Id for vertex number **vert** and return it in the Text **point_id**.

If the Element **super** is not of type **Super,** or the dimension Att_Point_Array is not set for **super**, then a non-zero return code is returned.

See Point Id Dimension for information on the Point Id dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the point id was successfully returned.

ID = 1626

Super String Vertex Symbol Functions

For definitions of the Vertex Symbols dimensions, see Vertex Symbol Dimensions

See Definitions of Super String Vertex Symbol Dimensions and Parameters
See Super String Use Vertex Symbol Functions
See Setting Super String Vertex Symbol Parameters

Definitions of Super String Vertex Symbol Dimensions and Parameters

Symbols can be placed on vertices of a super string.

The displayed symbol is defined by

(a) the position of the super string vertex

(b) the symbol name

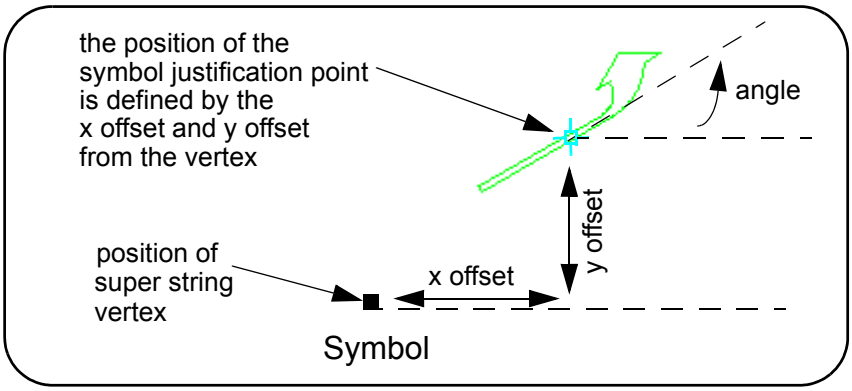
(c) angle of rotation of the symbol

(d) defining what is known as the ***symbol justification point*** in relation to the vertex

For symbols, the ***symbol justification point*** and the ***angle of the symbol*** are defined by:

(a) the *symbol justification point* is given as an ***x offset*** and a ***y offset*** from the vertex

(b) the *angle of the symbol* is given as a *counter clockwise* ***angle*** *of rotation* (measured from the x-axis) about the symbol justification point.



The vertex and justification point only coincide if the x offset and y offset values are both zero.

Super String Use Vertex Symbol Functions

Set_super_use_symbol(Element super,Integer use)

Name

Integer Set_super_use_symbol(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the vertex symbol dimension Att_Symbol_Value is used or removed.

See Vertex Symbol Dimensions for information on the Vertex Symbol dimensions or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the super string has **one** symbol for all vertices.
If **use** is **0**, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 797

Get_super_use_symbol(Element super,Integer &use)

Name

Integer Get_super_use_symbol(Element super,Integer &use)

Description

Query whether the vertex symbol dimension Att_Symbol_Value exists for the Element **super** of type **Super**.

See Vertex Symbol Dimensions for information on the Vertex Symbol dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists. That is, the super string has one symbol for all vertices.

use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 798

Set_super_use_vertex_symbol(Element super,Integer use)

Name

Integer Set_super_use_vertex_symbol(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the vertex symbol dimension Att_Symbol_Array is used or removed.

See Vertex Symbol Dimensions for information on the Vertex Symbol dimensions or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the super string has a **different** symbol on each vertex. If **use** is **0**, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 799

Get_super_use_vertex_symbol(Element super,Integer &use)

Name

Integer Get_super_use_vertex_symbol(Element super,Integer &use)

Description

Query whether the vertex symbol dimension Att_Symbol_Array exists for the super string.

See Vertex Symbol Dimensions for information on the Vertex Symbol dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists. That is, the super string has a **different** symbol on each vertex.

use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 800

Super_vertex_symbol_value_to_array(Element super)

Name

Integer Super_vertex_symbol_value_to_array(Element super)

Description

If for the super string **super** the dimension Att_Symbol_Value exists and the dimension Att_Symbol_Array does not exist then there will be one z value **zval** (height or level) for the entire string.

In this case (when the dimension Att_Symbol_Value exists and the dimension Att_Symbol_Array does not exist) this function sets the Att_Symbol_Array dimension and creates a new array for symbol at each vertex of **super.**

See Vertex Symbol Dimensions for information on the Height (ZCoord) dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 2175

Setting Super String Vertex Symbol Parameters

Set_super_vertex_symbol_style(Element super,Integer vert,Text sym)

Name

Integer Set_super_vertex_symbol_style(Element super,Integer vert,Text sym)

Description

For the super Element **super**, set the symbol on vertex number **vert** to be the symbol style named **sym**.

If there is only the one Symbol for the entire string then the symbol name for that symbol is set to **sym** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 801

Get_super_vertex_symbol_style(Element super,Integer vert,Text &sym)

Name

Integer Get_super_vertex_symbol_style(Element super,Integer vert,Text &s)

Description

For the super Element **super**, return the name of the symbol on vertex number **vert** in Text **sym**.

If there is only the one Symbol for the entire string then the symbol name for that symbol is returned in **sym** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 802

Set_super_vertex_symbol_colour(Element super,Integer vert,Integer col)

Name

Integer Set_super_vertex_symbol_colour(Element super,Integer vert,Integer col)

Description

For the super Element **super**, set the colour number of the symbol from the vertex number **vert** to be **col**.

If there is only the one Symbol for the entire string then the colour number of that symbol is set to **col** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 807

Get_super_vertex_symbol_colour(Element super,Integer vert,Integer &col)

Name

Integer Get_super_vertex_symbol_colour(Element super,Integer vert,Integer &col)

Description

For the super Element **super**, return as **col** the colour number of the symbol on vertex number **vert**.

If there is only the one Symbol for the entire string then the colour number of that symbol is returned in **col** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 808

Set_super_vertex_symbol_offset_width(Element super,Integer vert,Real x_offset)

Name

Integer Set_super_vertex_symbol_offset_width(Element super,Integer vert,Real x_offset)

Description

For the super Element **super**, set the x offset of the symbol from vertex number **vert** to be **x_offset**.

If there is only the one Symbol for the entire string then the x offset of that symbol is set to **x_offset** regardless of the value of **vert**.

See Definitions of Super String Vertex Symbol Dimensions and Parameters for the definition of x offset.

A return value of 0 indicates the function call was successful.

ID = 809

Get_super_vertex_symbol_offset_width(Element super,Integer vert,Real &x_offset)

Name

Integer Get_super_vertex_symbol_offset_width(Element super,Integer vert,Real &x_offset)

Description

For the super Element **super**, return as **x_offset** the x offset of the symbol from vertex number **vert**.

If there is only the one Symbol for the entire string then the x offset of that Symbol is returned in **x_offset** regardless of the value of **vert**.

See Definitions of Super String Vertex Symbol Dimensions and Parameters for the definition of x offset.

A return value of 0 indicates the function call was successful.

ID = 810

Set_super_vertex_symbol_offset_height(Element super,Integer vert,Real y_offset)

Name

Integer Set_super_vertex_symbol_offset_height(Element super,Integer vert,Real y_offset)

Description

For the super Element **super**, set the y offset of the symbol from the vertex number **vert** to be **y_offset**.

If there is only the one Symbol for the entire string then the y offset of that symbol is set to **y_offset** regardless of the value of **vert**.

See Definitions of Super String Vertex Symbol Dimensions and Parameters for the definition of y offset.

A return value of 0 indicates the function call was successful.

ID = 811

Get_super_vertex_symbol_offset_height(Element super,Integer vert,Real &y_offset)

Name

Integer Get_super_vertex_symbol_offset_height(Element super,Integer vert,Real &y_offset)

Description

For the super Element **super**, return as **y_offset** the y offset of the symbol from the vertex number **vert**.

If there is only the one Symbol for the entire string then the y offset of that Symbol is returned in **y_offset** regardless of the value of **vert**.

See Definitions of Super String Vertex Symbol Dimensions and Parameters for the definition of y offset.

A return value of 0 indicates the function call was successful.

ID = 812

Set_super_vertex_symbol_rotation(Element super,Integer vert,Real ang)

Name

Integer Set_super_vertex_symbol_rotation(Element super,Integer vert,Real ang)

Description

For the super Element **super**, set the angle of rotation of the symbol on vertex number **vert** to **ang**. **ang** is in radians and is measured counterclockwise from the x-axis.

angle is in radians and is measured counterclockwise from the x-axis.

If there is only the one Symbol for the entire string then the angle of rotation of that symbol is set to **ang** regardless of the value of **vert**.

See Definitions of Super String Vertex Symbol Dimensions and Parameters for the definition of angle of rotation of the symbol.

A return value of 0 indicates the function call was successful.

ID = 803

Get_super_vertex_symbol_rotation(Element super,Integer vert,Real &angle)

Name

Integer Get_super_vertex_symbol_rotation(Element super,Integer vert,Real &angle)

Description

For the super Element **super**, return the angle of rotation in **angle** of the symbol on vertex number **vert**.

angle is in radians and is measured counterclockwise from the x-axis.

If there is only the one angle of rotation for the entire string then the angle of rotation of that Symbol is returned in **ang** regardless of the value of **vert**.

See Definitions of Super String Vertex Symbol Dimensions and Parameters for the definition of angle of rotation of the symbol.

A return value of 0 indicates the function call was successful.

ID = 804

Set_super_vertex_symbol_size(Element super,Integer vert,Real sz)

Name

Integer Set_super_vertex_symbol_size(Element super,Integer vert,Real sz)

Description

For the super Element **super**, set the size of the symbol on vertex number **vert** to be **sz**.

If there is only the one Symbol for the entire string then the size of that symbol is set to **sz** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 805

Get_super_vertex_symbol_size(Element super,Integer vert,Real &sz)

Name

Integer Get_super_vertex_symbol_size(Element super,Integer vert,Real &sz)

Description

For the super Element **super**, return as **s** the size of the symbol on vertex number **vert**.

If there is only the one angle of rotation for the entire string then the angle of rotation of that Symbol is returned in **sz** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 806

Super String Pipe/Culvert Functions

For definitions of the Pipe and Culvert dimensions, see Pipe/Culvert Dimensions

See Definitions of Super String Pipe and Culvert Dimensions and Parameters
See Super String Use Pipe Functions
See Setting Super String Pipe/Culvert Parameters

Definitions of Super String Pipe and Culvert Dimensions and Parameters

A super string can be super pipe string and the super pipe string can be **either**

(a) a round pipe with a diameter and a thickness

or

(b) or a rectangular pipe (culvert) with a width, height and four thicknesses (top, bottom, left right).

As a round pipe string, it can have either one diameter and one wall thickness for all segments of the string, or it can have different diameters and wall thicknesses for each segment of the string.

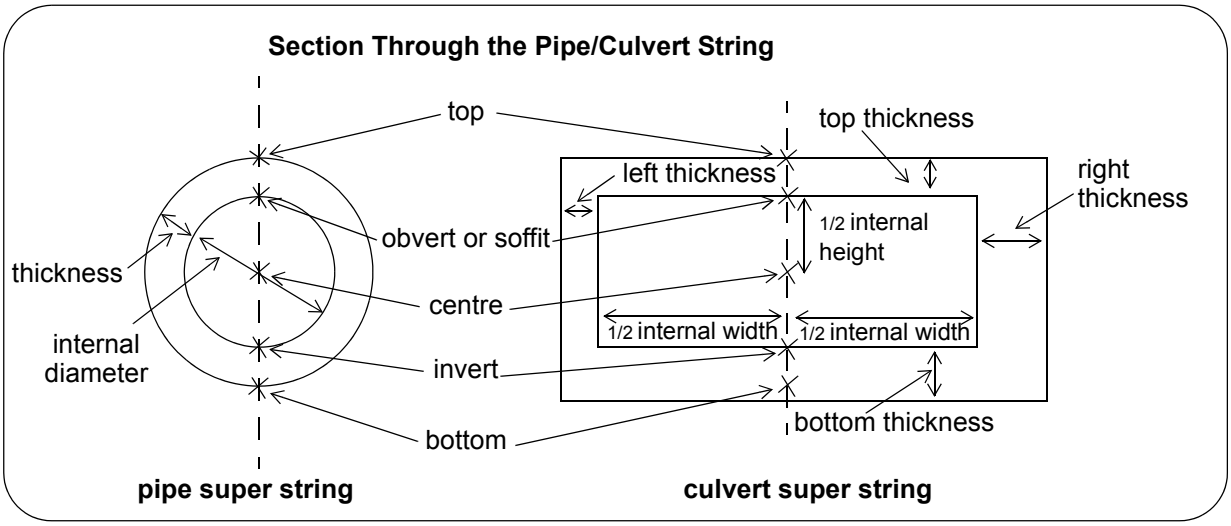
As a culvert string, it can have either one width, one height and four wall thicknesses (top, bottom, left and right) for all segments of the string, or it can have different heights, widths and four wall thicknesses (top, bottom, left and right) for each segment of the string.

The default value for wall thickness is zero.

external diameter of round pipe	= internal diameter + 2 * thickness
external width of culvert	= internal width + left thickness + right thickness
external height of culvert	= height + top thickness + bottom thickness

The centre of the culvert is defined to be the LJG?

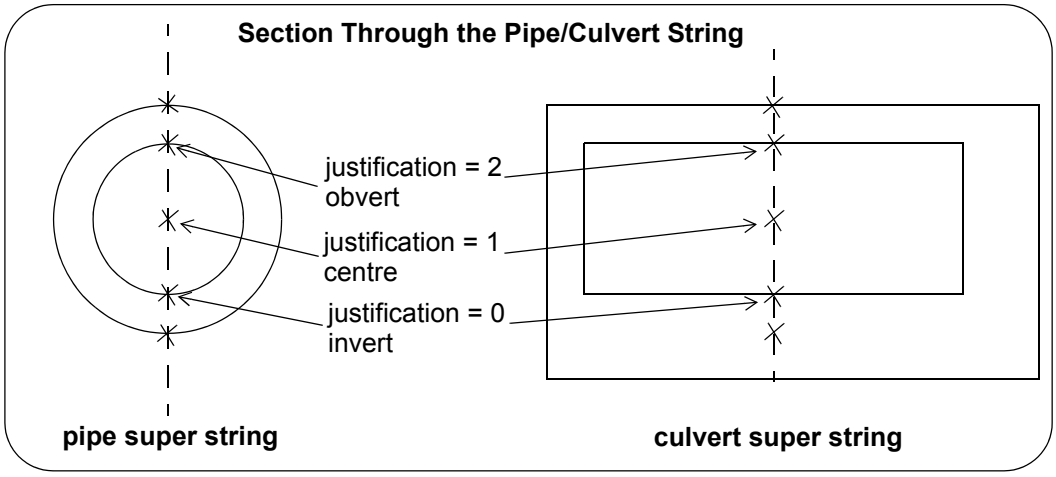
In practise pipes and culverts may also have a nominal diameter, width and height but there is no exact relationship between the nominal values and the interior or exterior values.



Pipe/Culvert Justification

Both the super pipe string and a super culvert string are defined in space by their (x,y,z) vertices but depending on the justification value, the (x,y,z) can represent either:

the invert of the pipe/culvert	justification = 0
the internal centre of the pipe/culvert	justification = 1
the obvert of the pipe/culvert	justification = 2



See Super String Use Pipe/Culvert Justify Dimensions
See Super String Use Pipe Functions
See Setting Super String Culvert Width, Height and Thicknesses

See Definitions of Super String Vertex Text Dimensions, Units and Annotation Parameters
See Super String Use Vertex Text Functions
See Super String Use Vertex Annotation Functions
See Setting Super String Vertex Text and Annotation Parameters

Super String Use Pipe Functions

Super pipes could have a diameter with an optional thickness (round pipe), or have a width and height with an four optional thicknesses (rectangular pipe or culvert).

Super String Use Round Pipe Dimensions

Set_super_use_pipe(Element elt,Integer use) for V10 onwards

Set_super_use_diameter(Element elt,Integer use) for V9

Name

Integer Set_super_use_pipe(Element elt,Integer use)

Integer Set_super_use_diameter(Element elt,Integer use)

Description

For the super string Element **elt**, define whether the pipe/culvert dimension Att_Diameter_Value is used or removed.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension Att_Diameter_Value is set That is, the pipe has one diameter and one thickness (V10) for the entire string (i.e. a constant pipe).
If **use** is **0**, the dimension is removed.

Note if any other pipe/culvert dimensions exist (besides Att_Pipe_Justify), this call is ignored.

This function has the new name for V10 onwards. The old call will still work.

A return value of 0 indicates the function call was successful.

ID = 704

Get_super_use_pipe(Element elt,Integer &use) for V10 onwards

Get_super_use_diameter(Element elt,Integer &use) for V9

Name

Integer Get_super_use_pipe(Element elt,Integer &use)

Integer Get_super_use_diameter(Element elt,Integer &use)

Description

Query whether the pipe/culvert dimension Att_Diameter_Value exists for the super string **elt**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists
use is returned as 0 if the dimension doesn't exist, or if it is a variable pipe string (i.e. a Att_Diameter_Array exists).

Note - if it is a constant pipe string (Att_Diameter_Value exists) and a variable pipe string (Att_Diameter_Array exists) then the variable pipe takes precedence.

This function has the new name for V10 onwards. The old call will still work.

A return value of 0 indicates the function call was successful.

ID = 705

Set_super_use_segment_pipe(Element elt,Integer use) for V10 onwards

Set_super_use_segment_diameter(Element elt,Integer use) for V9

Name

Integer Set_super_use_segment_pipe(Element elt,Integer use)

Integer Set_super_use_segment_diameter(Element elt,Integer use)

Description

For the super string Element **elt**, define whether the pipe/culvert dimension Att_Diameter_Array is used or removed.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension Att_Diameter_Array is set. That is, each pipe segment can have a different diameter and thickness (V10).
If **use** is 0, the dimension is removed.

Note if any other pipe/culvert dimensions exist (besides Att_Pipe_Justify), this call is ignored.

This function has the new name for V10 onwards. The old call will still work.

A return value of 0 indicates the function call was successful.

ID = 714

Get_super_use_segment_pipe(Element elt,Integer &use) for V10 onward

Get_super_use_segment_diameter(Element elt,Integer &use) for V9

Name

Integer Get_super_use_segment_pipe (Element elt,Integer &use)

Integer Get_super_use_segment_diameter (Element elt,Integer &use)

Description

Query whether the pipe/culvert dimension Att_Diameter_Array exists for the super string **elt**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

This function has the new name for V10 onwards. The old call will still work.

A return value of 0 indicates the function call was successful.

ID = 715

Super String Use Culvert Dimensions

Set_super_use_culvert(Element super,Integer use)

Name

Integer Set_super_use_culvert(Element super,Integer use)

Description

Tell the super string whether to use or remove the pipe/culvert dimension Att_Culvert_Value.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

Note if any other pipe/culvert dimensions exist (besides Att_Pipe_Justify), this call is ignored.

A return value of 0 indicates the function call was successful.

ID = 1247

Get_super_use_culvert(Element super,Integer &use)

Name

Integer Get_super_use_culvert(Element super,Integer &use)

Description

Query whether the pipe/culvert dimension Att_Culvert_Value exists for the super string.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension Att_Culvert_Value exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1246

Set_super_use_segment_culvert(Element super,Integer use)

Name

Integer Set_super_use_segment_culvert(Element super,Integer use)

Description

Tell the super string whether to use or remove the pipe/culvert dimension Att_Culvert_Array.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

Note if any other pipe/culvert dimensions exist (besides Att_Pipe_Justify), this call is ignored.

A return value of 0 indicates the function call was successful.

ID = 1251

Get_super_use_segment_culvert(Element super,Integer &use)

Name

Integer Get_super_use_segment_culvert(Element super,Integer &use)

Description

Query whether the pipe/culvert dimension Att_Culvert_Array exists for the super string.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension Att_Culvert_Array exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1250

Super String Use Pipe/Culvert Justify Dimensions

Set_super_use_pipe_justify(Element super,Integer use)

Name

Integer Set_super_use_pipe_justify(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the pipe/culvert dimension Att_Pipe_Justify is used or removed.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the pipe or culvert super string has a justification defined.

If **use** is **0**, the dimension is removed.

Note: the same justification flag is used whether the super string is a round pipe or a culvert and the justification applies for the entire string.

A return value of 0 indicates the function call was successful.

ID = 1255

Get_super_use_pipe_justify(Element super,Integer &use)

Name

Integer Get_super_use_pipe_justify(Element super,Integer &use)

Description

Query whether the pipe/culvert dimension Att_Pipe_Justify exists for the Element **super** of type **Super**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists
use is returned as 0 if the dimension doesn't exist.

Note: the same justification flag is used whether the super string is a round pipe or a culvert and the justification applies for the entire string.

A return value of 0 indicates the function call was successful.

ID = 1254

Setting Super String Pipe/Culvert Parameters

See Setting Super String Pipe/Culvert Justification
See Setting Super String Round Pipe Diameter and Thickness
See Setting Super String Culvert Width, Height and Thicknesses

See Superseded Setting Super String Round Pipe Diameter
See Superseded Setting Super String Culvert Width, Height and Thicknesses

Setting Super String Pipe/Culvert Justification

Integer Set_super_pipe_justify(Element super,Integer justify)

Name

Integer Set_super_pipe_justify(Element super,Integer justify)

Description

For the Element **super** of type **Super** which is a pipe or culvert string (i.e. Att_Diameter_Value, Att_Diameter_Array, Att_Culvert_Value or Att_Culvert_Array has been set), set the pipe/culvert justification to **justify**.

The values for **justify** are given in Pipe/Culvert Justification

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or a correct dimension is not allocated, this call fails and a non-zero function value is returned.

Note: the same justification flag is used whether the super string is a pipe or a culvert and the justification applies for the entire string.

A return value of 0 indicates the function call was successful

ID = 1256

Get_super_pipe_justify(Element super,Integer &justify)

Name

Integer Get_super_pipe_justify(Element super,Integer &justify)

Description

For the Element **super** of type **Super** which is a pipe or culvert string (i.e. Att_Diameter_Value, Att_Diameter_Array, Att_Culvert_Value or Att_Culvert_Array has been set), get the pipe/culvert justification and return it in **justify**.

The values for **justify** are given in Pipe/Culvert Justification

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or a correct dimension is not allocated, this call fails and a non-zero function value is returned.

Note: the same justification flag is used whether the super string is a pipe or a culvert and the justification applies for the entire string.

A return value of 0 indicates the function call was successful

ID = 1257

Setting Super String Round Pipe Diameter and Thickness

Set_super_pipe(Element super,Real diameter,Real thickness,Integer internal_diameter)

Name

Integer Set_super_pipe(Element super,Real diameter,Real thickness,Integer internal_diameter)

Description

For the Element **super** of type **Super** which is a **constant diameter** pipe string (i.e. the dimension flag Att_Diameter_Value has been set and Att_Diameter_Array has not been set), set the thickness to **thickness** and the internal diameter to **diameter** if internal_diameter = 1 or the external diameter to **diameter** if internal_diameter is non zero.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension is not allocated, this call fails and a non-zero function value is returned.

Note - Get_super_use_pipe can be called to make sure it is a constant diameter pipe string.

A return value of 0 indicates the function call was successful.

ID = 2645

Get_super_pipe(Element super,Real &diameter,Real thickness,Integer internal_diameter)

Name

Integer Get_super_pipe(Element super,Real &diameter,Real thickness,Integer internal_diameter)

Description

For the Element **super** of type **Super** which is a **constant diameter** round pipe string (i.e. Att_Diameter_Value has been set and Att_Diameter_Array has not been set), get the pipe thickness and return it in **thickness** and the internal diameter and return it in **internal_diameter**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension is not allocated, this call fails and a non-zero function value is returned.

Note - Get_super_use_pipe can be called to make sure it is a constant diameter round pipe string.

A return value of 0 indicates the function call was successful

ID = 2646

Set_super_segment_pipe(Element super,Integer seg,Real diameter, Real thickness,Integer internal_diameter)

Name

Integer Set_super_segment_pipe(Element super,Integer seg,Real diameter,Real thickness,Integer internal_diameter)

Description

For the super Element **super** and segment number **seg**, set the thickness to **thickness** and the internal diameter to **diameter** if **internal_diameter** = 1 or the external diameter to **diameter** if **internal_diameter** is non zero.

If **super** is not a variable pipe string then a non zero return value is returned.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

A return value of 0 indicates the function call was successful

ID = 2649

Get_super_segment_pipe(Element super,Integer seg,Real &diameter, Real &thickness,Integer &internal_diameter)

Name

Integer Get_super_segment_pipe(Element super,Integer seg,Real &diameter,Real &thickness,Integer &internal_diameter)

Description

For the super Element **super** and for segment number **seg,** get the pipe thickness and return it in **thickness**, and if the returned value of **internal_diameter** is 1 then return the internal diameter in **diameter** otherwise return the external diameter in **diameter**.

If **super** is not a variable pipe string then a non zero return value is returned.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

ID = 2650

Setting Super String Culvert Width, Height and Thicknesses

Set_super_culvert(Element super,Real width,Real height,Real left_thickness,Real right_thickness,Real top_thickness,Real bottom_thickness, Integer internal_width_height)

Name

Integer Set_super_culvert(Element super,Real width,Real height,Real left_thickness,Real right_thickness,Real top_thickness,Real bottom_thickness,Integer internal_width_height)

Description

For the Element **super** of type **Super** which is a **constant** width and height string (i.e.the pipe/culvert dimension flag Att_Culvert_Value has been set and Att_Culvert_Array not set), then

if **internal_width_height** =1 then set the culvert internal width to **w** and the internal height to **h**.

if **internal_width_height** is not 1 then set the culvert external width to **w** and the external height to **h**.

Set the left thickness to **left_thickness**, right thickness to **right_thickness,** top thickness to **top_thickness** and bottom thickness to **bottom_thickness**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension Att_Culvert_Value is not allocated, this call fails and a non-zero function value is returned.

A return value of 0 indicates the function call was successful.

Note - Get_super_use_culvert can be called to make sure it is a constant culvert string.

ID = 2647

Get_super_culvert(Element super,Real &width,Real &height,Real &left_thickness,Real &right_thickness,Real &top_thickness, Real &bottom_thickness,Integer &internal_width_height)

Name

Integer Get_super_culvert(Element super,Real &width,Real &height,Real &left_thickness,Real &right_thickness,Real &top_thickness,Real &bottom_thickness,Integer &internal_width_height)

Description

For the Element **super** of type **Super** which is a **constant** width and height string (i.e.the pipe/culvert dimension flag Att_Culvert_Value has been set and Att_Culvert_Array not set), then

if **internal_width_height** is returned as 1 then the culvert internal width is returned in **w** and the internal height returned in **h**.

if **internal_width_height** is not returned as 1 then the culvert external width is returned in **w** and the external height returned in **h**.

The left thickness is returned in **left_thickness**, right thickness in **right_thickness,** top thickness in **top_thickness** and bottom thickness in **bottom_thickness**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension is not allocated, this call fails and a non-zero function value is returned.

A return value of 0 indicates the function call was successful

Note - Get_super_use_culvert can be called to make sure it is a constant culvert string.

ID = 2648

Set_super_segment_culvert(Element super,Integer seg,Real width,Real height, Real left_thickness,Real right_thickness,Real top_thickness, Real bottom_thickness,Integer internal_width_height)

Name

Integer Set_super_segment_culvert(Element super,Integer seg,Real width,Real height,Real left_thickness,Real right_thickness,Real top_thickness,Real bottom_thickness,Integer internal_width_height)

Description

For the Element **super** of type **Super** which has culvert widths and heights for **each** segment (i.e.the pipe/culvert dimension flag Att_Culvert_Array has been set), then for segment number **seg**:

if **internal_width_height** =1 then set the culvert internal width to **w** and the internal height to **h**.

if **internal_width_height** is not 1 then set the culvert external width to **w** and the external height to **h**.

Set the left thickness to **left_thickness**, right thickness to **right_thickness,** top thickness to **top_thickness** and bottom thickness to **bottom_thickness**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension Att_Culvert_Array is not allocated, this call fails and a non-zero function value is returned.

A return value of 0 indicates the function call was successful.

Note - Get_super_use_segment_culvert can be called to make sure it is a variable segment culvert string.

ID = 2651

Get_super_segment_culvert(Element super,Integer seg,Real &width,Real &height,Real &left_thickness,Real &right_thickness,Real &top_thickness, Real &bottom_thickness,Integer &internal_width_height) For V10 only

Name

Integer Get_super_segment_culvert(Element super,Integer seg,Real &width,Real &height,Real &left_thickness,Real &right_thickness,Real &top_thickness,Real &bottom_thickness,Integer &internal_width_height)

Description

For the Element **super** of type **Super** which has culvert width and heights for **each** segment (i.e.the pipe/culvert dimension flag Att_Culvert_Array has been set), then for segment number **seg**:

if **internal_width_height** is returned as 1 then the culvert internal width is returned in **w** and the internal height returned in **h**.

if **internal_width_height** is not returned as 1 then the culvert external width is returned in **w** and the external height returned in **h**.

The left thickness is returned in **left_thickness**, right thickness in **right_thickness,** top thickness in **top_thickness** and bottom thickness in **bottom_thickness**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String

Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension is not allocated, this call fails and a non-zero function value is returned.

A return value of 0 indicates the function call was successful

Note - Get_super_use_segment_culvert can be called to make sure it is a variable segment culvert string.

ID = 2652

Superseded Setting Super String Round Pipe Diameter

From V10 onwards, round pipe strings can have a wall thickness so the following calls that do not return this extra value are now superseded and should not be used.

Set_super_pipe(Element super,Real diameter) for V10 and above

Set_super_diameter(Element super,Real diameter) for V9

Name

Integer Set_super_pipe (Element super,Real diameter)

Integer Set_super_diameter (Element super,Real diameter)

Description

For the Element **super** of type **Super** which is a **constant diameter** pipe string (i.e. the dimension flag Att_Diameter_Value has been set and Att_Diameter_Array has not been set), set the diameter to **diameter**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension is not allocated, this call fails and a non-zero function value is returned.

Note - Get_super_use_pipe can be called to make sure it is constant diameter pipe string.

This function has the new name for V10 onwards. The old call will still work.

A return value of 0 indicates the function call was successful.

ID = 706

Get_super_pipe(Element super,Real &diameter) for V10 onwards

Get_super_diameter(Element super,Real &diameter) for V9

Name

Integer Get_super_pipe(Element super,Real &diameter)

Integer Get_super_diameter(Element super,Real &diameter)

Description

For the Element **super** of type **Super** which is a **constant diameter** round pipe string (i.e. Att_Diameter_Value has been set and Att_Diameter_Array has not been set), get the pipe diameter and return it in **diameter**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension is not allocated, this call fails and a non-zero function value is returned.

This function has the new name for V10 onwards. The old call will still work.

Note - Get_super_use_pipe can be called to make sure it is a constant diameter pipe string.

A return value of 0 indicates the function call was successful

ID = 707

Set_super_segment_pipe(Element super,Integer seg,Real diameter) for V10 onwards

Set_super_segment_diameter(Element super,Integer seg,Real diameter) for V9

Name

Integer Set_super_segment_pipe(Element super,Integer seg,Real diameter)

Integer Set_super_segment_diameter(Element super,Integer seg,Real diameter)

Description

For the super Element **super**, set the pipe diameter for segment number **seg** to **diameter**.

For V10, if **super** is not a variable pipe string then a non zero return value is returned.
For V10,a return value of 0 indicates the function call was successful

For V9, the return code **is always** 0.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

Note - for V9, no error code is set if the string in not a variable pipe string. That needs to checked using the Get_super_use_pipe calls.

This function has the new name for V10 onwards. The old call will still work.

A return value of 0 indicates the function call was successful

ID = 716

Get_super_segment_pipe(Element super,Integer seg,Real &diameter) for V10 onward

Get_super_segment_diameter(Element super,Integer seg,Real &diameter) for V9

Name

Integer Get_super_segment_pipe(Element super,Integer seg,Real &diameter)

Integer Get_super_segment_diameter(Element super,Integer seg,Real &diameter)

Description

This function has the new name for V10 onwards. The old call will still work.

For the super Element **super**, get the pipe diameter for segment number **seg** and return it in **diameter**.

For V10, if **super** is not a variable pipe string then a non zero return value is returned.
For V10,a return value of 0 indicates the function call was successful

For V9, the return code **is always** 0.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

Note - for V9, no error code is set if the string in not a variable pipe string. That needs to checked using the Get_super_use_pipe calls.

ID = 717

Superseded Setting Super String Culvert Width, Height and Thicknesses

From V10 onwards, culvert strings can have four wall thicknesses (top, bottom, left and right) so the following calls that do not return theses extra values are now superseded and should not be used.

Set_super_culvert(Element super,Real w,Real h)

Name

Integer Set_super_culvert(Element super,Real w,Real h)

Description

For the Element **super** of type **Super** which is a **constant** width and height culvert string (i.e.the pipe/culvert dimension flag Att_Culvert_Value has been set), set the culvert width to **w** and the height to **h**.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension is not allocated Att_Culvert_Value, this call fails and a non-zero function value is returned.

A return value of 0 indicates the function call was successful.

Note - Get_super_use_culvert can be called to make sure it is a constant culvert string.

ID = 1249

Get_super_culvert(Element super,Real &w,Real &h)

Name

Integer Get_super_culvert(Element super,Real &w,Real &h)

Description

For the Element **super** of type **Super** which is a **constant** width and height culvert string (i.e.the pipe/culvert dimension flag Att_Culvert_Value has been set), get the culvert width and height and return them in **w** and **h** respectively.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension is not allocated, this call fails and a non-zero function value is returned.

A return value of 0 indicates the function call was successful

Note - Get_super_use_culvert can be called to make sure it is a constant culvert string.

ID = 1248

Set_super_segment_culvert(Element super,Integer seg,Real w,Real h)

Name

Integer Set_super_segment_culvert(Element super,Integer seg,Real w,Real h)

Description

For the Element **super** of type **Super** which has culvert widths and heights for **each** segment(i.e.the pipe/culvert dimension flag Att_Culvert_Array has been set), set the culvert width and height for segment number **seg** to be **w** and **h** respectively.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension Att_Culvert_Array is not allocated,

this call fails and a non-zero function value is returned.

A return value of 0 indicates the function call was successful.

Note - Get_super_use_segment_culvert can be called to make sure it is variable segment culvert string.

ID = 1253

Get_super_segment_culvert(Element super,Integer seg,Real &w,Real &h)

Name

Integer Get_super_segment_culvert(Element super,Integer seg,Real &w,Real &h)

Description

For the Element **super** of type **Super** which has culvert widths and heights for **each** segment(i.e.the pipe/culvert dimension flag Att_Culvert_Array has been set), get the culvert width and height for segment number **seg** and return them in **w** and **h** respectively.

See Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions or Super String Dimensions for information on all dimensions.

If the Element **super** is not of type **Super**, or the dimension Att_Culvert_Array is not allocated, this call fails and a non-zero function value is returned.

A return value of 0 indicates the function call was successful.

Note - *Get_super_use_segment_culvert* can be called to make sure it is variable segment culvert string.

ID = 1252

Super String Vertex Text and Annotation Functions

See Definitions of Super String Vertex Text Dimensions, Units and Annotation Parameters
See Super String Use Vertex Text Functions
See Super String Use Vertex Annotation Functions
See Setting Super String Vertex Text and Annotation Parameters

Definitions of Super String Vertex Text Dimensions, Units and Annotation Parameters

Super String Vertex text refers to the text at a super string vertex.

If super string text is required then the dimension to set is either

(a) the most common case of having a different text at each vertex (dimension Att_Vertex_Text_Array)

or

(b) the rare case of just the same text that is used for every vertex (dimension Att_Vertex_Text_Value)

Although vertex text may be defined, it will not display in a plan view, or on a plan plot, unless a Vertex Text Annotation dimension has been set. A Text Annotation controls the text size, colour, rotation etc.

So if super string vertex text is required to be drawn on a plan view then the dimension to set is either

(a) for the case of having a different text annotation at each vertex so that the annotation attributes can be modified at each vertex then set dimension Att_Vertex_Annotate_Array

or

(b) if there is just the one Annotation and its parameters are used for drawing the text on every vertex then set the dimension Att_Vertex_Annotate_Value (this is the case for the traditional 4d string).

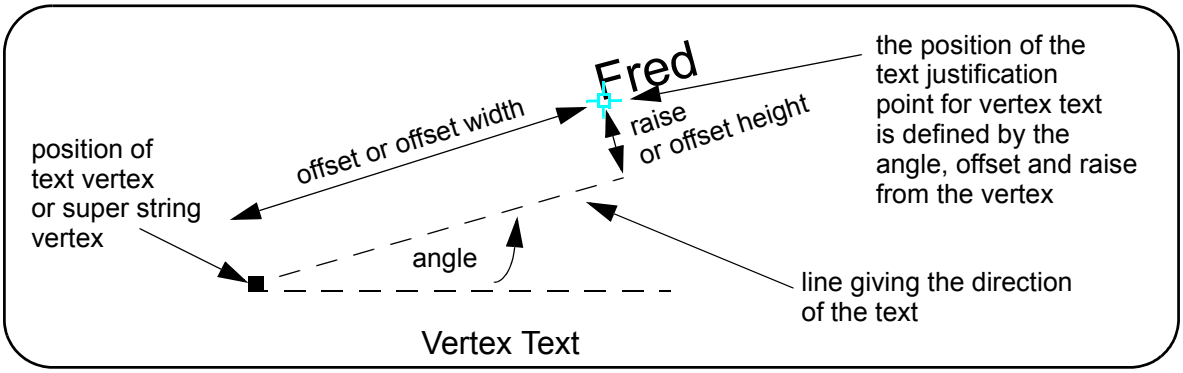
For definitions of the Vertex Text dimensions see Vertex Text Dimensions *and the Vertex Text Annotation dimensions see* Vertex Text Annotation Dimensions .

Vertex Text Annotation Definitions

For vertex text, the text ***justification point*** and the ***direction of the text*** are defined by:

(a) the *direction of the text* is given as a *counter clockwise* ***angle*** *of rotation* (measured from the x-axis) about the vertex *(default 0)*

(b) the *justification point* is given as an ***offset*** from the vertex *along the line through the vertex with the direction of the text*, and a perpendicular distance (called the ***raise***) from that offset point to the justification point (default 0).



The vertex and justification point only coincide if the offset and raise values are both zero.

Finally the text can be one of nine positions defined in relation to the (x,y) coordinates of the text justification point:

		top		
	3	6	9	
left	2	5	8	**right**
	1	4	7	
		bottom		

This is usually an Integer called the ***justification*** with a default value of 1.

Vertex Text Annotation Units

The units for text size is specified by an Integer whose value is

(a) 0 (the default) for the units are screen/pixel/device units

(b) 1 for world units

(c) 2 for paper units (millimetres on a plot).

Regardless of whether there is one Vertex Text Annotation for the entire string or a different Text Annotation for each vertex, there is only one ***units*** for text size used for all the Vertex Text of the string.

The units for text are used for the size of the text, and the offsets and raises for the text.

For Information on all the super string vertex text and vertex text annotations:

See Super String Use Vertex Text Functions
See Super String Use Vertex Annotation Functions
See Setting Super String Vertex Text and Annotation Parameters

Super String Use Vertex Text Functions

For definitions of the Vertex Text dimensions, see Vertex Text Dimensions

Set_super_use_vertex_text_value(Element super,Integer use)

Name

Integer Set_super_use_vertex_text_value(Element super,Integer use)

Description

Tell the super string **super** whether to use (set), or not use (remove), the dimension Att_Vertex_Text_Value.

A value for **use** of 1 sets the dimension and 0 removes it.

If Att_Vertex_Text_Value is used, then the *same* text is attached to all the vertices of the super string.

Note if the dimension Att_Vertex_Text_Array exists, this call is ignored.

See Vertex Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1237

Get_super_use_vertex_text_value(Element super,Integer &use)

Name

Integer Get_super_use_vertex_text_value(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Text_Value exists for the super string **super**.

use is returned as 1 if the dimension Att_Vertex_Text_Value exists.
use is returned as 0 if the dimension doesn't exist.

If the dimension Att_Vertex_Text_Value exists then the string has the same text for every vertex of the string.

See Vertex Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1238

Set_super_use_vertex_text_array(Element super,Integer use)

Name

Integer Set_super_use_vertex_text_array(Element super,Integer use)

Description

Tell the super string whether to use (set), or not use (remove), the dimension Att_Segment_Text_Array.

A value for **use** of 1 sets the dimension and 0 removes it.

If Att_Vertex_Text_Array is used, then there is different text at each vertex of the super string **super**.

See Vertex Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 742

Get_super_use_vertex_text_array(Element super,Integer &use)

Name

Integer Get_super_use_vertex_text_array(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Text_Array exists (is used) for the super string **super**.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

If Att_Vertex_Text_Array is used, then there is different text on each vertex of the of the string.

See Vertex Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 743

Super_vertex_text_value_to_array(Element super)

Name

Integer Super_vertex_text_value_to_array(Element super)

Description

If for the super string **super** the dimension Att_Vertex_Text_Value exists and the dimension Att_Vertex_Text_Array does not exist then there will be one Vertex Text **txt** for the entire string.

In this case (when the dimension Att_Vertex_Text_Value exists and the dimension Att_Vertex_Text_Array does not exist) this function sets the Att_Vertex_Text_Array dimension and new vertex text created for each vertex of **super** and the new vertex text is given the value **txt**.

See Vertex Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 2177

Super String Use Vertex Annotation Functions

For definitions of the Vertex Annotation dimensions, see Vertex Text Annotation Dimensions

Set_super_use_vertex_annotation_value(Element super,Integer use)

Name

Integer Set_super_use_vertex_annotation_value(Element super,Integer use)

Description

Tell the super string **super** whether to use, or not use, the dimension Att_Vertex_Annotate_Value.

If the dimension Att_Vertex_Annotate_Value exists and the dimension Att_Vertex_Annotate_Array doesn't exist then the string has the one annotation which is used for vertex text on **any** vertex of the string.

See Vertex Text Annotation Dimensions for information on the Text Annotation dimensions or Super String Dimensions for information on all the dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

Note if the dimension Att_Vertex_Annotate_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

ID = 750

Get_super_use_vertex_annotation_value(Element super,Integer &use)

Name

Integer Get_super_use_vertex_annotation_value(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Annotate_Value exists for the super string **super**.

If the dimension Att_Vertex_Annotate_Value exists and the dimension Att_Vertex_Annotate_Array doesn't exist then the string has the one annotation which is used for vertex text on **any** vertex of the string.

See Vertex Text Annotation Dimensions for information on the Text Annotation dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 751

Set_super_use_vertex_annotation_array(Element super,Integer use)

Name

Integer Set_super_use_vertex_annotation_array(Element super,Integer use)

Description

Tell the super string **super** whether to use, or not use, the dimension Att_Vertex_Annotate_Array.

If the dimension Att_Vertex_Annotate_Array exists then the string has a different annotation for the vertex text on each vertex of the string.

See Vertex Text Annotation Dimensions for information on the Text Annotation dimensions or Super String Dimensions for information on all the dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

A return value of 0 indicates the function call was successful.

ID = 752

Get_super_use_vertex_annotation_array(Element super,Integer &use)

Name

Integer Get_super_use_vertex_annotation_array(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Annotate_Array exists for the super string **super**.

If the dimension Att_Vertex_Annotate_Array exists then the string has a different annotation for the vertex text on each vertex of the string.

See Vertex Text Annotation Dimensions for information on the Text Annotation dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 753

Super_vertex_annotate_value_to_array(Element elt)

Name

Integer Super_vertex_annotate_value_to_array(Element elt)

Description

If for the super string **super** the dimension Att_Vertex_Annotate_Value exists and the dimension Att_Vertex_Annotate_Array does not exist then there will be one Annotation **annot** for the entire string.

In this case (when the dimension Att_Vertex_Annotate_Value exists and the dimension Att_Vertex_Annotate_Array does not exist), this function sets the Att_Vertex_Annotate_Array dimension and new Annotations created for each vertex of **super** and the new Annotation is given the value **annot**.

See Vertex Text Annotation Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 2178

Setting Super String Vertex Text and Annotation Parameters

Set_super_vertex_text(Element super,Integer vert,Text txt)

Name

Integer Set_super_vertex_text(Element super,Integer vert,Text txt)

Description

For the super Element **super**, set the vertex text at vertex number **vert** to be **txt**.

If there is only one Vertex Text for all the vertices then the text for that one Vertex Text is set to **txt** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 744

Get_super_vertex_text(Element super,Integer vert,Text &txt)

Name

Integer Get_super_vertex_text(Element super,Integer vert,Text &txt)

Description

For the super string **super**, return in **txt** the vertex text on vertex number **vert**.

If there is only one Vertex Text for all the vertices then the text for that one Vertex Text will be returned in **txt** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 745

Set_super_vertex_world_text(Element super)

Name

Integer Set_super_vertex_world_text(Element)

Description

Set the units for vertex text for the super string **super** to *World*. See Vertex Text Annotation Units .

A return value of 0 indicates the function call was successful.

ID = 747

Set_super_vertex_device_text(Element super)

Name

Integer Set_super_vertex_device_text(Element)

Description

Set the units for vertex text for the super string **super** to *Screen* (also known as Device or Pixel). See Vertex Text Annotation Units .

A return value of 0 indicates the function call was successful.

ID = 746

Set_super_vertex_paper_text(Element super)

Name

Integer Set_super_vertex_paper_text(Element super)

Description

For an Element **super** of type Super, set the text units for vertex text to be paper (that is millimetres).

See Vertex Text Annotation Units for the definition of segment text units.

If there is Textstyle_Data for the vertex text then this will override the *Set_super_vertex_device_text* call.

A return value of 0 indicates the function call was successful.

ID = 1633

Set_super_vertex_text_type(Element super,Integer type)

Name

Integer Set_super_vertex_text_type(Element super,Integer type)

Description

For the super Element **super**, set the vertex text units to be the value of **type**.

See Vertex Text Annotation Units for the definition of vertex text units.

A return value of 0 indicates the function call was successful.

ID = 748

Get_super_vertex_text_type(Element super,Integer &type)

Name

Integer Get_super_vertex_text_type(Element super,Integer &type)

Description

For the super Element **super**, return in **type** the value for the vertex text units for the vertex text of the string.

See Vertex Text Annotation Units for the definition of vertex text units.

A return value of 0 indicates the function call was successful.

ID = 749

Set_super_vertex_text_justify(Element super,Integer vert,Integer just)

Name

Integer Set_super_vertex_text_justify(Element super,Integer vert,Integer just)

Description

For the super string **super**, set the justification of the text on vertex number **vert** to **just**.

See Vertex Text Annotation Definitions for the definition of justification.

If there is only one Vertex Text Annotation for all the Vertex Text then the justification for that one Vertex Text Annotation is set to **just** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 754

Get_super_vertex_text_justify(Element super,Integer vert,Integer &just)

Name

Integer Get_super_vertex_text_justify(Element super,Integer vert,Integer &just)

Description

For the super string **super**, return the justification of the vertex text on vertex number **vert** in **just**.

See Vertex Text Annotation Definitions for the definition of justification.

If there is only one Vertex Text Annotation for all the Vertex Text then the justification for that one Vertex Text Annotation will be returned in **just** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 755

Set_super_vertex_text_offset_width(Element super,Integer vert,Real offset)

Name

Integer Set_super_vertex_text_offset_width(Element super,Integer vert,Real offset)

Description

For the super string **super**, set the offset (offset width) of the vertex text from vertex number **vert** to **offset**

See Vertex Text Annotation Definitions for the definition of offset (offset width).

If there is only one Vertex Text Annotation for all the Vertex Text then the offset width for that one Vertex Text Annotation is set to **offset** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 756

Get_super_vertex_text_offset_width(Element super,Integer vert,Real &offset)

Name

Integer Get_super_vertex_text_offset_width(Element super,Integer vert,Real &offset)

Description

For the super string **super**, return as **offset** the offset (offset width) of the vertex text from vertex number **vert**.

See Vertex Text Annotation Definitions for the definition of offset (offset width).

If there is only one Vertex Text Annotation for all the Vertex Text then the offset width for that one Vertex Text Annotation will be returned in **offset** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 757

Set_super_vertex_text_offset_height(Element super,Integer vert,Real raise)

Name

Integer Set_super_vertex_text_offset_height(Element super,Integer vert,Real raise)

Description

For the super string **super**, set the raise (offset height) of the vertex text for vertex number **vert** to **raise**.

See Vertex Text Annotation Definitions for the definition of raise (offset height)

If there is only one Vertex Text Annotation for all the Vertex Text then the raise for that one Vertex Text Annotation is set to **raise** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 758

Get_super_vertex_text_offset_height(Element super,Integer vert,Real &raise)

Name

Integer Get_super_vertex_text_offset_height(Element super,Integer vert,Real &raise)

Description

For the super string **super**, return as **raise** the raise of the vertex text from vertex number **vert**.

See Vertex Text Annotation Definitions for the definition of raise (offset height)

If there is only one Vertex Text Annotation for all the Vertex Text then the raise for that one Vertex Text Annotation will be returned in **raise** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 759

Set_super_vertex_text_colour(Element super,Integer vert,Integer col)

Name

Integer Set_super_vertex_text_colour(Element super,Integer vert,Integer col)

Description

For the super string **super**, set the colour number of the vertex text on the vertex number **vert** to be **col**.

If there is only one Vertex Text Annotation for all the Vertex Text then the colour number for that one Vertex Text Annotation is set to **col** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 1091

Get_super_vertex_text_colour(Element super,Integer vert,Integer &col)

Name

Integer Get_super_vertex_text_colour(Element super,Integer vert,Integer &col)

Description

For the super string **super**, return as **col** the colour number of the vertex text on vertex number **vert**.

If there is only one Vertex Text Annotion for all the Vertex Text then the colour for that one Vertex Text Annotation will be returned in **col** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 1092

Set_super_vertex_text_angle(Element super,Integer vert,Real ang)

Name

Integer Set_super_vertex_text_angle(Element super,Integer vert,Real ang)

Description

For the super string **super**, set the angle of rotation of the vertex text on vertex number **vert** to **ang**. **ang** is in radians and is measured counterclockwise from the x-axis.

See Vertex Text Annotation Definitions for the definition of angle.

If there is only one Vertex Text Annotion for all the Vertex Text then the angle for that one Vertex Text Annotation is set to **ang** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 760

Get_super_vertex_text_angle(Element super,Integer vert,Real &ang)

Name

Integer Get_super_vertex_text_angle(Element super,Integer vert,Real &ang)

Description

For the super string **super**, return the angle of rotation of the vertex text on vertex number **vert** in **ang**. **ang** is measured in radians and is measured counterclockwise from the x-axis.

See Vertex Text Annotation Definitions for the definition of angle.

If there is only one Vertex Text Annotion for all the Vertex Text then the angle for that one Vertex Text Annotation will be returned in **ang** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 761

Set_super_vertex_text_size(Element super,Integer vert,Real sz)

Name

Integer Set_super_vertex_text_size(Element super,Integer vert,Real sz)

Description

For the super Element **super**, set the size of the vertex text on vertex number **vert** to **sz**.

If there is only one Vertex Text Annotion for all the Vertex Text then the size for that one Vertex Text Annotation is set to **sz** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 762

Get_super_vertex_text_size(Element super,Integer vert,Real &sz)

Name

Integer Get_super_vertex_text_size(Element super,Integer vert,Real &sz)

Description

For the super string **super**, return the size of the vertex text on vertex number **vert** as **sz**.

If there is only one Vertex Text Annotion for all the Vertex Text then the size for that one Vertex Text Annotation will be returned in **sz** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 763

Set_super_vertex_text_x_factor(Element super,Integer vert,Real xf)

Name

Integer Set_super_vertex_text_x_factor(Element super,Integer vert,Real xf)

Description

For the super string **super**, set the x factor of the vertex text on vertex number **vert** to **xf**.

If there is only one Vertex Text Annotion for all the Vertex Text then the x factor for that one Vertex Text Annotation is set to **xf** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 764

Get_super_vertex_text_x_factor(Element super,Integer vert,Real &xf)

Name

Integer Get_super_vertex_text_x_factor(Element super,Integer vert,Real &x)

Description

For the super string **super**, return in **xf** the x factor of the vertex text on vertex number **vert**.

If there is only one Vertex Text Annotion for all the Vertex Text then the x factor for that one Vertex Text Annotation will be returned in **xf** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 765

Set_super_vertex_text_slant(Element super,Integer vert,Real sl)

Name

Integer Set_super_vertex_text_slant(Element super,Integer vert,Real sl)

Description

For the super string **super**, set the slant of the vertex text on vertex number **vert** to **sl**.

If there is only one Vertex Text Annotion for all the Vertex Text then the slant factor for that one Vertex Text Annotation is set to **sl** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 766

Get_super_vertex_text_slant(Element super,Integer vert,Real &sl)

Name

Integer Get_super_vertex_text_slant(Element super,Integer vert,Real &s)

Description

For the super string **super**, return as **sl** the slant of the vertex text on vertex number **vert**.

If there is only one Vertex Text Annotion for all the Vertex Text then the slant for that one Vertex Text Annotation will be returned in **sl** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 767

Set_super_vertex_text_style(Element super,Integer vert,Text ts)

Name

Integer Set_super_vertex_text_style(Element super,Integer vert,Text ts)

Description

For the super string **super**, set the textstyle of the vertex text on vertex number **vert** to **ts**.

If there is only one Vertex Text Annotion for all the Vertex Text then the textstyle for that one Vertex Text Annotation is set to **ts** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 768

Get_super_vertex_text_style(Element super,Integer vert,Text &ts)

Name

Integer Get_super_vertex_text_style(Element super,Integer vert,Text &ts)

Description

For the super string **super**, return as **ts** the textstyle of the vertex text on vertex number **vert**.

If there is only one Vertex Text Annotion for all the Vertex Text then the textstyle for that one Vertex Text Annotation will be returned in **ts** regardless of the value of **vert**.

A return value of 0 indicates the function call was successful.

ID = 769

Set_super_vertex_text_ttf_underline(Element super,Integer vert,Integer underline)

Name

Integer Set_super_vertex_text_ttf_underline(Element super super,Integer vert,Integer underline)

Description

For the Element **super** of type **Super,** set the underline state for the vertex text on vertex number **vert** to be **underline**.

If **underline** = 1, then for a true type font the text will be underlined.
If **underline** = 0, then text will not be underlined.

If there is only one Vertex Text Annotion for all the Vertex Text then the underline state for that one Vertex Text Annotation is set to **underline** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates **underline** was successfully set.

ID = 2600

Get_super_vertex_text_ttf_underline(Element super,Integer vert, Integer &underline)

Name

Integer Get_super_vertex_text_ttf_underline(Element super,Integer vert,Integer &underline)

Description

For the Element **super** of type **Super,** get the underline state for the vertex text on vertex number **vert** and return it as **underline**.

If **underline** = 1, then for a true type font the text will be underlined.
If **underline** = 0, then text will not be underlined.

If there is only one Vertex Text Annotion for all the Vertex Text then the underline state for that one Vertex Text Annotation will be returned in **underline** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates **underline** was successfully returned.

ID = 2601

Set_super_vertex_text_ttf_strikeout(Element super,Integer vert,Integer strikeout)

Name

Integer Set_super_vertex_text_ttf_strikeout(Element super,Integer vert,Integer strikeout)

Description

For the Element **super** of type **Super,** set the strikeout state for the vertex text on vertex number **vert** to be **strikeout**.

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

If there is only one Vertex Text Annotion for all the Vertex Text then the strikeout state for that one Vertex Text Annotation is set to **strikeout** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates **strikeout** was successfully set.

ID = 2602

Get_super_vertex_text_ttf_strikeout(Element super,Integer vert, Integer &strikeout)

Name

Integer Get_super_vertex_text_ttf_strikeout(Element super,Integer vert,Integer &strikeout)

Description

For the Element **super** of type **Super,** get the strikeout state for the vertex text on vertex number **vert** and return it as **strikeout**.

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

If there is only one Vertex Text Annotion for all the Vertex Text then the strikeout state for that one Vertex Text Annotation will be returned in **strikeout** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates **strikeout** was successfully returned.

ID = 2603

Set_super_vertex_text_ttf_italic(Element super,Integer vert,Integer italic)

Name

Integer Set_super_vertex_text_ttf_italic(Element super,Integer vert,Integer italic)

Description

For the Element **super** of type **Super,** set the italic state for the vertex text on vertex number **vert** to be **italic**.

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

If there is only one Vertex Text Annotion for all the Vertex Text then the italic state for that one Vertex Text Annotation is set to **italic** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates **italic** was successfully set.

ID = 2604

Get_super_vertex_text_ttf_italic(Element super,Integer vert,Integer &italic)

Name

Integer Get_super_vertex_text_ttf_italic(Element super,Integer vert,Integer &italic)

Description

For the Element **super** of type **Super,** get the italic state for the vertex text on vertex number **vert** and return it as **italic**.

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

If there is only one Vertex Text Annotion for all the Vertex Text then the italic state for that one Vertex Text Annotation will be returned in **italic** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates **italic** was successfully returned.

ID = 2605

Set_super_vertex_text_ttf_outline(Element elt,Integer vert,Integer outline)

Name

Integer Set_super_vertex_text_ttf_outline(Element elt,Integer vert,Integer outline)

Description

For the Element **super** of type **Super,** set the outline state for the vertex text on vertex number **vert** to be **outline**.

If **outline** = 1, then for a true type font the text will be only shown in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

If there is only one Vertex Text Annotion for all the Vertex Text then the outline state for that one Vertex Text Annotation is set to **outline** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates ***outline*** was successfully set.

ID = 2775

Get_super_vertex_text_ttf_outline(Element elt,Integer vert,Integer &outline)

Name

Integer Get_super_vertex_text_ttf_outline(Element elt,Integer vert,Integer &outline)

Description

For the Element **super** of type **Super,** get the outline state for the vertex text on vertex number **vert** and return it as **outline**.

If **outline** = 1, then for a true type font the text will be shown only in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

If there is only one Vertex Text Annotion for all the Vertex Text then the outline state for that one Vertex Text Annotation will be returned in **outline** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates ***outline*** was successfully returned.

ID = 2776

Set_super_vertex_text_ttf_weight(Element super,Integer vert,Integer weight)

Name

Integer Set_super_vertex_text_ttf_weight(Element super,Integer vert,Integer weight)

Description

For the Element **super** of type **Super,** set the weight for the vertex text on vertex number **vert** to be **weight**.

For the list of allowable weights, go to Allowable Weights

If there is only one Vertex Text Annotion for all the Vertex Text then the weight for that one Vertex Text Annotation is set to **weight** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates **weight** was successfully set.

ID = 2606

Get_super_vertex_text_ttf_weight(Element super,Integer vert,Integer &weight)

Name

Integer Get_super_vertex_text_ttf_weight(Element super,Integer vert,Integer &weight)

Description

For the Element **super** of type **Super,** get the weight for the vertex text on vertex number **vert** and return it as **weight**.

For the list of allowable weights, go to Allowable Weights

If there is only one Vertex Text Annotion for all the Vertex Text then the weight for that one Vertex Text Annotation will be returned in **weight** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Array or Att_Vertex_Value set.

A function return value of zero indicates **weight** was successfully returned.

ID = 2607

Set_super_vertex_text_whiteout(Element superstring,Integer vert,Integer c)

Name

Integer Set_super_vertex_text_whiteout(Element superstring,Integer vert,Integer c)

Description

For vertex number **vert** of the Super String Element **superstring**, set the colour number of the colour used for the whiteout box around the vertex text, to be **colour**.

If no text whiteout is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

If there is only one Vertex Text Annotion for all the Vertex Text then the colour number of the colour used for the whiteout box around the vertex text for that one Vertex Text Annotation is set to **c** regardless of the value of **vert**.

A function return value of zero indicates the colour number was successfully set.

ID = 2755

Get_super_vertex_text_whiteout(Element superstring,Integer vert,Integer &c)

Name

Integer Get_super_vertex_text_whiteout(Element superstring,Integer vert,Integer &c)

Description

For vertex number **vert** of the Super String Element **superstring**, get the colour number that is used for the whiteout box around the vertex text. The whiteout colour is returned as Integer **colour**.

NO_COLOUR is the returned as the colour number if whiteout is not being used.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

If there is only one Vertex Text Annotion for all the Vertex Text then the colour number that is used for the whiteout box around the vertex text for that one Vertex Text Annotation will be returned in **c** regardless of the value of **vert**.

A function return value of zero indicates the colour number was successfully returned.

ID = 2756

Set_super_vertex_text_border(Element superstring,Integer vert,Integer c)

Name

Integer Set_super_vertex_text_border(Element superstring,Integer vert,Integer c)

Description

For vertex number **vert** of the Super String Element **superstring**, set the colour number of the colour used for the border of the whiteout box around the vertex text, to be **colour**.

If no whiteout border is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

If there is only one Vertex Text Annotion for all the Vertex Text then the colour number of the colour used for the border of the whiteout box around the vertex text for that one Vertex Text Annotation is set to **c** regardless of the value of **vert**.

A function return value of zero indicates the colour number was successfully set.

ID = 2765

Get_super_vertex_text_border(Element superstring,Integer vert,Integer &c)

Name

Integer Get_super_vertex_text_border(Element superstring,Integer vert,Integer &c)

Description

For vertex number **vert** of the Super String Element **superstring**, get the colour number that is used for the border of the whiteout box around the vertex text. The whiteout border colour is returned as Integer **colour**.

NO_COLOUR is the returned as the colour number if there is no whiteout border.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

If there is only one Vertex Text Annotion for all the Vertex Text then the colour number that is used for the border of the whiteout box around the vertex text for that one Vertex Text Annotation will be returned in **c** regardless of the value of **vert**.

A function return value of zero indicates the colour number was successfully returned.

ID = 2766

Set_super_vertex_textstyle_data(Element super,Integer vert,Textstyle_Data d)

Name

Integer Set_super_vertex_textstyle_data(Element super,Integer vert,Textstyle_Data d)

Description

For the Element **super** of type **Super,** set the Textstyle_Data for the vertex text on vertex number **vert** to be **d**.

Setting a Textstyle_Data means that all the individual values that are contained in the Textstyle_Data are set rather than having to set each one individually.

LJG? if the value is blank in the Textstyle_Data and the value is already set for the vertex text, is the value left alone?

If there is only one Vertex Text Annotion for all the Vertex Text then the Textstyle_Data for that one Vertex Text Annotation is set to **d** regardless of the value of **vert**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Value set.

A function return value of zero indicates the Textstyle_Data was successfully set.

ID = 1663

Get_super_vertex_textstyle_data(Element elt,Integer vert,Textstyle_Data &d)

Name

Integer Get_super_vertex_textstyle_data(Element elt,Integer vert,Textstyle_Data &d)

Description

For the Element **super** of type **Super,** get the Textstyle_Data for the vertex text on vertex number **vert** and return it as **d**.

LJG? if a value is not set in the vertex text, what does it return?

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Vertex_Text_Value set.

If there is only one Vertex Text Annotion for all the Vertex Text then the Textstyle_Data for that

one Vertex Text Annotation will be returned in **d** regardless of the value of **vert**.

A function return value of zero indicates the Textstyle_Data was successfully returned.

ID = 1664

Super String Segment Text and Annotation Functions

See Definitions of Super String Segment Text Dimensions, Units and Annotation Parameters
See Super String Use Segment Text Functions
See Super String Use Segment Annotation Functions
See Setting Super String Segment Text and Annotation Parameters

Definitions of Super String Segment Text Dimensions, Units and Annotation Parameters

Super string Segment text is a special type of text that can only be placed on the *segment* of a super string. Unlike text at a vertex, the segment for segment text has a direction and the segment text is required to be parallel, or related to the segment direction.

If super string segment text is required then the dimension to set is either

(a) the most common case of having a different text on each segment (dimension Att_Segment_Text_Array)

or

(b) the rare case of just the same text that is used for every segment (dimension Att_Segment_Text_Value)

Although segment text may be defined, it will not display in a plan view, or on a plan plot, unless a Segment Text Annotation dimension has been set. A Text Annotation controls the text size, colour, rotation etc.

So if super string segment text is required to be drawn on a plan view then the dimension to set is either

(a) for the case of having a different text annotation for each segment so that the annotation attributes can be modified for each segment then set dimension Att_Segment_Annotate_Array

or

(b) if there is just the one Annotation and its parameters are used for drawing the text on every segment then set the dimension Att_Segment_Annotate_Value.

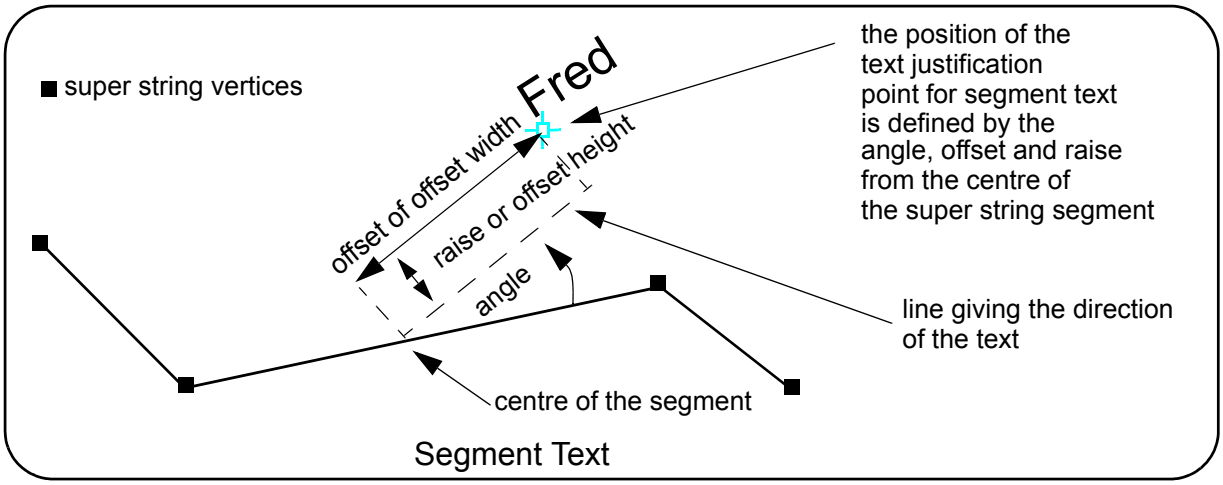
For definitions of the Vertex Text dimensions see Segment Text Dimensions *and the Vertex Text Annotation dimensions see* Segment Text Annotation Dimensions .

Segment Text Annotation Definitions

For segment text, the text ***justification point*** and the ***direction of the text*** are defined by:

(a) the *direction of the text* is given as a *counter clockwise* ***angle*** *of rotation,* measured from the segment, about the centre of the segment

(b) the *justification point* is given as an ***offset*** from the centre of the segment *along the line through the centre of the segment with the direction of the text*, and a perpendicular distance (called the ***raise***) from that offset point to the justification point.



The direction of the text is parallel to the segment if the angle is zero.

Note that these definitions are relative to the segment and if the vertex segment in any way, then the text also moves with it.

The vertex and justification point only coincide if the offset and raise values are both zero.

Finally the text can be one of nine positions defined in relation to the (x,y) coordinates of the text justification point:

		top		
	3	6	9	
left	2	5	8	**right**
	1	4	7	
		bottom		

This is usually an Integer called the ***justification*** with a default value of 1.

Segment Text Annotation Units

The units for text size is specified by an Integer whose value is

(a) 0 (the default) for the units are screen/pixel/device units

(b) 1 for world units

(c) 2 for paper units (millimetres on a plot).

Regardless of whether there is one Segment Text Annotation for the entire string or a different Text Annotation for each segment, there is only one ***units*** for text size used for all the Segment Text of the string.

The units for text are used for the size of the text, and the offsets and raises for the text.

For Information on all the super string segment text and segment text annotations:

See Super String Use Segment Text Functions
See Super String Use Segment Annotation Functions
See Setting Super String Segment Text and Annotation Parameters

Super String Use Segment Text Functions

For definitions of the Segment Text dimensions see Segment Text Dimensions

Set_super_use_segment_text_value(Element super,Integer use)

Name

Integer Set_super_use_segment_text_value(Element super,Integer use)

Description

Tell the super string **super** whether to use (set), or not use (remove) the dimension Att_Segment_Text_Value.

A value for **use** of 1 sets the dimension and 0 removes it.

If Att_Segment_Text_Value is used, then the same text is on all the segments of the super string.

Note if the dimension Att_Segment_Text_Array exists, this call is ignored.

See Vertex Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1239

Get_super_use_segment_text_value(Element super,Integer &use)

Name

Integer Get_super_use_segment_text_value(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Text_Value exists for the super string.

use is returned as 1 if the dimension Att_Segment_Text_Value exists.
use is returned as 0 if the dimension doesn't exist.

If the dimension Att_Segment_Text_Value exists then the string has the same text for every segment of the string.

See Segment Text Dimensions for information on the Segment Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1240

Set_super_use_segment_text_array(Element super,Integer use)

Name

Integer Set_super_use_segment_text_array(Element super,Integer use)

Description

Tell the super string **super** whether to use (set), or not use (remove), the dimension Att_Segment_Text_Array.

A value for **use** of 1 sets the dimension and 0 removes it.

If Att_Segment_Text_Array is used, then there is different text on each segment of the of the string.

See Segment Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1189

Get_super_use_segment_text_array(Element super,Integer &use)

Name

Integer Get_super_use_segment_text_array(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Text_Array exists for the super string **super**.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

If Att_Segment_Text_Array is used, then there is different text on each segment of the of the string.

See Segment Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1190

Super_segment_text_value_to_array(Element super)

Name

Integer Super_segment_text_value_to_array(Element super)

Description

If for the super string **super** the dimension Att_Segment_Text_Value exists and the dimension Att_Segment_Text_Array does not exist then there will be one Segment Text **txt** for the entire string.

In this case (when the dimension Att_Segment_Text_Value exists and the dimension Att_Segment_Text_Array does not exist) this function sets the Att_Segment_Text_Array dimension and new segment text created for each segment of **super** and the new segment text is given the value **txt**.

See Segment Text Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 2179

Super String Use Segment Annotation Functions

For definitions of the Segment Text dimensions see Segment Text Annotation Dimensions

Set_super_use_segment_annotation_value(Element super,Integer use)

Name

Integer Set_super_use_segment_annotation_value(Element super,Integer use)

Description

Tell the super string whether to use or remove, the dimension Att_Segment_Annotate_Value.

If the dimension Att_Segment_Annotate_Value exists and the dimension Att_Segment_Annotate_Array doesn't exist then the string has the one annotation which is used for segment text on **any** segment of the string.

See Vertex Text Annotation Dimensions for information on the Text Annotation dimensions or Super String Dimensions for information on all the dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

Note if the dimension Att_Segment_Annotate_Array exists, this call is ignored.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1193

Get_super_use_segment_annotation_value(Element super,Integer &use)

Name

Integer Get_super_use_segment_annotation_value(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Annotate_Value exists for the super string.

If the dimension Att_Segment_Annotate_Value exists and the dimension Att_Segment_Annotate_Array doesn't exist then the string has the one annotation which is used for segment text on **any** segment of the string.

See Vertex Text Annotation Dimensions for information on the Text Annotation dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1194

Set_super_use_segment_annotation_array(Element super,Integer use)

Name

Integer Set_super_use_segment_annotation_array(Element super,Integer use)

Description

Tell the super string whether to use or remove the dimension Att_Segment_Annotate_Array.

If the dimension Att_Segment_Annotate_Array exists then the string has a different annotation for the segment text on each segment of the string.

See Vertex Text Annotation Dimensions for information on the Text Annotation dimensions or Super String Dimensions for information on all the dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1195

Get_super_use_segment_annotation_array(Element super,Integer &use)

Name

Integer Get_super_use_segment_annotation_array(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Annotate_Array exists for the super string.

If the dimension Att_Segment_Annotate_Array exists then the string has a different annotation

for the segment text on each segment of the string.

See Vertex Text Annotation Dimensions for information on the Text Annotation dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1196

Super_segment_annotate_value_to_array(Element super)

Name

Integer Super_segment_annotate_value_to_array(Element super)

Description

If for the super string **super** the dimension Att_Segment_Annotate_Value exists and the dimension Att_Segment_Annotate_Array does not exist then there will be one Segment Text Annotate **annot** for the entire string.

In this case (when the dimension Att_Segment_Annotate_Value exists and the dimension Att_Segment_Annotate_Array does not exist) this function sets the Att_Segment_Annotate_Array dimension and new segment Annotates created for each segment of **super** and the new segment text Annotate is given the value **annot**

See Segment Text Annotation Dimensions for information on the Text dimensions or Super String Dimensions for information on all the dimensions.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 2180

Setting Super String Segment Text and Annotation Parameters

Set_super_segment_text(Element super,Integer seg,Text text)

Name

Integer Set_super_segment_text(Element super,Integer seg,Text text)

Description

For the super Element **super**, set the segment text at segment number **seg** to be **txt**.

If there is only one Segment Text for all the segments then the text for that one Segment Text is set to **txt** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1191

Get_super_segment_text(Element super,Integer seg,Text &text)

Name

Integer Get_super_segment_text(Element super,Integer seg,Text &text)

Description

For the super Element **super**, return in **txt** the segment text on segment number **seg**.

If there is only one Segment Text for all the segments then the text for that one Segment Text will be returned in **txt** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1192

Set_super_segment_world_text(Element super)

Name

Integer Set_super_segment_world_text(Element super)

Description

For an Element **super** of type Super, set the text unit for segment text to be world text.

See Segment Text Annotation Units for the definition of segment text units.

If there is Textstyle_Data for the segment text then this will override the *Set_super_segment_world_text* call.

A return value of 0 indicates the function call was successful.

ID = 1233

Set_super_segment_device_text(Element super)

Name

Integer Set_super_segment_device_text(Element super)

Description

For an Element **super** of type Super, set the text unit for segment text to be pixels (also known as device text or screen text).

See Segment Text Annotation Units for the definition of segment text units.

If there is Textstyle_Data for the segment text then this will override the *Set_super_segment_device_text* call.

A return value of 0 indicates the function call was successful.

ID = 1232

Set_super_segment_paper_text(Element super)

Name

Integer Set_super_segment_paper_text(Element super)

Description

For an Element **super** of type Super, set the text units for segment text to be paper (that is millimetres).

See Segment Text Annotation Units for the definition of segment text units.

If there is Textstyle_Data for the segment text then this will override the *Set_super_segment_device_text* call.

A return value of 0 indicates the function call was successful.

ID = 1634

Set_super_segment_text_type(Element super,Integer type)

Name

Integer Set_super_segment_text_type(Element super,Integer type)

Description

For the super Element **super**, set the segment text units to the value **type**.

See Segment Text Annotation Units for the definition of segment text units.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1234

Get_super_segment_text_type(Element super,Integer &type)

Name

Integer Get_super_segment_text_type(Element super,Integer &type)

Description

For the super Element **super**, return in **type** the value of the segment text units.

See Segment Text Annotation Units for the definition of vertex text units.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1235

Set_super_segment_text_justify(Element super,Integer seg,Integer just)

Name

Integer Set_super_segment_text_justify(Element super,Integer seg,Integer just)

Description

For the super string **super**, set the justification of the segment text on segment number **seg** to **just**.

See Segment Text Annotation Definitions for the definition of justification.

If there is only one Segment Text Annotation for all the Segment Text then the justification for that one Segment Text Annotation is set to **just** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1197

Get_super_segment_text_justify(Element super,Integer seg,Integer &just)

Name

Integer Get_super_segment_text_justify(Element super,Integer seg,Integer &just)

Description

For the super string **super**, return the justification of the segment text on segment number **seg** in **just**.

See Segment Text Annotation Definitions for the definition of justification.

If there is only one Segment Text Annotation for all the Segment Text then the justification for that

one Segment Text Annotation will be returned in **just** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1198

Set_super_segment_text_offset_width(Element super,Integer seg,Real off)

Name

Integer Set_super_segment_text_offset_width(Element super,Integer seg,Real o)ff

Description

For the super string **super**, set the offset (offset width) of the segment text on segment number **seg** to **off**.

See Segment Text Annotation Definitions for the definition of offset.

If there is only one Segment Text Annotation for all the Segment Text then the offset for that one Segment Text Annotation is set to **off** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1199

Get_super_segment_text_offset_width(Element super,Integer seg,Real &off)

Name

Integer Get_super_segment_text_offset_width(Element super,Integer seg,Real &off)

Description

For the super string **super**, return the offset (offset width) of the segment text on segment number **seg** in **off**.

See Segment Text Annotation Definitions for the definition of offset.

If there is only one Segment Text Annotation for all the Segment Text then the offset for that one Segment Text Annotation will be returned in **off** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1200

Set_super_segment_text_offset_height(Element super,Integer seg,Real raise)

Name

Integer Set_super_segment_text_offset_height(Element super,Integer seg,Real raise)

Description

For the super string **super**, set the raise (offset height) of the segment text on segment number **seg** to **raise**.

See Segment Text Annotation Definitions for the definition of raise.

If there is only one Segment Text Annotation for all the Segment Text then the raise for that one Segment Text Annotation is set to **raise** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1201

Get_super_segment_text_offset_height(Element super,Integer seg,Real &raise)

Name

Integer Get_super_segment_text_offset_height(Element super,Integer seg,Real &raise)

Description

For the super string **super**, return the raise (offset height) of the segment text on segment number **seg** in **raise**.

See Segment Text Annotation Definitions for the definition of raise.

If there is only one Segment Text Annotation for all the Segment Text then the raise for that one Segment Text Annotation will be returned in **raise** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1202

Set_super_segment_text_colour(Element super,Integer seg,Integer col)

Name

Integer Set_super_segment_text_colour(Element super,Integer seg,Integer col)

Description

For the super string **super**, set the colour number of the segment text on segment number **seg** to **col**.

If there is only one Segment Text Annotation for all the Segment Text then the colour number for that one Segment Text Annotation is set to **col** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1213

Get_super_segment_text_colour(Element super,Integer seg,Integer &col)

Name

Integer Get_super_segment_text_colour(Element super,Integer seg,Integer &col)

Description

For the super string **super**, return the colour number of the segment text on segment number **seg** in **col**.

If there is only one Segment Text Annotation for all the Segment Text then the colour number for that one Segment Text Annotation will be returned in **col** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1214

Set_super_segment_text_angle(Element super,Integer seg,Real ang)

Name

Integer Set_super_segment_text_angle(Element super,Integer seg,Real ang)

Description

For the super string **super**, set the angle of rotation of the segment text on segment number **seg** to **ang**.

See Segment Text Annotation Definitions for the definition of angle. **ang** is measured in radians and is measured counterclockwise from the direction of the segment.

If there is only one Segment Text Annotation for all the Segment Text then the angle for that one Segment Text Annotation is set to **angle** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1203

Get_super_segment_text_angle(Element super,Integer seg,Real &ang)

Name

Integer Get_super_segment_text_angle(Element super,Integer seg,Real &ang)

Description

For the super string **super**, return the angle of rotation of the segment text on segment number **seg** in **ang**.

See Segment Text Annotation Definitions for the definition of angle. **ang** is measured in radians and is measured counterclockwise from the direction of the segment.

If there is only one Segment Text Annotation for all the Segment Text then angle for that one Segment Text Annotation will be returned in **ang** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1204

Set_super_segment_text_size(Element super,Integer seg,Real sz)

Name

Integer Set_super_segment_text_size(Element super,Integer seg,Real sz)

Description

For the super string **super**, set the size of the segment text on segment number **seg** to **sz**.

If there is only one Segment Text Annotation for all the Segment Text then the size for that one Segment Text Annotation is set to **sz** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1205

Get_super_segment_text_size(Element super,Integer seg,Real &sz)

Name

Integer Get_super_segment_text_size(Element super,Integer seg,Real &sz)

Description

For the super string **super**, return the size of the segment text on segment number **seg** in **sz**.

If there is only one Segment Text Annotation for all the Segment Text then size for that one Segment Text Annotation will be returned in **sz** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1206

Set_super_segment_text_x_factor(Element super,Integer seg,Real xf)

Name

Integer Set_super_segment_text_x_factor(Element super,Integer seg,Real xf)

Description

For the super string **super**, set the x factor of the segment text on segment number **seg** to **xf**.

If there is only one Segment Text Annotation for all the Segment Text then the x factor for that one Segment Text Annotation is set to **xf** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1207

Get_super_segment_text_x_factor(Element super,Integer seg,Real &xf)

Name

Integer Get_super_segment_text_x_factor(Element super,Integer seg,Real &xf)

Description

For the super string **super**, return the x factor of the segment text on segment number **seg** in **xf**.

If there is only one Segment Text Annotation for all the Segment Text then the x factor for that one Segment Text Annotation will be returned in **xf** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1208

Set_super_segment_text_slant(Element super,Integer seg,Real sl)

Name

Integer Set_super_segment_text_slant(Element super,Integer seg,Real sl)

Description

For the super string **super**, set the slant of the segment text on segment number **seg** to **sl**.

If there is only one Segment Text Annotation for all the Segment Text then the slant for that one Segment Text Annotation is set to **sl** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1209

Get_super_segment_text_slant(Element super,Integer seg,Real &sl)

Name

Integer Get_super_segment_text_slant(Element super,Integer seg,Real &sl)

Description

For the super string **super**, return the slant of the segment text on segment number **seg** in **sl**.

If there is only one Segment Text Annotation for all the Segment Text then the slant for that one Segment Text Annotation will be returned in **sl** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1210

Set_super_segment_text_style(Element super,Integer seg,Text ts)

Name

Integer Set_super_segment_text_style(Element super,Integer seg,Text ts)

Description

For the super string **super**, set the textstyle of the segment text on segment number **seg** to **ts**.

If there is only one Segment Text Annotation for all the Segment Text then the textstyle for that one Segment Text Annotation is set to **ts** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1211

Get_super_segment_text_style(Element super,Integer seg,Text &ts)

Name

Integer Get_super_segment_text_style(Element super,Integer seg,Text &ts)

Description

For the super string **super**, return the textstyle of the segment text on segment number **seg** in **ts**.

If there is only one Segment Text Annotation for all the Segment Text then the textstyle for that one Segment Text Annotation will be returned in **ts** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A return value of 0 indicates the function call was successful.

ID = 1212

Set_super_segment_text_ttf_underline(Element super,Integer seg, Integer underline)

Name

Integer Set_super_segment_text_ttf_underline(Element super,Integer seg,Integer underline)

Description

For the super string **super**, set the underline state of the segment text on segment number **seg** to **underline**.

If **underline** = 1, then for a true type font the text will be underlined.
If **underline** = 0, then text will not be underlined.

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the underline state for

that one Segment Text Annotation is set to **underline** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates **underline** was successfully set.

ID = 2608

Get_super_segment_text_ttf_underline(Element super,Integer seg, Integer &underline)

Name

Integer Get_super_segment_text_ttf_underline(Element super,Integer seg,Integer &underline)

Description

For the super string **super**, return the underline state of the segment text on segment number **seg** in **underline**.

If **underline** = 1, then for a true type font the text will be underlined.
If **underline** = 0, then text will not be underlined.

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the underline state for that one Segment Text Annotation will be returned in **underline** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates **underline** was successfully returned.

ID = 2609

Set_super_segment_text_ttf_strikeout(Element super,Integer seg,Integer strikeout)

Name

Integer Set_super_segment_text_ttf_strikeout(Element super,Integer seg,Integer strikeout)

Description

For the super string **super**, set the strikeout state of the segment text on segment number **seg** to **strikeout**.

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the strikeout state for that one Segment Text Annotation is set to **strikeout** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates strikeout was successfully set.

ID = 2610

Get_super_segment_text_ttf_strikeout(Element super,Integer seg, Integer &strikeout)

Name

Integer Get_super_segment_text_ttf_strikeout(Element super,Integer seg,Integer &strikeout)

Description

For the super string **super**, return the strikeout state of the segment text on segment number **seg**

in **strikeout**.

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the strikeout state for that one Segment Text Annotation will be returned in **strikeout** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates **strikeout** was successfully returned.

ID = 2611

Set_super_segment_text_ttf_italic(Element super,Integer seg,Integer italic)

Name

Integer Set_super_segment_text_ttf_italic(Element super,Integer seg,Integer italic)

Description

For the super string **super**, set the italic state of the segment text on segment number **seg** to **italic**.

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the italic state for that one Segment Text Annotation is set to **italic** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates italic was successfully set.

ID = 2612

Get_super_segment_text_ttf_italic(Element super,Integer seg,Integer &italic)

Name

Integer Get_super_segment_text_ttf_italic(Element super,Integer seg,Integer &italic)

Description

For the super string **super**, return the italic state of the segment text on segment number **seg** in **italic**.

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the italic state for that one Segment Text Annotation will be returned in **italic** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates **italic** was successfully returned.

ID = 2613

Set_super_segment_text_ttf_outline(Element elt,Integer seg,Integer outline)

Name

Integer Set_super_segment_text_ttf_outline(Element elt,Integer seg,Integer outline)

Description

For the super string **super**, set the outline state of the segment text on segment number **seg** to **outline**.

If **outline** = 1, then for a true type font the text will be only shown in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the outline state for that one Segment Text Annotation is set to **outline** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates *outline* was successfully set.

ID = 2777

Get_super_segment_text_ttf_outline(Element elt,Integer seg,Integer &outline)

Name

Integer Get_super_segment_text_ttf_outline(Element elt,Integer seg,Integer &outline)

Description

For the super string **super**, return the outline state of the segment text on segment number **seg** in **outline**.

If **outline** = 1, then for a true type font the text will be shown only in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the outline state for that one Segment Text Annotation will be returned in **outline** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates **outline** was successfully returned.

ID = 2778

Set_super_segment_text_ttf_weight(Element super,Integer seg,Integer weight)

Name

Integer Set_super_segment_text_ttf_weight(Element super,Integer seg,Integer weight)

Description

For the super string **super**, set the weight of the segment text on segment number **seg** to **weight**.

If there is only one Segment Text Annotation for all the Segment Text then the weight for that one Segment Text Annotation is set to **weight** regardless of the value of **seg**.

For the list of allowable weights, go to Allowable Weights

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates *weight* was successfully set.

ID = 2614

Get_super_segment_text_ttf_weight(Element super,Integer seg,Integer &weight)

Name

Integer Get_super_segment_text_ttf_weight(Element super,Integer seg,Integer &weight)

Description

For the super string **super**, return the weight of the segment text on segment number **seg** in **weight**.

For the list of allowable weights, go to Allowable Weights

If there is only one Segment Text Annotation for all the Segment Text then the weight for that one Segment Text Annotation will be returned in **weight** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates **weight** was successfully returned.

ID = 2615

Set_super_segment_text_whiteout(Element superstring,Integer seg,Integer c)

Name

Integer Set_super_segment_text_whiteout(Element superstring,Integer seg,Integer c)

Description

For the super string **super**, set the colour number of the colour used for the whiteout box around the segment text on segment number **seg** to **c**.

If no text whiteout is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the colour number of the colour used for the whiteout box around the segment text for that one Segment Text Annotation is set to **c** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates the colour number was successfully set.

ID = 2757

Get_super_segment_text_whiteout(Element superstring,Integer seg,Integer &c)

Name

Integer Get_super_segment_text_whiteout(Element superstring,Integer seg,Integer &c)

Description

For the super string **super**, return the colour number that is used for the whiteout box around the segment text on segment number **seg** in **c**.

NO_COLOUR is the returned as the colour number if whiteout is not being used.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the colour number that is used for the whiteout box around the segment text for that one Segment Text Annotation will be returned in **c** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates the colour number was successfully returned.

ID = 2758

Set_super_segment_text_border(Element superstring,Integer seg,Integer c)

Name

Integer Set_super_segment_text_border(Element superstring,Integer seg,Integer c)

Description

For the super string **super**, set the colour number of the colour used for the border of the whiteout box around the segment text on segment number **seg** to **c**.

If no text whiteout border is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the colour number of the colour used for border of the whiteout box around the segment text for that one Segment Text Annotation is set to **c** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates the colour number was successfully set.

ID = 2767

Get_super_segment_text_border(Element superstring,Integer seg,Integer &c)

Name

Integer Get_super_segment_text_border(Element superstring,Integer seg,Integer &c)

Description

For the super string **super**, return the colour number that is used as the border of the whiteout box around the segment text on segment number **seg** in **c**.

NO_COLOUR is the returned as the colour number if whiteout is not being used.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

If there is only one Segment Text Annotation for all the Segment Text then the colour number that is used for the border around the whiteout box around the segment text for that one Segment Text Annotation will be returned in **c** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates the colour number was successfully returned.

ID = 2768

Set_super_segment_textstyle_data(Element elt,Integer seg,Textstyle_Data d)

Name

Integer Set_super_segment_textstyle_data(Element elt,Integer seg,Textstyle_Data d)

Description

For the super string **super**, set the Textstyle_Data of the segment text on segment number **seg** to **d**.

Setting a Textstyle_Data means that all the individual values that are contained in the Textstyle_Data are set rather than having to set each one individually.

LJG? if the value is blank in the Textstyle_Data and the value is already set for the segment text, is the value left alone?

If there is only one Segment Text Annotation for all the Segment Text then the Textstyle_Data for

that one Segment Text Annotation is set to **d** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates the Textstyle_Data was successfully set.

ID = 1665

Get_super_segment_textstyle_data(Element elt,Integer seg,Textstyle_Data &d)

Name

Integer Get_super_segment_textstyle_data(Element elt,Integer seg,Textstyle_Data &d)

Description

For the super string **super**, return the Textstyle_Data for the segment text on segment number **seg** in **d**.

Using a Textstyle_Data means that all the individual values for the Segment Text Annotation are returned in the Textstyle_Data rather than getting each one individually.

LJG? if a value is not set in the segment text, what does it return?

If there is only one Segment Text Annotation for all the Segment Text then the Textstyle_Data for that one Segment Text Annotation will be returned in **d** regardless of the value of **seg**.

A non-zero function return value is returned if **super** is not of type **Super**.

A function return value of zero indicates the Textstyle_Data was successfully returned.

ID = 1666

Super String Fills - Hatch/Solid/Bitmap/Pattern/ACAD Pattern Functions

For definitions of the Solid, Bitmap, Hatch and Fill dimensions, see Solid/Bitmap/Hatch/ Fill/Pattern/ ACAD Pattern Dimensions

See Super String Hatch Functions
See Super String Solid Fill Functions
See Super String Bitmap Functions
See Super String Patterns Functions
See Super String ACAD Patterns Functions

Super String Hatch Functions

Set_super_use_hatch(Element super,Integer use)

Name

Integer Set_super_use_hatch(Element super,Integer use)

Description

For the super string Element **super**, define whether the dimension Att_Hatch_Value is used or removed.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the super string can have 2 angle hatching.
If **use** is 0, the dimension is removed. If the string had hatching then the hatching will be removed.

A return value of 0 indicates the function call was successful.

ID = 1464

Get_super_use_hatch(Element super,Integer &use)

Name

Integer Get_super_use_hatch(Element super,Integer &use)

Description

Query whether the dimension Att_Hatch_Value exists for the super string **super**.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists and hatching is enabled for the string.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1465

Set_super_hatch_colour(Element super,Integer col_1,Integer col_2)

Name

Integer Set_super_hatch_colour(Element super,Integer col_1,Integer col_2)

Description

For the super Element **super**, set the colour of the first hatch lines to the Integer colour **col_1**and the colour of the second hatch lines to the Integer colour **col_2**.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1466

Get_super_hatch_colour(Element super,Integer &col_1,Integer &col_2)

Name

Integer Get_super_hatch_colour(Element super,Integer &col_1,Integer &col_2)

Description

For the super Element **super**, return the colour of the first hatch lines as **col_1** and the colour of the second hatch lines as **col_2**.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1467

Set_super_hatch_angle(Element super,Real ang_1,Real ang_2)

Name

Integer Set_super_hatch_angle(Element super,Real ang_1,Real ang_2)

Description

For the super Element **super**, set the angle of the first hatch lines to the angle **ang_1** and the angle of the second hatch lines to the angle **ang_2**. The angles are in radians and measured counterclockwise from the x-axis.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1468

Get_super_hatch_angle(Element super,Real &ang_1,Real &ang_2)

Name

Integer Get_super_hatch_angle(Element super,Real &ang_1,Real &ang_2)

Description

For the super Element **super**, return the angle of the first hatch lines as **ang_1** and the angle of the second hatch lines as **ang_2**. The angles are in radians and measured counterclockwise from the x-axis.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1469

Set_super_hatch_spacing(Element super,Real dist_1,Real dist_2)

Name

Integer Set_super_hatch_spacing(Element super,Real dist_1,Real dist_2)

Description

For the super Element **super**, set the distance between the first hatch lines to the **dist_1** and the

distance between the second hatch lines of **dist_2**. The units for **dist_1** and **dist_2** are given by other calls.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1470

Get_super_hatch_spacing(Element super,Real &dist_1,Real &dist_2)

Name

Integer Get_super_hatch_spacing(Element super,Real &dist_1,Real &dist_2)

Description

For the super Element **super**, return the distance of the first hatch lines as **dist_1** and the distance of the second hatch lines as **dist_2**. The units for **dist_1** and **dist_2** are given by other calls.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1471

Set_super_hatch_origin(Element super,Real x,Real y)

Name

Integer Set_super_hatch_origin(Element super,Real x,Real y)

Description

For the super Element **super**, both sets of hatch lines go through the point (**x**,**y**). The units for **x** and **y** are given by other calls.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1472

Get_super_hatch_origin(Element super,Real &x,Real &y)

Name

Integer Get_super_hatch_origin(Element super,Real &x,Real &y)

Description

For the super Element **super**, return the origin that both sets of hatch lines go through as (**x**,**y**). The units for **x** and **y** are given by other calls.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1473

Set_super_hatch_device(Element super)

Name

Integer Set_super_hatch_device(Element super)

Description

For the super Element **super**, set the units for the hatch spacing and the hatch origin to be device units.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1474

Set_super_hatch_world(Element super)

Name

Integer Set_super_hatch_world(Element super)

Description

For the super Element **super**, set the units for the hatch spacing and the hatch origin to be world units.

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1475

Set_super_hatch_type(Element super,Integer type)

Name

Integer Set_super_hatch_type(Element super,Integer type)

Description

For the super Element **super**, set the units for the hatch spacing and the hatch origin to be:

if type = 0 then device units
if type = 1 then world units
if type = 2 then paper units

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1476

Get_super_hatch_type(Element super,Integer &type)

Name

Integer Get_super_hatch_type(Element super,Integer &type)

Description

For the super Element **super**, get the units for the hatch spacing and the hatch origin. The units are returned as **type** and the values are:

if type = 0 then device units
if type = 1 then world units
if type = 2 then paper units

If hatching is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1477

Super String Solid Fill Functions

Set_super_use_solid(Element super,Integer use)

Name

Integer Set_super_use_solid(Element super,Integer use)

Description

For the super string Element **super**, define whether the dimension Att_Solid_Value is used or removed.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the super string can have solid fill.
If **use** is 0, the dimension is removed. If the string had solid fill then the solid fill will be removed.

A return value of zero indicates the function call was successful.

ID = 1478

Get_super_use_solid(Element super,Integer &use)

Name

Integer Get_super_use_solid(Element super,Integer &use)

Description

Query whether the dimension Att_Solid_Value exists for the super string **super**.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists and solid fill is enabled for the string.
use is returned as 0 if the dimension doesn't exist.

A return value of zero indicates the function call was successful.

ID = 1479

Set_super_solid_colour(Element super,Integer colour)

Name

Integer Set_super_solid_colour(Element super,Integer colour)

Description

For the super Element **super**, set the colour of the solid fill to the colour number **colour.**

If solid fill is not enabled for **super**, then a non-zero return code is returned.

A return value of zero indicates the function call was successful.

ID = 1480

Get_super_solid_colour(Element super,Integer &colour)

Name

Integer Get_super_solid_colour(Element super,Integer &colour)

Description

For the super Element **super**, get the colour number of the solid fill and return it in **colour.**

If solid fill is not enabled for **super**, then a non-zero return code is returned.

A return value of zero indicates the function call was successful.

ID = 1481

Set_super_solid_blend(Element super,Real blend)

Name

Integer Set_super_solid_blend(Element super,Real blend)

Description

For the super Element **super**, set the blend of the solid fill to the **blend.**

If solid fill is not enabled for **super**, then a non-zero return code is returned.

A return value of zero indicates the function call was successful.

ID = 2165

Get_super_solid_blend(Element super,Real &blend)

Name

Integer Get_super_solid_blend(Element super,Real &blend)

Description

For the super Element **super**, get the blend value of the solid fill and return it in **blend.**

blend will have a value between 0.0 for showing no colour fill, and 1.0 for showing full colour fill.

If solid fill is not enabled for **super**, then a non-zero return code is returned.

A return value of zero indicates the function call was successful.

ID = 2166

Super String Bitmap Functions

Set_super_use_bitmap(Element super,Integer use)

Name

Integer Set_super_use_bitmap(Element super,Integer use)

Description

For the super string Element **super**, define whether the dimension Att_Bitmap_Value is used or removed.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the super string can have bitmap fill.
If **use** is 0, the dimension is removed. If the string had a bitmap fill then the bitmap fill will be removed.

A return value of zero indicates the function call was successful.

ID = 1482

Get_super_use_bitmap(Element super,Integer &use)

Name

Integer Get_super_use_bitmap(Element super,Integer &use)

Description

Query whether the dimension Att_Bitmap_Value exists for the super string **super**.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists and bitmap fill is enabled for the string.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1483

Set_super_bitmap(Element super,Text filename)

Name

Integer Set_super_bitmap(Element super,Text filename)

Description

For the super Element **super**, set the bitmap to be the image in the file of name **filename**.

The image can be bmps or ?.

If bitmap fill is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1484

Get_super_bitmap(Element super,Text &filename)

Name

Integer Get_super_bitmap(Element super,Text &filename)

Description

For the super Element **super**, get the file name of the bitmap fill and return it in **filename.**

If bitmap fill is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1485

Set_super_bitmap_origin(Element super,Real x,Real y)

Name

Integer Set_super_bitmap_origin(Element super,Real x,Real y)

Description

For the super Element **super**, the left hand corner of the bitmap is placed at the point (**x**,**y**). The units for **x** and **y** are given in other functions.

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1486

Get_super_bitmap_origin(Element super,Real &x,Real &y)

Name

Integer Get_super_bitmap_origin(Element super,Real &x,Real &y)

Description

For the super Element **super**, return the (**x**,**y**) point of the left hand corner of the bitmap. The units for **x** and **y** are given in other functions.

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1487

Set_super_bitmap_transparent(Element super,Integer colour)

Name

Integer Set_super_bitmap_transparent(Element super,Integer colour)

Description

For the super Element **super**, set the colour with colour number **colour** to be transparent in the bitmap.

If bitmap fill is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1488

Get_super_bitmap_transparent(Element super,Integer &colour)

Name

Integer Get_super_bitmap_transparent(Element super,Integer &colour)

Description

For the super Element **super**, get the transparency colour and return it in **colour.**

If bitmap fill is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1489

Set_super_bitmap_device(Element super)

Name

Integer Set_super_bitmap_device(Element super)

Description

For the super Element **super**, set the units for the bitmap width and height to be device units.

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1490

Set_super_bitmap_world(Element super)

Name

Integer Set_super_bitmap_world(Element super)

Description

For the super Element **super**, set the units for the width and height of the bitmap to be world units.

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1491

Set_super_bitmap_type(Element super,Integer type)

Name

Integer Set_super_bitmap_type(Element super,Integer type)

Description

For the super Element **super**, set the units for the width and height of the bitmap to be:

if type = 0 then device units
if type = 1 then world units
if type = 2 then paper units

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1492

Get_super_bitmap_type(Element super,Integer &type)

Name

Integer Get_super_bitmap_type(Element super,Integer &type)

Description

For the super Element **super**, get the units for width and height of the bitmap. The units are returned as **type** and the values are:

if type = 0 then device units
if type = 1 then world units
if type = 2 then paper units

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1493

Set_super_bitmap_angle(Element super,Real ang)

Name

Integer Set_super_bitmap_angle(Element super,Real ang)

Description

For the super Element **super**, set the angle to rotate the bitmap to be **ang**. The angle is in radians and measured counterclockwise from the x-axis

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1494

Get_super_bitmap_angle(Element super,Real &ang)

Name

Integer Get_super_bitmap_angle(Element super,Real &ang)

Description

For the super Element **super**, get the angle of rotation of bitmap and return it in **ang**. The angle is in radians and measured counterclockwise from the x-axis

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1495

Set_super_bitmap_size(Element super,Real w,Real h)

Name

Integer Set_super_bitmap_size(Element super,Real w,Real h)

Description

For the super Element **super**, scale the bitmap to have the width **w** and height **h** in the units set in other bitmap calls.

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1496

Get_super_bitmap_size(Element super,Real &w,Real &h)

Name

Integer Get_super_bitmap_size(Element super,Real &w,Real &h)

Description

For the super Element **super**, get the width and height that the bitmap was scaled to. The width is returned in **w** and the height in **h**. The units have been set in other bitmap calls.

If bitmap is not enabled for **super**, then a non-zero return code is returned.

A return value of 0 indicates the function call was successful.

ID = 1497

Super String Patterns Functions

For definitions of the Pattern dimension, see Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions

Set_super_use_pattern(Element super,Integer use)

Name

Integer Set_super_use_pattern(Element super,Integer use)

Description

For the super string Element super, define whether the dimension Att_Pattern_Value is used or removed.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the super string can have a pattern.

If **use** is 0, the dimension is removed. If the string had a pattern then the pattern will be removed.

A return value of 0 indicates the function call was successful.

ID = 1686

Get_super_use_pattern(Element super,Integer &use)

Name

Integer Get_super_use_pattern(Element super,Integer &use)

Description

Query whether the dimension Att_Pattern_Value exists for the super string **super**.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1693

Super String ACAD Patterns Functions

For definitions of the ACAD Pattern dimension, see Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions

Set_super_use_acad_pattern(Element super,Integer use)

Name

Integer Set_super_use_acad_pattern(Element super,Integer use)

Description

For the super string Element super, define whether the dimension Att_Autocad_Pattern_Value is used or removed.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the super string can have an Autocad pattern.
If **use** is 0, the dimension is removed. If the string had an Autocad pattern then the Autocad pattern will be removed.

A return value of 0 indicates the function call was successful.

ID = 2141

Get_super_use_acad_pattern(Element super,Integer &use)

Name

Integer Get_super_use_acad_pattern(Element super,Integer &use)

Description

Query whether the dimension Att_Autocad_Pattern_Value exists for the super string super.

See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions for information on this dimension or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 2142

Super String Hole Functions

For definitions of the Hole dimension, see Hole Dimension

Set_super_use_hole(Element super,Integer use)

Name

Integer Set_super_use_hole(Element super,Integer use)

Description

For the super string Element **super**, define whether the dimension Att_Hole_Value is used or removed.

See Hole Dimension for information on the hole dimension or Super String Dimensions for information on all dimensions.

If **use** is 1, the dimension is set. That is, the super string can have holes.
If **use** is 0, the dimension is removed. If the string had holes then the holes will be removed.

A return value of 0 indicates the function call was successful.

ID = 1456

Get_super_use_hole(Element super,Integer &use)

Name

Integer Get_super_use_hole(Element super,Integer &use)

Description

Query whether the dimension Att_Hole_Value exists for the super string **super**.

See Hole Dimension for information on hole dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1457

Super_add_hole(Element super,Element hole)

Name

Integer Super_add_hole (Element super,Element hole)

Description

Add the Element **hole** as a hole to the super Element **super**.

The operation will fail if **super** already belongs to a model and a non-zero return value returned. So if an existing string in a model is to be used as a hole, the string must be copied and the copy used as the hole.

A return value of zero indicates the function call was successful.

ID = 1460

Get_super_holes(Element super,Integer &numberless)

Name

Integer Get_super_holes(Element super,Integer &numberless)

Description

For the Element **super** of type **Super**, the number of holes for the super string is returned as **no_holes.**

If holes are **not** enabled for the super string then a non-zero return code is returned and no_holes is set to 0.

A return value of 0 indicates the function call was successful.

ID = 1458

Super_get_hole(Element super,Integer hole_no,Element &hole)

Name

Integer Super_get_hole(Element super,Integer hole_no,Element &hole)

Description

For the Element **super** of type **Super**, the holes number **hole_no** is returned as the super Element **hole**.

If **hole** needs to be used in *12d Model* and added to a model, then the Element **hole** must be copied and added to the model.

If **hole_no** is less than zero or greater than the number of holes in **super**, then a non-zero return code is returned. The Element **hole** is then undefined.

A return value of 0 indicates the function call was successful.

ID = 1459

Super_delete_hole(Element super,Element hole)

Name

Integer Super_delete_hole(Element super,Element hole)

Description

If Super_get_hole is used to get the hole **hole** from the Element **super** then this option can be used to delete **hole** from **super.**

A return value of zero indicates the function call was successful.

ID = 1461

Super_delete_hole(Element super,Integer hole_no)

Name

Integer Super_delete_hole(Element super,Integer hole_no)

Description

Delete the hole number **hole_no** from the Element **super.**

If there is no hole **hole_no**, the operation will fail and a non-zero return value is returned.

A return value of zero indicates the function call was successful.

ID = 1462

Super_delete_all_holes(Element super)

Name

Integer Super_delete_all_holes(Element super)

Description

Delete all the holes from the Element **super.**

A return value of 0 indicates the function call was successful.

ID = 1463

Super String Segment Colour Functions

For definitions of the Colour dimension, see Colour Dimension

Set_super_use_segment_colour(Element super,Integer use)

Name

Integer Set_super_use_segment_colour(Element super,Integer use)

Description

Tell the super string whether to use or remove the colour dimension Att_Colour_Array.

A value for **use** of 1 sets the dimension and 0 removes it.

See Colour Dimension for information on Colour dimensions or Super String Dimensions for information on all dimensions.

A return value of 0 indicates the function call was successful.

ID = 726

Get_super_use_segment_colour(Element super,Integer &use)

Name

Integer Get_super_use_segment_colour(Element super,Integer &use)

Description

Query whether the colour dimension Att_Colour_Array exists for the super string.

use is returned as 1 if the dimension Att_Colour_Array exists, or 0 if the dimension doesn't exist.

See Colour Dimension for information on Colour dimensions or Super String Dimensions for information on all dimensions.

A return value of 0 indicates the function call was successful.

ID = 727

Set_super_segment_colour(Element super,Integer seg,Integer colour)

Name

Integer Set_super_segment_colour(Element super,Integer seg,Integer colour)

Description

For the Element **super** of type **Super,** set the colour number for the segment number **seg** to be **colour**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the colour dimension Att_Colour_Array set.

See Colour Dimension for information on Colour dimensions or Super String Dimensions for information on all dimensions.

A function return value of zero indicates **colour** was successfully set.

ID = 728

Get_super_segment_colour(Element super,Integer seg,Integer &colour)

Name

Integer Get_super_segment_colour(Element super,Integer seg,Integer &colour)

Description

For the Element **super** of type **Super,** get the colour number for the segment number **seg** and return it as **colour.**

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the colour dimension Att_Colour_Array set.

See Colour Dimension for information on Colour dimensions or Super String Dimensions for information on all dimensions.

A function return value of zero indicates **colour** was successfully returned.

ID = 729

Super String Segment Geometry Functions

For definitions of the Segment Geometry dimension, see Segment Geometry Dimension

To allow transitions to be used between vertices of a super string, the use of a Segment between vertices was introduced for super strings (see Segments).

Set_super_use_segment_geometry(Element super,Integer use)

Name

Integer Set_super_use_segment_geometry(Element super,Integer use)

Description

For the super string Element **super**, define whether the dimension Att_Geom_Array is used or removed.

If Att_Geom_Array exists, the string can have Segments (which can be straights, arcs or **transitions**) between the vertices of the super string.

See Segment Geometry Dimension for information on the Segment Geometry dimensions or Super String Dimensions for information on all the dimensions.

If **use** is **1**, the dimension is set. That is, the segments of the super string are not just straights but of type Segments (which can be straights, arcs or **transitions**).

If **use** is **0**, the dimension is removed. If the string had Segments for segments then they will be removed.

A return value of 0 indicates the function call was successful.

ID = 1838

Get_super_use_segment_geometry(Element super,Integer &use)

Name

Integer Get_super_use_segment_geometry(Element super,Integer &use)

Description

Query whether the dimension Att_Geom_Array exists for the super string super.

If Att_Geom_Array exists, the string can have Segments (which can be straights, arcs or **transitions**) between the vertices of the super string.

See Segment Geometry Dimension for information on the Segment Geometry dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists. That is, the segments of the super string are not just straights but of type Segments (which can be straights, arcs or **transitions**).

use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1839

Set_super_segment_spiral(Element elt,Integer seg,Spiral trans)

Name

Integer Set_super_segment_spiral(Element elt,Integer seg,Spiral trans)

Description

For the Element **super** of type **Super,** set the segment number **seg** to be the transition **trans**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not

have the dimension Att_Geom_Array set.

See Segment Geometry Dimension for information on the Segment Geometry dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the transition was successfully set.

ID = 1840

Get_super_segment_spiral(Element elt,Integer seg,Spiral &trans)

Name

Integer Get_super_segment_spiral(Element elt,Integer seg,Spiral &trans)

Description

For the Element **super** of type **Super,** get the Spiral for the segment number **seg** and return it as **trans**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Geom_Array set, or if the segment is not a Spiral.

See Segment Geometry Dimension for information on the Segment Geometry dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the Spiral was successfully returned.

ID = 1841

Set_super_segment_geometry(Element elt,Integer seg,Segment geom)

Name

Integer Set_super_segment_geometry(Element elt,Integer seg,Segment geom)

Description

For the Element **super** of type **Super,** set the segment number **seg** to be the Segment **geom**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Geom_Array set.

See Segment Geometry Dimension for information on the Segment Geometry dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the segment was successfully set.

ID = 1844

Get_super_segment_geometry(Element elt,Integer seg,Segment &geom)

Name

Integer Get_super_segment_geometry(Element elt,Integer seg,Segment &geom)

Description

For the Element **super** of type **Super,** get the Segment for the segment number **seg** and return it as **geom**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Geom_Array set.

See Segment Geometry Dimension for information on the Segment Geometry dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the Spiral was successfully returned.

ID = 1845

Super String Extrude Functions

For definitions of the Extrude dimensions, see Extrude Dimensions

Extruded an Element shape along a string means to take the (x,y) profile of shape and sweeping the (x,y) profile perpendicularly along the string.

A super string can have a list of Elements that are all to be extruded along the string. The Elements in the list are extruded in the order that they are in the list.

Note: the extrudes can be added as an Element where the (x,y) or the extrudes can come from the *extrudes.4d* file. The ones from the extrudes.4d can be more complex than just a simple profile swept along the string and include *interval* extrudes.

Set_super_use_extrude(Element super,Integer use)

Name

Integer Set_super_use_extrude(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Extrude_Value is used or removed.

If Att_Extrude_Value is set then an extrusion is allowed on the super string.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and an extrusion is allowed.
If **use** is 0, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 1679

Get_super_use_extrude(Element super,Integer &use)

Name

Integer Get_super_use_extrude(Element super,Integer &use)

Description

Query whether the dimension Att_Extrude_Value exists for the super string **super**.

If Att_Extrude_Value is set then an extrusion is allowed on the super string.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1680

Super_append_string_extrude(Element super,Element shape)

Name

Integer Super_append_string_extrude(Element super,Element shape)

Description

For the Element **super** of type **Super** which has the dimension Att_Extrude_Value set**,** add the Element **shape** to the list of Elements that are extruded along **super**. Note: **shape** must also be of type **Super**.

A non-zero function return value is returned if **super** or **shape** is not of type **Super**, or if the Dimension Att_Extrude_Value is not set.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the **shape** was successfully added to the list.

ID = 2643

Super_append_extrude(Element super,Text extrude_name)

Name

Integer Super_append_extrude(Element super,Text extrude_name)

Description

For the Element **super** of type **Super**, get the shape called **extrude_name** from the file *extrudes.4d* and append it to the list of extrudes for **super**.

Note: the extrudes in the extrudes.4d file can be more complex than just a simple profile swept along the string. It also included *interval extrudes*.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Extrude_Value is not set, or if there is no **extrude_name** in extrudes.4d.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1923

Super_append_string_extrude(Element string,Element shape,Integer use_string_colour,Integer shape_mirror,Real start_chainage,Real final_chainage)

Name

Integer Super_append_string_extrude(Element string,Element shape,Integer use_string_colour,Integer shape_mirror,Real start_chainage,Real final_chainage)

Description

what is shape_mirror 0/1

use_string_colour 1 use the **shape** string colour, 0 use **string** colour colour

<no description>

ID = 2644

Get_super_extrudes(Element super,Integer &num_extrudes)

Name

Integer Get_super_extrudes(Element super,Integer &num_extrudes)

Description

For the Element **super** of type **Super** and has the dimension Att_Extrude_Value set**,** get the number of Element that are in the list of extrudes for **super** and return it in **num_extrudes**.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension

Att_Extrude_Value is not set.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1921

Super_insert_extrude(Element super,Text extrude_name,Integer where)

Name

Integer Super_insert_extrude(Element super,Text extrude_name,Integer where)

Description

For the Element **super** of type **Super**, get the shape called **extrude_name** from the file extrudes.4d and insert into the list of extrudes at position number **where**. The existing extrudes from position number **where** upwards are all moved up one position in the list.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Extrude_Value is not set, or if there is no **extrude_name** in extrudes.4d.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1922

Super_delete_extrude(Element super,Integer extrude_num)

Name

Integer Super_delete_extrude(Element super,Integer extrude_num)

Description

For the Element **super** of type **Super**, delete the extrude in position number extrude_num from the list of extrusions for **super**.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Extrude_Value is not set.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1924

Super_delete_all_extrudes(Element super)

Name

Integer Super_delete_all_extrudes(Element super)

Description

Delete all extrudes.

For the Element **super** of type **Super**, delete all the extrudes from the list of extrusions for **super**.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Extrude_Value is not set.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1925

Set_super_extrude(Element super,Element shape)

Name

Integer Set_super_extrude(Element super,Element shape)

Description

LEGACY FUNCTION - DO NOT USE

Many moons ago there was only one profile that could be extruded along the string.
Later that was modified and there is now a list of profiles that are extruded.
This call is from before there was a list and will behave as if there is no list and will delete the list. Hence this option should not be used.

For the Element **super** of type **Super** which has the dimension Att_Extrude_Value set**,** set **shape** to be the Element that is extruded along **super**.

Note: **shape** must also be of type **Super**.

WARNING: If this function is called and there is a list of extrudes, the entire list will be deleted.

A non-zero function return value is returned if **super** or **shape** is not of type **Super**, or if the Dimension Att_Extrude_Value is not set.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the **shape** was successfully set.

ID = 1681

Get_super_extrude(Element super,Element &shape)

Name

Integer Get_super_extrude(Element super,Element &shape)

Description

LEGACY FUNCTION - DO NOT USE

Many moons ago there was only one profile that could be extruded along the string.
Later that was modified and there is now a list of profiles that are extruded.
This call will only return one profile. Hence this option should not be used.

For the Element **super** of type **Super** and has the dimension Att_Extrude_Value set**,** get the Element **shape** that defines the 2d profile that is extruded along **super**.

Note: **shape** will be of type **Super**.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Extrude_Value is not set.

See Extrude Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the **shape** was successfully returned.

ID = 1682

Super String Interval Functions

For definitions of the Interval dimensions, see Interval Dimensions

If Att_Interval_Value is set, then there is a Real *interval_distance* and a Real *chord_arc_distance* for the super string

if the plan length of a segment is greater than *interval_distance* then for triangulation purposes, extra temporary vertices are added into the super string so that the plan distance between each vertex is less than *interval_distance.* The z-value for the temporary vertices is interpolated from the z-values of the adjacent real vertices of the super string. If *interval_distance* is equal to zero, then no extra temporary vertices are added.

Also for each segment that is an arc, if the plan chord distance between the end points of the arc is greater than the *chord_arc_distance* then for triangulation purposes extra temporary vertices are added into the super string until the chord distance for each arc is less than *chord_arc_distance*. The z-value for the temporary vertices is interpolated from the z-values of the adjacent real vertices of the super string. If *chord_arc_distance* is equal to zero, then no extra temporary vertices are added

Set_super_use_interval(Element super,Integer use)

Name

Integer Set_super_use_interval(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Interval_Value is used or removed.

If Att_Interval_Value is set then there is a Real *interval_distance* and a Real *chord_arc_distance* stored for the super string.

See Interval Dimensions for information on the Interval dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and the two intervals are stored.
If **use** is 0, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 1702

Get_super_use_interval(Element super,Integer &use)

Name

Integer Get_super_use_interval(Element super,Integer &use)

Description

Query whether the dimension Att_Interval_Value exists for the super string **super**.

If Att_Interval_Value is set then there is a Real *interval_distance* and a Real *chord_arc_distance* stored for the super string.

See Interval Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1705

Set_super_interval_distance(Element super,Real value)

Name

Integer Set_super_interval_distance(Element super,Real value)

Description

For the Element **super** of type **Super** which has the dimension Att_Interval_Value set**,** set the *interval_distance* to **value**.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Interval_Value is not set.

See Interval Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the *interval_distance* was successfully set.

ID = 1704

Get_super_interval_distance(Element super,Real &value)

Name

Integer Get_super_interval_distance(Element super,Real &value)

Description

For the Element **super** of type **Super** and has the dimension Att_Interval_Value set**,** get the *interval_distance* for super and return it in **value***.*

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Interval_Value is not set.

See Interval Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the *interval_distance* was successfully returned.

ID = 1707

Set_super_interval_chord_arc(Element super,Real value)

Name

Integer Set_super_interval_chord_arc(Element super,Real value)

Description

Description

For the Element **super** of type **Super** which has the dimension Att_Interval_Value set**,** set the *chord_arc_distance* to **value**.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Interval_Value is not set.

See Interval Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the *chord_arc_distance* was successfully set.

ID = 1703

Get_super_interval_chord_arc(Element super,Real &value)

Name

Integer Get_super_interval_chord_arc(Element super,Real &value)

Description

For the Element **super** of type **Super** and has the dimension Att_Interval_Value set**,** get the *chord_arc_distance* for super and return it in **value**.

A non-zero function return value is returned if **super** is not of type **Super**, or if the Dimension Att_Interval_Value is not set.

See Interval Dimensions for information on the Extrude dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the *chord_arc_distance* was successfully returned.

ID = 1706

Super String Vertex Attributes Functions

For definitions of the Vertex Attributes dimensions, see User Defined Vertex Attributes Dimensions

Set_super_use_vertex_attribute(Element super,Integer use)

Name

Integer Set_super_use_vertex_attribute(Element super,Integer use)

Description

Tell the super string whether to use. or remove, the dimension Att_Vertex_Attribute_Array.

If Att_Vertex_Attribute_Array exists then there can be a type Attributes for each vertex.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and an Attributes is allowed on each vertex.
If **use** is 0, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 770

Get_super_use_vertex_attribute(Element super,Integer &use)

Name

Integer Get_super_use_vertex_attribute(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Attribute_Array exists for the super string.

If Att_Vertex_Attribute_Array exists then there can be a type Attributes for each vertex.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 771

Set_super_vertex_attributes(Element super,Integer vert,Attributes att)

Name

Integer Set_super_vertex_attributes(Element super,Integer vert,Attributes att)

Description

For the Element **super**, set the Attributes for the vertex number **vert** to **att**.

If the Element is not of type **Super,** or the dimension Att_Vertex_Attribute_Array is not set, then a non-zero return value is returned.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the attribute is successfully set.

ID = 2003

Get_super_vertex_attributes(Element super,Integer vert,Attributes &att)

Name

Integer Get_super_vertex_attributes(Element super,Integer vert,Attributes &att)

Description

For the Element **super**, return the Attributes for the vertex number **vert** as **att**.

If the Element is not of type **Super,** or the dimension Att_Vertex_Attribute_Array is not set, or the vertex number **vert** has no Attributes, then a non-zero return value is returned.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the attribute is successfully returned.

ID = 2002

Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Uid &uid)

Name

Integer Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Uid &uid)

Description

For the Element **super**, get the attribute called **att_name** for the vertex number **vert** and return the attribute value in **uid**. The attribute must be of type Uid.

If the Element is not of type **Super**, or the dimension Att_Vertex_Attribute_Array is not set, or the attribute is not of type Uid then a non-zero return value is returned.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2004

Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Attributes &att)

Name

Integer Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Attributes &att)

Description

For the Element **super**, get the attribute called **att_name** for the vertex number **vert** and return the attribute value in **att**. The attribute must be of type Attributes.

If the Element is not of type **Super,** or the dimension Att_Vertex_Attribute_Array is not set, or the attribute is not of type Attributes then a non-zero return value is returned.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2005

Get_super_vertex_attribute(Element elt,Integer vert,Integer att_no,Uid &uid)

Name

Integer Get_super_vertex_attribute(Element elt,Integer vert,Integer att_no,Uid &uid)

Description

For the Element **super**, get the attribute with number **att_no** for the vertex number **vert** and return the attribute value in **uid**. The attribute must be of type Uid.

If the Element is not of type **Super,** or the dimension Att_Vertex_Attribute_Array is not set, or the attribute is not of type Uid then a non-zero return value is returned.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 2006

Get_super_vertex_attribute(Element elt,Integer vert,Integer att_no,Attributes &att)

Name

Integer Get_super_vertex_attribute(Element elt,Integer vert,Integer att_no,Attributes &att)

Description

For the Element **super**, get the attribute with number **att_no** for the vertex number **vert** and return the attribute value in **att**. The attribute must be of type Attributes.

If the Element is not of type **Super,** or the dimension Att_Vertex_Attribute_Array is not set, or the attribute is not of type Attributes then a non-zero return value is returned.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 2007

Set_super_vertex_attribute(Element elt,Integer vert,Text att_name,Uid uid)

Name

Integer Set_super_vertex_attribute(Element elt,Integer vert,Text att_name,Uid uid)

Description

For the Element **super** and on the vertex number **vert**,
if the attribute called **att_name** does not exist then create it as type Uid and give it the value **uid**.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **uid**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2008

Set_super_vertex_attribute(Element elt,Integer vert,Text att_name,Attributes att)

Name

Integer Set_super_vertex_attribute(Element elt,Integer vert,Text att_name,Attributes att)

Description

For the Element **super** and on the vertex number **vert**,
if the attribute called **att_name** does not exist then create it as type Attributes and give it the value **att**.
if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2009

Set_super_vertex_attribute(Element elt,Integer vert,Integer att_no,Uid uid)

Name

Integer Set_super_vertex_attribute(Element elt,Integer vert,Integer att_no,Uid uid)

Description

For the Element **super** and on the vertex number **vert**, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **uid**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 2010

Set_super_vertex_attribute(Element elt,Integer vert,Integer att_no,Attributes att)

Name

Integer Set_super_vertex_attribute(Element elt,Integer vert,Integer att_no,Attributes att)

Description

For the Element **super** and on the vertex number **vert**, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 2011

Super_vertex_attribute_exists(Element elt,Integer vert,Text att_name,Integer &num)

Name

Integer Super_vertex_attribute_exists(Element elt,Integer vert,Text att_name,Integer &num)

Description

Checks to see if for vertex number **vert**, an attribute of name **att_name** exists, and if it does, return the number of the attribute as **num**.

A non-zero function return value indicates the attribute exists and its number was successfully returned.

A zero function return value indicates the attribute does not exist, or the number was not successfully returned.

Warning - this is the opposite to most 12dPL function return values

ID = 773

Super_vertex_attribute_exists(Element elt,Integer vert,Text att_name)

Name

Integer Super_vertex_attribute_exists(Element elt,Integer vert,Text att_name)

Description

Checks to see if for vertex number **vert**, an attribute of name **att_name** exists.

A non-zero function return value indicates the attribute exists.

A zero function return value indicates the attribute does not exist.

Warning - this is the opposite to most 12dPL function return values

ID = 772

Super_vertex_attribute_delete(Element super,Integer vert,Integer att_no)

Name

Integer Super_vertex_attribute_delete(Element super,Integer vert,Integer att_no)

Description

For the Element **super**, delete the attribute with attribute number **att_no** for vertex number **vert**.

If the Element **super** is not of type **Super** or **super** has no vertex number **vert**, then a non-zero return code is returned.

A function return value of zero indicates the attribute was deleted.

ID = 775

Super_vertex_attribute_delete(Element super,Integer vert,Text att_name)

Name

Integer Super_vertex_attribute_delete(Element super,Integer vert,Text att_name)

Description

For the Element **super**, delete the attribute with the name **att_name** for vertex number **vert**.

If the Element **super** is not of type **Super** or **super** has vertex number **vert**, then a non-zero return code is returned.

A function return value of zero indicates the attribute was deleted.

ID = 774

Super_vertex_attribute_delete_all(Element super,Integer vert)

Name

Integer Super_vertex_attribute_delete_all(Element super,Integer vert)

Description

Delete all the attributes of vertex number **vert** of the super string **super**.

A function return value of zero indicates the function was successful.

ID = 776

Super_vertex_attribute_dump(Element super,Integer vert)

Name

Integer Super_vertex_attribute_dump(Element super,Integer vert)

Description

Write out information to the Output Window about the vertex attributes for vertex number **vert** of the super string **super**.

A function return value of zero indicates the function was successful.

ID = 777

Super_vertex_attribute_debug(Element super,Integer vert)

Name

Integer Super_vertex_attribute_debug(Element super,Integer vert)

Description

Write out even more information to the Output Window about the vertex attributes for vertex number **vert** of the super string **super**.

A function return value of zero indicates the function was successful.

ID = 778

Get_super_vertex_number_of_attributes(Element super,Integer vert,Integer &no_atts)

Name

Integer Get_super_vertex_number_of_attributes(Element super,Integer vert,Integer &no_atts)

Description

Get the total number of attributes for vertex number **vert** of the Element **super**.

The total number of attributes is returned in Integer **no_atts**.

A function return value of zero indicates the number of attributes was successfully returned.

ID = 779

Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Text &txt)

Name

Integer Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Text &txt)

Description

For the Element **super**, get the attribute called **att_name** for the vertex number **vert** and return the attribute value in **txt**. The attribute must be of type **Text**.

If the Element is not of type **Super** or the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 780

Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Integer &int)

Name

Integer Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Integer &int)

Description

For the Element **super**, get the attribute called **att_name** for the vertex number **vert** and return the attribute value in **int**. The attribute must be of type **Integer**.

If the Element is not of type **Super** or the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 781

Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Real &real)

Name

Integer Get_super_vertex_attribute(Element super,Integer vert,Text att_name,Real &real)

Description

For the Element **super**, get the attribute called **att_name** for the vertex number **vert** and return the attribute value in **real**. The attribute must be of type **Real**.

If the Element is not of type **Super** or the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 782

Get_super_vertex_attribute(Element super,Integer vert,Integer att_no,Text &txt)

Name

Integer Get_super_vertex_attribute(Element super,Integer vert,Integer att_no,Text &txt)

Description

For the Element **super**, get the attribute number **att_no** for the vertex number **vert** and return the attribute value in **txt**. The attribute must be of type **Text**.

If the Element is not of type **Super** or the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 783

Get_super_vertex_attribute(Element super,Integer vert,Integer att_no,Integer &int)

Name

Integer Get_super_vertex_attribute(Element super,Integer vert,Integer att_no,Integer &int)

Description

For the Element **super**, get the attribute number **att_no** for the vertex number **vert** and return the attribute value in **int**. The attribute must be of type **Integer**.

If the Element is not of type **Super** or the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 784

Get_super_vertex_attribute(Element super,Integer vert,Integer att_no,Real &real)

Name

Integer Get_super_vertex_attribute(Element super,Integer vert,Integer att_no,Real &real)

Description

For the Element **super**, get the attribute number **att_no** for the vertex number **vert** and return the attribute value in **real**. The attribute must be of type **Real**.

If the Element is not of type **Super** or the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 785

Get_super_vertex_attribute_name(Element super,Integer vert,Integer att_no,Text &txt)

Name

Integer Get_super_vertex_attribute_name(Element super,Integer vert,Integer att_no,Text &txt)

Description

For vertex number **vert** of the Element **super**, get the name of the attribute number **att_no**. The attribute name is returned in **txt**.

A function return value of zero indicates the attribute name was successfully returned.

ID = 786

Get_super_vertex_attribute_length(Element super,Integer vert,Text att_name,Integer &att_len)

Name

Integer Get_super_vertex_attribute_length(Element super,Integer vert,Text att_name,Integer &att_len)

Description

For vertex number **vert** of the Element **super**, get the length (in bytes) of the attribute with the name **att_name**. The attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for user attributes of type **Text** and **Binary**.

ID = 789

Get_super_vertex_attribute_length(Element super,Integer vert,Integer att_no,Integer &att_len)

Name

Integer Get_super_vertex_attribute_length(Element super,Integer vert,Integer att_no,Integer &att_len)

Description

For vertex number **vert** of the Element **super**, get the length (in bytes) of the attribute number **att_no**. The attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for attributes of type Text and Binary.

ID = 790

Get_super_vertex_attribute_type(Element super,Integer vert,Text att_name,Integer &att_type)

Name

Integer Get_super_vertex_attribute_type(Element super,Integer vert,Text att_name,Integer &att_type)

Description

For vertex number **vert** of the Element **super**, get the type of the attribute with name **att_name**. The attribute type is returned in **att_type**.

A function return value of zero indicates the attribute type was successfully returned.

ID = 787

Get_super_vertex_attribute_type(Element super,Integer vert,Integer att_no,Integer &att_type)

Name

Integer Get_super_vertex_attribute_type(Element super,Integer vert,Integer att_no,Integer &att_type)

Description

For vertex number **vert** of the Element **super**, get the type of the attribute with attribute number **att_no**. The attribute type is returned in **att_type**.

A function return value of zero indicates the attribute type was successfully returned.

ID = 788

Set_super_vertex_attribute(Element super,Integer vert,Text att_name,Text txt)

Name

Integer Set_super_vertex_attribute(Element super,Integer vert,Text att_name,Text txt)

Description

For the Element **super** and on the vertex number **vert**,
if the attribute called **att_name** does not exist then create it as type Text and give it the value **txt**.
if the attribute called **att_name** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 791

Set_super_vertex_attribute(Element super,Integer vert,Text att_name,Integer int)

Name

Integer Set_super_vertex_attribute(Element super,Integer vert,Text att_name,Integer int)

Description

For the Element **super** and on the vertex number **vert**,
if the attribute called **att_name** does not exist then create it as type Integer and give it the value **int**.
if the attribute called **att_name** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 792

Set_super_vertex_attribute(Element super,Integer vert,Text att_name,Real real)

Name

Integer Set_super_vertex_attribute(Element super,Integer vert,Text att_name,Real real)

Description

For the Element **super** and on the vertex number **vert**,
if the attribute called **att_name** does not exist then create it as type Real and give it the value **real**.
if the attribute called **att_name** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 793

Set_super_vertex_attribute(Element super,Integer vert,Integer att_no,Text txt)

Name

Integer Set_super_vertex_attribute(Element super,Integer vert,Integer att_no,Text txt)

Description

For the Element **super** and on the vertex number **vert**,
if the attribute with number **att_no** does not exist then create it as type Text and give it the value **txt**.
if the attribute with number **att_no** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 794

Set_super_vertex_attribute(Element super,Integer vert,Integer att_no,Integer int)

Name

Integer Set_super_vertex_attribute(Element super,Integer vert,Integer att_no,Integer int)

Description

For the Element **super** and on the vertex number **vert**,
if the attribute with number **att_no** does not exist then create it as type Integer and give it the value **int**.
if the attribute with number **att_no** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 795

Set_super_vertex_attribute(Element super,Integer vert,Integer att_no,Real real)

Name

Integer Set_super_vertex_attribute(Element super,Integer vert,Integer att_no,Real real)

Description

For the Element **super** and on the vertex number **vert**,
if the attribute with number **att_no** does not exist then create it as type Real and give it the value **real**.
if the attribute with number **att_no** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 796

Super String Segment Attributes Functions

For definitions of the Segment Attributes dimensions, see User Defined Vertex Attributes Dimensions

Set_super_use_segment_attribute(Element super,Integer use)

Name

Integer Set_super_use_segment_attribute(Element super,Integer use)

Description

Tell the super string whether to use or remove the dimension Att_Segment_Attribute_Array.

If the dimension Att_Segment_Attribute_Array exists then there can be an Attributes on each segment.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

A return value of 0 indicates the function call was successful.

ID = 1060

Get_super_use_segment_attribute(Element super,Integer &use)

Name

Integer Get_super_use_segment_attribute(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Attribute_Array exists for the super string.

If the dimension Att_Segment_Attribute_Array exists then there can be an Attributes on each segment.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1061

Get_super_segment_attributes(Element elt,Integer seg,Attributes &att)

Name

Integer Get_super_segment_attributes(Element elt,Integer seg,Attributes &att)

Description

For the Element **super**, return the Attributes for the segment number **seg** as **att**.

If the Element is not of type **Super**, or Att_Segment_Attribute_Array dimension is not set, or the segment number **seg** has no attribute then a non-zero return value is returned.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the attribute is successfully returned.

ID = 2012

Set_super_segment_attributes(Element elt,Integer seg,Attributes att)

Name

Integer Set_super_segment_attributes(Element elt,Integer seg,Attributes att)

Description

For the Element **super**, set the Attributes for the segment number **seg** to **att**.

If the Element is not of type **Super,** or Att_Segment_Attribute_Array dimension is not set, then a non-zero return value is returned.

See User Defined Vertex Attributes Dimensions for information on the Attributes dimensions or Super String Dimensions for information on all the dimensions.

A function return value of zero indicates the attribute is successfully set.

ID = 2013

Get_super_segment_attribute(Element super,Integer seg,Text att_name,Uid &uid)

Name

Integer Get_super_segment_attribute(Element super,Integer seg,Text att_name,Uid &uid)

Description

For the Element **super**, get the attribute called **att_name** for the segment number **seg** and return the attribute value in **uid**. The attribute must be of type Uid.

If the Element is not of type **Super** or the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2014

Get_super_segment_attribute(Element super,Integer seg,Text att_name, Attributes &att)

Name

Integer Get_super_segment_attribute(Element super,Integer seg,Text att_name,Attributes &att)

Description

For the Element **super**, get the attribute called **att_name** for the segment number **seg** and return the attribute value in **att**. The attribute must be of type Attributes.

If the Element is not of type **Super** or the attribute is not of type **Attributes** then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2015

Get_super_segment_attribute(Element super,Integer seg,Integer att_no,Uid &uid)

Name

Integer Get_super_segment_attribute(Element super,Integer seg,Integer att_no,Uid &uid)

Description

For the Element **super**, get the attribute with number **att_no** for the segment number **seg** and

return the attribute value in **uid**. The attribute must be of type Uid.

If the Element is not of type **Super** or the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 2016

Get_super_segment_attribute(Element super,Integer seg,Integer att_no, Attributes &att)

Name

Integer Get_super_segment_attribute(Element super,Integer seg,Integer att_no,Attributes &att)

Description

For the Element **super**, get the attribute with number **att_no** for the segment number **seg** and return the attribute value in **att**. The attribute must be of type Attributes.

If the Element is not of type **Super** or the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 2017

Set_super_segment_attribute(Element super,Integer seg,Text att_name,Uid uid)

Name

Integer Set_super_segment_attribute(Element super,Integer seg,Text att_name,Uid uid)

Description

For the Element **super** and on the segment number **seg**,
if the attribute called **att_name** does not exist then create it as type Uid and give it the value **uid**.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **uid**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2018

Set_super_segment_attribute(Element super,Integer seg,Text att_name, Attributes att)

Name

Integer Set_super_segment_attribute(Element super,Integer seg,Text att_name,Attributes att)

Description

For the Element **super** and on the segment number **seg**,
if the attribute called **att_name** does not exist then create it as type Attributes and give it the value **att**.

if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_name.

ID = 2019

Set_super_segment_attribute(Element super,Integer seg,Integer att_no,Uid uid)

Name

Integer Set_super_segment_attribute(Element super,Integer seg,Integer att_no,Uid uid)

Description

For the Element **super** and on the segment number **seg**, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **uid**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 2020

Set_super_segment_attribute(Element super,Integer seg,Integer att_no,Attributes att)

Name

Integer Set_super_segment_attribute(Element super,Integer seg,Integer att_no,Attributes att)

Description

For the Element **super** and on the segment number **seg**, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 2021

Super_segment_attribute_exists(Element elt,Integer seg,Text att_name)

Name

Integer Super_segment_attribute_exists(Element elt,Integer seg,Text att_name)

Description

Checks to see if for segment number **seg**, an attribute of name **att_name** exists.

A non-zero function return value indicates the attribute exists.

A zero function return value indicates the attribute does not exist.

Warning - this is the opposite to most 12dPL function return values

ID = 1062

Super_segment_attribute_exists(Element elt,Integer seg,Text att_name,Integer &num)

Name

Integer Super_segment_attribute_exists(Element elt,Integer seg,Text att_name,Integer &num)

Description

Checks to see if for segment number **seg**, an attribute of name **att_name** exists, and if it does, return the number of the attribute as **num**.

A non-zero function return value indicates the attribute exists and its number was successfully returned.

A zero function return value indicates the attribute does not exist, or the number was not successfully returned.

Warning - this is the opposite to most 12dPL function return values

ID = 1063

Super_segment_attribute_delete (Element super,Integer seg,Text att_name)

Name

Integer Super_segment_attribute_delete (Element super,Integer seg,Text att_name)

Description

For the Element **super**, delete the attribute with the name **att_name** for segment number **seg**.

If the Element **super** is not of type **Super** or **super** has no segment number **seg**, then a non-zero return code is returned.

A function return value of zero indicates the attribute was deleted.

ID = 1064

Super_segment_attribute_delete (Element super,Integer seg,Integer att_no)

Name

Integer Super_segment_attribute_delete (Element super,Integer seg,Integer att_no)

Description

For the Element **super**, delete the attribute with attribute number **att_no** for segment number **seg**.

If the Element **super** is not of type **Super** or **super** has no segment number **seg**, then a non-zero return code is returned.

A function return value of zero indicates the attribute was deleted.

ID = 1065

Super_segment_attribute_delete_all (Element super,Integer seg)

Name

Integer Super_segment_attribute_delete_all (Element super,Integer seg)

Description

Delete all the attributes of segment number **seg** of the super string **super**.

A function return value of zero indicates the function was successful.

ID = 1066

Super_segment_attribute_dump (Element super,Integer seg)

Name

Integer Super_segment_attribute_dump (Element super,Integer seg)

Description

Write out information to the Output Window about the segment attributes for segment number **seg** of the super string **super**.

A function return value of zero indicates the function was successful.

ID = 1067

Super_segment_attribute_debug (Element super,Integer seg)

Name

Integer Super_segment_attribute_debug (Element super,Integer seg)

Description

Write out even more information to the Output Window about the segment attributes for segment number **seg** of the super string **super**.

A function return value of zero indicates the function was successful.

ID = 1068

Get_super_segment_number_of_attributes(Element super,Integer seg,Integer &no_atts)

Name

Integer Get_super_segment_number_of_attributes(Element elt,Integer seg,Integer &no_atts)

Description

Get the total number of attributes for segment number **seg** of the Element **super**.

The total number of attributes is returned in Integer **no_atts**.

A function return value of zero indicates the number of attributes was successfully returned.

A return value of 0 indicates the function call was successful.

ID = 1069

Get_super_segment_attribute (Element super,Integer seg,Text att_name,Text &text)

Name

Integer Get_super_segment_attribute (Element super,Integer seg,Text att_name,Text &text)

Description

For the Element **super**, get the attribute called **att_name** for the segment number **seg** and return the attribute value in **text**. The attribute must be of type **Text**.

If the Element is not of type **Super** or the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1070

Get_super_segment_attribute (Element super,Integer seg,Text att_name,Integer &int)

Name

Integer Get_super_segment_attribute (Element super,Integer seg,Text att_name,Integer &int)

Description

For the Element **super**, get the attribute called **att_name** for the segment number **seg** and return the attribute value in **int**. The attribute must be of type **Integer**.

If the Element is not of type **Super** or the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1071

Get_super_segment_attribute (Element super,Integer seg,Text att_name,Real &real)

Name

Integer Get_super_segment_attribute (Element super,Integer seg,Text att_name,Real &real)

Description

For the Element **super**, get the attribute called **att_name** for the segment number **seg** and return the attribute value in **real**. The attribute must be of type **Real**.

If the Element is not of type **Super** or the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1072

Get_super_segment_attribute (Element super,Integer seg,Integer att_no,Text &txt)

Name

Integer Get_super_segment_attribute (Element super,Integer seg,Integer att_no,Text &txt)

Description

For the Element **super**, get the attribute number **att_no** for the segment number **seg** and return the attribute value in **txt**. The attribute must be of type **Text**.

If the Element is not of type **Super** or the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1073

Get_super_segment_attribute (Element super,Integer seg,Integer att_no,Integer &int)

Name

Integer Get_super_segment_attribute (Element super,Integer seg,Integer att_no,Integer &int)

Description

For the Element **super**, get the attribute number **att_no** for the segment number **seg** and return the attribute value in **int**. The attribute must be of type **Integer**.

If the Element is not of type **Super** or the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 1074

Get_super_segment_attribute (Element super,Integer seg,Integer att_no,Real &real)

Name

Integer Get_super_segment_attribute (Element super,Integer seg,Integer att_no,Real &real)

Description

For the Element **super**, get the attribute number **att_no** for the segment number **seg** and return the attribute value in **real**. The attribute must be of type **Real**.

If the Element is not of type **Super** or the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 1075

Get_super_segment_attribute_name (Element super,Integer seg,Integer att_no,Text &txt)

Name

Integer Get_super_segment_attribute_name (Element super,Integer seg,Integer att_no,Text &txt)

Description

For segment number **seg** of the Element **super**, get the name of the attribute number **att_no**. The attribute name is returned in **txt**.

A function return value of zero indicates the attribute name was successfully returned.

ID = 1076

Get_super_segment_attribute_type (Element super,Integer seg,Text att_name,Integer &att_type)

Name

Integer Get_super_segment_attribute_type (Element super,Integer seg,Text att_name,Integer &att_type)

Description

For segment number **seg** of the Element **super**, get the type of the attribute with name **att_name**. The attribute type is returned in **att_type**.

A function return value of zero indicates the attribute type was successfully returned.

ID = 1077

Get_super_segment_attribute_type (Element super,Integer seg,Integer att_no,Integer &att_type)

Name

Integer Get_super_segment_attribute_type (Element super,Integer seg,Integer att_no,Integer &att_type)

Description

For segment number **seg** of the Element **super**, get the type of the attribute with attribute number **att_no**. The attribute type is returned in **att_type**.

A function return value of zero indicates the attribute type was successfully returned.

ID = 1078

Get_super_segment_attribute_length(Element super,Integer seg,Text att_name,Integer &att_len)

Name

Integer Get_super_segment_attribute_length(Element super,Integer seg,Text att_name,Integer &att_len)

Description

For segment number **seg** of the Element **super**, get the length (in bytes) of the attribute with the name **att_name**. The attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for user attributes of type **Text** and **Binary.**

ID = 1079

Get_super_segment_attribute_length(Element super,Integer seg,Integer att_no,Integer &att_len)

Name

Integer Get_super_segment_attribute_length(Element super,Integer seg,Integer att_no,Integer &att_len)

Description

For segment number **seg** of the Element **super**, get the length (in bytes) of the attribute number **att_no**. The attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for attributes of type **Text** and **Binary**.

ID = 1080

Set_super_segment_attribute (Element super,Integer seg,Text att_name,Text txt)

Name

Integer Set_super_segment_attribute (Element super,Integer seg,Text att_name,Text txt)

Description

For the Element **super** and on the segment number **seg**,
if the attribute called **att_name** does not exist then create it as type Text and give it the value **txt**.
if the attribute called **att_name** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1081

Set_super_segment_attribute (Element super,Integer seg,Text att_name,Integer in)

Name

Integer Set_super_segment_attribute (Element super,Integer seg,Text att_name,Integer int)

Description

For the Element **super** and on the segment number **seg**,
if the attribute called **att_name** does not exist then create it as type Integer and give it the value **int**.
if the attribute called **att_name** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1082

Set_super_segment_attribute (Element super,Integer seg,Text att_name,Real real)

Name

Integer Set_super_segment_attribute (Element super,Integer seg,Text att_name,Real real)

Description

For the Element **super** and on the segment number **seg**,
if the attribute called **att_name** does not exist then create it as type Real and give it the value **real**.
if the attribute called **att_name** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1083

Set_super_segment_attribute (Element super,Integer seg,Integer att_no,Text txt)

Name

Integer Set_super_segment_attribute (Element super,Integer seg,Integer att_no,Text txt)

Description

For the Element **super** and on the segment number **seg**,
if the attribute with number **att_no** does not exist then create it as type Text and give it the value **txt**.

if the attribute with number **att_no** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 1084

Set_super_segment_attribute (Element super,Integer seg,Integer att_no,Integer in)

Name

Integer Set_super_segment_attribute (Element super,Integer seg,Integer att_no,Integer int)

Description

For the Element **super** and on the segment number **seg**,
if the attribute with number **att_no** does not exist then create it as type Integer and give it the value **int**.
if the attribute with number **att_no** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute number att_no.

ID = 1085

Set_super_segment_attribute(Element super,Integer seg,Integer att_no,Real real)

Name

Integer Set_super_segment_attribute(Element super,Integer seg,Integer att_no,Real real)

Description

For the Element **super** and on the segment number **seg**,
if the attribute with number **att_no** does not exist then create it as type Real and give it the value **real**.
if the attribute with number **att_no** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 1086

Super String Uid Functions

For definitions of the UID dimensions, see UID Dimensions

If Att_Vertex_UID_Array is used, then there is an Integer (referred to as a uid) stored at each vertex of the super string. Note that this is an Integer and not a variable of type Uid.

This is used by 12d Solutions to store special backtracking numbers on each vertex (for example for survey data reduction or with the underlying super string in a super alignment).

See Super String Vertex Uid
See Super String Segment Uid

Super String Vertex Uid

Set_super_use_vertex_uid(Element super,Integer use)

Name

Integer Set_super_use_vertex_uid(Element super,Integer use)

Description

WARNING - Reserved for 12d Solutions Staff Only.

Tell the super string **super** whether to use (set), or not use (remove), the dimension Att_Vertex_UID_Array.

A value for **use** of 1 sets the dimension and 0 removes it.

If Att_Vertex_UID_Array is used, then there is an Integer (referred to as a uid) stored at each vertex of the super string.

This is used by 12d Solutions to store special backtracking numbers on each vertex (for example for survey data reduction or with the underlying super string in a super alignment).

See UID Dimensions for information on the Vertex UID dimension or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1572

Get_super_use_vertex_uid(Element super,Integer &use)

Name

Integer Get_super_use_vertex_uid(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_UID_Array exists (is used) for the super string **super**.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

If Att_Vertex_UID_Array is used, then there is an Integer (referred to as a uid) stored at each vertex of the super string.

This is used by 12d Solutions to store special backtracking numbers on each vertex (for example for survey data reduction or with the underlying super string in a super alignment).

See UID Dimensions for information on the Vertex UID dimension or Super String Dimensions for information on all the dimensions.

ID = 1573

Set_super_vertex_uid(Element super,Integer vert,Integer num)

Name

Integer Set_super_vertex_uid(Element super,Integer vert,Integer num)

Description

WARNING - Reserved for 12d Solutions Staff Only.

For the super Element **super**, set the vertex uid at vertex number **vert** to be **num**.

A return value of 0 indicates the function call was successful.

ID = 1574

Get_super_vertex_uid(Element super,Integer vert,Integer &num)

Name

Integer Get_super_vertex_uid(Element super,Integer vert,Integer &num)

Description

For the super Element **super**, get the vertex uid at vertex number **vert** and return it in **num**.

A return value of 0 indicates the function call was successful.

ID = 1575

Super String Segment Uid

Set_super_use_segment_uid(Element super,Integer use)

Name

Integer Set_super_use_segment_uid(Element super,Integer use)

Description

WARNING - Reserved for 12d Solutions Staff Only.

Tell the super string **super** whether to use (set), or not use (remove), the dimension Att_Segment_UID_Array.

A value for **use** of 1 sets the dimension and 0 removes it.

If Att_Segment_UID_Array is used, then there is an Integer stored at each segment of the super string.

This is used by 12d Solutions to store special backtracking numbers on each segment (for example for survey data reduction or with the underlying super string in a super alignment).

See UID Dimensions for information on the Segment UID dimension or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 1576

Get_super_use_segment_uid(Element super,Integer &use)

Name

Integer Get_super_use_segment_uid(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_UID_Array exists (is used) for the super string

super.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

If Att_Segment_UID_Array is used, then there is an Integer stored at each segment of the super string.

This is used by 12d Solutions to store special backtracking numbers on each segment (for example for survey data reduction or with the underlying super string in a super alignment).

See UID Dimensions for information on the Segment UID dimension or Super String Dimensions for information on all the dimensions.

ID = 1577

Set_super_segment_uid(Element super,Integer seg,Integer num)

Name

Integer Set_super_segment_uid(Element super,Integer seg,Integer num)

Description

WARNING - Reserved for 12d Solutions Staff Only.

For the super Element **super**, set the number called uid at segment number **seg** to be **num**.

A return value of 0 indicates the function call was successful.

ID = 1578

Get_super_segment_uid(Element super,Integer seg,Integer &num)

Name

Integer Get_super_segment_uid(Element super,Integer seg,Integer &num)

Description

For the super Element **super**, get the number called the uid on segment number **seg** and return it in **num**.

A return value of 0 indicates the function call was successful.

ID = 1579

Super String Vertex Image Functions

For definitions of the Visibility dimensions, see Vertex Image Dimensions

See Super String Use Vertex Image Functions
See Setting Super String Vertex Image Functions

Super String Use Vertex Image Functions

Set_super_use_vertex_image_value(Element super,Integer use)

Name

Integer Set_super_use_vertex_image_value(Element super,Integer use)

Description

For the super string Element super, define whether the dimension Att_Vertex_Image_Value is used. If the dimension Att_Vertex_Image_Value is set then there can be one image attached to each vertex.

See Vertex Image Dimensions for information on the Vertex Image dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set. That is, the super string can have an image attached to each vertex (it can be a different image at each vertex).

If **use** is 0, the dimension is removed. If the string had images then the images will be removed.

A return value of 0 indicates the function call was successful.

ID = 1767

Get_super_use_vertex_image_value(Element super,Integer &use)

Name

Integer Get_super_use_vertex_image_value(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Image_Value exists for the super string super.

If the dimension Att_Vertex_Image_Value is set then there can be one image attached to each vertex.

See Vertex Image Dimensions for information on the Vertex Image dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1768

Set_super_use_vertex_image_array(Element super,Integer use)

Name

Integer Set_super_use_vertex_image_array(Element super,Integer use)

Description

For the super string Element super, define whether the dimension Att_Vertex_Image_Array is used, or removed, for the super string super.

If the dimension Att_Vertex_Image_Array is set then there can be more than one image attached to each vertex.

See Vertex Image Dimensions for information on the Vertex Image dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set. That is, each super string vertex can have a number of images attached to it.

If **use** is 0, the dimension is removed. If the super string vertex had images then the images will be removed.

A return value of 0 indicates the function call was successful.

ID = 1769

Get_super_use_vertex_image_array(Element super,Integer &use)

Name

Integer Get_super_use_vertex_image_array(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Image_Array exists for the super string super.

If the dimension Att_Vertex_Image_Array is set then there can be more than one image attached to each vertex.

See Vertex Image Dimensions for information on the Vertex Image dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists. That is, each super string vertex can have a number of images attached to it.

use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1770

Super_vertex_image_value_to_array(Element super)

Name

Integer Super_vertex_image_value_to_array(Element super)

Description

If for the super string **super** the dimension Att_Vertex_Image_Value exists and the dimension Att_Vertex_Image_Array does not exist then there will be one image **img** for the entire string.

In this case (when the dimension Att_Vertex_Image_Value exists and the dimension Att_ZCoord_Array does not exist) this function sets the Att_Vertex_Image_Array dimension and creates a new image for each vertex of **super** and it is given the value **img**.

See Height Dimensions for information on the Height (ZCoord) dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 2176

Setting Super String Vertex Image Functions

Super_vertex_image_delete(Element elt,Integer vertex_num,Integer image_num)

Name

Integer Super_vertex_image_delete(Element super,Integer vertex_num,Integer image_num)

Description

For the super Element **super**, delete image number **image_num** from vertex number **vertex_num.**

A return value of 0 indicates the function call was successful.

ID = 1862

Super_vertex_image_delete_all(Element super,Integer vertex_num)

Name

Integer Super_vertex_image_delete_all(Element super,Integer vertex_num)

Description

For the super Element **super**, delete all the images on vertex number **vertex_num**.

A return value of 0 indicates the function call was successful.

ID = 1863

Get_super_vertex_number_of_images(Element super,Integer vertex_num,Integer &num_images)

Name

Integer Get_super_vertex_number_of_images(Element super,Integer vertex_num,Integer &num_images)

Description

For the super Element **super**, return in **num_images** the number of images on vertex number **vertex_num**.

A return value of 0 indicates the function call was successful.

ID = 1864

Get_super_vertex_image_type(Element elt,Integer vertex,Integer image_no,Text &image_type)

Name

Integer Get_super_vertex_image_type(Element elt,Integer vertex,Integer image_no,Text &image_type)

Description

what is image_type? (it is URL etc)

<no description>

ID = 1865

Super_vertex_add_URL(Element super,Integer vertex,Text url)

Name

Integer Super_vertex_add_URL(Element super,Integer vertex,Text url)

Description

image_vertex_array or value. Set the vertex to have text which is treated as url.

<no description>

ID = 1771

Get_super_vertex_URL(Element elt,Integer vertex,Integer image_no,Text &url)

Name

Integer Get_super_vertex_URL(Element elt,Integer vertex,Integer image_no,Text &url)

Description

get url. If not url type then error.

<no description>

ID = 1866

Get_Super_vertex_plan_image(Element super,Integer vertex,Integer image_no,Text &url,Real &width,Real &height,Real &angle,Real &offset_x,Real &offset_y)

Name

Integer Get_Super_vertex_plan_image(Element super,Integer vertex,Integer image_no,Text &url,Real &width,Real &height,Real &angle,Real &offset_x,Real &offset_y)

Description

an image type

<no description>

ID = 1867

Super String Visibility Functions

For definitions of the Visibility dimensions, see Visibility Dimensions

See Super String Combined Visibility
See Super String Vertex Visibility
See Super String Segment Visibility

Super String Combined Visibility

Set_super_use_visibility(Element super,Integer use)

Name

Integer Set_super_use_visibility(Element super,Integer use)

Description

Tell the super string whether to use, or remove, the dimension Att_Visible_Array.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

A return value of 0 indicates the function call was successful.

ID = 718

Get_super_use_visibility(Element super,Integer &use)

Name

Integer Get_super_use_visibility(Element super,Integer &use)

Description

Query whether the dimension Att_Visible_Array exists for the super string.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 719

Super String Vertex Visibility

Set_super_use_vertex_visibility_value(Element super,Integer use)

Name

Integer Set_super_use_vertex_visibility_value(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Vertex_Visible_Value is used or removed.

If Att_Vertex_Visible_Value is set and Att_Vertex_Visible_Array is not set, then there is only one visibility value for all vertices in **super.**

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions

for information on all the dimensions.

If Att_Vertex_Visible_Value is set then the visibility is the same for all vertices in **super**.

If **use** is 1, the dimension is set and the visibility is the same for **all** vertices.

If **use** is 0, the dimension is removed.

Note that if the dimension Att_Vertex_Visible_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

ID = 1580

Get_super_use_vertex_visibility_value(Element super,Integer &use)

Name

Integer Get_super_use_vertex_visibility_value(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Visible_Value exists for the super string **super**. If Att_Vertex_Visible_Value is set then there is one visibility value for all vertices in **super**.

If Att_Vertex_Visible_Value is set and Att_Vertex_Visible_Array is not set, then there is only one visibility value for all vertices in **super.**

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1581

Set_super_use_vertex_visibility_array(Element super,Integer use)

Name

Integer Set_super_use_vertex_visibility_array(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Vertex_Visible_Array is used or removed.

If Att_Vertex_Visible_Array is set then there can be a different visibility defined for each vertex in **super**.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and the visibility is different for each vertex.

If **use** is 0, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 1582

Get_super_use_vertex_visibility_array(Element super,Integer &use)

Name

Integer Get_super_use_vertex_visibility_array(Element super,Integer &use)

Description

Query whether the dimension Att_Vertex_Visible_Array exists for the super string **super**.

If Att_Vertex_Visible_Array is set then there can be a different visibility defined for each vertex in **super**.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1583

Set_super_vertex_visibility(Element super,Integer vert,Integer visibility)

Name

Integer Set_super_vertex_visibility(Element super,Integer vert,Integer visibility)

Description

For the Element **super** (which must be of type **Super**), set the visibility value for vertex number **vert** and to **visibility**.

If **visibility** is 1, the vertex is visible.
If **visibility** is 0, the vertex is invisible.

If the Element **super** is not of type **Super,** or Att_Vertex_Visible_Array is not set for **super**, then a non-zero return code is returned.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 734

Get_super_vertex_visibility(Element super,Integer vert,Integer &visibility)

Name

Integer Get_super_vertex_visibility(Element super,Integer vert,Integer &visibility)

Description

For the Element **super** (which must be of type **Super**), get the visibility value for vertex number **vert** and return it in the Integer **visibility**.

If **visibility** is 1, the vertex is visible.
If **visibility** is 0, the vertex is invisible.

If the Element **super** is not of type **Super,** or Att_Vertex_Visible_Array is not set for **super**, then a non-zero return code is returned.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 735

Super String Segment Visibility

Set_super_use_segment_visibility_value(Element super,Integer use)

Name

Integer Set_super_use_segment_visibility_value(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Segment_Visible_Value is used or removed.

If Att_Segment_Visible_Value is set and Att_Segment_Visible_Array is not set, then the visibility is the same for all segments in **super**.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and the visibility is the same for **all** segments.
If **use** is 0, the dimension is removed.

Note that if the dimension Att_Segment_Visible_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

ID = 1588

Get_super_use_segment_visibility_value(Element super,Integer &use)

Name

Integer Get_super_use_segment_visibility_value(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Visible_Value exists for the super string **super**.

If Att_Segment_Visible_Value is set and Att_Segment_Visible_Array is not set, then the visibility is the same for all segments in **super**.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1589

Set_super_use_segment_visibility_array(Element super,Integer use)

Name

Integer Set_super_use_segment_visibility_array(Element super,Integer use)

Description

For Element **super** of type **Super**, define whether the dimension Att_Segment_Visible_Array is used or removed.

If Att_Segment_Visible_Array is set then there can be a different visibility defined for each segment in **super**.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

If **use** is 1, the dimension is set and the visibility is different for each segment.
If **use** is 0, the dimension is removed.

A return value of 0 indicates the function call was successful.

ID = 1590

Get_super_use_segment_visibility_array(Element super,Integer &use)

Name

Integer Get_super_use_segment_visibility_array(Element super,Integer &use)

Description

Query whether the dimension Att_Segment_Visible_Array exists for the super string **super**.

If Att_Segment_Visible_Array is set then there can be a different visibility defined for each segment in **super**.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

ID = 1591

Set_super_segment_visibility(Element super,Integer seg,Integer visibility)

Name

Integer Set_super_segment_visibility(Element super,Integer seg,Integer visibility)

Description

For the Element **super** (which must be of type **Super**), set the visibility value for segment number **seg** to **visibility**.

If **visibility** is 1, the segment is visible.
If **visibility** is 0, the segment is invisible.

If the Element **super** is not of type **Super,** or Att_Segment_Visible_Array is not set for **super**, then a non-zero return code is returned.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 720

Get_super_segment_visibility(Element super,Integer seg,Integer &visibility)

Name

Integer Get_super_segment_visibility(Element super,Integer seg,Integer &visibility)

Description

For the Element **super** (which must be of type **Super**), get the visibility value for segment number **seg** and return it in the Integer **visibility**.

If **visibility** is 1, the segment is visible.
If **visibility** is 0, the segment is invisible.

If the Element **super** is not of type **Super,** or Att_Segment_Visible_Array is not set for **super**, then a non-zero return code is returned.

See Visibility Dimensions for information on the Visibility dimensions or Super String Dimensions

for information on all the dimensions.

A return value of 0 indicates the function call was successful.

ID = 721

Examples of Setting Up Super Strings

See 2d Super String
See 2d Super String with Arcs
See 3d Super String
See Polyline Super String
See Pipe Super String
See Culvert Super String
See Polyline Pipe Super String
See 4d Super String

2d Super String

A 2d string consists of (x,y) values at each vertex of the string and a **constant height** for the entire string. There are only straight segments joining the vertices.

Creating a 2d Super String with Straight Segments

To defined a super string ***super*** with num_vert vertices, and for it to have a constant height 30 say:

```
#include "setups.h"
Element super;
// need dimension 1 Att_ZCoord_Value to have the value 1 and all other dimensions are 0
Integer flag1 = String_Super_Bit(ZCoord_Value);
// NOTE:  this is the same as    flag1 = 1;        // dimension 1 only
super = Create_super(flag1, num_vert);
Set_super_2d_level(super,30.0);
Set_colour(super,4);              // cyan in the standard colours.4d
```

The data could then be loaded into *super* using repeated calls of

```
Set_super_vertex_coord(super,i,x,y,30.0);
```

where (x,y) are the coordinates of the ith vertex of *super, height is 30, i i*s the vertex index.

Checking for a 2d Super String

To check if a super string Element, ***super***, has a constant height (z-value), use the code:

```
Integer ret_h_value, use_h_value, ret_z_array, use_z_array;
ret_z_array = Get_super_use_3d(super, use_z_array);
ret_h_value = Get_super_use_2d(super, use_h_value);
```

If *ret*_z_array is 0 and *use_z_array* is 1 (from the *Get_super_use_3d* call) then the super string ***super*** has an array of z-values and so **isn't** like a 2d super string.

If the above does not hold then:

If *ret_h_value* is 0 and *use_h_value* is 0 (from the *Get_super_use_2d* call) then the super string ***super*** has a constant height dimension and is like a 2d string.

To find out the actual height of the 2d super string, use

```
Real height;
Get_super_2d_level(super,height);
```

The coordinate data can be read out of the super string *super* using repeated calls of

```
Get_super_vertex_coord(super,i,x,y,z);
```

where (x,y) are the coordinates of the ith vertex of *super*. The value z can be ignored if the height of the 2d string is already known.

2d Super String with Arcs

Unlike the superseded 2d string, it is possible to defined a super string ***super*** with a constant height for the entire string but rather than just having straight line segments between vertices, the segments may be arcs.

Creating a 2d Super String with Arc Segments

So to defined a super string ***super*** with num_vert vertices, and for it to have a constant height 30 say but also to have arc segments:

```
#include "setups.h"
Element super;
// need dimension 1 Att_ZCoord_Value, dimension 3 Att_Radius_Array and
// dimension 4 Att_Major_Array to have the value 1 and all other dimensions are 0
Integer flag1 = String_Super_Bit(ZCoord_Value)|String_Super_Bit(Radius_Array)
                |String_Super_Bit(Major_Array);
// NOTE:  this is the same as    flag1 = 13;        // dimensions 1, 3 and 4 only

super = Create_super(flag1, num_vert);
Set_super_2d_level(super,30.0);
Set_colour(super,4);              // cyan in the standard colours.4d
```

The data could then be loaded into *super* using repeated calls of

```
Set_super_data(super,i,x,y,30.0,r,b);
```

where (x,y) are the coordinates of the ith vertex of *super* and Real r and Integer b are the radius and major/minor arc bulge for the arc between vertex i and vertex i+1.

Checking for a 2d Super String with Arc Segments

To check if a super string Element, ***super***, has a constant height (z-value) and arc segments, use the code:

```
Integer ret_h_value, use_h_value, ret_z_array, use_z_array;
Integer ret_r_array, use_r_array, ret_b_array, use_b_array;
ret_z_array = Get_super_use_3d(super, use_z_array);
ret_h_value = Get_super_use_2d(super, use_h_value);
ret_r_array = Get_super_use_segment_radius(super, use_r_array);
// note - setting the super string to have radius array also forces it to have a major/minor arc
// bulge array
```

If *ret_z_*array is 0 and *use_z_array* is 1 (from the *Get_super_use_3d* call) then the super string ***super*** has an array of z-values and so **isn't** like a 2d super string.

If the above does not hold then:

If *ret_h_value* is 0 and *use_h_value* is 0 (from the *Get_super_use_2d* call) then the super string ***super*** has a constant height dimension and is like a 2d string.

To find out the actual height of the 2d super string, use

```
Real height;
```

```
Get_super_2d_level(super,height);
```

The coordinate data can be read out of the super string *super* using repeated calls of

```
Get_super_data(super,i,x,y,z,r,b);
```

where (x,y) are the coordinates of the ith vertex of *super* and Real r and Integer b will give the radius and major/minor arc bulge. The value z can be ignored if the height of the 2d string is already known.

3d Super String

A traditional 3d string consists of (x,y,z) values at each vertex of the string with straight line segments between each vertex.

Creating a 3d Super String with Straight Segments

To defined a super string ***super*** with num_vert vertices and different z-values at each vertex:

```
#include "setups.h"
Element super;
// need dimension 2 Att_ZCoord_Array (2) to have the value 1 and all other dimensions are 0
Integer flag1 = String_Super_Bit(ZCoord_Array);
// NOTE:  this is the same as    flag1 = 2;        // dimension 2 only
super = Create_super(flag1, num_vert);
Set_colour(super,4);              // cyan in the standard colours.4d
```

The data could then be loaded into *super* using repeated calls of

```
Set_super_vertex_coord(super,i,x,y,z);
```

where (x,y,z) are the coordinates of the ith vertex of *super*.

Checking for a 3d Super String

To check if a super string Element, ***super***, has a variable z-value, use the code:

```
Integer ret_z_array, use_z_array;
ret_z_array = Get_super_use_3d(super, use_z_array);
```

If *ret_z_*array is 0 and *use_z_array* is 1 (from the *Get_super_use_3d* call) then the super string ***super*** has an array of z-values and so is like a 3d super string.

The coordinate data can be read out of the super string *super* using repeated calls of

```
Get_super_vertex_coord(super,i,x,y,z);
```

where (x,y,z) are the coordinates of the ith vertex of *super*.

Polyline Super String

A traditional polyline string consists of (x,y,z) values at each vertex of the string and straight line **or arc** segments between each vertex. So each vertex has values (x,y,z,r,b) where r is the radius of the arc from this segment to the next segment and b is a major/minor arc bulge.

Creating a Polyline Super String (3d Super String with Arc Segments)

Unlike the old 3d string, it is possible to defined a super string ***super*** with a (x,y,z) coordinates at each vertex but rather than just having straight line segments between vertices, the segments may be arcs. This is then the traditional polyline string.

So to defined a super string ***super*** with num_vert vertices, with variable z, and also to have arc segments:

```
#include "setups.h"

Element super;

// need dimension Att_ZCoord_Array (2), dimension Att_Radius_Array (3) and
// dimension Att_Major_Array (4) to have the value 1 and all other dimensions are 0

Integer flag1 = String_Super_Bit(ZCord_Array)|String_Super_Bit(Radius_Array)
                |String_Super_Bit(Major_Array);

// NOTE:  this is the same as    flag1 = 14;        // dimensions 2, 3 and 4 only
// Att_Major_Array does not actually have to be set because it is automatically set with
// Att_Radius_Array

super = Create_super(flag1, num_vert);

Set_colour(super,4);            // cyan in the standard colours.4d
```

The data could then be loaded into *super* using repeated calls of

```
Set_super_data(super,i,x,y,y,z,r,b);
```

where (x,y,z) are the coordinates of the ith vertex of *super* and r and f are the radius and major/minor arc bulge for the arc between vertex i and vertex i+1.

NOTE: if the dimensions were not set when the super string was first created, then they can be created later using the Super_string_use calls. For example

```
Set_super_use_3d_level(super,1);        // sets on the Att_ZCoord_Array dimension
```

Checking for a Polyline Super String

To check if a super string Element, ***super*** has a variable z-value and allows a radius for each segment between vertices, use the code:

```
Integer ret_z_array, use_z_array;

Integer ret_r_array, use_r_array, ret_b_array, use_b_array;


ret_z_array = Get_super_use_3d(super, use_z_array);

ret_r_array = Get_super_use_segment_radius(super, use_r_array);

// note - setting the super string to have radius array also forces it to have a major/minor arc array
```

If *ret*_z_array is 0 and *use_z_array* is 1 (from the *Get_super_use_3d* call) then the super string ***super*** has an array of z-values and so is like a 3d string.

If *ret*_r_array is 0 and *use_r_array* is 1 (from the *Get_super_use_segment_radius* call) then the

super string ***super*** has an array of radii for the segments and so is like a polyline string.

The coordinate data can be read out of the super string *super* using repeated calls of

```
Get_super_data(super,i,x,y,z,r,b);
```

where (x,y,z) are the coordinates of the ith vertex of *super* and Real r and Integer b will give the radius and major/minor arc bulge flag for the segment from vertex i to vertex i+1.

Pipe Super String

A traditional pipe string consists of (x,y,z) values at each vertex of the string with straight line segments between each vertex, plus a diameter for the entire string. There is also a justification (invert, obvert, centre) for what ALL the z values represent for the pipe string.

Creating a Pipe Super String with Straight Segments

To defined a super string ***super*** with num_vert vertices and different z-values at each vertex, plus a pipe diameter and justification for the entire string:

```
#include "setups.h"
Element super;
// need dimension 2 Att_ZCoord_Array (2), Att_Pipe_Justify (23)
// and Att_Diameter_Value (5) to have the value 1, and all other dimensions are 0
Integer flag1 = String_Super_Bit(ZCoord_Array)|String_Super_Bit(Pipe_Justify)|
                String_Super_Bit(Diameter_Value);
super = Create_super(flag1, num_vert);
Set_super_pipe_justify(super,2);        // obvert justification for pipe string
Set_super_pipe(super,0.5,0.0,1));       // set the string internal diameter to 0.5 units and
                                        // 0 wall thickness
Set_colour(super,4);        // cyan in the standard colours.4d
```

The data could then be loaded into *super* using repeated calls of

```
Set_super_vertex_coord(super,i,x,y,z);
```

where (x,y,z) are the coordinates of the obvert of the ith vertex of *super*.

NOTE: if the dimensions were not set when the super string was first created, then they can be created later using the Super_string_use calls. For example

```
Set_super_use_3d_level(super,1);      // sets on the Att_ZCoord_Array dimension
Set_super_use_pipe(super,1);          // sets on the Att_Diameter_Value dimension
Set_super_use_pipe_justify(super,1);  // sets on the Att_Pipe_Justify dimension
```

Checking for a Pipe Super String

To check if a super string Element, ***super***, has a variable z-value, a diameter and a pipe justification, use the code:

```
Integer ret_z_array, use_z_array;
Integer ret_diam_value, use_diam_value;
Integer ret_justification_value, use_justification_value;

ret_z_array = Get_super_use_3d(super, use_z_array);
ret_diam_value = Get_super_use_pipe(super, use_diam_value);
ret_justification_value = Get_super_use_pipe_justify(super, use_justification_value);
```

If *ret*_z_array is 0 and *use_z_array* is 1 (from the *Get_super_use_3d* call) then the super string ***super*** has an array of z-values and so is like a 3d super string.

If *ret*_diam_value is 0 and *use_diam_value* is 1 (from the *Get_super_use_pipe* call) then the

super string ***super*** has a diameter for the entire string.

If *ret*_justification_value is 0 and *use_justification_value* is 1 (from the *Get_super_use_pipe_justify* call) then the super string ***super*** has a justification value to use for each vertex of the string.

The coordinate data can be read out of the super string *super* using repeated calls of

```
Get_super_vertex_coord(super,i,x,y,z);
```

where (x,y,z) are the coordinates of the ith vertex of *super*.

The diameter and thickness for the super string *super* can be obtained by the call

```
Real diameter, thickness;

Integer internal_diameter;
Get_super_pipe(super,diameter,internal_diameter);
```

The justification for the super string *super* can be obtained by the call

```
integer justify;
Get_super_pipe_justify(super,justify);
```

Culvert Super String

A simple box culvert consists of (x,y,z) values at each vertex of the string with straight line segments between each vertex, plus the one width and height for the entire string. There is also a justification (invert, obvert, centre) for what ALL the z values represent for the pipe string.

Creating a Culvert Super String with Straight Segments

To defined a super string ***super*** with num_vert vertices and different z-values at each vertex, plus a constant culvert width and height and justification for the entire string:

```
#include "setups.h"
   Element super;
// need dimension 2 Att_ZCoord_Array (2), Att_Pipe_Justify (23) and Att_Culvert_Value (24)
// to have the value 1, and all other dimensions are 0
   Integer flag1 = String_Super_Bit(ZCoord_Array)|String_Super_Bit(Pipe_Justify)|
                   String_Super_Bit(Culvert_Value);
   super = Create_super(flag1, num_vert);
   Set_super_pipe_justify(super,2);          // obvert justification for pipe string
   Set_super_culvert(super,10,5,1,1,1,1,1));        // set the string internal width to 10 units,
                                             // internal height to 5, and wall thickness of 1
   Set_colour(super,4);          // cyan in the standard colours.4d
```

The data could then be loaded into *super* using repeated calls of

```
   Set_super_vertex_coord(super,i,x,y,z);
```

where (x,y,z) are the coordinates of the obvert of the ith vertex of *super*.

NOTE: if the dimensions were not set when the super string was first created, then they can be created later using the Super_string_use calls. For example

```
   Set_super_use_3d_level(super,1);      // sets on the Att_ZCoord_Array dimension
   Set_super_use_pipe(super,1);          // sets on the Att_Diameter_Value dimension
   Set_super_use_pipe_justify(super,1);  // sets on the Att_Pipe_Justify dimension
```

Checking for a Culvert Super String with Constant Width and Height

To check if a super string Element, ***super***, has a variable z-value, a constant width and height and a pipe justification, use the code:

```
   Integer ret_z_array, use_z_array;
   Integer ret_culvert_value, use_culvert_value;
   Integer ret_justification_value, use_justification_value;

   ret_z_array = Get_super_use_3d(super, use_z_array);
   ret_culvert_value = Get_super_use_culvert(super, use_culvert_value);
   ret_justification_value = Get_super_use_pipe_justify(super, use_justification_value);
```

If *ret*_z_array is 0 and *use_z_array* is 1 (from the *Get_super_use_3d* call) then the super string ***super*** has an array of z-values and so is like a 3d super string.

If *ret*_culvert_value is 0 and *use_culvert_value* is 1 (from the *Get_super_use_culvert* call) then the super string ***super*** has one width and height for the entire string.

If *ret*_justification_value is 0 and *use_justification_value* is 1 (from the *Get_super_use_pipe_justify* call) then the super string ***super*** has a justification value to use for each vertex of the string.

The coordinate data can be read out of the super string *super* using repeated calls of

```
Get_super_vertex_coord(super,i,x,y,z);
```

where (x,y,z) are the coordinates of the ith vertex of *super*.

The width, height and four thicknesses for the super string *super* can be obtained by the call

```
Real width, height, left_thick, right_thick, top_thick, bottom_thick;

Integer internal_width, height;
Get_super_culvert(super,width,height,left_thick,right_thick,
                  top_thick,bottom_thick,internal_width_height);
```

The justification for the super string *super* can be obtained by the call

```
integer justify;
Get_super_pipe_justify(super,justify);
```

Polyline Pipe Super String

Unlike the old pipe string, it is possible to defined a super string ***super*** with a (x,y,z) coordinates at each vertex but rather than just having straight line segments between vertices, the segments may be arcs, plus a diameter and justification for the entire string. There is NO equivalent superseded string.

Creating a Polyline Pipe Super String

So to defined a super string ***super*** with num_vert vertices, with variable z, arc segments, diameter and justification:

```
#include "setups.h"
Element super;
// need dimensions Att_ZCoord_Array (2), Att_Radius_Array (3), Att_Major_Array (4),
// Att_Pipe_Justify (23) and Att_Diameter_Value (5) to have the value 1
// and all other dimensions the value 0
Integer flag1 = String_Super_Bit(ZCord_Array)|String_Super_Bit(Radius_Array)
                |String_Super_Bit(Major_Array)|String_Super_Bit(Pipe_Justify)
                |String_Super_Bit(Diameter_Value);
super = Create_super(flag1, num_vert);
Set_super_pipe_justify(super,0);          // invert justification for polyline pipe string
Set_super_pipe(super,0.5,0.0,1);          // set the string internal diameter to 0.5 units
//                                        // 0 wall thickness
Set_colour(super,4);          // cyan in the standard colours.4d
```

The data could then be loaded into *super* using repeated calls of

```
Set_super_data(super,i,x,y,y,z,r,b);
```

where (x,y,z) are the coordinates of the ith vertex of *super* and r and b are the radius and major/minor arc bulge for the arc between vertex i and vertex i+1.

NOTE: if the dimensions were not set when the super string was first created, then they can be created later using the Super_string_use calls. For example

```
Set_super_use_3d_level(super,1);        // sets on the Att_ZCoord_Array dimension
Set_super_use_segment_radius(super,1);  // sets on the Att_Radius_Array dimension
Set_super_use_pipe(super,1);            // sets on the Att_Diameter_Value dimension
Set_super_use_pipe_justify(super,1);    // sets on the Att_Pipe_Justify dimension
```

Checking for a Polyline Pipe Super String

To check if a super string Element, ***super*** has a variable z-value, allows a radius for each segment between vertices, and a diameter and justification for the string, use the code:

```
Integer ret_z_array, use_z_array;
Integer ret_r_array, use_r_array, ret_f_array, use_f_array;
Integer ret_diam_value, use_diam_value;
Integer ret_justification_value, use_justification_value;

ret_z_array = Get_super_use_3d(super, use_z_array);
```

```
   ret_r_array = Get_super_use_segment_radius(super, use_r_array);
// note - setting the super string to have a radius array also forces it to have
// a major/minor arc array
   ret_diam_value = Get_super_use_pipe(super, use_diam_value);
   ret_justification_value = Get_super_use_pipe_justify(super, use_justification_value);
```

If *ret*_z_array is 0 and *use_z_array* is 1 (from the *Get_super_use_3d* call) then the super string ***super*** has an array of z-values and so is like a 3d super string.

If *ret*_r_array is 0 and *use_r_array* is 1 (from the *Get_super_use_segment_radius* call) then the super string ***super*** has an array of radii for the segments and so is like a polyline string.

If *ret*_diam_value is 0 and *use_diam_value* is 1 (from the *Get_super_use_pipe* call) then the super string ***super*** has a diameter for the entire string.

If *ret*_justification_value is 0 and *use_justification_value* is 1 (from the *Get_super_use_pipe_justify* call) then the super string ***super*** has a justification value to use for each vertex of the string.

The coordinate data can be read out of the super string *super* using repeated calls of

```
Get_super_data(super,i,x,y,z,r,b);
```

where (x,y,z) are the coordinates of the ith vertex of *super* and Real r and Integer b will give the radius and major/minor arc bulge for the segment from vertex i to vertex i+1.

The diameter for the super string *super* can be obtained by the call

```
Real diameter;
Get_super_pipe(super,diameter);
```

The justification for the super string *super* can be obtained by the call

```
integer justify;
Get_super_pipe_justify(super,justify);
```

4d Super String

A traditional 4d string consists of different (x,y,z) values at each vertex (with straight line segments between each vertex) and also a different text at each vertex. So each vertex has the values (x,y,z,t) where (x,y,z) are the coordinates of the vertex and t is the text at the vertex.

The 4d string also has drawing information to describe how the text is drawn on a plan view or plot. All the text is drawn in the same way.

Creating a 4d Super String with Straight Segments

To defined a super string ***super*** with num_vert vertices and different z-values and text at each vertex. There are only straight segments between the vertices and all the text is drawn the same way: World units will be used for the text size.

```
#include "setups.h"
Element super;
// need dimensions Att_ZCoord_Array (2), Att_Vertex_Text_Array (7),
// Att_Vertex_Annotate_Value (14) and Att_Vertex_World_Annotate (30) to have the value 1
// and all other dimensions are 0
Integer flag1 = String_Super_Bit(ZCord_Array)|String_Super_Bit(Vertex_Text_Array)
                |String_Super_Bit(Vertex_Annotate_Value)
                |String_Super_Bit(Vertex_World_Annotate);
//
super = Create_super(flag1, num_vert);
Set_colour(super,4);          // cyan in the standard colours.4d
```

The drawing information for the text is set by

```
Set_super_vertex_text_style(super,1,"Arial");      // 1 is ignored, textstyle "Arial"
Set_super_vertex_text_colour(super,1,5);        // 1 is ignored, colour number is 5
Set_super_vertex_text_size(super,1,2.0);        // 1 is ignored, size is 2 world units
```

The data could then be loaded into *super* using repeated calls of

```
Set_super_vertex_coord(super,i,x,y,z);
Set_super_vertex_text(super,i,txt);
```

where (x,y,z) are the coordinates of the ith vertex of *super* and txt is the Text at vertex i.

NOTE: if the dimensions were not set when the super string was first created, then they can be created later using the Super_string_use calls. For example

```
Set_super_use_3d_level(super,1);        // sets on the Att_ZCoord_Array dimension
Set_super_use_vertex_text_array(super,1);   // sets on the Att_Vertex_Text_Array dimension
Set_super_use_vertex_annotation_value(super,1);   // sets on the
                                                  //Att_Vertex_Annotate_Value dimension
```

Checking for a 4d Super String

To check if a super string Element, ***super***, has a variable z-value, use the code:

```
Integer ret_z_array, use_z_array, ret_t_array, use_t_array;
ret_z_array = Get_super_use_3d(super, use_z_array);
```

```
ret_t_array = Get_super_use_vertex_text_array(super, use_t_array);
```

If *ret_z_*array is 0 and *use_z_array* is 1 (from the *Get_super_use_3d* call) then the super string ***super*** has an array of z-values and so is like a 3d super string.

If *ret*_t_array is 0 and *use_t_array* is 1 (from the *Get_super_use_vertex_text_array* call) then the super string ***super*** also has an array of text values and so is like a 4d string.

The coordinate data can be read out of the super string *super* using repeated calls of

```
Get_super_vertex_coord(super,i,x,y,z);
Get_super_vertex_text(super,i,txt);
```

where (x,y,z) are the coordinates of the ith vertex of *super*, and txt is the Text at the ith vertex.

Super Alignment String Element

A Super Alignment string holds both the horizontal and vertical information needed in defining entities such as the centre line of a road.

Horizontal intersection points (hips), lines, arcs and transitions (such as spirals) are used to define the plan geometry.

Vertical intersection points (vips), lines and parabolic and circular curves are used to define the vertical geometry.

The process to define an Super Alignment string is

(a) create an Super Alignment Element

(b) add the horizontal geometry

(c) perform a Calc_alignment on the string

(d) add the vertical geometry

(e) perform a Calc_alignment

For an existing Super Alignment string, there are functions to get the positions of all critical points (such as horizontal and vertical tangent points, spiral points, curve centres) for the string.

The functions used to create new Super Alignment strings and make inquiries and modifications to existing Alignment strings now follow.

Element Create_super_align()

Name

Element Create_align()

Description

Create an Element of type **Super_Alignment**.

The function return value gives the actual Element created.

If the Super Alignment string could not be created, then the returned Element will be null.

ID = 2120

Create_super_align(Element seed)

Name

Element Create_align(Element seed)

Description

Create an Element of type **Super_Alignment**, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

If the Super Alignment string could not be created, then the returned Element will be null.

ID = 2121

Is_super_alignment_solved(Element super_alignment)

Name

Integer Is_super_alignment_solved(Element super_alignment)

Description

Check if the geometry of the Element **super_alignment** solves.

The Element **super_alignment** must be of type Super_Alignment.

A no-zero function return value indicates that the geometry will solve.

A zero function return value indicates the geometry for the will **not** solve, or that **super_alignment** is not of type Super_Alignment.

Warning this is the opposite of most 12dPL function return values.

ID = 2680

Arc String Element

A 12d Model **Arc** string is similar to the entity Arc in that it is a helix which projects onto an arc in the (x,y) plane.

The Element type Arc has a radius and three dimensional co-ordinates for its centre, start and end points. The radius can be positive or negative.

A positive radius indicates that the direction of travel between the start and end points is in the clockwise direction (right hand curve).

A negative radius indicates that the direction of travel between the start and end points is in the anti-clockwise direction (left hand curve).

Unlike the variable of type Arc, the Element arc string has Element header information and can be added to 12d Model models. Thus arc strings can be drawn on a 12d Model view and stored in the 12d Model database.

Create_arc(Arc arc)

Name

Element Create_arc(Arc arc)

Description

Create an Element of type **Arc** from the Arc **arc**.

The arc string has the same centre, radius, start and end points as the Arc **arc**.

The function return value gives the actual Element created.

If the arc string could not be created, then the returned Element will be null.

ID = 294

Create_arc(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3)

Name

Element Create_arc(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3)

Description

Create an Element of type **Arc** through three given points.

The arc string has start point (x1,y1,z1), an intermediate point (x2,y2,z2) on the arc and the end point (x3,y3,z3).

The centre and radius of the arc will be automatically calculated.

The function return value gives the actual Element created.

If the arc string could not be created, then the returned Element will be null.

ID = 312

Create_arc(Real xc,Real yc,Real zc,Real rad,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze)

Name

Element Create_arc(Real xc,Real yc,Real zc,Real rad,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze)

Description

Create an Element of type **Arc** with centre (**xc**,**yc**,**zc**), radius **rad**, start point (**xs,ys,zs**) and end point (**xe,ye,ze**).

The function return value gives the actual Element created.

If the arc string could not be created, then the returned Element will be null.

ID = 296

Create_arc(Real xc,Real yc,Real zc,Real rad,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze)

Name

Element Create_arc(Real xc,Real yc,Real zc,Real rad,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze)

Description

Create an Element of type **Arc** with centre (xc,yc,zc), and radius **rad**.

The points (**xs,ys,zs**) and (**xe,ye,ze**) define the start and end points respectively for the arc. If either of the points do not lie on the plan circle with centre (xc,yc) and radius **rad**, then the point is dropped perpendicularly onto the plan circle to define the (x,y) co-ordinates for the relevant start or end point.

The function return value gives the actual Element created.

If the arc string could not be created, then the returned Element will be null.

ID = 296

Create_arc(Real xc,Real yc,Real zc,Real xs,Real ys,Real zs,Real sweep)

Name

Element Create_arc(Real xc,Real yc,Real zc,Real xs,Real ys,Real zs,Real sweep)

Description

Create an Element of type **Arc** with centre point (**xc,yc,zc**), start point (**xs,ys,zs**) and sweep angle **sweep**.

The absolute radius is calculated as the distance between the centre and start point of the arc. The sign of the radius comes from the sweep angle.

The sweep angle is measured in a clockwise direction from the line joining the centre to the arc start point. The units for sweep angles are radians.

Hence the sweep angle is measured in radians and a positive value indicates a clockwise direction and a positive radius.

The end point of the arc will be automatically created.

The function return value gives the actual Element created.

If the arc string could not be created, then the returned Element will be null.

ID = 313

Create_arc(Real xc,Real yc,Real zc,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze,Integer dir)

Name

Element Create_arc(Real xc,Real yc,Real zc,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze,Integer dir)

Description

Create an Element of type **Arc** with centre (**xc,yc,zc**), start point (**xs,ys,zs**) and end point

(**xe,ye,ze**).

The absolute radius is calculated as the distance between the centre and start point of the arc.

If **dir** is positive, the radius is taken to be positive.

If **dir** is negative, the radius is taken to be negative.

The function return value gives the actual Element created.

If the arc string could not be created, then the returned Element will be null.

ID = 314

Create_arc_2(Real xs,Real ys,Real zs,Real rad,Real arc_length,Real start_angle)

Name

Element Create_arc_2(Real xs,Real ys,Real zs,Real rad,Real arc_length,Real start_angle)

Description

Create an Element of type **Arc** with radius **rad**. The arc starts at the point (xs,ys,zs) with tangent angle **start_angle** and total arc length **arc_length**.

The centre and end points will be automatically created.

The function return value gives the actual Element created.

If the arc string could not be created, then the returned Element will be null.

ID = 316

Create_arc_3(Real xs,Real ys,Real zs,Real rad,Real arc_length,Real chord_angle)

Name

Element Create_arc_3(Real xs,Real ys,Real zs,Real rad,Real arc_length,Real chord_angle)

Description

Create an Element of type **Arc** with radius **rad**. The arc starts at the point (xs,ys,zs) with a chord angle **chord_angle** and total arc length **arc_length**.

The centre and end points will be automatically created.

The function return value gives the actual Element created.

If the arc string could not be created, then the returned Element will be null.

ID = 317

Set_arc_centre(Element elt,Real xc,Real yc,Real zc)

Name

Integer Set_arc_centre(Element elt,Real xc,Real yc,Real zc)

Description

Set the centre point of the Arc string given by Element **elt** to (**xc,yc,zc**).

The start and end points are also translated by the plan distance between the old and new centre.

A function return value of zero indicates the centre was successfully modified.

ID = 319

Get_arc_centre(Element elt,Real &xc,Real &yc,Real &zc)

Name

Integer Get_arc_centre(Element elt,Real &xc,Real &yc,Real &zc)

Description

Get the centre point for Arc string given by Element **elt**.

The centre of the arc is (**xc,yc,zc**).

A function return value of zero indicates the centre was successfully returned.

ID = 318

Set_arc_radius(Element elt,Real rad)

Name

Integer Set_arc_radius(Element elt,Real rad)

Description

Set the radius of the Arc string given by Element **elt** to **rad**. The new radius must be non-zero.

The start and end points are projected radially so that they still lie on the arc.

A function return value of zero indicates the radius was successfully modified.

ID = 321

Get_arc_radius(Element elt,Real &rad)

Name

Integer Get_arc_radius(Element elt,Real &rad)

Description

Get the radius for Arc string given by Element **elt**.

The radius is given by **rad**.

A function return value of zero indicates the radius was successfully returned.

ID = 320

Set_arc_start(Element elt,Real xs,Real ys,Real zs)

Name

Integer Set_arc_start(Element elt,Real xs,Real ys,Real zs)

Description

Set the start point of the Arc string given by Element **elt** to (**xs,ys,zs**).

If the start point does not lie on the arc, then the point (xs,ys,zs) is projected radially onto the arc and the projected point taken as the start point.

A function return value of zero indicates the start point was successfully modified.

ID = 323

Get_arc_start(Element elt,Real &xs,Real &ys,Real &zs)

Name

Integer Get_arc_start(Element elt,Real &xs,Real &ys,Real &zs)

Description

Get the start point for Arc string given by Element **elt**.

The start of the arc is (**xs,ys,zs**).

A function return value of zero indicates that the start point was successfully returned.

ID = 322

Set_arc_end(Element elt,Real xe,Real ye,Real ze)

Name

Integer Set_arc_end(Element elt,Real xe,Real ye,Real ze)

Description

Set the end point of the Arc string given by Element **elt** to (**xe,ye,ze**).

If the end point does not lie on the arc, then the point (xe,ye,ze) is projected radially onto the arc and the projected point taken as the end point.

A function return value of zero indicates the end point was successfully modified.

ID = 325

Get_arc_end(Element elt,Real &xe,Real &ye,Real &ze)

Name

Integer Get_arc_end(Element elt,Real &xe,Real &ye,Real &ze)

Description

Get the end point for Arc string given by Element **elt**.

The end of the arc is (**xe,ye,ze**).

A function return value of zero indicates that the end point was successfully returned.

ID = 324

Set_arc_data(Element elt,Real xc,Real yc,Real zc, Real rad,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze)

Name

Integer Set_arc_data(Element elt,Real xc,Real yc,Real zc,Real rad,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze)

Description

Set the data for the Arc string given by Element **elt**.

The arc is given the centre (**xc,yc,zc**), radius rad and start and end points (**xs,ys,zs**) and (**xe,ye,ze**) respectively.

A function return value of zero indicates the arc data was successfully set.

ID = 327

Get_arc_data(Element elt,Real &xc,Real &yc,Real &zc,Real &rad,Real &xs,Real &ys,Real &zs,Real &xe,Real &ye,Real &ze)

Name

Integer Get_arc_data(Element elt,Real &xc,Real &yc,Real &zc,Real &rad,Real &xs,Real &ys,Real &zs,Real &xe,Real &ye,Real &ze)

Description

Get the data for the Arc string given by Element **elt**.

The arc has centre (**xc**,**yc**,**zc**), radius **rad** and start and end points (**xs**,**ys**,**zs**) and (**xe**,**ye**,**ze**) respectively.

A function return value of zero indicates that the arc date was successfully returned.

ID = 326

Circle String Element

A 12d Model Circle string is a circle in the (x,y) plane with a constant z value (height).

Create_circle(Real xc,Real yc,Real zc,Real rad)

Name

Element Create_circle(Real xc,Real yc,Real zc,Real rad)

Description

Create an Element of type **Circle** with centre (**xc,yc**), radius **rad** and z value (height) **zc**.

The function return value gives the actual Element created.

If the circle string could not be created, then the returned Element will be null.

ID = 307

Create_circle(Real xc,Real yc,Real zc, Real xp,Real yp,Real zp)

Name

Element Create_circle(Real xc,Real yc,Real zc,Real xp,Real yp,Real zp)

Description

Create an Element of type **Circle** with centre (**xc,yc**) and point (**xp,yp**) on the circle.

The height of the circle is **zc**.

The radius of the circle will be automatically calculated.

The function return value gives the actual Element created.

If the circle string could not be created, then the returned Element will be null.

ID = 308

Create_circle(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3)

Name

Element Create_circle(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3)

Description

Create an Element of type **Circle** going through the three points (**x1,y1**), (**x2,y2**) and (**x3,y3**).

The height of the circle is **z1**.

The centre and radius of the circle will be automatically created.

The function return value gives the actual Element created.

If the circle string could not be created, then the returned Element will be null.

ID = 309

Set_circle_data(Element elt,Real xc,Real yc,Real zc,Real rad)

Name

Integer Set_circle_data(Element elt,Real xc,Real yc,Real zc,Real rad)

Description

Set the data for the Circle string given by Element **elt**.

The centre of the circle is set to (**xc,yc,zc**), the height to **zc** and the radius to **rad**.

A function return value of zero indicates success.

ID = 311

Get_circle_data(Element elt,Real &xc,Real &yc,Real &zc,Real &rad)

Name

Integer Get_circle_data(Element elt,Real &xc,Real &yc,Real &zc,Real &rad)

Description

Get the data for the Circle string given by Element **elt**.

The centre of the circle is (**xc,yc,zc**), height **zc**

and radius **rad**.

A function return value of zero indicates success.

ID = 310

Text String Element

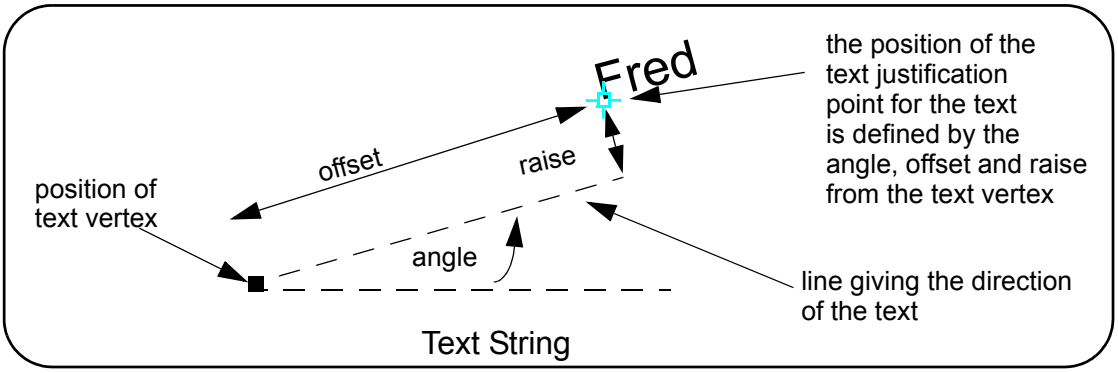
A Text String consists of text positioned with respect to the text vertex point (x,y).

The text is defined by parameters that can be individually set, or set all at once by setting a Textstyle_Data.

The current parameters contained in the Textstyle_Data structure and used for a Text String are:

the text itself, text style, colour, height, offset, raise, justification, angle, slant, xfactor, italic, strikeout, underlines, weight, whiteout, border and a name.

The parameters are described in the section Textstyle Data



The following functions are used to create new text strings and make inquiries and modifications to existing text strings.

Create_text(Text text,Real x,Real y,Real size,Integer colour)

Name

Element Create_text(Text text,Real x,Real y,Real size,Integer colour)

Description

Creates an Element of type **Text**.

The Element is at position (**x,y**), has Text **text** of size **size** and colour **colour**. The other data is defaulted.

The function return value gives the actual Element created.

If the text string could not be created, then the returned Element will be null.

ID = 174

Create_text(Text text,Real x,Real y,Real size,Integer colour,Real ang)

Name

Element Create_text(Text text,Real x,Real y,Real size,Integer colour,Real ang)

Description

Creates an Element of type **Text**.

The Element is at position (**x,y**), has Text **text** of size **size**, colour **colour** and angle **ang**. The other data is defaulted.

The function return value gives the actual Element created.

If the text string could not be created, then the returned Element will be null.

ID = 175

Create_text(Text text,Real x,Real y,Real size,Integer colour,Real ang,Integer justif)

Name

Element Create_text(Text text,Real x,Real y,Real size,Integer colour,Real ang,Integer justif)

Description

Creates an Element of type **Text**.

The Element is at position (**x,y**), has Text **text** of size **size**, colour **colour**, angle **ang** and justification **justif**. The other data is defaulted.

The function return value gives the actual Element created.

If the text string could not be created, then the returned Element will be null.

ID = 176

Create_text(Text text,Real x,Real y,Real size,Integer colour,Real ang,Integer justif, Integer size_mode)

Name

Element Create_text(Text text,Real x,Real y,Real size,Integer colour,Real ang,Integer justif,Integer size_mode)

Description

Creates an Element of type Text.

The Element is at position (**x,y**), has Text **text** of size **size**, colour **colour**, angle **ang**, justification **justif** and size mode **size_mode**. The other data is defaulted.

The function return value gives the actual Element created.

If the text string could not be created, then the returned Element will be null.

ID = 177

Create_text(Text text,Real x,Real y,Real size,Integer colour,Real ang,Integer justif,Integer size_mode,Real offset_distance,Real rise_distance)

Name

Element Create_text(Text text,Real x,Real y,Real size,Integer colour,Real ang,Integer justif,Integer size_mode,Real offset_distance,Real rise_distance)

Description

Creates an Element of type **Text**.

The Element is at position (**x,y**), has Text **text** of size **size**, colour **colour**, angle **ang**, justification **justif**, size mode **size_mode**, offset **offset_distance** and rise **rise_distance**.

The function return value gives the actual Element created.

If the text string could not be created, then the returned Element will be null.

ID = 178

Set_text_data(Element elt,Text text,Real x,Real y,Real size,Integer colour,Real

ang,Integer justif,Integer size_mode,Real offset_distance,Real rise_distance)

Name

Integer Set_text_data(Element elt,Text text,Real x,Real y,Real size,Integer colour,Real ang,Integer justif,Integer size_mode,Real offset_distance,Real rise_distance)

Description

Set values for each of the text parameters.

For a diagram, see Textstyle Data .

A function return value of zero indicates that the text data was successfully set.

ID = 180

Get_text_data(Element elt,Text &text,Real &x,Real &y,Real &size,Integer &colour,Real &ang,Integer &justification,Integer &size_mode,Real &offset_dist,Real &rise_dist)

Name

Integer Get_text_data(Element elt,Text &text,Real &x,Real &y,Real &size,Integer &colour,Real &ang,Integer &justification,Integer &size_mode,Real &offset_dist,Real &rise_dist)

Description

Get the values for each of the text parameters.

For a diagram, see Textstyle Data .

A function return value of zero indicates that the text data was successfully returned.

ID = 179

Set_text_value(Element elt,Text text)

Name

Integer Set_text_value(Element elt,Text text)

Description

Set the actual text of the text Element **elt**.

The text is given as Text **text**.

A function return value of zero indicates the data was successfully set.

ID = 461

Get_text_value(Element elt,Text &text)

Name

Integer Get_text_value(Element elt,Text &text)

Description

Get the actual text of the text Element **elt**.

The text is returned as Text **text**.

A function return value of zero indicates the data was successfully returned.

ID = 453

Set_text_textstyle_data(Element elt,Textstyle_Data d)

Name

Integer Set_text_textstyle_data(Element elt,Textstyle_Data d)

Description

For the Element **elt** of type **Text,** set the Textstyle_Data to be **d**.

Setting a Textstyle_Data means that all the individual values that are contained in the Textstyle_Data are set rather than having to set each one individually.

LJG? if the value is blank in the Textstyle_Data and the value is already set for the text string, is the value left alone?

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates the Textstyle_Data was successfully set.

ID = 1669

Get_text_textstyle_data(Element elt,Textstyle_Data &d)

Name

Integer Get_text_textstyle_data(Element elt,Textstyle_Data &d)

Description

For the Element **elt** of type **Text,** get the Textstyle_Data for the string and return it as **d**.

LJG? if a value is not set in the text string, what does it return?

A non-zero function return value is returned if **elt** is not of type **Text**.

A function return value of zero indicates the Textstyle_Data was successfully returned.

ID = 1670

Get_text_length(Element elt,Real &length)

Name

Integer Get_text_length(Element elt,Real &length)

Description

Get the length of the characters of the text Element **elt**.

The text length is returned as Real **length**.

A function return value of zero indicates the data was successfully returned.

ID = 580

Set_text_xy(Element elt,Real x,Real y)

Name

Integer Set_text_xy(Element elt,Real x,Real y)

Description

Set the base position of for the text Element **elt**.

The position is given as Real (**x,y**).

A function return value of zero indicates the data was successfully set.

ID = 462

Get_text_xy(Element elt,Real &x,Real &y)

Name

Integer Get_text_xy(Element elt,Real &x,Real &y)

Description

Get the base position of for the text Element **elt**.

The position is returned as Real (**x,y**).

A function return value of zero indicates the data was successfully returned.

ID = 454

Set_text_units(Element elt,Integer units_mode)

Name

Integer Set_text_units(Element elt,Integer units_mode)

Description

Set the units used for the text parameters of the text Element **elt**.

The mode is given as Integer **units_mode**.

For the values of **units_mode**, see Textstyle Data .

A function return value of zero indicates the data was successfully set.

ID = 466

Get_text_units(Element elt,Integer &units_mode)

Name

Integer Get_text_units(Element elt,Integer &units_mode)

Description

Get the units used for the text parameters of the text Element **elt**.

The mode is returned as Integer **units_mode**.

For the values of **units_mode**, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 458

Set_text_size(Element elt,Real size)

Name

Integer Set_text_size(Element elt,Real size)

Description

Set the size of the characters of the text Element **elt**.

The text size is returned as Real **size**.

A function return value of zero indicates the data was successfully set.

ID = 463

Get_text_size(Element elt,Real &size)

Name

Integer Get_text_size(Element elt,Real &size)

Description

Get the size of the characters of the text Element **elt**.

The text size is returned as Real **size**.

A function return value of zero indicates the data was successfully returned.

ID = 455

Set_text_justify(Element elt,Integer justify)

Name

Integer Set_text_justify(Element elt,Integer justify)

Description

Set the justification used for the text Element **elt**.

The justification is given as Integer **justify**.

*For the values of **justify** and their meaning, see* Textstyle Data .

A function return value of zero indicates the data was successfully set.

ID = 465

Get_text_justify(Element elt,Integer &justify)

Name

Integer Get_text_justify(Element elt,Integer &justify)

Description

Get the justification used for the text Element **elt**.

The justification is returned as Integer **justify**.

*For the values of **justify** and their meaning, see* Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 457

Set_text_angle(Element elt,Real ang)

Name

Integer Set_text_angle(Element elt,Real ang)

Description

Set the angle of rotation (in radians) about the text (**x,y**) point of the text Element **elt**.

The angle is given as Real **ang**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully set.

ID = 464

Get_text_angle(Element elt,Real &ang)

Name

Integer Get_text_angle(Element elt,Real &ang)

Description

Get the angle of rotation (in radians) about the text (x,y) point of the text Element **elt** and return the angle as **ang**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 456

Set_text_offset(Element elt,Real offset)

Name

Integer Set_text_offset(Element elt,Real offset)

Description

Set the offset distance of the text Element **elt**.

The offset is given as Real **offset**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully set.

ID = 467

Get_text_offset(Element elt,Real &offset)

Name

Integer Get_text_offset(Element elt,Real &offset)

Description

Get the offset distance of the text Element **elt**.

The offset is returned as Real **offset**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 459

Set_text_rise(Element elt,Real rise)

Name

Integer Set_text_rise(Element elt,Real rise)

Description

Set the rise distance of the text Element **elt**.

The rise is returned as Real **rise**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully set.

ID = 468

Get_text_rise(Element elt,Real &rise)

Name

Integer Get_text_rise(Element elt,Real &rise)

Description

Get the rise distance of the text Element **elt**.

The rise is returned as Real **rise**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 460

Set_text_height(Element elt,Real height)

Name

Integer Set_text_height(Element elt,Real height)

Description

Set the height of the characters of the text Element **elt**.

The text height is given as Real **height**.

A function return value of zero indicates the data was successfully set.

ID = 584

Get_text_height(Element elt,Real &height)

Name

Integer Get_text_height(Element elt,Real &height)

Description

Get the height of the characters of the text Element **elt**.

The text height is returned as Real **height**.

A function return value of zero indicates the data was successfully returned.

ID = 579

Set_text_slant(Element elt,Real slant)

Name

Integer Set_text_slant(Element elt,Real slant)

Description

Set the slant of the characters of the text Element **elt**.

The text slant is given as Real **slant**.

A function return value of zero indicates the data was successfully set.

ID = 585

Get_text_slant(Element elt,Real &slant)

Name

Integer Get_text_slant(Element elt,Real &slant)

Description

Get the slant of the characters of the text Element **elt**.

The text slant is returned as Real **slant**.

A function return value of zero indicates the data was successfully returned.

ID = 581

Set_text_style(Element elt,Text style)

Name

Integer Set_text_style(Element elt,Text style)

Description

Set the style of the characters of the text Element **elt**.

The text style is given as Text **style**.

A function return value of zero indicates the data was successfully set.

ID = 587

Get_text_style(Element elt,Text &style)

Name

Integer Get_text_style(Element elt,Text &style)

Description

Get the style of the characters of the text Element **elt**.

The text style is returned as Text **style**.

A function return value of zero indicates the data was successfully returned.

ID = 583

Set_text_x_factor(Element elt,Real xfact)

Name

Integer Set_text_x_factor(Element elt,Real xfact)

Description

Set the x factor of the characters of the text Element **elt**.

The text x factor is given as Real **xfact**.

A function return value of zero indicates the data was successfully set.

ID = 586

Get_text_x_factor(Element elt,Real &xfact)

Name

Integer Get_text_x_factor(Element elt,Real &xfact)

Description

Get the x factor of the characters of the text Element **elt**.

The text x factor is returned as Real **xfact**.

A function return value of zero indicates the data was successfully returned.

ID = 582

Set_text_ttf_underline(Element elt,Integer underline)

Name

Integer Set_text_ttf_underline(Element elt,Integer underline)

Description

For the Element **elt** of type **Text,** set the underline state to **underline**.

If **underline** = 1, then for a true type font the text will be underlined.
If **underline** = 0, then text will not be underlined.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates underlined was successfully set.

ID = 2596

Get_text_ttf_underline(Element elt,Integer &underline)

Name

Integer Get_text_ttf_underline(Element elt,Integer &underline)

Description

For the Element **elt** of type **Text,** get the underline state and return it in **underline**.

If **underline** = 1, then for a true type font the text will be underlined.
If **underline** = 0, then text will not be underlined.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates underlined was successfully returned.

ID = 2592

Set_text_ttf_strikeout(Element elt,Integer strikeout)

Name

Integer Set_text_ttf_strikeout(Element elt,Integer strikeout)

Description

For the Element **elt** of type **Text,** set the strikeout state to **strikeout**.

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates strikeout was successfully set.

ID = 2597

Get_text_ttf_strikeout(Element elt,Integer &strikeout)

Name

Integer Get_text_ttf_strikeout(Element elt,Integer &strikeout)

Description

For the Element **elt** of type **Text,** get the strikeout state and return it in **strikeout**.

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates strikeout was successfully returned.

ID = 2593

Set_text_ttf_italic(Element elt,Integer italic)

Name

Integer Set_text_ttf_italic(Element elt,Integer italic)

Description

For the Element **elt** of type **Text,** set the italic state to **italic**.

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates italic was successfully set.

ID = 2598

Get_text_ttf_italic(Element elt,Integer &italic)

Name

Integer Get_text_ttf_italic(Element elt,Integer &italic)

Description

For the Element **elt** of type **Text,** get the italic state and return it in **italic**.

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates italic was successfully returned.

ID = 2594

Set_text_ttf_outline(Element elt,Integer outline)

Name

Integer Set_text_ttf_outline(Element elt,Integer outline)

Description

For the Element **elt** of type **Text,** set the outline state to **outline**.

If **outline** = 1, then for a true type font the text will be only shown in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates ***outline*** was successfully set.

ID = 2772

Get_text_ttf_outline(Element elt,Integer &outline)

Name

Integer Get_text_ttf_outline(Element elt,Integer &outline)

Description

For the Element **elt** of type **Text,** get the outline state and return it in **outline**.

If **outline** = 1, then for a true type font the text will be shown only in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates ***outline*** was successfully returned.

ID = 2771

Set_text_ttf_weight(Element elt,Integer weight)

Name

Integer Set_text_ttf_weight(Element elt,Integer weight)

Description

For the Element **elt** of type **Text,** set the font weight to **weight**.

For the list of allowable weights, go to Allowable Weights

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates weight was successfully set.

ID = 2599

Get_text_ttf_weight(Element elt,Integer &weight)

Name

Integer Get_text_ttf_weight(Element elt,Integer &weight)

Description

For the Element **elt** of type **Text,** get the font weight and return it in **weight**.

For the list of allowable weights, go to Allowable Weights

A non-zero function return value is returned if **elt** is not of type **Text.**

A function return value of zero indicates weight was successfully returned.

ID = 2595

Set_text_whiteout(Element text,Integer colour)

Name

Integer Set_text_whiteout(Element text,Integer colour)

Description

For the Text Element **text**, set the colour number of the colour used for the whiteout box around the text, to be **colour**.

If no text whiteout is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully set.

ID = 2752

Get_text_whiteout(Element text,Integer &colour)

Name

Integer Get_text_whiteout(Element text,Integer &colour)

Description

For the Text Element **text**, get the colour number that is used for the whiteout box around the text. The whiteout colour is returned as Integer **colour**.

NO_COLOUR is the returned as the colour number if whiteout is not being used.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully returned.

ID = 2751

Set_text_border(Element text,Integer colour)

Name

Integer Set_text_border(Element text,Integer colour)

Description

For the Text Element **text**, set the colour number of the colour used for the border of the whiteout box around the text, to be **colour**.

If no whiteout border is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully set.

ID = 2762

Get_text_border(Element text,Integer &colour)

Name

Integer Get_text_border(Element text,Integer &colour)

Description

For the Text Element **text**, get the colour number that is used for the border of the whiteout box around the text. The whiteout border colour is returned as Integer **colour**.

NO_COLOUR is the returned as the colour number if there is no whiteout border.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff)

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully returned.

ID = 2761

Pipeline String Element

Integer Create_pipeline()

Name

Integer Create_pipeline()

Description

Create a pipeline.

A function return value of zero indicates the pipeline was created successfully.

ID = 1264

Create_pipeline(Element seed)

Name

Integer Create_pipeline(Element seed)

Description

Create an Element of type **Pipeline**, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

A function return value of zero indicates the **pipeline** was created successfully.

ID = 1265

Set_pipeline_diameter(Element pipeline,Real diameter)

Name

Integer Set_pipeline_diameter(Element pipeline,Real diameter)

Description

Set the **diameter** for pipeline.

Type of the diameter must be **Real**.

A function return value of zero indicates the **diameter** was successfully set.

ID = 1266

Get_pipeline_diameter(Element pipeline,Real &diameter)

Name

Integer Get_pipeline_diameter(Element pipeline,Real &diameter)

Description

Get the **diameter** from the Element **pipeline**.

The type of **diameter** must be **Real**.

A function return value of zero indicates the **diameter** was returned successfully.

ID = 1268

Set_pipeline_length(Element pipeline,Real length)

Name

Integer Set_pipeline_length(Element pipeline,Real length)

Description

Set the **length** for pipeline.

Type of the length must be **Real**.

A function return value of zero indicates the **length** was successfully set.

ID = 1267

Get_pipeline_length(Element pipeline,Real &length)

Name

Integer Get_pipeline_length(Element pipeline,Real &length)

Description

Get the **length** from the Element **pipeline**.

The type of **length** must be **Real**.

A function return value of zero indicates the **length** was returned successfully.

ID = 1269

Drainage String Element

Drainage Definitions

See Drainage Definitions - Pits and Pipes
See Drainage Definitions - Connection Points
See Drainage Definitions - Flow Direction
See Drainage Definitions - Drainage Network, Junction, Trunk

Drainage Definitions - Pits and Pipes

The **drainage** string is used in the ***Drainage*** modules (Drainage, Drainage Analysis and Dynamic Drainage Analysis) and also in the ***Sewer*** (Waste Water) module.

Drainage strings have a special attribute (**sewertype**) to denote whether the drainage string represents a storm water (sewertype = 0 the default) or a waste water (foul water or sewer) string (sewertype = 1) but both will be referred to as a ***drainage*** string.

A **drainage** string consists of two parts:

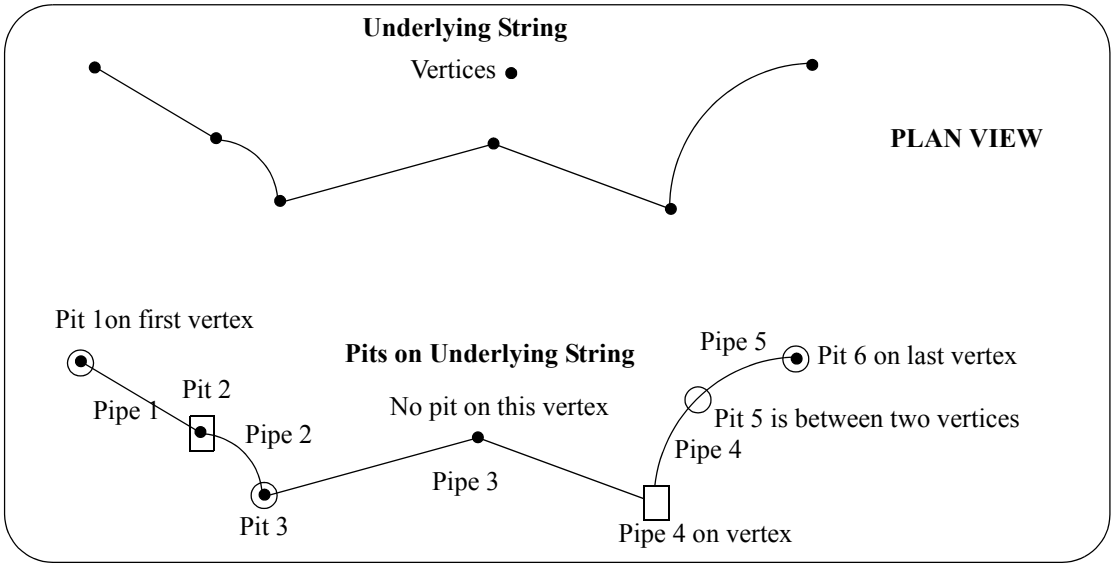
(a) a string of vertices with straight or arc segments that defines the underlying geometry of the drainage string

(b) information about pits (or maintenance holes) and pipes.

Pits (maintenance holes or manholes) can be located anywhere on the underlying string but are normally located on actual vertices of the underlying string. There must be a pit on the first and last vertices of the underlying string.

Pits can be circular or rectangular, and have a depth, cover, grate and sump levels as well as wall and bottom thicknesses.

Pipe are the conduits connecting the pits, and pipes can be round or rectangular.



The number of pits is return in ***npits*** by the function Get_drainage_pits(Element drain,Integer &npits) and the ***number of pipes*** = ***number of pits*** - 1.

Drainage Definitions - Connection Points

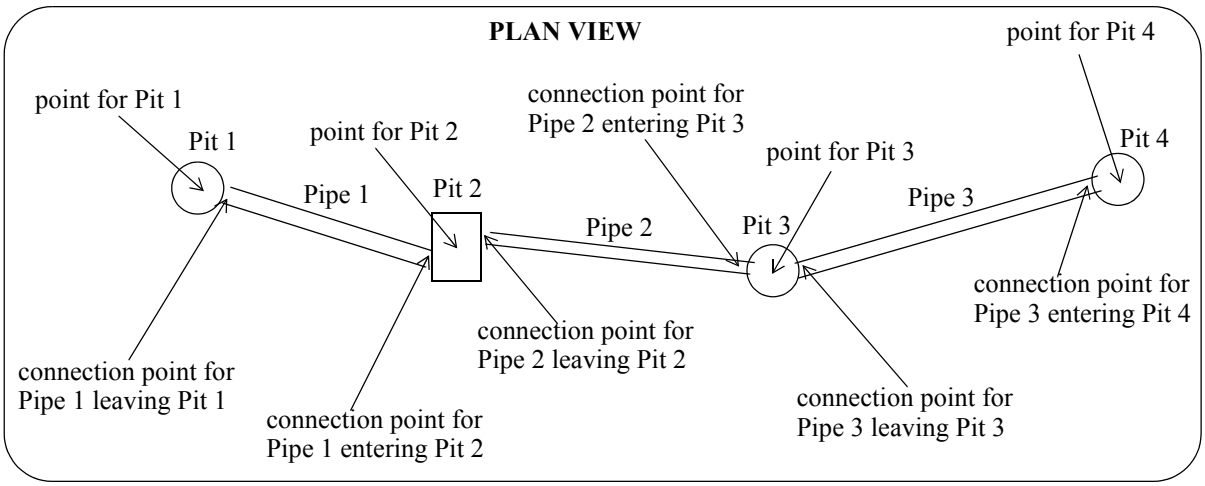
Although a pipe must go between two pits, the ends of the pipe do not have to be on the centre of the pit at each end but stop at what are called connection points.

If connection points are being used, then there will be

(a) **one** connection point for the first pit (for the pipe leaving the first pit) and the underlying string will have a vertex for the pit and one for the connection point

(b) **one** connection point for the last pit (for the pipe entering the last pit) and the underlying string will have a vertex for the pit and one for the connection point

(c) **two** connection points for each pit between the end pits (one for the pipe entering/leaving from the left of the pit and the other for the pipe entering/leaving from the right of the pit) and the underlying string will have a vertex for the pit and one for each of the two connection points

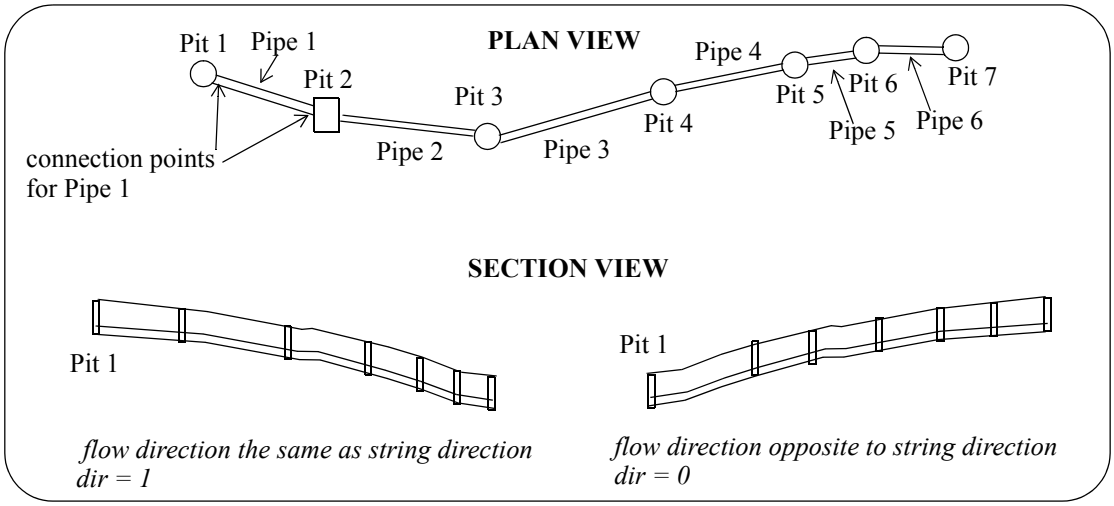


Drainage Definitions - Flow Direction

A drainage string has a ***flow direction*** which is either in the same as the direction of the drainage string (dir = 1), or is in the opposite direction to the direction of the drainage string (dir = 0). The direction of a string is the chainage direction of the string.

Storm water strings are usually designed with the flow direction the same as the drainage string direction and so when profiled in a section view, most of the pipes slope down to the right.

Water water strings are usually designed with the flow direction the opposite to the drainage string direction and so when profiled in a section view, most of the pipes slope up to the right.



Drainage Definitions - Drainage Network, Junction, Trunk

In 12d Model, a **drainage network** is defined to be all the drainage strings in the ***same*** model. So all the drainage strings in the same model are considered to be part of the same drainage network. If you have two different drainage networks, then they must be in different models. In particular, all the drainage strings of type storm water need to be in a different model to those of type waste water.

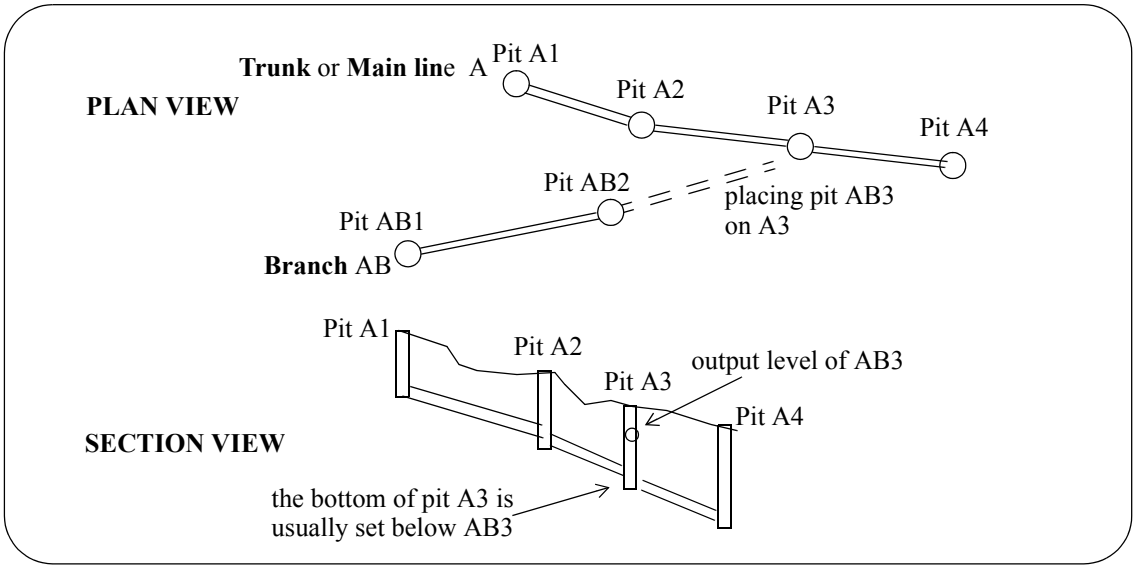
To model one drainage string AB connecting into another drainage string A in the same network (model), the pit at the end of string AB must have exactly the same (x,y) location as the pit on the drainage string A where the connection occurs. This situation represents a **junction** and the pit at the end of AB is called the **junction** pit and the pit in A, is the controlling pit, and is either a **trunk** or a **diverging** pit.

The two pits are then considered to be the same pit and all the information for the pit resides on the controlling pit.

A branch is defined as a drainage string that flows into a non-outlet pit of another drainage string. Thus the flow direction of the drainage string is important.

The drainage string that the water flows into from a branch drainage string is referred to as a **trunk** line (for that branch string).

A trunk line may also be a branch for another downstream trunk line.



For more information on drainage strings, see the ***12d Model Reference*** manual.

The following functions are used to create new drainage strings and make inquiries and modifications to existing drainage strings.

See Underlying Drainage String Functions
See General Drainage String Functions
See Drainage String Pits
See Drainage Pit Type Information in the drainage.4d File
See Drainage String Pit Attributes
See Drainage String Pipes
See Drainage Pipe Type Information in the drainage.4d File
See Drainage String Pipe Attributes
See Drainage String House Connections - For Sewer Module Only

Underlying Drainage String Functions

A **drainage** string consists of two parts:

(a) a string of vertices with straight or arc segments that defines the underlying geometry of the drainage string

(b) information about pits (maintenance holes or manholes) and pipes.

See Drainage Definitions

The following functions are for defining the underlying geometry of the drainage string.

Drainage pit information starts in the section Drainage String Pits and drainage pipe information starts in the section Drainage String Pipes .

Create_drainage(Integer num_verts,Integer num_pits)

Name

Element Create_drainage(Integer num_verts,Integer num_pits)

Description

Create an Element of type Drainage with room for **num_verts** vertices in the underlying string, and room for **num_pits** pits.

The actual data of the drainage string is set after the string is created.

If the drainage string could not be created, then the returned Element will be null.

ID = 490

Create_drainage(Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_verts, Integer num_pits)

Name

Element Create_drainage(Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_verts, Integer num_pits)

Description

Create an Element of type drainage.

The Element has **num_verts** vertices with (x,y,z) values for the vertices given in the Real arrays **x[], y[] and z[]**, and the radii of the arcs for the segments between the vertices given by the Real radius array **r[]** and the Integer bulge array **b[]** (Bulge arrayb=1 for major arc >180 degrees, b = 1 for minor arc < 180 degrees).

The drainage string also contains Integer **num_pits** pits.

The function return value gives the actual Element created.

If the drainage string could not be created, then the returned Element will be null.

ID = 489

Set_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_verts)

Name

Integer Set_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_verts)

Description

Set the (x,y,z,r,b) data for the first **num_verts** vertices of the drainage Element **drain**.

This function allows the user to modify a large number of vertices of the string in one call.

The maximum number of vertices that can be set is given by the number of vertices in the string.

The (x,y,z,r,b) values for each string vertex are given in the Real arrays **x[], y[], z[], r[]** and **b[]**.

The number of vertices to be set is given by Integer **num_verts**

If the Element **drain** is not of type Drainage, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Note

This function can not create new Drainage Elements but only modify existing Drainage Elements.

ID = 2100

Get_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer max_verts,Integer &num_verts)

Name

Integer Get_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer max_verts,Integer &num_verts)

Description

Get the (x,y,z,r,b) data for the first **max_verts** points of the drainage Element drain.

The (x,y,z,r,b) values at each string vertex are returned in the Real arrays **x[], y[], z[], r[] and b[]**.

The maximum number of vertices that can be returned is given by **max_verts** (usually the size of the arrays). The vertex data returned starts at the first vertex and goes up to the minimum of **max_verts** and the number of vertices in the string.

The actual number of vertices returned is returned by Integer **num_verts**

num_verts <= max_verts

If the Element **drain** is not of type Drainage, then **num_pts** is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

ID = 2097

Set_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_verts,Integer start_vert)

Name

Integer Set_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_verts,Integer start_vert)

Description

For the drainage Element **drain**, set the (x,y,z,r,b) data for **num_verts** vertices, starting at vertex number **start_vert**.

This function allows the user to modify a large number of vertices of the string in one call starting at vertex number **start_vert** rather than vertex one.

The maximum number of vertices that can be set is given by the difference between the number of vertices in the string and the value of **start_vert**.

The (x,y,z,r,f) values for the string vertices are given in the Real arrays **x[], y[], z[], r[]** and **b[]**.

The number of the first string vertex to be modified is **start_vert**.

The total number of vertices to be set is given by Integer **num_verts**

If the Element **drain** is not of type Drainage, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Notes

(a) A **start_vert** of one gives the same result as the function Set_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer num_verts) .

(b) This function can not create new Drainage Elements but only modify existing Drainage Elements.

ID = 2101

Get_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer max_verts,Integer &num_verts,Integer start_vert)

Name

Integer Get_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer max_verts,Integer &num_verts,Integer start_vert)

Description

For a drainage Element **drain**, get the (x,y,z,r,b) data for **max_verts** points starting at vertex number **start_vert**.

This routine allows the user to return the data from a drainage string in user specified chunks. This is necessary if the number of vertices in the string is greater than the size of the arrays available to contain the information.

The maximum number of vertices that can be returned is given by **max_verts** (usually the size of the arrays). For this function, the vertex data returned starts at vertex number start_vert rather than vertex one.

The (x,y,z,r,b) values at each string vertex are returned in the Real arrays **x[], y[], z[], r[]** and **b[]**.

The actual number of vertices returned is given by Integer **num_verts**

num_verts <= max_verts

If the Element **drain** is not of type Drainage, then **num_verts** is set to zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Note

A **start_vert** of one gives the same result as for the function Get_drainage_data(Element drain,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer max_verts,Integer &num_verts) .

ID = 2098

Set_drainage_data(Element drain,Integer i,Real x,Real y,Real z,Real r,Integer b)

Name

Integer Set_drainage_data(Element drain,Integer i,Real x,Real y,Real z,Real r,Integer b)

Description

Set the (x,y,z,r,f) data for the ith vertex of the string.

The x value is given in Real **x**.

The y value is given in Real **y**.
The z value is given in Real **z**.
The radius value is given in Real **r**.
The minor/major value is given in Integer b. if **b** = 0, arc < 180 degrees; if **b** = 1, arc >180 degrees.

A function return value of zero indicates the data was successfully set.

ID = 2102

Get_drainage_data(Element drain,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &b)

Name

Integer Get_drainage_data(Element drain,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &b)

Description

Get the (x,y,z,r,f) data for the **i**th vertex of the Element **drain**.

The x value is returned in Real **x**.
The y value is returned in Real **y**.
The z value is returned in Real **z**.
The radius value is returned in Real **r**.
The minor/major value is returned in Integer **b**.
If minor/major is 0, arc < 180.
If minor/major is 1, arc > 180

A function return value of zero indicates the data was successfully returned.

ID = 2099

Go to the next section General Drainage String Functions or return to Drainage String Element .

General Drainage String Functions

Set_drainage_outfall_height(Element drain,Real ht)

Name

Integer Set_drainage_outfall_height(Element drain,Real ht)

Description

Set the outfall height of the drainage Element **drain** to the value **ht**.

A function return value of zero indicates the outfall height was successfully set.

ID = 491

Get_drainage_outfall_height(Element drain,Real &ht)

Name

Integer Get_drainage_outfall_height(Element drain,Real &ht)

Description

Get the outfall height of the drainage Element **drain** and return it as **ht**.

A function return value of zero indicates the outfall height was successfully returned.

ID = 492

Set_drainage_ns_tin(Element drain,Tin tin)

Name

Integer Set_drainage_ns_tin(Element drain,Tin tin)

Description

For the drainage string **drain**, set the natural surface Tin to be **tin**.

A function return value of zero indicates the tin was successfully set.

ID = 1275

Get_drainage_ns_tin(Element drain,Tin &tin)

Name

Integer Get_drainage_ns_tin(Element drain,Tin &tin)

Description

For the drainage string **drain**, get the natural surface Tin and return it in **tin**.

A function return value of zero indicates the tin was successfully returned.

ID = 1274

Set_drainage_fs_tin(Element drain,Tin tin)

Name

Integer Set_drainage_fs_tin(Element drain,Tin tin)

Description

For the drainage string **drain**, set the finished surface Tin to be **tin**.

A function return value of zero indicates the tin was successfully set.

ID = 1273

Get_drainage_fs_tin(Element drain,Tin &tin)

Name

Integer Get_drainage_fs_tin(Element drain,Tin &tin)

Description

For the drainage string **drain**, get the finished surface Tin and return it in **tin**.

A function return value of zero indicates the tin was successfully returned.

ID = 1272

Set_drainage_flow(Element drain,Integer dir)

Name

Integer Set_drainage_flow(Element drain,Integer dir)

Description

Set the flow direction of the drainage Element **drain**

The flow direction is given as Integer **dir**.

dir = 1 means the flow direction is the same as the string direction. That is, the flow direction is the same as the chainage direction of the drainage string.

dir = 0 means the flow direction is opposite to the string direction. That is, the flow direction is the opposite direction to the chainage direction of the drainage string.

See Drainage Definitions .

A function return value of zero indicates the flow direction was successfully set.

ID = 539

Get_drainage_flow(Element drain,Integer &dir)

Name

Integer Get_drainage_flow(Element drain,Integer &dir)

Description

Get the flow direction of the drainage Element **drain** and return the flow direction **dir**.

dir = 1 means the flow direction is the same as the string direction. That is, the flow direction is the same as the chainage direction of the drainage string.

dir = 0 means the flow direction is opposite to the string direction. That is, the flow direction is the opposite direction to the chainage direction of the drainage string.

See Drainage Definitions .

A function return value of zero indicates the flow direction was successfully returned.

ID = 540

Set_drainage_float(Element drain,Integer string_pit_float)

Name

Integer Set_drainage_float(Element drain,Integer string_pit_float)

Description

For the Element **drain,** which must be of type *Drainage*, set the value of the flag for the string floating pit to **string_pit_float**.

Note: If a pit does not have a pit_float value set for the pit, then the pit uses the **string_pit_float** value.

A pit can be given its own pit_float value using the call Set_drainage_pit_float(Element drain,Integer pit,Integer pit_float) .

If **string_pit_float** = 1, the top of a pit automatically takes its level (height) from the finished surface tin for the drainage string **drain**.

If **string_pit_float** = 0, the top of the pit level is fixed.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the string_pit_float was successfully set.

ID = 1271

Get_drainage_float(Element drain,Integer &string_pit_float)

Name

Integer Get_drainage_float(Element drain,Integer &string_pit_float)

Description

For the Element **drain,** which must be of type *Drainage*, return the value of the flag for the string floating pit in **string_pit_float**.

Note: If a pit does not have a *pit_float* value set for the pit, then the pit uses the **string_pit_float** value.

A pit can be given its own pit_float value using the call Set_drainage_pit_float(Element drain,Integer pit,Integer pit_float) .

If **string_pit_float** = 1, the top of a pit automatically takes its level (height) from the finished surface tin for the drainage string **drain**.

If **string_pit_float** = 0, the top of the pit level is fixed.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the string_pit_float value was successfully returned.

ID = 1270

Get_drainage_trunk(Element drain,Element &trunk)

Name

Integer Get_drainage_trunk(Element drain,Element &trunk)

Description

For the drainage string **drain**, determine if **drain** flows into a trunk string.

If there **is a trunk** string then it is returned as **trunk** and the function return value is **0**. If a trunk exists, then **drain** is a branch string.

If there is **no trunk** string and the downstream end of **drain** is an **outlet** then the function return value is **44**.

For all other cases, the function return value is non zero but not 44.

See Drainage Definitions .

ID = 1444

Drainage_default_grading_to_end(Element drain,Integer pipe_num)

Name

Integer Drainage_default_grading_to_end(Element drain,Integer pipe_num)

Description

For the Element **drain**, which must be of type *Drainage*, grade from pipe number **pipe_num** to the end of the string using the minimum grade, cover etc for the **drain**.

The drainage flow direction is essential to the grading algorithm.

A function return value of zero indicates the string was successfully graded.

ID = 1700

Drainage_grade_to_end(Element drain,Integer pipe_num,Real slope)

Name

Integer Drainage_grade_to_end(Element drain,Integer pipe_num,Real slope)

Description

For the Element **drain**, which must be of type *Drainage*, grade from pipe number **pipe_num** to the end of the string using the slope **slope** where the units for slope are 1:in. That is, 1 vertical :in **slope** horizontal

The drainage flow direction is essential to the grading algorithm.

A function return value of zero indicates the string was successfully graded.

ID = 1701

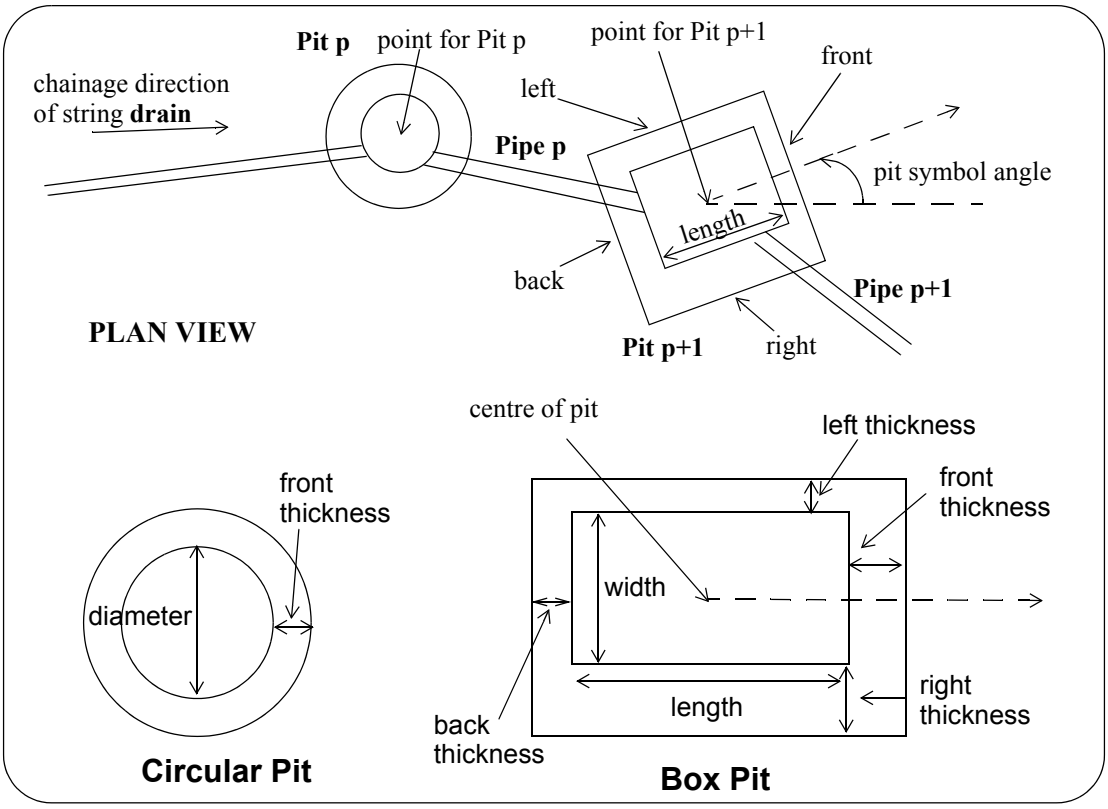
Go to the next section Drainage String Pits or return to Drainage String Element .

Drainage String Pits

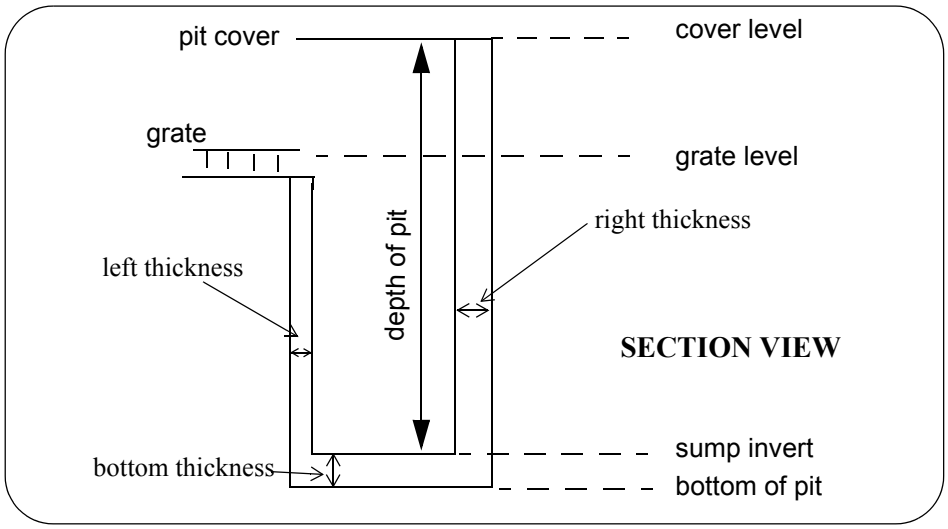
Drainage Pit Definitions

For a circle drainage pit, the point for the pit is the centre of the circle of the pit.

For a rectangular drainage pit, the point for the pit is the centre of the *internal* walls of the pit and the rotation of the pit is defined by the ***pit symbol angle*** which is measured in the counter clockwise direction from the x-axis. The pit *length* is defined in the direction of the pit symbol angle and pit *width* is in the direction perpendicular to the *length*. The *front*, *back*, *left* and *right* are all defined in relation to line going through the centre of the pit and with the pit symbol angle.



Drainage Pit Cross Section



Get_drainage_pits(Element drain,Integer &npits)

Name

Integer Get_drainage_pits(Element drain,Integer &npits)

Description

For the Element drain, which must of type *Drainage*, get the number of pits for the string and return it in **npits**. The number of pipes in **npits** - 1.

The **i'**th pipe goes from the i'th pit to the (i+1)'th pit.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 530

Set_drainage_pit(Element drain,Integer p,Real x,Real y,Real z)

Name

Integer Set_drainage_pit(Element drain,Integer p,Real x,Real y,Real z)

Description

Set the x,y & z for the **p**th pit of the string Element **drain**.

The x coordinate of the pit is given as Real **x**.

The y coordinate of the pit is given as Real **y**.

The z coordinate of the pit is given as Real **z**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 532

Get_drainage_pit(Element drain,Integer p,Real &x,Real &y,Real &z)

Name

Integer Get_drainage_pit(Element drain,Integer p,Real &x,Real &y,Real &z)

Description

Get the x,y & z for the **p**th pit of the string Element **drain**.

The x coordinate of the pit is returned in Real **x**.

The y coordinate of the pit is returned in Real **y**.

The z coordinate of the pit is returned in Real **z** (the cover level).

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 531

Set_drainage_pit_name(Element drain,Integer p,Text name)

Name

Integer Set_drainage_pit_name(Element drain,Integer p,Text name)

Description

For the Element **drain,** which must be of type *Drainage*, set the name for the **p**th pit to **name**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 513

Get_drainage_pit_name(Element drain,Integer p,Text &name)

Name

Integer Get_drainage_pit_name(Element drain,Integer p,Text &name)

Description

For the Element **drain,** which must be of type *Drainage*, get the name for the **p**th pit and return it in **name**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 507

Set_drainage_pit_colour(Element drain,Integer p,Integer colour)

Name

Integer Set_drainage_pit_colour(Element drain,Integer pit,Integer colour)

Description

For the Element **drain**, which must of type *Drainage*, set the colour of the **p**th pit to colour number **colour**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2781

Get_drainage_pit_colour(Element drain,Integer p,Integer &colour)

Name

Integer Get_drainage_pit_colour(Element drain,Integer p,Integer &colour)

Description

For the Element **drain**, which must of type *Drainage*, return the colour number of the **p**th pit in **colour**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2780

Set_drainage_pit_diameter(Element drain,Integer p,Real diameter)

Name

Integer Set_drainage_pit_diameter(Element drain,Integer p,Real diameter)

Description

For the Element **drain**, which must of type *Drainage*, set the diameter for the **p**th pit to **diameter**.

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 511

Get_drainage_pit_diameter(Element drain,Integer p,Real &diameter)

Name

Integer Get_drainage_pit_diameter(Element drain,Integer p,Real &diameter)

Description

For the Element **drain**, which must of type *Drainage*, return the diameter of the **p**th pit in **diameter**.

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 505

Set_drainage_pit_symbol_angle(Element drain,Integer p,Real angle)

Name

Integer Set_drainage_pit_symbol_angle(Element drain,Integer p,Real angle)

Description

For the Element **drain**, which must of type *Drainage*, set the angle for the **p**th pit to **angle**. **angle** is used for both the physical pit, and a symbol used for the pit in a *Drainage Plan Plot*.

angle is in radians and measured in the counter clockwise direction from the x-axis.

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2872

Get_drainage_pit_symbol_angle(Element drain,Integer pit,Real &angle)

Name

Integer Get_drainage_pit_symbol_angle(Element drain,Integer pit,Real &angle)

Description

For the Element **drain**, which must of type *Drainage*, return the angle of the **p**th pit in **angle**. **angle** is used for both the physical pit, and a symbol used for the pit in a *Drainage Plan Plot.*

angle is in radians and measured in the counter clockwise direction from the x-axis.

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 2871

Set_drainage_pit_width(Element drain,Integer p,Real width)

Name

Integer Set_drainage_pit_width(Element drain,Integer p,Real width)

Description

For the Element **drain**, which must of type *Drainage*, set the width for the **p**th pit to **width**.

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2876

Get_drainage_pit_width(Element drain,Integer p,Real &width)

Name

Integer Get_drainage_pit_width(Element drain,Integer p,Real &width)

Description

For the Element **drain**, which must of type *Drainage*, return the width of the **p**th pit in **width**.

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 2877

Set_drainage_pit_length(Element drain,Integer p,Real length)

Name

Integer Set_drainage_pit_length(Element drain,Integer p,Real length)

Description

For the Element **drain**, which must of type *Drainage*, set the length for the **p**th pit to **length**.

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2878

Get_drainage_pit_length(Element drain,Integer p,Real &length)

Name

Integer Get_drainage_pit_length(Element drain,Integer p,Real &length)

Description

For the Element **drain**, which must of type *Drainage*, return the length of the **p**th pit in **length**.

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 2879

Set_drainage_pit_float_sump(Element drain,Integer pit,Integer sump_float)

Name

Integer Set_drainage_pit_float_sump(Element drain,Integer pit,Integer sump_float)

Description

For the Element **drain,** which must be of type *Drainage*, and pit number **pit**, set the flag for the floating sump invert level to **sump_float**.

If **sump_float** = 1, the invert level of the sump automatically moves to be the invert level of the lowest pipe coming into the pit, plus the sump offset (which is defined by an attribute).

If **sump_float** = 0, the invert level of the sump is fixed and is explicitly set by the call Set_drainage_pit_sump_level(Element drain,Integer pit,Real level) .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the floating sump level flag was successfully set.

ID = 2786

Get_drainage_pit_float_sump(Element element,Integer pit,Integer &sump_float)

Name

Integer Get_drainage_pit_float_sump(Element element,Integer pit,Integer &sump_float)

Description

For the Element **drain,** which must be of type *Drainage*, and pit number **pit**, return the flag for the floating sump invert level as **sump_float**.

If **sump_float** = 1, the invert level of the sump automatically moves to be the invert level of the lowest pipe coming into the pit, plus the sump offset (which is defined by an attribute).

If **sump_float** = 0, the invert level of the sump is fixed and is explicitly set by the call Set_drainage_pit_sump_level(Element drain,Integer pit,Real level) .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the floating sump level flag was successfully returned.

ID = 2787

Set_drainage_pit_sump_level(Element drain,Integer pit,Real level)

Name

Integer Set_drainage_pit_sump_level(Element drain,Integer pit,Real level)

Description

For the Element **drain,** which must be of type *Drainage*, and pit number **pit**, set the pit sump invert level to **level**.

This value is only used when the pit floating sump level flag is set to 1. See Set_drainage_pit_float_sump(Element drain,Integer pit,Integer sump_float) .

See Drainage Pit Cross Section .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the sump invert level was successfully set.

ID = 2788

Get_drainage_pit_sump_level(Element drain,Integer pit,Real &level)

Name

Integer Get_drainage_pit_sump_level(Element drain,Integer pit,Real &level)

Description

invert of the sump

For the Element **drain,** which must be of type Drainage, and pit number **pit**, return the invert level of the sump as **level**.

See Drainage Pit Cross Section .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the sump invert level was successfully returned.

ID = 2789

Set_drainage_pit_thickness(Element drain,Integer p,Real bottom,Real front,Real back,Real left,Real right)

Name

Integer Set_drainage_pit_thickness(Element drain,Integer p,Real bottom,Real front,Real back,Real left,Real right)

Description

For the Element **drain**, which must of type *Drainage*, set the thicknesses for the **p**th pit to **bottom, front back, left and right** where

bottom is the thickness of the bottom of the pit
front is the thickness for a round pit and the front thickness for a rectangular pit
back is the back thickness for a rectangular pit and not used for a round pit
left is the left thickness for a rectangular pit and not used for a round pit
right is the right thickness for a rectangular pit and not used for a round pit

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2870

Get_drainage_pit_thickness(Element drain,Integer p,Real &bottom,Real &front,Real &back,Real &left,Real &right)

Name

Integer Get_drainage_pit_thickness(Element drain,Integer p,Real &bottom,Real &front,Real &back,Real &left,Real &right)

Description

For the Element **drain**, which must of type *Drainage*, get the thicknesses for the **p**th pit and return them in **bottom, front back, left and right** where

bottom is the thickness of the bottom of the pit
front is the thickness for a round pit, and the front thickness for a rectangular pit
back is the back thickness for a rectangular pit and not used for a round pit
left is the left thickness for a rectangular pit and not used for a round pit
right is the right thickness for a rectangular pit and not used for a round pit

See Drainage Pit Definitions .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the thicknesses was successfully returned.

ID = 2869

Set_drainage_use_connection_points(Element drain,Integer use_connection_points)

Name

Integer Set_drainage_use_connection_points(Element drain,Integer use_connection_points)

Description

For the Element **drain,** which must be of type *Drainage*, set whether pit connection points are used or not.

If **use_connection_points** = 0, pit connection points are not used.
If **use_connection_points** = 1, pit connection points are used.

If connection points are to be used and there are no custom connection points defined for the pit in the *drainage.4d* file, then every pipe goes to the centre of the closest rectangular side, or onto the circle for circular pits.

If connection points are to be used and there are custom connection points defined for the pit in the drainage.4d file, then the pipes go to the closest connection point.

See Drainage Definitions for connection points.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the **use_connection_points** flag was successfully set.

ID = 2790

Get_drainage_use_connection_points(Element drain,Integer &use_connection_points)

Name

Integer Get_drainage_use_connection_points(Element drain,Integer &use_connection_points)

Description

For the Element **drain,** return the pit connection point mode for the string in **use_connection_points**.

If **use_connection_points** = 0, pit connection points are not used for **drain.**
If **use_connection_points** = 1, pit connection points are used for **drain**.

See Drainage Definitions for connection points.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the **use_connection_points** flag was successfully returned.

ID = 2791

Drainage_Adjust_Pit_Connection_Points(Element drain,Integer pit)

Name

Integer Drainage_Adjust_Pit_Connection_Points(Element drain,Integer pit)

Description

For the Element **drain,** which must be of type *Drainage*, recalculate the pit connection points for pit number **pit**.

Note that this needs to be done if the pit was moved or changed. For example, changing the diameter of the pit.

See Drainage Definitions for connection points.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the connection points were successfully adjusted.

ID = 2792

Drainage_Adjust_Pit_Connection_Points_All(Element drain)

Name

Integer Drainage_Adjust_Pit_Connection_Points_All(Element drain)

Description

For the Element **drain,** which must be of type Drainage, recalculate the pit connection points for all the pits in **drain**.

Note that this needs to be done if pits were moved or changed. For example, changing the diameter of the pits.

See Drainage Definitions for connection points.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the connection points were successfully adjusted.

ID = 2793

Get_drainage_pit_connection_points(Element drain,Integer pit,Real &lx,Real &ly,Real &rx,Real &ry)

Name

Integer Get_drainage_pit_connection_points(Element drain,Integer pit,Real &lx,Real &ly,Real &rx,Real &ry)

Description

For the Element **drain,** which must be of type *Drainage*, return the pit connection points for pit number **pit**.

The coordinates of the pit connection point for the pipe that comes into the pit from the left are returned as (**lx**,**ly**).

The coordinates of the pit connection point for the pipe that goes out of the pit to the right are returned as (**rx**,**ry**).

See Drainage Definitions for connection points.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the connection points were successfully returned.

ID = 2847

Set_drainage_pit_inverts(Element drain,Integer p,Real lhs,Real rhs)

Name

Integer Set_drainage_pit_inverts(Element drain,Integer p,Real lhs,Real rhs)

Description

For the Element **drain,** which must be of type *Drainage*, set the invert levels of the pipes of **drain** entering/leaving the **p**th pit.

The invert level of the *pipe* entering/leaving the *left side* of the pit is set to Real **lhs**.

The invert level of the *pipe* entering/leaving the *right side* of the pit is set to Real **rhs**.

See Drainage Pipe Definitions for invert levels.

Note: this is setting the invert levels of the *pipes* entering/leaving the **p**th pit.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 514

Get_drainage_pit_inverts(Element drain,Integer p,Real &lhs,Real &rhs)

Name

Integer Get_drainage_pit_inverts(Element drain,Integer p,Real &lhs,Real &rhs)

Description

For the Element **drain,** which must be of type *Drainage*, get the invert levels of the pipes of **drain** entering/leaving the **p**th pit.

The invert level of the pipe entering/leaving the *left side* of the pit is returned in **lhs**.

The invert level of the pipe entering/leaving the *right side* of the pit is returned in **rhs**.

See Drainage Pipe Definitions for invert levels.

Note: this is getting the invert levels of the *pipes* entering/leaving the **p**th pit.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 508

Get_drainage_pit_angle(Element drain,Integer p,Real &ang)

Name

Integer Get_drainage_pit_angle(Element drain,Integer p,Real &ang)

Description

For the Element **drain**, which must of type *Drainage*, get the *angle* between pipes of **drain** entering and leaving the **p**th pit, and return the angle as **ang**.

Note: this is not the angle of the drainage pit itself which is returned by the call Get_drainage_pit_symbol_angle(Element drain,Integer pit,Real &angle) .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 517

Get_drainage_pit_angle (Element drain,Integer p,Real &ang,Integer trunk)

Name

Integer Get_drainage_pit_angle(Element drain,Integer p,Real &ang,Integer trunk)

Description

For the Element **drain**, which must of type *Drainage*, for the **p**th pit, get the *angle* between incoming pipe and the outgoing pipe, and return it as **ang**. **ang** is in radians.

If the drainage string is using connection points, the direction of the pipes at the connection points are used.

If the drainage string is NOT using connection points, the direction of the pipes at the pit centre are used.

trunk controls the action to be taken when *the pit is at the downstream end of the drainage string*.

If **trunk** in non-zero, then a trunk line will be searched for to obtain the outgoing pipe. If no trunk line is found, **ang** = 0.

If **trunk** is zero, **ang** = 0.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 1294

Get_drainage_pit_chainage(Element drain,Integer p,Real &chainage)

Name

Integer Get_drainage_pit_chainage(Element drain,Integer p,Real &chainage)

Description

For the Element **drain,** which must be of type *Drainage,* return the chainage for the **p**th pit in **chainage**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 520

Get_drainage_pit_chainages(Element drain,Integer pit,Real &ch_lcp,Real &ch_centre,Real &ch_rcp)

Name

Integer Get_drainage_pit_chainages(Element drain,Integer pit,Real &ch_lcp,Real &ch_centre,Real &ch_rcp)

Description

For the Element **drain,** which must be of type *Drainage*, and for pit number **pit,** return the chainages of the pit connection points and the chainage of the *centre* of the pit.

The chainage of the pit connection point for the pipe that comes into the pit from the left is returned as **ch_lcp**.

The chainage of the pit connection point for the pipe that goes out of the pit to the right is returned as **ch_rcp**.

The chainage of the centre of the pit is returned as **ch_centre**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the chainages were successfully returned.

ID = 2848

Get_drainage_pit_shape(Element drain,Integer pit,Integer mode,Element

&super_inside,Element &super_outside)

Name

Integer Get_drainage_pit_shape(Element drain,Integer pit,Integer mode,Element &super_inside,Element &super_outside)

Description

For the Element **drain,** which must be of type *Drainage*, return the *plan* shape of the inside of pit number **pit** as the super string **super_inside** and the *plan* shape of the outside of the pit as **super_outside**.

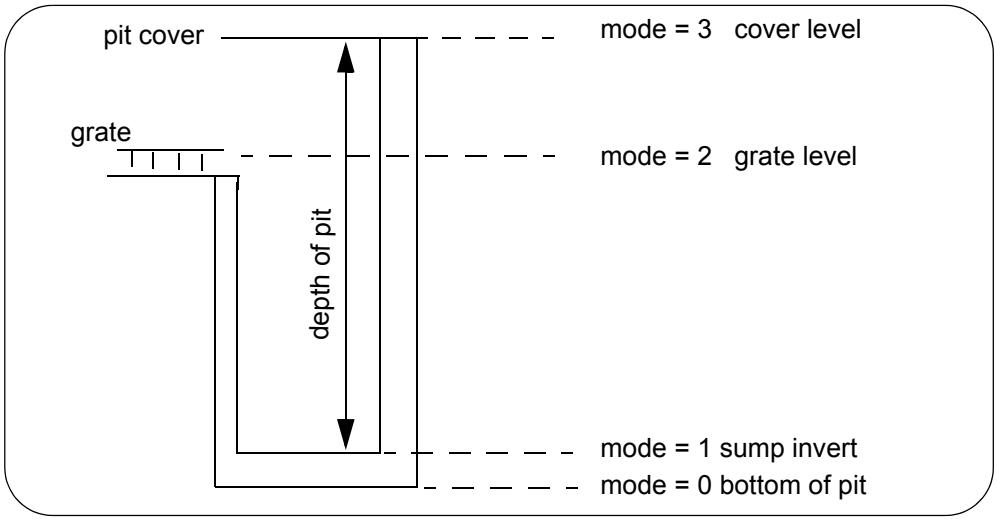
So for a circular pit with a wall thickness, a super string representing a circle of the diameter of the pit is the *super_inside* and a circle of (diameter + 2*thickness) is the *super_outside*.

If **mode** = 0, the shapes are given the z-value of the bottom of the pit (sump bottom).

If **mode** = 1, the shapes are given the z-value of the invert of the sump.

If **mode** = 2, the shapes are given the z-value of the grate.

If **mode** = 3, the shapes are given the z-value of the cover.



If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the shapes were successfully returned.

ID = 2849

Set_drainage_pit_float(Element drain,Integer pit,Integer pit_float)

Name

Integer Set_drainage_pit_float(Element drain,Integer pit,Integer pit_float)

Description

For the Element **drain,** which must be of type *Drainage*, and pit number **pit**, set the flag for the floating pit level to **pit_float**.

If **pit_float** = 1, the top of the pit automatically takes its level (height) from the finished surface tin for the drainage string **drain**.

If **pit_float** = 0, the top of the pit level is fixed.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the **pit_floa**t value was successfully set.

ID = 1277

Get_drainage_pit_float(Element drain,Integer pit,Integer &pit_float)

Name

Integer Get_drainage_pit_float(Element drain,Integer pit,Integer &pit_float)

Description

For the Element **drain,** which must be of type *Drainage*, and pit number **pit**, return the flag for the floating pit level as **pit_float**.

If **pit_float** = 1, the top of the pit automatically takes its level (height) from the finished surface tin for the drainage string **drain**.

If **pit_float** = 0, the top of the pit level is fixed.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the **pit_float** value was successfully returned.

ID = 1276

Set_drainage_pit_hgl(Element drain,Integer p,Real hgl)

Name

Integer Set_drainage_pit_hgl(Element drain,Integer p,Real hgl)

Description

For the Element **drain,** which must be of type *Drainage*, set the hgl level for the centre of the **p**th pit of the string to **hgl**.

If **hgl** is null then the hgl for the surface is not drawn.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 1241

Get_drainage_pit_hgl(Element drain,Integer p,Real &hgl)

Name

Integer Get_drainage_pit_hgl(Element drain,Integer p,Real &hgl)

Description

For the Element **drain,** which must be of type *Drainage*, get the hgl level for centre of the **p**th pit and return it in **hgl**.

If **hgl** is null then the hgl for the surface is not drawn.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 1242

Set_drainage_pit_surface_hgl(Element element,Integer pit,Real surface_hgl)

Name

Integer Set_drainage_pit_surface_hgl(Element element,Integer pit,Real surface_hgl)

Description

For the Element **drain,** which must be of type *Drainage*, set the surface hgl level for the centre of

the **p**th pit of the string, to **surface_hgl**.

If **surface_hgl** is null then the hgl for the surface is not drawn.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2785

Get_drainage_pit_surface_hgl(Element element,Integer pit,Real &surface_hgl

Name

Integer Get_drainage_pit_surface_hgl(Element element,Integer pit,Real &surface_hgl)

Description

For the Element **drain,** which must be of type *Drainage*, get the surface hgl level for the centre of the **p**th pit of the string, and return it in **surface_hgl**.

If **surface_hgl** is null then the hgl for the surface is not drawn.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2784

Set_drainage_pit_hgls(Element drain,Integer p,Real lhs,Real rhs)

Name

Integer Set_drainage_pit_hgls(Element drain,Integer p,Real lhs,Real rhs)

Description

For the Element **drain,** which must be of type *Drainage*, set the hgl levels of the pipes of **drain** entering/leaving the **p**th pit.

The hgl level of the pipe entering/leaving the left side of the pit is given as Real **lhs**.

The hgl level of the entering/leaving right side of the pit is given as Real **rhs**.

Note: this is setting the hgl levels for the *pipes* entering/leaving the **p**th pit, **not** the hgl of the pit.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 538

Get_drainage_pit_hgls(Element drain,Integer p,Real &lhs,Real &rhs)

Name

Integer Get_drainage_pit_hgls(Element drain,Integer p,Real &lhs,Real &rhs)

Description

For the Element **drain,** which must be of type *Drainage*, get the hgl levels of the pipes of **drain** entering/leaving the **p**th pit.

The hgl level of the pipe entering/leaving the left side of the pit is returned in Real **lhs**.

The hgl level of the pipe entering/leaving the right side of the pit is returned in Real **rhs**.

Note: this is getting the hgl levels of the *pipes* entering/leaving the **p**th pit, **not** the hgl of the pit.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 535

Set_drainage_pit_road_chainage(Element drain,Integer p,Real chainage)

Name

Integer Set_drainage_pit_road_chainage(Element drain,Integer p,Real chainage)

Description

For the Element **drain,** which must be of type *Drainage*, set the road chainage for the **p**th pit to **chainage**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 515

Get_drainage_pit_road_chainage(Element drain,Integer p,Real &chainage)

Name

Integer Get_drainage_pit_road_chainage(Element drain,Integer p,Real &chainage)

Description

For the Element **drain,** which must be of type *Drainage,* return the road chainage for the **p**th pit in **chainage**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 509

Set_drainage_pit_road_name(Element drain,Integer p,Text name)

Name

Integer Set_drainage_pit_road_name(Element drain,Integer p,Text name)

Description

For the Element **drain,** which must be of type *Drainage,* set the road name for the **p**th pit to **name**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 516

Get_drainage_pit_road_name(Element drain,Integer p,Text &name)

Name

Integer Get_drainage_pit_road_name(Element drain,Integer p,Text &name)

Description

For the Element **drain,** which must be of type *Drainage,* return the road name for the **p**th pit in **name**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 510

Set_drainage_pit_type(Element drain,Integer p,Text type)

Name

Integer Set_drainage_pit_type(Element drain,Integer p,Text type)

Description

For the Element **drain,** which must be of type *Drainage,* set the type for the **p**th pit to **type**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 512

Get_drainage_pit_type(Element drain,Integer p,Text &type)

Name

Integer Get_drainage_pit_type(Element drain,Integer p,Text &type)

Description

For the Element **drain,** which must be of type *Drainage,* return the type for the **p**th pit in **type**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 506

Get_drainage_pit_branches(Element drain,Integer p,Dynamic_Element &branches)

Name

Integer Get_drainage_pit_branches(Element drain,Integer p,Dynamic_Element &branches)

Description

For the Element **drain,** which must be of type *Drainage,* this function returns a list of the branches (each branch is a Drainage string) that flow into the **p**th pit of **drain**. The list of branches is returned in the Dynamic_Element **branches**.

Note: a branch is defined as a drainage string that flows into a non-outlet pit of another drainage string. Thus the flow direction of the drainage string is important.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 1443

Get_drainage_pit_depth(Element drain,Integer p,Real &depth)

Name

Integer Get_drainage_pit_depth(Element drain,Integer p,Real &depth)

Description

For the Element **drain,** which must be of type *Drainage,* return the depth of the **p**th pit in **depth**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

See _for the def inion of pit depth.

A function return value of zero indicates the data was successfully returned.

ID = 519

Get_drainage_pit_drop(Element drain,Integer p,Real &drop)

Name

Integer Get_drainage_pit_drop(Element drain,Integer p,Real &drop)

Description

For the Element **drain,** which must be of type *Drainage,* return the drop through the **p**th pit in **drop**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 518

Get_drainage_pit_ns(Element drain,Integer n,Real &ns_ht)

Name

Integer Get_drainage_pit_ns(Element drain,Integer n,Real &ns_ht)

Description

For the Element **drain,** which must be of type *Drainage,* return the *height* from the natural surface tin at the location of the centre of the **n**th pit in **ns_ht**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 521

Get_drainage_pit_fs(Element drain,Integer n,Real &fs_ht)

Name

Integer Get_drainage_pit_fs(Element drain,Integer n,Real &fs_ht)

Description

For the Element **drain,** which must be of type *Drainage,* return the *height* from the finished surface tin at the location of the centre of the **n**th pit in **fs_ht**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 522

Go to the next section Drainage Pit Type Information in the drainage.4d File or return to Drainage String Element .

Drainage Pit Type Information in the drainage.4d File

Get_drainage_number_of_manhole_types(Integer &num_types)

Name

Integer Get_drainage_number_of_manhole_types(Integer &num_types)

Description

Get the number of pit (manhole, maintenance hole) types from the *drainage.4d* file and return the number in **num_types**.

A function return value of zero indicates the data was successfully returned.

ID = 2077

Get_drainage_manhole_type(Integer i,Text &type)

Name

Integer Get_drainage_manhole_type(Integer i,Text &type)

Description

Get the name of the **i**'th manhole type from the *drainage.4d* file and return the name in ***type***.

A function return value of zero indicates the data was successfully returned.

ID = 2078

Get_drainage_manhole_length(Text type,Real &length)

Name

Integer Get_drainage_manhole_length(Text type,Real &length)

Description

For the manhole of type **type** from the *drainage.4d* file, return the *length* as given by the keyword "mhsize" in ***length*** (the *length* and *width* are given by the keyword "mhsize").

If there is no such manhole type, -1 is returned as the function return value.

If the length does not exist for the manhole type ***type***, -2 is returned as the function return value.

A function return value of zero indicates the data was successfully returned.

ID = 2079

Get_drainage_manhole_width(Text type,Real &width)

Name

Integer Get_drainage_manhole_width(Text type,Real &width)

Description

For the manhole of type **type** from the *drainage.4d* file, return the *width* as given by the keyword "mhsize" in ***width*** (the *length* and *width* are given by the keyword "mhsize").

If there is no such manhole type, -1 is returned as the function return value.

If the width does not exist for manhole type ***type***, -2 is returned as the function return value.

A function return value of zero indicates the data was successfully returned.

ID = 2080

Get_drainage_manhole_description(Text type,Text &description)

Name

Integer Get_drainage_manhole_description(Text type,Text &description)

Description

Get the *description* of the manhole of type **type** from the *drainage.4d* file and return the description in ***description***.

If there is no such manhole type, -1 is returned as the function return value.

If the description does not exist for manhole type ***type***, -2 is returned as the function return value.

A function return value of zero indicates the data was successfully returned.

ID = 2081

Get_drainage_manhole_notes(Text type,Text &notes)

Name

Integer Get_drainage_manhole_notes(Text type,Text &notes)

Description

Get the *notes* of the manhole of type **type** from the *drainage.4d* file and return the notes in ***notes***.

If there is no such manhole type, -1 is returned as the function return value.

If notes do not exist for manhole type ***type***, -2 is returned as the function return value.

A function return value of zero indicates the data was successfully returned.

ID = 2082

Get_drainage_manhole_group(Text type,Text &group)

Name

Integer Get_drainage_manhole_group(Text type,Text &group)

Description

Get the *group* of the manhole of type **type** from the *drainage.4d* file and return the group in ***group***.

If there is no such manhole type, -1 is returned as the function return value.

If group does not exist for manhole type ***type***, -2 is returned as the function return value.

A function return value of zero indicates the data was successfully returned.

ID = 2083

Get_drainage_manhole_capacities(Text type,Real &multi,Real &fixed, Real &percent,Real &coeff,Real &power)

Name

Integer Get_drainage_manhole_capacities(Text type,Real &multi,Real &fixed,Real &percent,Real &coeff,Real &power)

Description

From the *drainage.4d file*, for the manhole of type **type** return the values for the generic Inlet

capacities from the file for:

```
cap_multi       // if undefined the default is 1
cap_fixed       // if undefined the default is 0
cap_percent     // if undefined the default is 0
cap_coeff       // if undefined the default is 0
cap_power       // if undefined the default is 1
```

A function return value of zero indicates the data was successfully returned.

ID = 2084

Get_drainage_number_of_sag_curves(Text type,Integer &n)

Name

Integer Get_drainage_number_of_sag_curves(Text type,Integer &n)

Description

From the *drainage.4d* file, for the manhole of type **type**, get the number of sag capacity curves (cap_curve_sag) and return the number in ***n***.

A function return value of zero indicates the number was successfully returned.

ID = 2085

Get_drainage_sag_curve_name(Text type,Text &name)

Name

Integer Get_drainage_sag_curve_name(Text type,Text &name)

Description

From the *drainage.4d file*, for the manhole of type **type**, return the name of the sag capacity curve (cap_curve_sag) in ***name***.

A function return value of zero indicates the data was successfully returned.

ID = 2086

Get_drainage_manhole_capacities_sag(Text type,Real &multi,Real &fixed,Real &percent,Real &coeff,Real &power)

Name

Integer Get_drainage_manhole_capacities_sag(Text type,Real &multi,Real &fixed,Real &percent,Real &coeff,Real &power)

Description

From the *drainage.4d* file, for the manhole of type **type**, return the sag capacity curve (cap_curve_sag) values from the file for:

```
cap_multi       // if undefined the default is 1
cap_fixed       // if undefined the default is 0
cap_percent     // if undefined the default is 0
cap_coeff       // if undefined the default is 0
cap_power       // if undefined the default is 1
```

A function return value of zero indicates the data was successfully returned.

ID = 2087

Get_drainage_number_of_sag_curve_coords(Text type,Integer &n)

Name

Integer Get_drainage_number_of_sag_curve_coords(Text type,Integer &n)

Description

From the *drainage.4d* file, for the manhole of type **type**, return the number of coordinates in the sag capacity curve (cap_curve_sag) in **n**.

Note - **n** may be 0.

A function return value of zero indicates the number was successfully returned.

ID = 2088

Get_drainage_sag_curve_coords(Text type,Real Depth[],Real Qin[],Integer nmax,Integer &num)

Name

Integer Get_drainage_sag_curve_coords(Text type,Real Depth[],Real Qin[],Integer nmax,Integer &num)

Description

From the *drainage.4d* file, for the manhole of type **type**, return the coordinates for the sag capacity curve (cap_curve_sag) in **Depth[]** and **Qin[]**.

nmax is the size of the arrays **Depth**[] and **Qin**[], and **num** returns the actual number of coordinates.

A function return value of zero indicates the coordinates were successfully returned.

ID = 2089

Get_drainage_number_of_grade_curves(Text type,Integer &n)

Name

Integer Get_drainage_number_of_grade_curves(Text type,Integer &n)

Description

From the *drainage.4d* file, for the manhole of type **type**, get the number of grade curves (cap_curve_grade) and return the number in ***n***.

A function return value of zero indicates the number was successfully returned.

ID = 2090

Get_drainage_grade_curve_name(Text type,Integer i,Text &name)

Name

Integer Get_drainage_grade_curve_name(Text type,Integer i,Text &name)

Description

From the *drainage.4d* file, for the manhole of type **type**, return the name of the **i**'th grade curve (cap_curve_grade) in ***name***.

A function return value of zero indicates the name was successfully returned.

ID = 2091

Get_drainage_grade_curve_threshold(Text type,Text name,Integer

&by_grade,Real &road_grade,Integer &by_xfall,Real &road_xfall)

Name

Integer Get_drainage_grade_curve_threshold(Text type,Text name,Integer &by_grade,Real &road_grade,Integer &by_xfall,Real &road_xfall)

Description

From the *drainage.4d* file, for the manhole of type **type**, and the capacity on grade curve called **name**:

if the keyword "road_grade**"** exists then **by_grade** is set to 1 and the road on grade value is returned in **road_grade**. Otherwise **by_grade** is set to 0.

if the keyword "road_crossfall**"** exists then **by_crossfall** is set to 1 and the road crossfall value is returned in **road_xfall**. Otherwise **by_xfall** is set to 0.

A function return value of zero indicates the values were successfully returned.

ID = 2092

Get_drainage_manhole_capacities_grade(Text type,Text name,Real &multi,Real &fixed,Real &percent,Real &coeff,Real &power)

Name

Integer Get_drainage_manhole_capacities_grade(Text type,Text name,Real &multi,Real &fixed,Real &percent,Real &coeff,Real &power)

Description

From the drainage.4d file, for the manhole of type **type**, and the capacity on grade curve called **name**, return the sag capacity curve (cap_curve_grade) values from the file for:

```
cap_multi       // if undefined the default is 1
cap_fixed       // if undefined the default is 0
cap_percent     // if undefined the default is 0
cap_coeff       // if undefined the default is 0
cap_power       // if undefined the default is 1
```

A function return value of zero indicates the data was successfully returned.

ID = 2093

Get_drainage_number_of_grade_curve_coords(Text type,Text name,Integer &n)

Name

Integer Get_drainage_number_of_grade_curve_coords(Text type,Text name,Integer &n)

Description

From the drainage.4d file, for the manhole of type **type**, and the capacity on grade curve called **name**, return the number of coordinates in the on grade capacity curve (cap_curve_grade) in **n**.

Note - **n** may be 0.

A function return value of zero indicates the number was successfully returned.

ID = 2094

Get_drainage_grade_curve_coords(Text type,Text name,Real Qa[],Real Qin[],Integer nmax,Integer &n)

Name

Integer Get_drainage_grade_curve_coords(Text type,Text name,Real Qa[],Real Qin[],Integer nmax,Integer &n)

Description

From the drainage.4d file, for the manhole of type **type**, and the capacity on grade curve called **name,** return the coordinates for the on grade capacity curve (cap_curve_grade) in **Qa[]** and **Qin[]**.

nmax is the size of the arrays **Qa**[] and **Qin**[], and **num** returns the actual number of coordinates.

A function return value of zero indicates the coordinates were successfully returned.

ID = 2095

Get_drainage_manhole_config(Text type,Text &cap_config)

Name

Integer Get_drainage_manhole_config(Text type,Text &cap_config)

Description

From the drainage.4d file, for the manhole of type **type**, return the value of the keyword "cap_config" in ***cap_config***.

The value of cap_config must be:

"g" - for an on grade pit
"s" - for an sag pit
or
"m" - for a manhole sealed pit.

If the value of **cap_config** is not "g", "s" or "m" then a non zero function return value is returned.

A function return value of zero indicates the value was successfully returned.

ID = 2103

Get_drainage_manhole_diam(Text type,Real &diameter)

Name

Integer Get_drainage_manhole_diam(Text type,Real &diameter)

Description

From the drainage.4d file, for the manhole of type **type**, return the value of the keyword "mhdiam" in ***diameter***.

A function return value of zero indicates the value was successfully returned.

ID = 2104

Go to the next section Drainage String Pit Attributes or return to Drainage String Element .

Drainage String Pit Attributes

Get_drainage_pit_attribute_length(Element drain,Integer pit,Integer att_no,Integer &att_len)

Name

Integer Get_drainage_pit_attribute_length(Element drain,Integer pit,Integer att_no,Integer &att_len)

Description

For pit number **pit** of the Element **drain**, get the length (in bytes) of the attribute number **att_no**. The attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for attributes of type **Text** and **Binary.**

ID = 1005

Get_drainage_pit_attribute_length(Element drain,Integer pit,Text att_name,Integer &att_len)

Name

Integer Get_drainage_pit_attribute_length(Element drain,Integer pit,Text att_name,Integer &att_len)

Description

For pit number **pit** of the Element **drain**, get the length (in bytes) of the attribute with the name **att_name**. The attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for user attributes of type Text and Binary.

ID = 1004

Get_drainage_pit_attribute_type(Element drain,Integer pit,Integer att_no,Integer &att_type)

Name

Integer Get_drainage_pit_attribute_type(Element drain,Integer pit,Integer att_no,Integer &att_type)

Description

For pit number **pit** of the Element **drain**, get the type of the attribute with attribute number **att_no**. The attribute type is returned in **att_type**.

A function return value of zero indicates the attribute type was successfully returned.

ID = 1003

Get_drainage_pit_attribute_type(Element drain,Integer pit,Text att_name,Integer &att_type)

Name

Integer Get_drainage_pit_attribute_type(Element drain,Integer pit,Text att_name,Integer &att_type)

Description

For pit number **pit** of the Element **drain**, get the type of the attribute with name **att_name**. The attribute type is returned in **att_type**.

A function return value of zero indicates the attribute type was successfully returned.

ID = 1002

Get_drainage_pit_attribute_name(Element drain,Integer pit,Integer att_no,Text &name)

Name

Integer Get_drainage_pit_attribute_name(Element drain,Integer pit,Integer att_no,Text &name)

Description

For pit number **pit** of the Element **drain**, get the name of the attribute number **att_no**. The attribute name is returned in **name**.

A function return value of zero indicates the attribute name was successfully returned.

ID = 1001

Get_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Real &real)

Name

Integer Get_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Real &real)

Description

For the Element **drain**, get the attribute with number **att_no** for the pit number **pit** and return the attribute value in **real**. The attribute must be of type Real.

If the Element is not of type **Drainage** or the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute with attribute number att_no.

ID = 1000

Get_drainage_pit_attribute (Element drain,Integer pit,Integer att_no,Integer &int)

Name

Integer Get_drainage_pit_attribute (Element drain,Integer pit,Integer att_no,Integer &int)

Description

For the Element **drain**, get the attribute with number **att_no** for the pit number **pit** and return the attribute value in **int**. The attribute must be of type Integer.

If the Element is not of type **Drainage** or the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute with attribute number att_no.

ID = 999

Get_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Text &txt)

Name

Integer Get_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Text &txt)

Description

For the Element **drain**, get the attribute with number **att_no** for the pit number **pit** and return the attribute value in **txt**. The attribute must be of type Text.

If the Element is not of type **Drainage** or the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 998

Get_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Real &real)

Name

Integer Get_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Real &real)

Description

For the Element **drain**, get the attribute called **att_name** for the pit number **pit** and return the attribute value in **real**. The attribute must be of type Real.

If the Element is not of type **Drainage** or the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute called att_name.

ID = 997

Get_drainage_pit_number_of_attributes(Element drain,Integer pit,Integer &no_atts)

Name

Integer Get_drainage_pit_number_of_attributes(Element drain,Integer pit,Integer &no_atts)

Description

Get the total number of attributes for pit number **pit** of the Element **drain**.

The total number of attributes is returned in Integer **no_atts**.

A function return value of zero indicates the number of attributes was successfully returned.

ID = 994

Get_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Text &txt)

Name

Integer Get_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Text &txt)

Description

For the Element **drain**, get the attribute called **att_name** for the pit number **pit** and return the attribute value in **txt**. The attribute must be of type Text.

If the Element is not of type **Drainage** or the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute called att_name.

ID = 995

Get_drainage_pit_attribute (Element drain,Integer pit,Text att_name,Integer &int)

Name

Integer Get_drainage_pit_attribute (Element drain,Integer pit,Text att_name,Integer &int)

Description

For the Element **drain**, get the attribute called **att_name** for the pit number **pit** and return the attribute value in **int**. The attribute must be of type Integer.

If the Element is not of type **Drainage** or the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute called att_name.

ID = 996

Get_drainage_pit_attributes(Element drain,Integer pit,Attributes &att)

Name

Integer Get_drainage_pit_attributes(Element drain,Integer pit,Attributes &att)

Description

For the Element **drain**, return the Attributes for the pit number **pit** as **att**.

If the Element is not of type **Drainage** or the pit number **pit** has no attribute then a non-zero return value is returned.

A function return value of zero indicates the attribute is successfully returned.

ID = 2022

Set_drainage_pit_attributes(Element drain,Integer pit,Attributes att)

Name

Integer Set_drainage_pit_attributes(Element drain,Integer pit,Attributes att)

Description

For the Element **drain**, set the Attributes for the pit number **pit** to **att**.

If the Element is not of type **Drainage** then a non-zero return value is returned.

A function return value of zero indicates the attribute is successfully set.

ID = 2023

Get_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Uid &uid)

Name

Integer Get_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Uid &uid)

Description

For the Element **drain**, get the attribute called **att_name** for the pit number **pit** and return the attribute value in **uid**. The attribute must be of type Uid.

If the Element is not of type **Drainage** or the attribute is not of type Uid then a non-zero return

value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_name.

ID = 2024

Get_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Attributes &att)

Name

Integer Get_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Attributes &att)

Description

For the Element **drain**, get the attribute called **att_name** for the pit number **pit** and return the attribute value in **att**. The attribute must be of type Attributes.

If the Element is not of type **Drainage** or the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_name.

ID = 2025

Get_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Uid &uid)

Name

Integer Get_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Uid &uid)

Description

For the Element **drain**, get the attribute with number **att_no** for the pit number **pit** and return the attribute value in **uid**. The attribute must be of type Uid.

If the Element is not of type **Drainage** or the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 2026

Get_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Attributes &att)

Name

Integer Get_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Attributes &att)

Description

For the Element **drain**, get the attribute with number **att_no** for the pit number **pit** and return the attribute value in **att**. The attribute must be of type Attributes.

If the Element is not of type **Drainage** or the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number att_no.

ID = 2027

Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Uid uid)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Uid uid)

Description

For the Element **drain** and on the pit number **pit**,
if the attribute called **att_name** does not exist then create it as type Uid and give it the value **uid**.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **uid**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2028

Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Attributes att)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Attributes att)

Description

For the Element **drain** and on the pit number **pit**,
if the attribute called **att_name** does not exist then create it as type Attributes and give it the value **att**.
if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_name.

ID = 2029

Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Uid uid)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Uid uid)

Description

For the Element **drain** and on the pit number **pit**, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **uid**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 2030

Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Attributes att)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Attributes att)

Description

For the Element **drain** and on the pit number **pit**, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 2031

Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Real real)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Real real)

Description

For the Element **drain** and on the pit number **pit**,
if the attribute with number **att_no** does not exist then create it as type Real and give it the value **real**.
if the attribute with number **att_no** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 1011

Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Integer int)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Integer int)

Description

For the Element **drain** and on the pit number **pit**,
if the attribute with number **att_no** does not exist then create it as type Integer and give it the value **int**.
if the attribute with number **att_no** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 1010

Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Text txt)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Integer att_no,Text txt)

Description

For the Element **drain** and on the pit number **pit**,
if the attribute with number **att_no** does not exist then create it as type Text and give it the value **txt**.
if the attribute with number **att_no** does exist and it is type Text then set its value to **txt**.

If the attribute exists and is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 1009

Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Real real)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Real real)

Description

For the Element **drain** and on the pit number **pit**,
if the attribute called **att_name** does not exist then create it as type Real and give it the value **real**.
if the attribute called **att_name** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1008

Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Integer int)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Integer int)

Description

For the Element **drain** and on the pit number **pit**
if the attribute called **att_name** does not exist then create it as type Integer and give it the value **int**.
if the attribute called **att_name** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1007

Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Text txt)

Name

Integer Set_drainage_pit_attribute(Element drain,Integer pit,Text att_name,Text txt)

Description

For the Element **drain** and on the pit number **pit**,
if the attribute called **att_name** does not exist then create it as type Text and give it the value **txt**.
if the attribute called **att_name** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pit_attribute_type call can be used to get the type of the attribute called att_name.

ID = 1006

Drainage_pit_attribute_exists(Element drain,Integer pit,Text att_name)

Name

Integer Drainage_pit_attribute_exists (Element drain,Integer pit,Text att_name)

Description

For the Element **drain**, checks to see if an attribute with the name **att_name** exists for pit number **pit**.

A non-zero function return value indicates that an attribute of that name exists.

If the attribute does not exist, or **drain** is not of type Drainage, or there is no pit number **pit,** a **zero** function return value is returned.

Warning - this is the opposite of most 12dPL function return values.

ID = 987

Drainage_pit_attribute_exists (Element drain,Integer pit,Text name,Integer &no)

Name

Integer Drainage_pit_attribute_exists (Element drain,Integer pit,Text name,Integer &no)

Description

For the Element **drain**, checks to see if an attribute with the name **att_name** exists for pit number **pit**.

If the attribute of that name exists, its attribute number is returned is **no**.

A non-zero function return value indicates that an attribute of that name exists.

If the attribute does not exist, or **drain** is not of type Drainage, or there is no pit number **pit,** a **zero** function return value is returned.

Warning - this is the opposite of most 12dPL function return values.

ID = 988

Drainage_pit_attribute_delete (Element drain,Integer pit,Text att_name)

Name

Integer Drainage_pit_attribute_delete (Element drain,Integer pit,Text att_name)

Description

For the Element **drain**, delete the attribute with the name **att_name** for pit number **pit**.

If the Element **drain** is not of type **Drainage** or **drain** has no pit number **pit**, then a non-zero return code is returned.

A function return value of zero indicates the attribute was deleted.

ID = 989

Drainage_pit_attribute_delete (Element drain,Integer pit,Integer att_no)

Name

Integer Drainage_pit_attribute_delete (Element drain,Integer pit,Integer att_no)

Description

For the Element **drain**, delete the attribute with attribute number **att_no** for pit number **pit**.

If the Element **drain** is not of type **Drainage** or **drain** has no pit number **pit**, then a non-zero return code is returned.

A function return value of zero indicates the attribute was deleted.

ID = 990

Drainage_pit_attribute_delete_all (Element drain,Integer pit)

Name

Integer Drainage_pit_attribute_delete_all (Element drain,Integer pit)

Description

Delete all the attributes of pit number **pit** of the drainage string **drain**.

A function return value of zero indicates the function was successful.

ID = 991

Drainage_pit_attribute_dump (Element drain,Integer pit)

Name

Integer Drainage_pit_attribute_dump (Element drain,Integer pit)

Description

Write out information to the Output Window about the pit attributes for pit number **pit** of the drainage string **drain**.

A function return value of zero indicates the function was successful.

ID = 992

Drainage_pit_attribute_debug (Element drain,Integer pit)

Name

Integer Drainage_pit_attribute_debug (Element drain,Integer pit)

Description

Write out even more information to the Output Window about the pit attributes for pit number **pit** of the drainage string **drain**.

A function return value of zero indicates the function was successful.

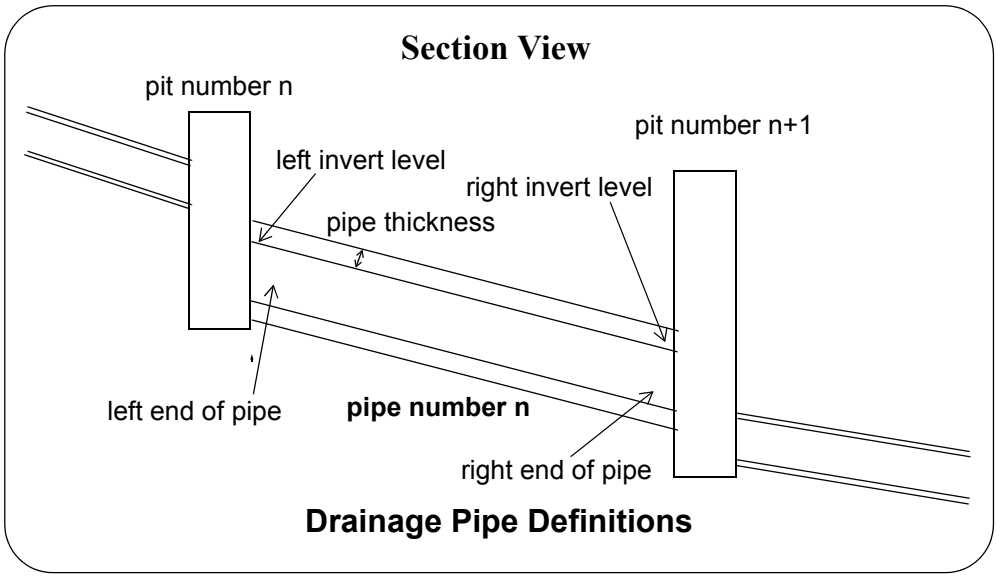
ID = 993

Go to the next section Drainage String Pipes or return to Drainage String Element .

Drainage String Pipes

Drainage Pipe Definitions

Drainage **pipe number n** goes from drainage pit number n and pit number n+1. The **left end** of the pipe is the end closest to pit n, and the **right end** is the end closes to pit n+1.



Drainage Pipe Cross Sections

A drainage pipe can have either a Circular, Box or Trapezoid cross section depending on whether only a diameter is defined (circular), only a diameter and a width are defined (box), or a diameter, width and top width are defined (trapezoid). The box and trapezoid will be referred to as *non round* pipes.

Pipes can also have thicknesses.

For a round pipe, there is only one thickness.

For a non round pipe, there is a top_thickness, bottom_thickness, left_thickness and right_thickness. Note that the left and right are defined when going in the chainage direction of the pipe.

So diameter, width and top width refer to the ***internal dimensions*** of the pipe and for a

round pipe, the *external diameter* = diameter + 2 * thickness

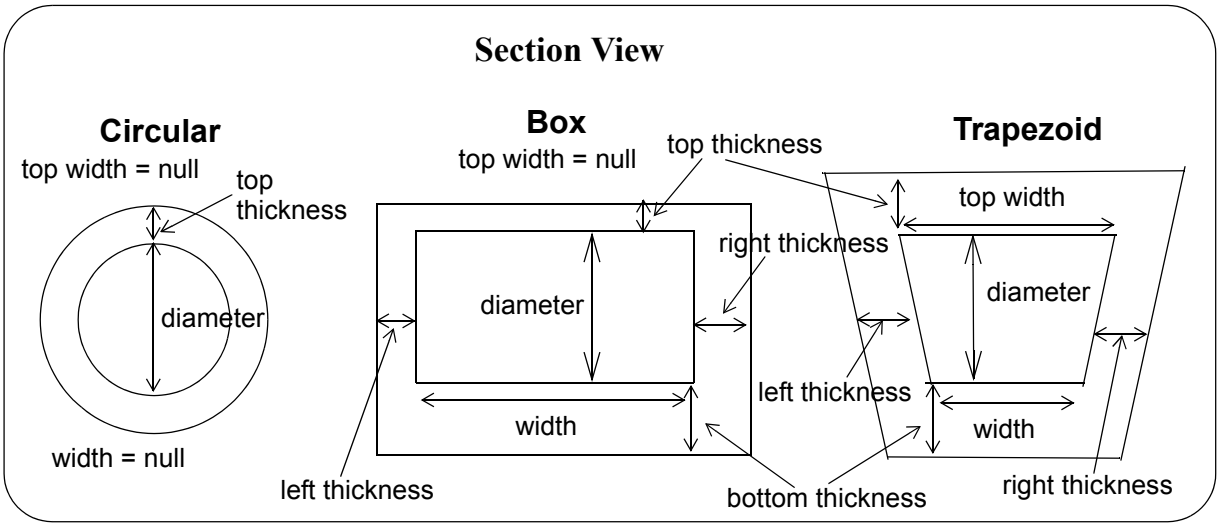
box pipe, the *external diameter* = diameter + top thickness + bottom thickness

the *external width* = width + left thickness + right thickness

trapezoid pipe, the *external diameter* = diameter + top thickness + bottom thickness

the *external width* = width + left thickness + right thickness

the *external top width* = top width + left thickness + right thickness


Section View
Circular
top width = null
top
thickness
diameter
width = null
Box
top width = null
top thickness
right thickness
diameter
width
left thickness
bottom thickness
Trapezoid
top width
diameter
left thickness
width
right thickness

Set_drainage_pipe_inverts(Element drain,Integer p,Real lhs,Real rhs)

Name

Integer Set_drainage_pipe_inverts(Element drain,Integer p,Real lhs,Real rhs)

Description

Set the pipe invert levels for the **p**th pipe of the string Element **drain**.

The invert level of the left hand end of the pipe is given as Real **lhs**.

The invert level of the right hand end of the pipe is given as Real **rhs**.

See Drainage Pipe Definitions .

Note: pipe invert levels can also be set using the call Set_drainage_pit_inverts(Element drain,Integer p,Real lhs,Real rhs) .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 536

Get_drainage_pipe_inverts(Element drain,Integer p,Real &lhs,Real &rhs)

Name

Integer Get_drainage_pipe_inverts(Element drain,Integer p,Real &lhs,Real &rhs)

Description

Get the pipe invert levels for the **p**th pipe of the string Element **drain**.

The invert level of the pipe of the left hand end of the pipe is returned in Real **lhs**.

The invert level of the right hand end of the pipe is returned in Real **rhs**.

See Drainage Pipe Definitions .

Note: pipe invert levels can also be returned using the call Get_drainage_pit_inverts(Element drain,Integer p,Real &lhs,Real &rhs) .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 533

Set_drainage_pipe_number_of_pipes(Element drain,Integer pipe,Integer n)

Name

Integer Set_drainage_pipe_number_of_pipes(Element drain,Integer pipe,Integer n)

Description

For the Element **drain,** which must be of type Drainage, and for the pipe number **pipe**, set the number of pipes to be **n**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the number was successfully set.

ID = 2852

Get_drainage_pipe_number_of_pipes(Element drain,Integer pipe,Integer &n)

Name

Integer Get_drainage_pipe_number_of_pipes(Element drain,Integer pipe,Integer &n)

Description

For the Element **drain,** which must be of type Drainage, and for the pipe number **pipe**, return the number of pipes as **n**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the number was successfully returned.

ID = 2853

Set_drainage_pipe_colour(Element drain,Integer p,Integer colour)

Name

Integer Set_drainage_pipe_colour(Element drain,Integer p,Integer colour)

Description

Set the colour of the **p**th pipe of the Element **drain** to colour number **colour**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2783

Get_drainage_pipe_colour(Element drain,Integer p,Integer &colour)

Name

Integer Get_drainage_pipe_colour(Element drain,Integer p,Integer &colour)

Description

Get the colour number of the **p**th pipe of the Element **drain** and return the colour number in **colour**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 2782

Set_drainage_pipe_name(Element drain,Integer p,Text name)

Name

Integer Set_drainage_pipe_name(Element drain,Integer p,Text name)

Description

Set the pipe name for the **p**th pipe of the string Element **drain**.

The pipe name is given as Text **name**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 502

Get_drainage_pipe_name(Element drain,Integer p,Text &name)

Name

Integer Get_drainage_pipe_name(Element drain,Integer p,Text &name)

Description

Get the pipe name for the **p**th pipe of the string Element **drain**.

The pipe name is returned in Text **name**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 497

Set_drainage_pipe_type(Element drain,Integer p,Text type)

Name

Integer Set_drainage_pipe_type(Element drain,Integer p,Text type)

Description

Set the pipe type for the **p**th pipe of the string Element **drain**.

The pipe type is given as Text **type**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 501

Get_drainage_pipe_type(Element drain,Integer p,Text &type)

Name

Integer Get_drainage_pipe_type(Element drain,Integer p,Text &type)

Description

Get the pipe type for the **p**th pipe of the string Element **drain**.

The pipe type is returned in Text **type**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 496

Set_drainage_pipe_cover(Element drain,Integer pipe,Real cover)

Name

Integer Set_drainage_pipe_cover(Element drain,Integer pipe,Real cover)

Description

For the Element **drain,** which must be of type *Drainage,* set the minimum cover for pipe number **pipe**, to **cover**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 1442

Get_drainage_pipe_cover(Element drain,Integer pipe,Real &minc,Real &maxc)

Name

Integer Get_drainage_pipe_cover(Element drain,Integer pipe,Real &minc,Real &maxc)

Description

For the Element **drain,** which must be of type *Drainage,* return the minimum cover value for pipe number **pipe**, in **cover**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 1441

Set_drainage_pipe_diameter(Element drain,Integer p,Real diameter)

Name

Integer Set_drainage_pipe_diameter(Element drain,Integer p,Real diameter)

Description

Set the pipe diameter for the **p**th pipe of the string Element **drain**.

The pipe diameter is given as Real **diameter**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 500

Set_drainage_pipe_width(Element drain,Integer pipe,Real &width)

Name

Integer Set_drainage_pipe_width(Element drain,Integer pipe,Real &width)

Description

For the Element **drain,** which must be of type Drainage, and pipe number **pipe**, set the width of the pipe to the value **width**.

If a width is not to be used then set a null value for **width**.

See Drainage Pipe Cross Sections .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the width was successfully set.

ID = 2857

Set_drainage_pipe_top_width(Element drain,Integer pipe,Real &top_width)

Name

Integer Set_drainage_pipe_top_width(Element drain,Integer pipe,Real &top_width)

Description

For the Element **drain,** which must be of type Drainage, and pipe number **pipe**, set the top width of the pipe to the value **top_width**.

If a top width is not to be used then set a null value for **top_width**.

See Drainage Pipe Cross Sections .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the top width was successfully set.

ID = 2858

Get_drainage_pipe_diameter(Element drain,Integer p,Real &diameter)

Name

Integer Get_drainage_pipe_diameter(Element drain,Integer p,Real &diameter)

Description

Get the pipe diameter for the pth pipe of the string Element **drain**.

The pipe diameter is returned in Real **diameter**.

See Drainage Pipe Cross Sections .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 495

Get_drainage_pipe_width(Element drain,Integer pipe,Real &width)

Name

Integer Get_drainage_pipe_width(Element drain,Integer pipe,Real &width)

Description

For the Element **drain,** which must be of type Drainage, and pipe number **pipe**, get the width of the pipe and return it in **width**.

If a width is not to be used then a null value is returned for **width**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

See Drainage Pipe Cross Sections .

A function return value of zero indicates the width was successfully returned.

ID = 2855

Get_drainage_pipe_top_width(Element drain,Integer pipe,Real &top_width)

Name

Integer Get_drainage_pipe_top_width(Element drain,Integer pipe,Real &top_width)

Description

For the Element **drain,** which must be of type Drainage, and pipe number **pipe**, get the top width of the pipe and return it in **top_width**.

If a top width is not to be used then a null value is returned for **top_width**.

See Drainage Pipe Cross Sections .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the top width was successfully returned.

ID = 2856

Get_drainage_pipe_thickness(Element drain,Integer pipe,Real &top,Real &bottom,Real &left,Real &right)

Name

Integer Get_drainage_pipe_thickness(Element drain,Integer pipe,Real &top,Real &bottom,Real &left,Real &right)

Description

For the Element **drain,** which must be of type Drainage, and pipe number **pipe**, set the pipe thicknesses to **top**, **bottom**, **left** and **right** where

top is the thickness for a round pipe, and the top thickness for a non round pipe.
bottom is the thickness of the bottom of the pipe for a non round pipe.
left is the thickness of the left of the pipe for a non round pipe.
right is the thickness of the right of the pipe for a non round pipe.

See Drainage Pipe Cross Sections .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the thicknesses were successfully set.

ID = 2867

Set_drainage_pipe_thickness(Element drain,Integer pit,Real top,Real bottom,Real left,Real right)

Name

Integer Set_drainage_pipe_thickness(Element drain,Integer pit,Real top,Real bottom,Real left,Real right)

Description

For the Element **drain,** which must be of type Drainage, and pipe number **pipe**, return the pipe thicknesses in **top**, **bottom**, **left** and **right** where

top is the thickness for a round pipe, and the top thickness for a non round pipe.
bottom is the thickness of the bottom of the pipe for a non round pipe.
left is the thickness of the left of the pipe for a non round pipe.
right is the thickness of the right of the pipe for a non round pipe.

See Drainage Pipe Cross Sections .

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the thicknesses were successfully returned.

ID = 2868

Get_drainage_pipe_intersects_pit(Element drain,Integer pipe,Real offset,Real &lx,Real &ly,Real &lch,Real &rx,Real &ry,Real &rch)

Name

Integer Get_drainage_pipe_intersects_pit(Element drain,Integer pipe,Real offset,Real &lx,Real &ly,Real &lch,Real &rx,Real &ry,Real &rch)

Description

For the Element **drain,** which must be of type Drainage, and for pipe number **pipe**, get the (x,y) coordinates and chainage of the intersection of the pipe offset (in the (x,y) pane) by the distance **offset**, with the pits at either end of the offset pipe.

If **offset** is positive then the pipe is offset to the right of the original pipe, and to the left when the offset is negative. Left and right are defined with respect to the direction of the pipe.

The coordinates of the intersection of the pipe with the left hand pit are returned as (**lx**,**ly**) and the chainage of the intersection point as **lch**.

The coordinates of the intersection of the pipe with the right hand pit are returned as (**rx**,**ry**) and the chainage of the intersection point as **rch**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the values were successfully returned.

ID = 2851

Get_drainage_pipe_shape(Element element,Integer pipe,Integer mode,Dynamic_Element &super_inside,Dynamic_Element &super_outside)

Name

Integer Get_drainage_pipe_shape(Element element,Integer pipe,Integer mode,Dynamic_Element &super_inside,Dynamic_Element &super_outside)

Description

For the Element **drain,** which must be of type *Drainage*, return as super strings, the shape of the insides of the pipes in the Dynamic_Element **super_inside** and the shape of the outsides of the pipes in the Dynamic_Element **super_outside**. The number of pipes, separation and thickness settings are used in generating all the shapes.

So this function returns a list of the super strings that “draw” the plan view of the inside and outside of the pipes.

For a circular pipe with wall thickness, the super_inside string is a super string with a plan box shape with a width of the diameter of the pipe and a length equal to the length of the pipe. And super_outside has a width equal to (diameter + 2*thickness).

For a rectangular pipe with a wall thicknesses, the super_inside is a super string with a plan box shape with a width of the diameter of the pipe and a length equal to the length of the pipe. And super_outside has a width equal to (diameter + left_thickness + right_thickness)

mode controls the z values assigned to the super strings.

If **mode** = 0, the shapes are given the z-value of the invert levels of the pipes.
If **mode** = 1, the shapes are given the z-value of the centre levels of the pipes.
If **mode** = 2, the shapes are given the z-value of the obvert levels of the pipes.

A function return value of 2 indicates the super strings could not be created.

A function return value of zero indicates the shapes were successfully returned.

ID = 2854

Get_drainage_pipe_shape(Element drain,Integer pipe,Integer mode,Real offset,Element &super_inside,Element &super_outside)

Name

Integer Get_drainage_pipe_shape(Element drain,Integer pipe,Integer mode,Real offset,Element &super_inside,Element &super_outside)

Description

For the Element **drain,** which must be of type Drainage, return the shape of the inside of pipe number **pipe** as the super string **super_inside** and the shape of the outside of the pipe as **super_outside**, and the shapes are offset in the (x,y) plane from the pipe by the distance **offset**.

If **offset** is positive then the shapes are offset to the right of the pipe and to the left when the offset is negative. Left and right is defined with respect to the direction of the pipe.

So this function returns a list of the super strings that “draw” the plan view of the inside and outside of the pipe offset by the given value **offset**.

For for a circular pipe with a wall thickness, the super_inside is a super string with a plan box shape with a width of the diameter of the pipe and a length equal to the length of the pipe. And super_outside has a width equal to (diameter + 2*thickness).

For a rectangular pipe with a wall thicknesses, the super_inside is a super string with a plan box shape with a width of the diameter of the pipe and a length equal to the length of the pipe. And

super_outside has a width equal to (diameter + left_thickness + right_thickness)

If **mode** = 0, the shapes are given the z-value of the invert levels of the pipe.
If **mode** = 1, the shapes are given the z-value of the centre levels of the pipe.
If **mode** = 2, the shapes are given the z-value of the obvert levels of the pipe.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the shapes were successfully returned.

Note: the number of pipes and separation are not used for generating the shapes and offset is use instead. For generating shapes using number of pipes and separation, see Get_drainage_pipe_shape(Element element,Integer pipe,Integer mode,Dynamic_Element &super_inside,Dynamic_Element &super_outside)

ID = 2850

Set_drainage_pipe_hgls(Element drain,Integer p,Real lhs,Real rhs)

Name

Integer Set_drainage_pipe_hgls(Element drain,Integer p,Real lhs,Real rhs)

Description

Set the pipe hgl levels for the **p**th pipe of the string Element drain.

The hgl level of the left hand side of the pipe is set to **lhs**.

The hgl level of the right hand side of the pipe is set to **rhs**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 537

Get_drainage_pipe_hgls(Element drain,Integer p,Real &lhs,Real &rhs)

Name

Integer Get_drainage_pipe_hgls(Element drain,Integer p,Real &lhs,Real &rhs)

Description

Get the pipe HGL levels for the **p**th pipe of the string Element **drain**.

The hgl level of the left hand side of the pipe is returned in **lhs**.

The hgl level of the right hand side of the pipe is returned in **rhs**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 534

Set_drainage_pipe_velocity(Element drain,Integer p,Real velocity)

Name

Integer Set_drainage_pipe_velocity(Element drain,Integer p,Real velocity)

Description

Get the pipe flow velocity for the **p**th pipe of the string Element **drain**.

The velocity of the pipe is returned in Real **velocity**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 499

Get_drainage_pipe_velocity(Element drain,Integer p,Real &velocity)

Name

Integer Get_drainage_pipe_velocity(Element drain,Integer p,Real &velocity)

Description

Get the flow velocity for the **p**th pipe of the string Element drain.

The velocity is returned in Real **velocity**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 494

Set_drainage_pipe_flow(Element drain,Integer p,Real flow)

Name

Integer Set_drainage_pipe_flow(Element drain,Integer p,Real flow)

Description

Get the pipe flow volume for the **p**th pipe of the string Element **drain**.

The velocity of the pipe is returned in Real **flow**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully set.

ID = 498

Get_drainage_pipe_flow(Element drain,Integer p,Real &flow)

Name

Integer Get_drainage_pipe_flow(Element drain,Integer p,Real &flow)

Description

Get the flow volume for the **p**th pipe of the string Element **drain**.

The volume is returned in Real **velocity**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 493

Get_drainage_pipe_length(Element drain,Integer p,Real &length)

Name

Integer Get_drainage_pipe_length(Element drain,Integer p,Real &length)

Description

Get the pipe length for the **p**th pipe of the string Element **drain**.

The length of the pipe is returned in Real **length**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 503

Get_drainage_pipe_grade(Element drain,Integer p,Real &grade)

Name

Integer Get_drainage_pipe_grade(Element drain,Integer p,Real &grade)

Description

Get the pipe grade for the **p**th pipe of the string Element **drain**.

The grade of the pipe is returned in Real **grade**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 504

Get_drainage_pipe_ns(Element drain,Integer p,Real ch[],Real ht[],Integer max_pts,Integer &npts)

Name

Integer Get_drainage_pipe_ns(Element drain,Integer p,Real ch[],Real ht[],Integer max_pts,Integer &npts)

Description

For the drainage string **drain,** get the heights along the **p**th pipe from the natural surface tin.

Because the pipe is long then there will be more than one height and the heights are returned in chainage order along the pipe. The heights are returned in the arrays **ch** (for chainage) and **ht**.

The maximum number of natural surface points that can be returned is given by **max_pts** (usually the size of the arrays).

The actual number of points of natural surface is returned in **npts**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 523

Get_drainage_pipe_fs(Element drain,Integer p,Real ch[],Real ht[],Integer max_pts,Integer &npts)

Name

Integer Get_drainage_pipe_fs(Element drain,Integer p,Real ch[],Real ht[],Integer max_pts,Integer &npts)

Description

For the drainage string **drain,** get the heights along the **p**th pipe from the finished surface tin.

Because the pipe is long then there will be more than one height and the heights are returned in chainage order along the pipe. The heights are returned in the arrays **ch** (for chainage) and **ht**.

The maximum number of finished surface points that can be returned is given by **max_pts** (usually the size of the arrays).

The actual number of points of finished surface is returned in **npts**.

If **drain** is not an Element of type *Drainage* then a non zero function return code is returned.

A function return value of zero indicates the data was successfully returned.

ID = 524

Go to the next section Drainage Pipe Type Information in the drainage.4d File or return to Drainage String Element .

Drainage Pipe Type Information in the drainage.4d File

Get_drainage_number_of_pipe_types(Integer &n)

Name

Integer Get_drainage_number_of_pipe_types(Integer &n)

Description

Get the number of pipe types (classes) from the *drainage.4d* file and return the number in *n*.

A function return value of zero indicates the data was successfully returned.

ID = 2271

Get_drainage_pipe_type(Integer i,Text &type)

Name

Integer Get_drainage_pipe_type(Integer i,Text &type)

Description

Get the name of the **i**'th pipe type (class) from the *drainage.4d* file and return the name in *type*.

A function return value of zero indicates the data was successfully returned.

ID = 2272

Get_drainage_pipe_roughness(Text type,Real &roughness,Integer &roughness_type)

Name

Integer Get_drainage_pipe_roughness(Text type,Real &roughness,Integer &roughness_type)

Description

For the pipe type ***type***, return from the *drainage.4d* file, the roughness in *roughness* and roughness type in *roughness_type*. Roughness type is MANNING (0) or COLEBROOK (1).

If pipe type ***type*** does not exist, then a non-zero return value is returned.

A function return value of zero indicates the data was successfully returned.

ID = 2273

Go to the next section Drainage String Pipe Attributes or return to Drainage String Element .

Drainage String Pipe Attributes

Set_drainage_pipe_attributes(Element drain,Integer pipe,Attributes att)

Name

Integer Set_drainage_pipe_attributes(Element drain,Integer pipe,Attributes att)

Description

For the Element **drain**, set the Attributes for the pipe number **pipe** to **att**.

If the Element is not of type **Drainage** then a non-zero return value is returned.

A function return value of zero indicates the attribute is successfully set.

ID = 2033

Get_drainage_pipe_attributes(Element drain,Integer pipe,Attributes &att)

Name

Integer Get_drainage_pipe_attributes(Element drain,Integer pipe,Attributes &att)

Description

For the Element **drain**, return the Attributes for the pipe number **pipe** as **att**.

If the Element is not of type **Drainage** or the pipe number **pipe** has no attribute then a non-zero return value is returned.

A function return value of zero indicates the attribute is successfully returned.

ID = 2032

Get_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Uid &uid)

Name

Integer Get_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Uid &uid)

Description

For the Element **drain**, get the attribute called **att_name** for the pipe number **pipe** and return the attribute value in **uid**. The attribute must be of type Uid.

If the Element is not of type **Drainage** or the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_name.

ID = 2034

Get_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Attributes &att)

Name

Integer Get_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Attributes &att)

Description

For the Element **drain**, get the attribute called **att_name** for the pipe number **pipe** and return the attribute value in **att**. The attribute must be of type Attributes.

If the Element is not of type **Drainage** or the attribute is not of type **Attributes** then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_name.

ID = 2035

Get_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Uid &uid)

Name

Integer Get_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Uid &uid)

Description

For the Element **drain** get the attribute with number **att_no** for the pipe number **pipe** and return the attribute value in **uid**. The attribute must be of type Uid.

If the Element is not of type **Drainage** or the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number att_no.

ID = 2036

Get_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no, Attributes &att)

Name

Integer Get_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Attributes &att)

Description

For the Element **drain**, get the attribute with number **att_no** for the pipe number **pipe** and return the attribute value in **att**. The attribute must be of type Attributes.

If the Element is not of type **Drainage** or the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number att_no.

ID = 2037

Set_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Uid uid)

Name

Integer Set_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Uid uid)

Description

For the Element **drain** and on the pipe number **pipe**,
if the attribute called **att_name** does not exist then create it as type Uid and give it the value **uid**.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **uid**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 2038

Set_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name, Attributes att)

Name

Integer Set_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Attributes att)

Description

For the Element **drain** and on the pipe number **pipe**,
if the attribute called **att_name** does not exist then create it as type Attributes and give it the value **att**.
if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_name.

ID = 2039

Set_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Uid uid)

Name

Integer Set_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Uid uid)

Description

For the Element **drain** and on the pipe number **pipe**, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **uid**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 2040

Set_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no, Attributes att)

Name

Integer Set_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Attributes att)

Description

For the Element **drain** and on the pipe number **pipe**, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called att_no.

ID = 2041

Get_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Text &txt)

Name

Integer Get_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Text &txt)

Description

For the Element **drain**, get the attribute called **att_name** for the pipe number **pipe** and return the attribute value in **txt**. The attribute must be of type Text.

If the Element is not of type **Drainage** or the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1020

Get_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Integer &int)

Name

Integer Get_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Integer &int)

Description

For the Element **drain**, get the attribute called **att_name** for the pipe number **pipe** and return the attribute value in **int**. The attribute must be of type Integer.

If the Element is not of type **Drainage** or the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1021

Get_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Real &real)

Name

Integer Get_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Real &real)

Description

For the Element **drain**, get the attribute called **att_name** for the pipe number **pipe** and return the attribute value in **real**. The attribute must be of type Real.

If the Element is not of type **Drainage** or the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1022

Get_drainage_pipe_attribute (Element drain,Integer pipe,Integer att_no,Text &txt)

Name

Integer Get_drainage_pipe_attribute (Element drain,Integer pipe,Integer att_no,Text &txt)

Description

For the Element **drain**, get the attribute with number **att_no** for the pipe number **pipe** and return the attribute value in **txt**. The attribute must be of type Text.

If the Element is not of type **Drainage** or the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1023

Get_drainage_pipe_attribute (Element drain,Integer pipe,Integer att_no,Integer &int)

Name

Integer Get_drainage_pipe_attribute (Element drain,Integer pipe,Integer att_no,Integer &int)

Description

For the Element **drain**, get the attribute with number **att_no** for the pipe number **pipe** and return the attribute value in **int**. The attribute must be of type Integer.

If the Element is not of type **Drainage** or the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1024

Get_drainage_pipe_attribute (Element drain,Integer pipe,Integer att_no,Real &real)

Name

Integer Get_drainage_pipe_attribute (Element drain,Integer pipe,Integer att_no,Real &real)

Description

For the Element **drain**, get the attribute with number **att_no** for the pipe number **pipe** and return the attribute value in **real**. The attribute must be of type Real.

If the Element is not of type **Drainage** or the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1025

Drainage_pipe_attribute_exists(Element drain,Integer pipe,Text att_name)

Name

Integer Drainage_pipe_attribute_exists (Element drain,Integer pipe,Text att_name)

Description

For the Element **drain**, checks to see if an attribute with the name **att_name** exists for pipe number **pipe**.

A non-zero function return value indicates that an attribute of that name exists.

If the attribute does not exist, or **drain** is not of type Drainage, or there is no pipe number **pipe,** a **zero** function return value is returned.

Warning this is the opposite of most 12dPL function return values.

ID = 1012

Drainage_pipe_attribute_exists (Element drain, Integer pipe,Text name,Integer &no)

Name

Integer Drainage_pipe_attribute_exists (Element drain, Integer pipe,Text name,Integer &no)

Description

For the Element **drain**, checks to see if an attribute with the name **att_name** exists for pipe number **pipe**.

If the attribute of that name exists, its attribute number is returned is **no**.

A non-zero function return value indicates that an attribute of that name exists.

If the attribute does not exist, or **drain** is not of type Drainage, or there is no pipe number **pipe,** a **zero** function return value is returned.

Warning this is the opposite of most 12dPL function return values.

ID = 1013

Drainage_pipe_attribute_delete (Element drain,Integer pipe,Text att_name)

Name

Integer Drainage_pipe_attribute_delete (Element drain,Integer pipe,Text att_name)

Description

For the Element **drain**, delete the attribute with the name **att_name** for pipe number **pipe**.

If the Element **drain** is not of type **Drainage** or **drain** has no pipe number **pipe**, then a non-zero return code is returned.

A function return value of zero indicates the attribute was deleted.

ID = 1014

Drainage_pipe_attribute_delete (Element drain,Integer pipe,Integer att_no)

Name

Integer Drainage_pipe_attribute_delete (Element drain,Integer pipe,Integer att_no)

Description

For the Element **drain**, delete the attribute with attribute number **att_no** for pipe number **pipe**.

If the Element **drain** is not of type **Drainage** or **drain** has no pipe number **pipe**, then a non-zero return code is returned.

A function return value of zero indicates the attribute was deleted.

ID = 1015

Drainage_pipe_attribute_delete_all (Element drain,Integer pipe)

Name

Integer Drainage_pipe_attribute_delete_all (Element drain,Integer pipe)

Description

Delete all the attributes of pipe number **pipe** of the drainage string **drain**.

A function return value of zero indicates the function was successful.

ID = 1016

Drainage_pipe_attribute_dump (Element drain,Integer pipe)

Name

Integer Drainage_pipe_attribute_dump (Element drain,Integer pipe)

Description

Write out information to the Output Window about the pipe attributes for pipe number **pipe** of the drainage string **drain**.

A function return value of zero indicates the function was successful.

ID = 1017

Drainage_pipe_attribute_debug (Element drain,Integer pipe)

Name

Integer Drainage_pipe_attribute_debug (Element drain,Integer pipe)

Description

Write out even more information to the Output Window about the pipe attributes for pipe number **pipe** of the drainage string **drain**.

A function return value of zero indicates the function was successful.

ID = 1018

Get_drainage_pipe_number_of_attributes(Element drain,Integer pipe,Integer &no_atts)

Name

Integer Get_drainage_pipe_number_of_attributes(Element drain,Integer pipe,Integer &no_atts)

Description

Get the total number of attributes for pipe number **pipe** of the Element **drain**.

The total number of attributes is returned in Integer **no_atts**.

A function return value of zero indicates the number of attributes was successfully returned.

ID = 1019

Get_drainage_pipe_attribute_length (Element drain,Integer pipe,Text

att_name,Integer &att_len)

Name

Integer Get_drainage_pipe_attribute_length (Element drain,Integer pipe,Text att_name,Integer &att_len)

Description

For pipe number **pipe** of the Element **drain**, get the length (in bytes) of the attribute with the name **att_name**. The attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for user attributes of type **Text** and **Binary.**

ID = 1029

Get_drainage_pipe_attribute_length (Element drain,Integer pipe,Integer att_no,Integer &att_len)

Name

Integer Get_drainage_pipe_attribute_length (Element drain,Integer pipe,Integer att_no,Integer &att_len)

Description

For pipe number **pipe** of the Element **drain**, get the length (in bytes) of the attribute number **att_no**. The attribute length is returned in **att_len**.

A function return value of zero indicates the attribute length was successfully returned.

Note - the length is useful for attributes of type **Text** and **Binary.**

ID = 1030

Get_drainage_pipe_attribute_name(Element drain,Integer pipe,Integer att_no,Text &name)

Name

Integer Get_drainage_pipe_attribute_name(Element drain,Integer pipe,Integer att_no,Text &name)

Description

For pipe number **pipe** of the Element **drain**, get the name of the attribute number **att_no**. The attribute name is returned in **name**.

A function return value of zero indicates the attribute name was successfully returned.

ID = 1026

Get_drainage_pipe_attribute_type(Element drain,Integer pipe,Text att_name,Integer &att_type)

Name

Integer Get_drainage_pipe_attribute_type(Element drain,Integer pipe,Text att_name,Integer &att_type)

Description

For pipe number **pipe** of the Element **drain**, get the type of the attribute with name **att_name**. The attribute type is returned in **att_type**.

A function return value of zero indicates the attribute type was successfully returned.

ID = 1027

Get_drainage_pipe_attribute_type(Element drain,Integer pipe,Integer att_no,Integer &att_type

Name

Integer Get_drainage_pipe_attribute_type(Element drain,Integer pipe,Integer att_no,Integer &att_type)

Description

For pipe number **pipe** of the Element **drain**, get the type of the attribute with attribute number **att_no**. The attribute type is returned in **att_type**.

A function return value of zero indicates the attribute type was successfully returned.

ID = 1028

Set_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Text txt)

Name

Integer Set_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Text txt)

Description

For the Element **drain** and on the pipe number **pipe**,
if the attribute called **att_name** does not exist then create it as type Text and give it the value **txt**.
if the attribute called **att_name** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute called att_name.

ID = 1031

Set_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Integer int)

Name

Integer Set_drainage_pipe_attribute (Element drain,Integer pipe,Text att_name,Integer int)

Description

For the Element **drain** and on the pipe number **pipe**,
if the attribute called **att_name** does not exist then create it as type Integer and give it the value **int**.
if the attribute called **att_name** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1032

Set_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Real real)

Name

Integer Set_drainage_pipe_attribute(Element drain,Integer pipe,Text att_name,Real real)

Description

For the Element **drain** and on the pipe number **pipe**,
if the attribute called **att_name** does not exist then create it as type Real and give it the value **real**.
if the attribute called **att_name** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1033

Set_drainage_pipe_attribute (Element drain,Integer pipe,Integer att_no,Text txt)

Name

Integer Set_drainage_pipe_attribute (Element drain,Integer pipe,Integer att_no,Text txt)

Description

For the Element **drain** and on the pipe number **pipe**,
if the attribute with number **att_no** does not exist then create it as type Text and give it the value **txt**.
if the attribute with number **att_no** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute number att_no.

ID = 1034

Set_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Integer int)

Name

Integer Set_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Integer int)

Description

For the Element **drain** and on the pipe number **pipe**,
if the attribute with number **att_no** does not exist then create it as type Integer and give it the value **int**.
if the attribute with number **att_no** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute number att_no.

ID = 1035

Set_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Real real)

Name

Integer Set_drainage_pipe_attribute(Element drain,Integer pipe,Integer att_no,Real real)

Description

For the Element **drain** and on the pipe number **pipe**,

if the attribute with number **att_no** does not exist then create it as type Real and give it the value **real**.
if the attribute with number **att_no** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_drainage_pipe_attribute_type call can be used to get the type of the attribute number **att_no**.

ID = 1036

Go to the next section Drainage String House Connections - For Sewer Module Only or return to Drainage String Element .

Drainage String House Connections - For Sewer Module Only

Get_drainage_hcs(Element drain,Integer &no_hcs)

Name

Integer Get_drainage_hcs(Element drain,Integer &no_hcs)

Description

Get the number of house connections for the string Element **drain**.

The number of house connection is returned in Integer **no_hcs**.

A function return value of zero indicates the data was successfully returned.

ID = 590

Get_drainage_hc(Element drain,Integer h,Real &x,Real &y,Real &z)

Name

Integer Get_drainage_hc(Element drain,Integer h,Real &x,Real &y,Real &z)

Description

Get the x,y & z for the **h**th house connection of the string Element **drain**.

The x coordinate of the house connection is returned in Real **x**.
The y coordinate of the house connection is returned in Real **y**.
The z coordinate of the house connection is returned in Real **z**.

A function return value of zero indicates the data was successfully returned.

ID = 591

Set_drainage_hc_adopted_level(Element drain,Integer hc,Real level)

Name

Integer Set_drainage_hc_adopted_level(Element drain,Integer hc,Real level)

Description

For the drainage string **drain**, set the adopted level for the **h'**th house connection to **level**.

A function return value of zero indicates the data was successfully set.

ID = 1302

Get_drainage_hc_adopted_level(Element drain,Integer h,Real &level)

Name

Integer Get_drainage_hc_adopted_level(Element drain,Integer h,Real &level)

Description

Get the adopted level for the **h'**th house connection of the string Element drain.

The adopted level of the house connection is returned in Real **level**.

A function return value of zero indicates the data was successfully returned.

ID = 598

Set_drainage_hc_bush(Element drain,Integer hc,Text bush)

Name

Integer Set_drainage_hc_bush(Element drain,Integer hc,Text bush)

Description

For the drainage string **drain**, set the bush type for the **h'**th house connection to **bush**.

A function return value of zero indicates the data was successfully set.

ID = 1310

Get_drainage_hc_bush(Element drain,Integer h,Text &bush)

Name

Integer Get_drainage_hc_bush(Element drain,Integer h,Text &bush)

Description

Get the bush type for the **h'**th house connection of the string Element **drain**.

The bush type of the house connection is returned in Text **bush**.

A function return value of zero indicates the data was successfully returned.

ID = 606

Set_drainage_hc_colour(Element drain,Integer hc,Integer colour)

Name

Integer Set_drainage_hc_colour(Element drain,Integer hc,Integer colour)

Description

For the drainage string **drain**, set the colour number for the **h'**th house connection to **colour**.

A function return value of zero indicates the data was successfully set.

ID = 1307

Get_drainage_hc_colour(Element drain,Integer h,Integer &colour)

Name

Integer Get_drainage_hc_colour(Element drain,Integer h,Integer &colour)

Description

Get the colour for the **h'**th house connection of the string Element **drain**.

The colour of the house connection is returned in Integer **colour**.

A function return value of zero indicates the data was successfully returned.

ID = 603

Set_drainage_hc_depth(Element drain,Integer hc,Real depth)

Name

Integer Set_drainage_hc_depth(Element drain,Integer hc,Real depth)

Description

For the drainage string **drain**, set the depth for the **h'**th house connection to **depth**.

A function return value of zero indicates the data was successfully set.

ID = 1305

Get_drainage_hc_depth(Element drain,Integer h,Real &depth)

Name

Integer Get_drainage_hc_depth(Element drain,Integer h,Real &depth)

Description

Get the depth for the **h'**th house connection of the string Element **drain**.

The depth of the house connection is returned in Real **depth**.

A function return value of zero indicates the data was successfully returned.

ID = 601

Set_drainage_hc_diameter(Element drain,Integer hc,Real diameter)

Name

Integer Set_drainage_hc_diameter(Element drain,Integer hc,Real diameter)

Description

For the drainage string **drain**, set the diameter for the **h'**th house connection to **diameter**.

A function return value of zero indicates the data was successfully set.

ID = 1306

Get_drainage_hc_diameter(Element drain,Integer h,Real &diameter)

Name

Integer Get_drainage_hc_diameter(Element drain,Integer h,Real &diameter)

Description

Get the diameter for the **h'**th house connection of the string Element **drain**.

The diameter of the house connection is returned in Real **diameter**.

A function return value of zero indicates the data was successfully returned.

ID = 602

Set_drainage_hc_grade(Element drain,Integer hc,Real grade)

Name

Integer Set_drainage_hc_grade(Element drain,Integer hc,Real grade)

Description

For the drainage string **drain**, set the grade for the **h'**th house connection to **grade**.

A function return value of zero indicates the data was successfully set.

ID = 1304

Get_drainage_hc_grade(Element drain,Integer h,Real &grade)

Name

Integer Get_drainage_hc_grade(Element drain,Integer h,Real &grade)

Description

Get the grade for the **h'**th house connection of the string Element **drain**.

The grade of the house connection is returned in Real **grade**.

A function return value of zero indicates the data was successfully returned.

ID = 600

Set_drainage_hc_hcb(Element drain,Integer hc,Integer hcb)

Name

Integer Set_drainage_hc_hcb(Element drain,Integer hc,Integer hcb)

Description

For the drainage string **drain**, set the hcb for the **h'**th house connection to **hcb**.

A function return value of zero indicates the data was successfully set.

ID = 1300

Get_drainage_hc_hcb(Element drain,Integer h,Integer &hcb)

Name

Integer Get_drainage_hc_hcb(Element drain,Integer h,Integer &hcb)

Description

Get the hcb for the **h'**th house connection of the string Element **drain**.

The hcb of the house connection is returned in Integer **hcb**.

A function return value of zero indicates the data was successfully returned.

ID = 596

Set_drainage_hc_length(Element drain,Integer hc,Real length)

Name

Integer Set_drainage_hc_length(Element drain,Integer hc,Real length)

Description

For the drainage string **drain**, set the length for the **h'**th house connection to **length**.

A function return value of zero indicates the data was successfully set.

ID = 1303

Get_drainage_hc_length(Element drain,Integer h,Real &length)

Name

Integer Get_drainage_hc_length(Element drain,Integer h,Real &length)

Description

Get the length for the **h'**th house connection of the string Element **drain**.

The length of the house connection is returned in Real **length**.

A function return value of zero indicates the data was successfully returned.

ID = 599

Set_drainage_hc_level(Element drain,Integer hc,Real level)

Name

Integer Set_drainage_hc_level(Element drain,Integer hc,Real level)

Description

For the drainage string **drain**, set the level for the **h'**th house connection to **level**.

A function return value of zero indicates the data was successfully set.

ID = 1301

Get_drainage_hc_level(Element drain,Integer h,Real &level)

Name

Integer Get_drainage_hc_level(Element drain,Integer h,Real &level)

Description

Get the level for the **h'**th house connection of the string Element **drain**.

The level of the house connection is returned in Real **level**.

A function return value of zero indicates the data was successfully returned.

ID = 597

Set_drainage_hc_material(Element drain,Integer hc,Text material)

Name

Integer Set_drainage_hc_material(Element drain,Integer hc,Text material)

Description

For the drainage string **drain**, set the material for the **h'**th house connection to **material**.

A function return value of zero indicates the data was successfully set.

ID = 1309

Get_drainage_hc_material(Element drain,Integer h,Text &material)

Name

Integer Get_drainage_hc_material(Element drain,Integer h,Text &material)

Description

Get the material for the **h'**th house connection of the string Element **drain**.

The material of the house connection is returned in Text **material**.

A function return value of zero indicates the data was successfully returned.

ID = 605

Set_drainage_hc_name(Element drain,Integer hc,Text name)

Name

Integer Set_drainage_hc_name(Element drain,Integer hc,Text name)

Description

For the drainage string **drain**, set the name for the **h'**th house connection to **name**.

A function return value of zero indicates the data was successfully set.

ID = 1299

Get_drainage_hc_name(Element drain,Integer h,Text &name)

Name

Integer Get_drainage_hc_name(Element drain,Integer h,Text &name)

Description

Get the name for the **h'**th house connection of the string Element **drain**.

The name of the house connection is returned in Text **name**.

A function return value of zero indicates the data was successfully returned.

ID = 595

Set_drainage_hc_side(Element drain,Integer hc,Integer side)

Name

Integer Set_drainage_hc_side(Element drain,Integer hc,Integer side)

Description

For the drainage string **drain**, set the side for the **h'**th house connection by the value of **side**.

when **side** = -1, the house connection is on the left side of the string.
when **side** = 1, the house connection is on the right side of the string.

A function return value of zero indicates the data was successfully set.

ID = 1298

Get_drainage_hc_side(Element drain,Integer h,Integer &side)

Name

Integer Get_drainage_hc_side(Element drain,Integer h,Integer &side)

Description

Get the side for the **h'**th house connection of the string Element **drain**.

The side of the house connection is returned in Integer **side**.

If **side** = -1, the house connection is on the left side of the string.
If **side** = 1, the house connection is on the right side of the string.

A function return value of zero indicates the data was successfully returned.

ID = 594

Set_drainage_hc_type(Element drain,Integer hc,Text type)

Name

Integer Set_drainage_hc_type(Element drain,Integer hc,Text type)

Description

For the drainage string **drain**, set the hc type for the **h'**th house connection to **type**.

A function return value of zero indicates the data was successfully set.

ID = 1308

Get_drainage_hc_type(Element drain,Integer h,Text &type)

Name

Integer Get_drainage_hc_type(Element drain,Integer h,Text &type)

Description

Get the type for the **h'**th house connection of the string Element **drain**.

The type of the house connection is returned in Text **type**.

A function return value of zero indicates the data was successfully returned.

ID = 604

Get_drainage_hc_chainage(Element drain,Integer h,Real &chainage)

Name

Integer Get_drainage_hc_chainage(Element drain,Integer h,Real &chainage)

Description

Get the chainage for the **h'**th house connection of the string Element **drain**.

The chainage of the house connection is returned in Real **chainage**.

A function return value of zero indicates the data was successfully returned.

ID = 592

Get_drainage_hc_ip(Element drain,Integer h,Integer &ip)

Name

Integer Get_drainage_hc_ip(Element drain,Integer h,Integer &ip)

Description

Get the intersect point for the **h'**th house connection of the string Element **drain**.

The intersection point of the house connection is returned in Integer **ip**.

A function return value of zero indicates the data was successfully returned.

ID = 593

Go to the next major section Feature String Element or return to Drainage String Element .

Feature String Element

A 12d Model Feature string is a circle with a z-value at the centre but only null values on the circumference.

Create_feature()

Name

Element Create_feature()

Description

Create an Element of type **Feature**

The function return value gives the actual Element created.

If the feature string could not be created, then the returned Element will be null.

ID = 872

Create_feature(Element seed)

Name

Element Create_feature(Element seed)

Description

Create an Element of type **Feature** and set the colour, name, style etc. of the new string to be the same as those from the Element **Seed**.

The function return value gives the actual Element created.

If the Feature string could not be created, then the returned Element will be null.

ID = 873

Create_feature(Text name,Integer colour,Real xc,Real yc,Real zc,Real rad)

Name

Element Create_feature(Text name,Integer colour,Real xc,Real yc,Real zc,Real rad)

Description

Create an Element of type **Feature** with name **name**, colour **colour**, centre (**xc,yc**), radius **rad** and z value (height) **zc**.

The function return value gives the actual Element created.

If the Feature string could not be created, then the returned Element will be null.

ID = 874

Get_feature_centre(Element elt,Real &xc,Real &yc,Real &zc)

Name

Integer Get_feature_centre(Element elt,Real &xc,Real &yc,Real &zc)

Description

Get the centre point for Feature string given by Element **elt**.

The centre of the Feature is (**xc,yc,zc**).

A function return value of zero indicates the centre was successfully returned.

ID = 876

Set_feature_centre(Element elt,Real xc,Real yc,Real zc)

Name

Integer Set_feature_centre(Element elt,Real xc,Real yc,Real zc)

Description

Set the centre point of the Feature string given by Element **elt** to (**xc,yc,zc**).

A function return value of zero indicates the centre was successfully modified.

ID = 875

Get_feature_radius(Element elt,Real &rad)

Name

Integer Get_feature_radius(Element elt,Real &rad)

Description

Get the radius for Feature string given by Element **elt** and return it in **rad**.

A function return value of zero indicates the radius was successfully returned.

ID = 878

Set_feature_radius(Element elt,Real rad)

Name

Integer Set_feature_radius(Element elt,Real rad)

Description

Set the radius of the Feature string given by Element **elt** to **rad**. The new radius must be non-zero.

A function return value of zero indicates the radius was successfully modified.

ID = 877

Interface String Element

A Interface string consists of (x,y,z,flag) values at each point of the string where flag is the cut-fill flag.

If the cut-fill flag is

-2	the surface was not reached
-1	the point was in cut
0	the point was on the surface
1	the point was in fill

The following functions are used to create new Interface strings and make inquiries and modifications to existing Interface strings.

Create_interface(Real x[],Real y[],Real z[],Integer f[],Integer num_pts)

Name

Element Create_interface(Real x[],Real y[],Real z[],Integer f[],Integer num_pts)

Description

Create an Element of type **Interface**.

The Element has **num_pts** points with (x,y,z,flag) values given in the Real arrays **x[]**, **y[]**, **z[]** and Integer array **f[]**.

The function return value gives the actual Element created.

If the Interface string could not be created, then the returned Element will be null.

ID = 181

Create_interface(Integer num_pts)

Name

Element Create_interface(Integer num_pts)

Description

Create an Element of type **Interface** with room for **num_pts** (x,y,z,flag) points.

The actual x, y, z and flag values of the Interface string are set after the string is created.

If the Interface string could not be created, then the returned Element will be null.

ID = 451

Create_interface(Integer num_pts,Element seed)

Name

Element Create_interface(Integer num_pts,Element seed)

Description

Create an Element of type Interface with room for **num_pts** (x,y,z,flag) points, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

The actual x, y, z and flag values of the Interface string are set after the string is created.

If the Interface string could not be created, then the returned Element will be null.

ID = 668

Get_interface_data(Element elt,Real x[],Real y[],Real z[], Integer f[],Integer max_pts,Integer &num_pts)

Name

Integer Get_interface_data(Element elt,Real x[],Real y[],Real z[],Integer f[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z,flag) data for the first **max_pts** points of the Interface Element **elt**.

The (x,y,z,flag) values at each string point are returned in the Real arrays **x[]**, **y[]**, **z[]** and Integer array **f[]**.

The maximum number of points that can be returned is given by **max_pts** (usually the size of the arrays). The point data returned starts at the first point and goes up to the minimum of **max_pts** and the number of points in the string.

The actual number of points returned is given by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type Interface, then **num_pts** is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

ID = 182

Get_interface_data(Element elt,Real x[],Real y[],Real z[],Integer f[],Integer max_pts,Integer &num_pts,Integer start_pt)

Name

Integer Get_interface_data(Element elt,Real x[],Real y[],Real z[],Integer f[],Integer max_pts,Integer &num_pts,Integer start_pt)

Description

For a Interface Element **elt**, get the (x,y,z,flag) data for **max_pts** points starting at the point number **start_pt**.

This routine allows the user to return the data from a Interface string in user specified chunks. This is necessary if the number of points in the string is greater than the size of the arrays available to contain the information.

As in the previous function, the maximum number of points that can be returned is given by **max_pts** (usually the size of the arrays).

However, for this function, the point data returned starts at point number start_pt rather than point one.

The (x,y,z,text) values at each string point are returned in the Real arrays **x[]**, **y[]**, **z[]** and Integer array **f[]**.

The actual number of points returned is given by Integer **num_pts**

num_pts <= max_pts

If the Element elt is not of type Interface, then **num_pts** is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Note

A start_pt of one gives the same result as for the previous function.

ID = 183

Get_interface_data(Element elt,Integer i,Real &x,Real &y,Real &z,Integer &f)

Name

Integer Get_interface_data(Element elt,Integer i,Real &x,Real &y,Real &z,Integer &f)

Description

Get the (x,y,z,flag) data for the ith point of the string.

The x value is returned in Real **x**.

The y value is returned in Real **y**.

The z value is returned in Real **z**.

The flag value is returned in Integer **f**.

A function return value of zero indicates the data was successfully returned.

ID = 184

Set_interface_data(Element elt,Real x[],Real y[],Real z[],Integer f[],Integer num_pts)

Name

Integer Set_interface_data(Element elt,Real x[],Real y[],Real z[],Integer f[],Integer num_pts)

Description

Set the (x,y,z,flag) data for the first **num_pts** points of the Interface Element **elt**.

This function allows the user to modify a large number of points of the string in one call.

The maximum number of points that can be set is given by the number of points in the string.

The (x,y,z,flag) values at each string point are given in the Real arrays **x[]**, **y[]**, **z[]** and Integer array **f[]**.

The number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type Interface, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Note

This function can not create new Interface Elements but only modify existing Interface Elements.

ID = 185

Set_interface_data(Element elt,Real x[],Real y[],Real z[],Integer f[],Integer num_pts,Integer start_pt)

Name

Integer Set_interface_data(Element elt,Real x[],Real y[],Real z[],Integer f[],Integer num_pts,Integer start_pt)

Description

For the Interface Element **elt**, set the (x,y,z,flag) data for **num_pts** points starting at point number **start_pt**.

This function allows the user to modify a large number of points of the string in one call starting at point number **start_pt**

rather than point one.

The maximum number of points that can be set is given by the difference between the number of points in the string and the value of **start_pt**.

The (x,y,z,flag) values for the string points are given in the Real arrays **x[]**, **y[]**, **z[]** and Integer array **f[]**.

The number of the first string point to be modified is **start_pt**.

The total number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type Interface, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Notes

(a) A start_pt of one gives the same result as the previous function.

(b) This function can not create new Interface Elements but only modify existing Interface Elements.

ID = 186

Set_interface_data(Element elt,Integer i,Real x,Real y,Real z,Integer flag)

Name

Integer Set_interface_data(Element elt,Integer i,Real x,Real y,Real z,Integer flag)

Description

Set the (x,y,z,flag) data for the **i**th point of the string.

The x value is given in Real **x**.

The y value is given in Real **y**.

The z value is given in Real **z**.

The flag value is given in Integer flag.

A function return value of zero indicates the data was successfully set.

ID = 187

Face String Element

A face string consists of (x,y,z) values at each vertex of the string. The string can be filled with a colour or a hatch pattern

The following functions are used to create new face strings and make inquiries and modifications to existing face strings.

Create_face(Real x[],Real y[],Real z[],Integer num_pts)

Name

Element Create_face(Real x[],Real y[],Real z[],Integer num_pts)

Description

The Element has num_pts points with (x,y,z) values given in the Real arrays **x[], y[]** and **z[]**.

The function return value gives the actual Element created.

If the face string could not be created, then the returned Element will be null.

ID = 1215

Create_face(Integer num_npts)

Name

Element Create_face(Integer num_npts)

Description

Create an Element of type **face** with room for **num_pts** (x,y,z) points.

The actual x, y and z values of the face string are set after the string is created.

If the face string could not be created, then the returned Element will be null.

ID = 1216

Create_face(Integer num_npts,Element seed)

Name

Element Create_face(Integer num_npts,Element seed)

Description

Create an Element of type face with room for **num_pts** (x,y) points, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

The actual x, y and z values of the face string are set after the string is created.

If the face string could not be created, then the returned Element will be null.

ID = 1217

Get_face_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts)

Name

Integer Get_face_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z) data for the first **max_pts** vertices of the face Element elt.

The (x,y,z) values at each string vertex are returned in the Real arrays **x[], y[]** and **z[]**.

The maximum number of vertices that can be returned is given by max_pts (usually the size of the arrays). The vertex data returned starts at the first vertex and goes up to the minimum of max_pts and the number of vertices in the string.

The actual number of vertices returned is returned by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type face, then num_pts is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

ID = 78

Get_face_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts,Integer start_pt)

Name

Integer Get_face_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts,Integer start_pt)

Description

For a face Element **elt**, get the (x,y,z) data for **max_pts** vertices starting at vertex number **start_pt**.

This routine allows the user to return the data from a face string in user specified chunks.

This is necessary if the number of vertices in the string is greater than the size of the arrays available to contain the information.

As in the previous function, the maximum number of points that can be returned is given by **max_pts** (usually the size of the arrays).

However, for this function, the vertex data returned starts at vertex number **start_pt** rather than vertex one.

The (x,y,z) values at each string vertex is returned in the Real arrays **x[], y[]** and **z[]**.

The actual number of vertices returned is given by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type face, then **num_pts** is set to zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Note

A start_pt of one gives the same result as for the previous function.

ID = 79

Set_face_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts)

Name

Integer Set_face_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts)

Description

Set the (x,y,z) data for the first **num_pts** vertices of the face Element **elt**.

This function allows the user to modify a large number of vertices of the string in one call.

The maximum number of vertices that can be set is given by the number of vertices in the string.

The (x,y,z) values for each string vertex is given in the Real arrays **x[], y[]** and **z[]**.

The number of vertices to be set is given by Integer **num_pts**

If the Element **elt** is not of type face, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Note

This function can not create new face Elements but only modify existing face Elements.

ID = 80

Set_face_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts,Integer start_pt)

Name

Integer Set_face_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts,Integer start_pt)

Description

For the face Element **elt**, set the (x,y,z) data for num_pts vertices, starting at vertex number **start_pt**.

This function allows the user to modify a large number of vertices of the string in one call starting at vertex number **start_pt** rather than the first vertex (vertex one).

The maximum number of vertices that can be set is given by the difference between the number of vertices in the string and the value of start_pt.

The (x,y,z) values for the string vertices are given in the Real arrays **x[], y[]** and **z[]**.

The number of the first string vertex to be modified is **start_pt**.

The total number of vertices to be set is given by Integer num_pts

If the Element **elt** is not of type face, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Notes

(a) A start_pt of one gives the same result as the previous function.

(b) This function can not create new face Elements but only modify existing face Elements.

ID = 81

Get_face_data(Element elt,Integer i,Real &x,Real &y,Real &z)

Name

Integer Get_face_data(Element elt,Integer i,Real &x,Real &y,Real &z)

Description

Get the (x,y,z) data for the **i**th vertex of the string.

The x value is returned in Real **x**.
The y value is returned in Real **y**.
The z value is returned in Real **z**.

A function return value of zero indicates the data was successfully returned.

ID = 82

Set_face_data(Element elt,Integer i,Real x,Real y,Real z)

Name

Integer Set_face_data(Element elt,Integer i,Real x,Real y,Real z)

Description

Set the (x,y,z) data for the **i**th vertex of the string.

The x value is given in Real **x**.
The y value is given in Real **y**.
The z value is given in Real **z**.

A function return value of zero indicates the data was successfully set.

ID = 83

Get_face_hatch_distance(Element elt,Real &dist)

Name

Integer Get_face_hatch_distance(Element elt,Real &dist)

Description

Get the distance between the hatch lines for the face string **elt**. The distance is returned as **dist**

A function return value of zero indicates the data was successfully returned.

ID = 1218

Set_face_hatch_distance(Element elt,Real dist)

Name

Integer Set_face_hatch_distance(Element elt,Real dist)

Description

Set the distance between the hatch lines for the face string **elt** to be **dist**

The distance is given in world units.

A function return value of zero indicates the data was successfully set.

ID = 1219

Get_face_hatch_angle(Element elt,Real &ang)

Name

Integer Get_face_hatch_angle(Element elt,Real &ang)

Description

Get the angle of the hatch lines for the face string **elt**. The angle is returned as **ang**.

The angle is given in radians and is measured in the counter-clockwise direction from the x-axis.

A function return value of zero indicates the data was successfully returned.

ID = 1220

Set_face_hatch_angle(Element elt,Real ang)

Name

Integer Set_face_hatch_angle(Element elt,Real ang)

Description

Set the angle of the hatch lines for the face string **elt** to be **ang**

A function return value of zero indicates the data was successfully set.

ID = 1221

Get_face_hatch_colour(Element elt,Integer &colour)

Name

Integer Get_face_hatch_colour(Element elt,Integer &colour)

Description

Get the colour of the solid fill for the face string **elt**. The colour number is returned as **colour**.

A function return value of zero indicates the data was successfully returned.

ID = 1222

Set_face_hatch_colour(Element elt,Integer colour)

Name

Integer Set_face_hatch_colour(Element elt,Integer colour)

Description

Set the colour of the solid fill for the face string **elt** to the colour number **colour**.

A function return value of zero indicates the data was successfully set.

ID = 1223

Get_face_edge_colour(Element elt,Integer &colour)

Name

Integer Get_face_edge_colour(Element elt,Integer &colour)

Description

Get the colour of the edge of the face string **elt**. The colour number is returned as **colour**.

A function return value of zero indicates the data was successfully returned.

ID = 1224

Set_face_edge_colour(Element elt,Integer colour)

Name

Integer Set_face_edge_colour(Element elt,Integer colour)

Description

Set the colour of the edge of the face string **elt** to the colour number **colour**.

A function return value of zero indicates the data was successfully set.

ID = 1225

Get_face_hatch_mode(Element elt,Integer &mode)

Name

Integer Get_face_hatch_mode(Element elt,Integer &mode)

Description

Get the mode of the hatch of the face string **elt**. The value of mode is returned as **mode**.

If the mode is 1, then the hatch pattern is drawn when the face is on a plan view.
If the mode is 0, then the hatch pattern is not drawn when the face is on a plan view.

A function return value of zero indicates the data was successfully returned.

ID = 1226

Set_face_hatch_mode(Element elt,Integer mode)

Name

Integer Set_face_hatch_mode(Element elt,Integer mode)

Description

Set the mode of the hatch pattern of the face string **elt** to the value **mode**.

If the mode is 1, then the hatch pattern is drawn when the face is on a plan view.
If the mode is 0, then the hatch pattern is not drawn when the face is on a plan view.

A function return value of zero indicates the data was successfully set.

ID = 1227

Get_face_fill_mode(Element elt,Integer &mode)

Name

Integer Get_face_fill_mode(Element elt,Integer &mode)

Description

Get the mode of the fill of the face string **elt**. The value of mode is returned as **mode**.

If the mode is 1, then the face is filled with the face colour when the face is on a plan view.
If the mode is 0, then the face is not filled when the face is on a plan view.

A function return value of zero indicates the data was successfully returned.

ID = 1228

Set_face_fill_mode(Element elt,Integer mode)

Name

Integer Set_face_fill_mode(Element elt,Integer mode)

Description

Set the mode of the fill of the face string **elt** to the value **mode**.

If the mode is 1, then the face is filled with the face colour when the face is on a plan view.
If the mode is 0, then the face is not filled when the face is on a plan view.

A function return value of zero indicates the data was successfully set.

ID = 1229

Get_face_edge_mode(Element elt,Integer &mode)

Name

Integer Get_face_edge_mode(Element elt,Integer &mode)

Description

Get the mode of the edge of the face string **elt**. The value of mode is returned as **mode**.

If the mode is 1, then the edge is drawn with the edge colour when the face is on a plan view.
If the mode is 0, then the edge is not drawn when the face is on a plan view.

A function return value of zero indicates the data was successfully returned.

ID = 1230

Set_face_edge_mode(Element elt,Integer mode)

Name

Integer Set_face_edge_mode(Element elt,Integer mode)

Description

Set the mode for displaying the edge of the face string **elt** to the value **mode**.

If the mode is 1, then the edge is drawn with the edge colour when the face is on a plan view.
If the mode is 0, then the edge is not drawn when the face is on a plan view.

A function return value of zero indicates the data was successfully set.

ID = 1231

Plot Frame Element

A Plot Frame string consists of data for producing plan plots.

The following functions are used to create new plot frames and make inquiries and modifications to existing plot frames.

Create_plot_frame(Text name)

Name

Element Create_plot_frame(Text name)

Description

Create an Element of type Plot_Frame.

The function return value gives the actual Element created.

If the plot frame could not be created, then the returned Element will be null.

ID = 607

Get_plot_frame_name(Element elt,Text &name)

Name

Integer Get_plot_frame_name(Element elt,Text &name)

Description

Get the name of the plot frame in Element **elt**.

The name value is returned in Text **name**.

A function return value of zero indicates the data was successfully returned.

ID = 608

Get_plot_frame_scale(Element elt,Real &scale)

Name

Integer Get_plot_frame_scale(Element elt,Real &scale)

Description

Get the scale of the plot frame in Element **elt**.

The scale value is returned in Real **scale**. The value for scale is 1:**scale**.

A function return value of zero indicates the data was successfully returned.

ID = 609

Get_plot_frame_rotation(Element elt,Real &rotation)

Name

Integer Get_plot_frame_rotation(Element elt,Real &rotation)

Description

Get the rotation of the plot frame in Element **elt**.

The name value is returned in Real rotation. The units for **rotation** are radians.

A function return value of zero indicates the data was successfully returned.

ID = 610

Get_plot_frame_origin(Element elt,Real &x,Real &y)

Name

Integer Get_plot_frame_origin(Element elt,Real &x,Real &y)

Description

Get the origin of the plot frame in Element **elt**.

The x origin value is returned in Real **x**.

The y origin value is returned in Real **y**.

A function return value of zero indicates the data was successfully returned.

ID = 611

Get_plot_frame_sheet_size(Element elt,Real &w,Real &h)

Name

Integer Get_plot_frame_sheet_size(Element elt,Real &w,Real &h)

Description

Get the sheet size of the plot frame in Element **elt**.

The width value is returned in Real **w**.

The height value is returned in Real **h**.

A function return value of zero indicates the data was successfully returned.

ID = 612

Get_plot_frame_sheet_size(Element elt,Text &size)

Name

Integer Get_plot_frame_sheet_size(Element elt,Text &size)

Description

Get the sheet size of the plot frame in Element **elt**.

The sheet size is returned in Text **size**.

A function return value of zero indicates the data was successfully returned.

ID = 613

Get_plot_frame_margins(Element elt,Real &l,Real &b,Real &r,Real &t)

Name

Integer Get_plot_frame_margins(Element elt,Real &l,Real &b,Real &r,Real &t)

Description

Get the sheet margins of the plot frame in Element **elt**.

The left margin value is returned in Real **l**.

The bottom margin value is returned in Real **b**.

The right margin value is returned in Real **r**.

The top margin value is returned in Real **t**.

A function return value of zero indicates the data was successfully returned.

ID = 614

Get_plot_frame_text_size(Element elt,Real &text_size)

Name

Integer Get_plot_frame_text_size(Element elt,Real &text_size)

Description

Get the text size of the plot frame in Element **elt**.

The text size is returned in Text **text_size**.

A function return value of zero indicates the data was successfully returned.

ID = 615

Get_plot_frame_draw_border(Element elt,Integer &draw_border)

Name

Integer Get_plot_frame_draw_border(Element elt,Integer &draw_border)

Description

Get the draw border of the plot frame in Element **elt**.

The draw border flag is returned in Integer **draw_border**.

A function return value of zero indicates the data was successfully returned.

ID = 616

Get_plot_frame_draw_viewport(Element elt,Integer &draw_viewport)

Name

Integer Get_plot_frame_draw_viewport(Element elt,Integer &draw_viewport)

Description

Get the draw viewport of the plot frame in Element **elt**.

The draw viewport flag is returned in Integer **draw_viewport**.

A function return value of zero indicates the data was successfully returned.

ID = 617

Get_plot_frame_draw_title_file(Element elt,Integer &draw_title)

Name

Integer Get_plot_frame_draw_title_file(Element elt,Integer &draw_title)

Description

Get the draw title file of the plot frame in Element **elt**.

The draw title file flag is returned in Integer **draw_title**.

A function return value of zero indicates the data was successfully returned.

ID = 618

Get_plot_frame_colour(Element elt,Integer &colour)

Name

Integer Get_plot_frame_colour(Element elt,Integer &colour)

Description

Get the colour of the plot frame in Element **elt**.

The colour value is returned Integer **colour**.

A function return value of zero indicates the data was successfully returned.

ID = 619

Get_plot_frame_textstyle(Element elt,Text &textstyle)

Name

Integer Get_plot_frame_textstyle(Element elt,Text &textstyle)

Description

Get the textstyle of the plot frame in Element **elt**.

The textstyle value is returned in Text **textstyle**.

A function return value of zero indicates the data was successfully returned.

ID = 620

Get_plot_frame_plotter(Element elt,Integer &plotter)

Name

Integer Get_plot_frame_plotter(Element elt,Integer &plotter)

Description

Get the plotter of the plot frame in Element **elt**.

The plotter value is returned in Integer **plotter**.

A function return value of zero indicates the data was successfully returned.

ID = 621

Get_plot_frame_plotter_name(Element elt,Text &plotter_name)

Name

Integer Get_plot_frame_plotter_name(Element elt,Text &plotter_name)

Description

Get the plotter name of the plot frame in Element **elt**.

The plotter name is returned in the Text **plotter_name**.

A function return value of zero indicates the plotter _name was returned successfully.

ID = 686

Get_plot_frame_plot_file(Element elt,Text &plot_file)

Name

Integer Get_plot_frame_plot_file(Element elt,Text &plot_file)

Description

Get the plot file of the plot frame in Element **elt**.

The plot file value is returned in Text **plot_file**.

A function return value of zero indicates the data was successfully returned.

ID = 622

Get_plot_frame_title_1(Element elt,Text &title)

Name

Integer Get_plot_frame_title_1(Element elt,Text &title)

Description

Get the first title line of the plot frame in Element **elt**.

The title line value is returned in Text **title**.

A function return value of zero indicates the data was successfully returned.

ID = 623

Get_plot_frame_title_2(Element elt,Text &title)

Name

Integer Get_plot_frame_title_2(Element elt,Text &title)

Description

Get the second title line of the plot frame in Element **elt**.

The title line value is returned in Text **title**.

A function return value of zero indicates the data was successfully returned.

ID = 624

Get_plot_frame_title_file(Element elt,Text &title_file)

Name

Integer Get_plot_frame_title_file(Element elt,Text &title_file)

Description

Get the title file of the plot frame in Element **elt**.

The title file value is returned in Text **title_file**.

A function return value of zero indicates the data was successfully returned.

ID = 625

Set_plot_frame_name(Element elt,Text name)

Name

Integer Set_plot_frame_name(Element elt,Text name)

Description

Set the name of the plot frame in Element **elt**.

The name value is defined in Text **name**.

A function return value of zero indicates the data was successfully set.

ID = 626

Set_plot_frame_scale(Element elt,Real scale)

Name

Integer Set_plot_frame_scale(Element elt,Real scale)

Description

Set the scale of the plot frame in Element **elt**.

The scale value is defined in Real **scale**.

A function return value of zero indicates the data was successfully set.

ID = 627

Set_plot_frame_rotation(Element elt,Real rotation)

Name

Integer Set_plot_frame_rotation(Element elt,Real rotation)

Description

Set the rotation of the plot frame in Element elt.

The rotation value is defined in Real rotation.

A function return value of zero indicates the data was successfully set.

ID = 628

Set_plot_frame_origin(Element elt,Real x,Real y)

Name

Integer Set_plot_frame_rotation(Element elt,Real rotation)

Description

Set the rotation of the plot frame in Element **elt**

The rotation value is defined in Real **rotation**.

A function return value of zero indicates the data was successfully set.

Set_plot_frame_origin(Element elt,Real x,Real y)

Name

Integer Set_plot_frame_origin(Element elt,Real x,Real y)

Description

Set the origin of the plot frame in Element **elt**.

The x origin value is defined in Real **x**.

The y origin value is defined in Real **y**.

A function return value of zero indicates the data was successfully set.

ID = 629

Set_plot_frame_sheet_size(Element elt,Real w,Real h)

Name

Integer Set_plot_frame_sheet_size(Element elt,Real w,Real h)

Description

Set the sheet size of the plot frame in Element **elt**.

The width value is defined in Real **w**.

The height value is defined in Real **h**.

A function return value of zero indicates the data was successfully set.

ID = 630

Set_plot_frame_sheet_size(Element elt,Text size)

Name

Integer Set_plot_frame_sheet_size(Element elt,Text size)

Description

Set the sheet size of the plot frame in Element **elt**.

The sheet size is defined in Text **size**.

A function return value of zero indicates the data was successfully set.

ID = 631

Set_plot_frame_margins(Element elt,Real l,Real b,Real r,Real t)

Name

Integer Set_plot_frame_margins(Element elt,Real l,Real b,Real r,Real t)

Description

Set the sheet margins of the plot frame in Element **elt**.

The left margin value is defined in Real **l**.

The bottom margin value is defined in Real **b**.

The right margin value is defined in Real **r**.

The top margin value is defined in Real **t**.

A function return value of zero indicates the data was successfully set.

ID = 632

Set_plot_frame_text_size(Element elt,Real text_size)

Name

Integer Set_plot_frame_text_size(Element elt,Real text_size)

Description

Set the text size of the plot frame in Element **elt**.

The text size is defined in Text **text_size**.

A function return value of zero indicates the data was successfully set.

ID = 633

Set_plot_frame_draw_border(Element elt,Integer draw_border)

Name

Integer Set_plot_frame_draw_border(Element elt,Integer draw_border)

Description

Set the draw border of the plot frame in Element **elt**.

The draw border flag is defined in Integer **draw_border**.

A function return value of zero indicates the data was successfully set.

ID = 634

Set_plot_frame_draw_viewport(Element elt,Integer draw_viewport)

Name

Integer Set_plot_frame_draw_viewport(Element elt,Integer draw_viewport)

Description

Set the draw viewport of the plot frame in Element **elt**.

The draw viewport flag is defined in Integer **draw_viewport**.

A function return value of zero indicates the data was successfully set.

ID = 635

Set_plot_frame_draw_title_file(Element elt,Integer draw_title)

Name

Integer Set_plot_frame_draw_title_file(Element elt,Integer draw_title)

Description

Set the draw title file of the plot frame in Element **elt**.

The draw title file flag is defined in Integer **draw_title**.

A function return value of zero indicates the data was successfully set.

ID = 636

Set_plot_frame_colour(Element elt,Integer colour)

Name

Integer Set_plot_frame_colour(Element elt,Integer colour)

Description

Set the colour of the plot frame in Element **elt**.

The colour value is defined Integer **colour**.

A function return value of zero indicates the data was successfully set.

ID = 637

Set_plot_frame_textstyle(Element elt,Text textstyle)

Name

Integer Set_plot_frame_textstyle(Element elt,Text textstyle)

Description

Set the textstyle of the plot frame in Element **elt**.

The textstyle value is defined in Text **textstyle**

A function return value of zero indicates the data was successfully set.

ID = 638

Set_plot_frame_plotter(Element elt,Integer plotter)

Name

Integer Set_plot_frame_plotter(Element elt,Integer plotter)

Description

Set the plotter of the plot frame in Element **elt**.

The plotter value is defined in Integer **plotter**.

A function return value of zero indicates the data was successfully set.

ID = 639

Set_plot_frame_plotter_name(Element elt,Text plotter_name)

Name

Integer Set_plot_frame_plotter_name(Element elt,Text plotter_name)

Description

Set the plotter name of the plot frame in Element **elt**.

The plotter name is given in the Text **plotter_name**.

A function return value of zero indicates the plotter name was successfully set.

ID = 687

Set_plot_frame_plot_file(Element elt,Text plot_file)

Name

Integer Set_plot_frame_plot_file(Element elt,Text plot_file)

Description

Set the plot file of the plot frame in Element **elt**

The plot file value is defined in Text **plot_file**.

A function return value of zero indicates the data was successfully set.

ID = 640

Set_plot_frame_title_1(Element elt,Text title_1)

Name

Integer Set_plot_frame_title_1(Element elt,Text title_1)

Description

Set the first title line of the plot frame in Element **elt**.

The title line value is defined in Text **title_1**.

A function return value of zero indicates the data was successfully set.

ID = 641

Set_plot_frame_title_2(Element elt,Text title_2)

Name

Integer Set_plot_frame_title_2(Element elt,Text title_2)

Description

Set the second title line of the plot frame in Element **elt**.

The title line value is defined in Text **title_2**.

A function return value of zero indicates the data was successfully set.

ID = 642

Set_plot_frame_title_file(Element elt,Text title_file)

Name

Integer Set_plot_frame_title_file(Element elt,Text title_file)

Description

Set the title file of the plot frame in Element **elt**

The title file value is defined in Text **title_file**.

A function return value of zero indicates the data was successfully set.

ID = 643

Strings Replaced by Super Strings

From *12d Model 9* onwards, super strings are replacing many of the earlier string types used in earlier versions of ***12d Model***.

See 2d Strings
See 3d Strings
See 4d Strings
See Pipe Strings
See Polyline Strings

2d Strings

A 2d string consists of (x,y) values at each point of the string and a constant height for the entire string.

The following functions are used to create new 2d strings and make inquiries and modifications to existing 2d strings.

Note: From ***12d Model 9*** onwards, 2d strings have been replaced by Super strings.

For setting up a Super 2d String rather than the superseded 2d string see 2d Super String .

Create_2d(Real x[],Real y[],Real zvalue,Integer num_pts)

Name

Element Create_2d(Real x[],Real y[],Real zvalue,Integer num_pts)

Description

Create an Element of type **2d**.

The Element has **num_pts** points with (x,y) values given in the Real arrays **x[]** and **y[]**.

The height of the string is given by the Real **zvalue**.

The function return value gives the actual Element created.

If the 2d string could not be created, then the returned Element will be null.

ID = 77

Create_2d(Integer num_pts)

Name

Element Create_2d(Integer num_pts)

Description

Create an Element of type **2d** with room for **num_pts** (x,y) points.

The actual x and y values and the height of the 2d string are set after the string is created.

If the 2d string could not be created, then the returned Element will be null.

ID = 448

Create_2d(Integer num_pts,Element seed)

Name

Element Create_2d(Integer num_pts,Element seed)

Description

Create an Element of type 2d with room for **num_pts** (x,y) points, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

The actual x and y values and the height of the 2d string are set after the string is created.

If the 2d string could not be created, then the returned Element will be null.

ID = 665

Get_2d_data(Element elt,Real x[],Real y[],Real &zvalue,Integer max_pts,Integer &num_pts)

Name

Integer Get_2d_data(Element elt,Real x[],Real y[],Real &zvalue,Integer max_pts,Integer &num_pts)

Description

Get the string height and the (x,y) data for the first **max_pts** points of the 2d Element **elt**.

The x and y values at each string point are returned in the Real arrays **x[]** and **y[]**.

The maximum number of points that can be returned is given by max_pts (usually the size of the arrays). The point data returned starts at the first point and goes up to the minimum of max_pts and the number of points in the string.

The actual number of points returned is given by Integer **num_pts**

num_pts <= max_pts

The height of the 2d string is returned in the Real **zvalue**.

If the Element **elt** is not of type 2d, then num_pts is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

ID = 69

Get_2d_data(Element elt,Real x[],Real y[],Real &zvalue,Integer max_pt,Integer &num_pts,Integer start_pt)

Name

Integer Get_2d_data(Element elt,Real x[],Real y[],Real &zvalue,Integer max_pt,Integer &num_pts,Integer start_pt)

Description

For a 2d Element **elt**, get the string height and the (x,y) data for **max_pts** points starting at point number **start_pt**.

This routine allows the user to return the data from a 2d string in user specified chunks. This is necessary if the number of points in the string is greater than the size of the arrays available to contain the information.

As in the previous function, the maximum number of points that can be returned is given by max_pts (usually the size of the arrays).

However, for this function, the point data returned starts at point number **start_pt** rather than point one.

The (x,y) values at each string point are returned in the Real arrays **x[]** and **y[]**.

The actual number of points returned is given by Integer **num_pts**

num_pts <= max_pts

The height of the 2d string is returned in the Real **zvalue**.

If the Element **elt** is not of type 2d, then num_pts is set to zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Note

A **start_pt** of one gives the same result as for the previous function.

ID = 70

Get_2d_data(Element elt,Integer i,Real &x,Real &y)

Name

Integer Get_2d_data(Element elt,Integer i,Real &x,Real &y)

Description

Get the (**x,y**) data for the ith point of the string.

The x value is returned in Real **x**.

The y value is returned in Real **y**.

A function return value of zero indicates the data was successfully returned.

ID = 73

Get_2d_data(Element elt,Real &z)

Name

Integer Get_2d_data(Element elt,Real &z)

Description

Get the height of the 2d string given by Element **elt**.

The height of the string is returned in Real **z**.

A function return value of zero indicates the height was successfully returned.

ID = 75

Set_2d_data(Element elt,Real x[],Real y[],Integer num_pts)

Name

Integer Set_2d_data(Element elt,Real x[],Real y[],Integer num_pts)

Description

Set the (x,y) data for the first **num_pts** points of the 2d Element **elt**.

This function allows the user to modify a large number of points of the string in one call.

The maximum number of points that can be set is given by the number of points in the string.

The (x,y) values at each string point are given in the Real arrays **x[]** and **y[]**.

The number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type 2d, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Note

This function can not create new 2d Elements - it only modifies existing 2d Elements.

ID = 71

Set_2d_data(Element elt,Real x[],Real y[],Integer num_pts,Integer start_pt)

Name

Integer Set_2d_data(Element elt,Real x[],Real y[],Integer num_pts,Integer start_pt)

Description

For the 2d Element **elt**, set the (x,y) data for **num_pts** points starting at point number **start_pt**.

This function allows the user to modify a large number of points of the string in one call starting at point number **start_pt** rather than point one.

The maximum number of points that can be set is given by the difference between the number of points in the string and the value of **start_pt**.

The (x,y) values for the string points are given in the Real arrays **x[]** and **y[]**.

The number of the first string point to be modified is **start_pt**.

The total number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type 2d, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Notes

(a) A start_pt of one gives the same result as the previous function.

(b) This function can not create new 2d Elements but only modify existing 2d Elements.

ID = 72

Set_2d_data(Element elt,Integer i,Real x,Real y)

Name

Integer Set_2d_data(Element elt,Integer i,Real x,Real y)

Description

Set the (x,y) data for the ith point of the string.

The x value is given in Real **x**.

The y value is given in Real **y**.

A function return value of zero indicates the data was successfully set.

ID = 74

Set_2d_data(Element elt,Real z)

Name

Integer Set_2d_data(Element elt,Real z)

Description

Modify the height of the 2d Element **elt**.

The new height is given in the Real **z**.

A function return value of zero indicates the height was successfully set.

ID = 76

3d Strings

A 3d string consists of (x,y,z) values at each point of the string.

The following functions are used to create new 3d strings and make inquiries and modifications to existing 3d strings.

Note: From ***12d Model 9*** onwards, 3d strings have been replaced by Super strings.

For setting up a Super 3d String rather than the superseded 3d string see 3d Super String .

Create_3d(Line line)

Name

Element Create_3d(Line line)

Description

Create an Element of type **3d** from the Line **line**.

The created Element will have two points with co-ordinates equal to the end points of the Line **line**.

The function return value gives the actual Element created.

If the 3d string could not be created, then the returned Element will be null.

ID = 295

Create_3d(Real x[],Real y[],Real z[],Integer num_pts)

Name

Element Create_3d(Real x[],Real y[],Real z[],Integer num_pts)

Description

Create an Element of type **3d**.

The Element has **num_pts** points with (x,y,z) values given in the Real arrays **x[]**, **y[]** and **z[]**.

The function return value gives the actual Element created.

If the 3d string could not be created, then the returned Element will be null.

ID = 84

Create_3d(Integer num_pts)

Name

Element Create_3d(Integer num_pts)

Description

Create an Element of type **3d** with room for **num_pts** (x,y,z) points.

The actual x, y and z values of the 3d string are set after the string is created.

If the 3d string could not be created, then the returned Element will be null.

ID = 449

Create_3d(Integer num_pts,Element seed)

Name

Element Create_3d(Integer num_pts,Element seed)

Description

Create an Element of type 3d with room for **num_pts** (x,y) points, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

The actual x, y and z values of the 3d string are set after the string is created.

If the 3d string could not be created, then the returned Element will be null.

ID = 666

Get_3d_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts)

Name

Integer Get_3d_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z) data for the first **max_pts** points of the 3d Element **elt**.

The (x,y,z) values at each string point are returned in the Real arrays **x[]**, **y[]** and **z[]**.

The maximum number of points that can be returned is given by max_pts (usually the size of the arrays). The point data returned starts at the first point and goes up to the minimum of max_pts and the number of points in the string.

The actual number of points returned is returned by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type 3d, then num_pts is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Get_3d_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts,Integer start_pt)

Name

Integer Get_3d_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts,Integer start_pt)

Description

For a 3d Element **elt**, get the (x,y,z) data for **max_pts** points starting at point number **start_pt**.

This routine allows the user to return the data from a 3d string in user specified chunks. This is necessary if the number of points in the string is greater than the size of the arrays available to contain the information.

As in the previous function, the maximum number of points that can be returned is given by **max_pts** (usually the size of the arrays).

However, for this function, the point data returned starts at point number **start_pt** rather than point one.

The (x,y,z) values at each string point are returned in the Real arrays **x[]**, **y[]** and **z[]**.

The actual number of points returned is given by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type 3d, then **num_pts** is set to zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Note

A start_pt of one gives the same result as for the previous function.

Get_3d_data(Element elt,Integer i, Real &x,Real &y,Real &z)

Name

Integer Get_3d_data(Element elt,Integer i, Real &x,Real &y,Real &z)

Description

Get the (x,y,z) data for the ith point of the string.

The x value is returned in Real **x**.

The y value is returned in Real **y**.

The z value is returned in Real **z**.

A function return value of zero indicates the data was successfully returned.

Set_3d_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts)

Name

Integer Set_3d_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts)

Description

Set the (x,y,z) data for the first **num_pts** points of the 3d Element **elt**.

This function allows the user to modify a large number of points of the string in one call.

The maximum number of points that can be set is given by the number of points in the string.

The (x,y,z) values for each string point are given in the Real arrays **x[]**, **y[]** and **z[]**.

The number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type 3d, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Note

This function can not create new 3d Elements but only modify existing 3d Elements.

ID = 80

Set_3d_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts,Integer start_pt)

Name

Integer Set_3d_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts,Integer start_pt)

Description

For the 3d Element **elt**, set the (x,y,z) data for num_pts points, starting at point number **start_pt**.

This function allows the user to modify a large number of points of the string in one call starting at point number **start_pt** rather than point one.

The maximum number of points that can be set is given by the difference between the number of points in the string and the value of **start_pt**.

The (x,y,z) values for the string points are given in the Real arrays **x[]**, **y[]** and **z[]**.

The number of the first string point to be modified is **start_pt**.

The total number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type 3d, then nothing is modified and the function return value is set to

a non-

zero value.

A function return value of zero indicates the data was successfully set.

Notes

(a) A start_pt of one gives the same result as the previous function.

(b) This function can not create new 3d Elements but only modify existing 3d Elements.

Set_3d_data(Element elt,Integer i,Real x,Real y,Real z)

Name

Integer Set_3d_data(Element elt,Integer i,Real x,Real y,Real z)

Description

Set the (x,y,z) data for the ith point of the string.

The x value is given in Real **x**.

The y value is given in Real **y**.

The z value is given in Real **z**.

A function return value of zero indicates the data was successfully set.

ID = 83

4d Strings

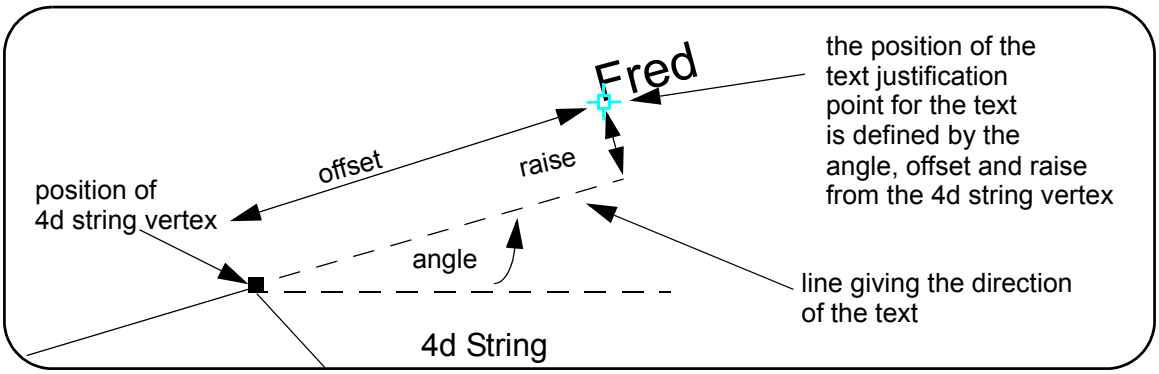
A 4d string consists of (x,y,z,text) values at each **vertex** of the 4d string.

All the texts in a 4d string have the same text parameters and the parameters can be individually set, or all set at once by setting a Textstyle_Data.

The current parameters contained in the Textstyle_Data structure and used for the texts of a 4d String are:

the text itself, text style, colour, height, offset, raise, justification, angle, slant, xfactor, italic, strikeout, underlines, weight, whiteout, border and a name.

The parameters are described in the section Textstyle Data



The following functions are used to create new 4d strings and make inquiries and modifications to existing 4d strings.

Note: From ***12d Model 9*** onwards, 4d strings have been replaced by Super strings.

For setting up a Super 4d String rather than the superseded 4d string see 4d Super String .

Create_4d(Real x[],Real y[],Real z[],Text t[],Integer num_pts)

Name

Element Create_4d(Real x[],Real y[],Real z[],Text t[],Integer num_pts)

Description

Create an Element of type **4d**. The Element has num_pts points with (x,y,z,text) values given in the Real arrays **x[]**, **y[]**, **z[]** and Text array **t[]**.

The function return value gives the actual Element created.

If the 4d string could not be created, then the returned Element will be null.

ID = 91

Create_4d(Integer num_pts)

Name

Element Create_4d(Integer num_pts)

Description

Create an Element of type **4d** with room for **num_pts** (x,y,z,text) points.

The actual x, y, z and text values of the 4d string are set after the string is created.

If the 4d string could not be created, then the returned Element will be null.

ID = 450

Create_4d(Integer num_pts,Element seed)

Name

Element Create_4d(Integer num_pts,Element seed)

Description

Create an Element of type 4d with room for **num_pts** (x,y) points, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

The actual x, y, z and text values of the 4d string are set after the string is created.

If the 4d string could not be created, then the returned Element will be null.

ID = 667

Set_4d_data(Element elt,Real x[],Real y[],Real z[], Text t[],Integer num_pts)

Name

Integer Set_4d_data(Element elt,Real x[],Real y[],Real z[],Text t[],Integer num_pts)

Description

Set the (x,y,z,text) data for the first **num_pts** points of the 4d Element **elt**.

This function allows the user to modify a large number of points of the string in one call.

The maximum number of points that can be set is given by the number of points in the string.

The (x,y,z,text) values at each string point are given in the Real arrays **x[]**, **y[]**, **z[]** and Text array **t[]**.

The number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type 4d, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Note

This function can not create new 4d Elements but only modify existing 4d Elements.

ID = 87

Set_4d_data(Element elt,Real x[],Real y[],Real z[],Text t[],Integer num_pts,Integer start_pt)

Name

Integer Set_4d_data(Element elt,Real x[],Real y[],Real z[],Text t[],Integer num_pts,Integer start_pt)

Description

For the 4d Element **elt**, set the (x,y,z,text) data for **num_pts** points, starting at point number **start_pt**.

This function allows the user to modify a large number of points of the string in one call starting at point number **start_pt** rather than point one.

The maximum number of points that can be set is given by the difference between the number of points in the string and the value of **start_pt**.

The (x,y,z,text) values for the string points are given in the Real arrays **x[]**, **y[]**, **z[]** and Text array **t[]**.

The number of the first string point to be modified is start_pt.

The total number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type 4d, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Notes

(a) A start_pt of one gives the same result as the previous function.

(b) This function can not create new 4d Elements but only modify existing 4d Elements.

ID = 88

Set_4d_data(Element elt,Integer i,Real x,Real y,Real z,Text t)

Name

Integer Set_4d_data(Element elt,Integer i,Real x,Real y,Real z,Text t)

Description

Set the (x,y,z,text) data for the **i**th point of the string.

The x value is given in Real **x**.

The y value is given in Real **y**.

The z value is given in Real **z**.

The text value is given in Text **t**.

A function return value of zero indicates the data was successfully set.

ID = 90

Get_4d_data(Element elt,Real x[],Real y[],Real z[],Text t[],Integer max_pts,Integer &num_pts)

Name

Integer Get_4d_data(Element elt,Real x[],Real y[],Real z[],Text t[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z,text) data for the first **max_pts** points of the 4d Element elt.

The (x,y,z,text) values at each string point are returned in the Real arrays **x[]**, **y[]**, **z[]** and Text array **t[]**.

The maximum number of points that can be returned is given by **max_pts** (usually the size of the arrays). The point data returned starts at the first point and goes up to the minimum of max_pts and the number of points in the string.

The actual number of points returned is returned by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type 4d, then **num_pts** is set to zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

ID = 85

Get_4d_data(Element elt,Real x[],Real y[],Real z[],Text t[],Integer max_pts,Integer &num_pts,Integer start_pt)

Name

Integer Get_4d_data(Element elt,Real x[],Real y[],Real z[],Text t[],Integer max_pts,Integer &num_pts,Integer start_pt)

Description

For a 4d Element **elt**, get the (x,y,z,text) data for **max_pts** points starting at point number **start_pt**.

This routine allows the user to return the data from a 4d string in user specified chunks. This is necessary if the number of points in the string is greater than the size of the arrays available to contain the information.

As in the previous function, the maximum number of points that can be returned is given by **max_pts** (usually the size of the arrays).

However, for this function, the point data returned starts at point number **start_pt** rather than point one.

The (x,y,z,text) values at each string point are returned in the Real arrays **x[]**, **y[]**, **z[]** and Text array **t[]**.

The actual number of points returned is given by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type 4d, then **num_pts** is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Note

A start_pt of one gives the same result as for the previous function.

ID = 86

Get_4d_data(Element elt,Integer i,Real &x,Real &y,Real &z,Text &t)

Name

Integer Get_4d_data(Element elt,Integer i,Real &x,Real &y,Real &z,Text &t)

Description

Get the (x,y,z,text) data for the **i**th point of the string.

The x value is returned in Real **x**.

The y value is returned in Real **y**.

The z value is returned in Real **z**.

The text value is returned in Text **t**.

A function return value of zero indicates the data was successfully returned.

ID = 89

Set_4d_textstyle_data(Element elt,Textstyle_Data d)

Name

Integer Set_4d_textstyle_data(Element elt,Textstyle_Data d)

Description

For the Element **elt** of type **4d,** set the Textstyle_Data to be **d**.

Setting a Textstyle_Data means that all the individual values that are contained in the Textstyle_Data are set rather than having to set each one individually.

LJG? if the value is blank in the Textstyle_Data and the value is already set for the 4d string, is the value left alone?

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates the Textstyle_Data was successfully set.

ID = 1667

Get_4d_textstyle_data(Element elt,Textstyle_Data &d)

Name

Integer Get_4d_textstyle_data(Element elt,Textstyle_Data &d)

Description

For the Element **elt** of type **4d,** get the Textstyle_Data for the string and return it as **d**.

LJG? if a value is not set in the 4d string, what does it return?

A non-zero function return value is returned if **elt** is not of type **4d**.

A function return value of zero indicates the Textstyle_Data was successfully returned.

ID = 1668

Set_4d_units(Element elt,Integer units_mode)

Name

Integer Set_4d_units(Element elt,Integer units_mode)

Description

Set the units used for the text parameters of the 4d Element **elt**.

The mode is given as Integer **units_mode**.

For the values of **units_mode**, see Textstyle Data .

A function return value of zero indicates the data was successfully set.

ID = 447

Get_4d_units(Element elt,Integer &units_mode)

Name

Integer Get_4d_units(Element elt,Integer &units_mode)

Description

Get the units used for the text parameters of the 4d Element **elt**.

The mode is returned as Integer **units_mode**.

For the values of **units_mode**, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 441

Set_4d_size(Element elt,Real size)

Name

Integer Set_4d_size(Element elt,Real size)

Description

Set the size of the characters of the 4d text of the Element **elt**.

The text size is given as Real **size**.

A function return value of zero indicates the data was successfully set.

ID = 442

Get_4d_size(Element elt,Real &size)

Name

Integer Get_4d_size(Element elt,Real &size)

Description

Get the size of the characters of the 4d text of the Element **elt**.

The text size is returned as Real **size**.

A function return value of zero indicates the data was successfully returned.

ID = 436

Set_4d_justify(Element elt,Integer justify)

Name

Integer Set_4d_justify(Element elt,Integer justify)

Description

Set the justification used for the text parameters of the 4d Element **elt**.

The justification is given as Integer **justify**.

*For the values of **justify** and their meaning, see* Textstyle Data .

A function return vale of zero indicates the data was successfully set.

ID = 446

Get_4d_justify(Element elt,Integer &justify)

Name

Integer Get_4d_justify(Element elt,Integer &justify)

Description

Get the justification used for the text parameters of the 4d Element **elt**.

The justification is returned as Integer **justify**.

*For the values of **justify** and their meaning, see* Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 440

Set_4d_angle(Element elt,Real angle)

Name

Integer Set_4d_angle(Element elt,Real angle)

Description

Set the angle of rotation (in radians) about each 4d point (x,y) of the text of the 4d Element **elt**.

The angle is given as Real **angle**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully set.

ID = 445

Get_4d_angle(Element elt,Real &angle)

Name

Integer Get_4d_angle(Element elt,Real &angle)

Description

Get the angle of rotation (in radians) about each 4d point (x,y) of the text of the 4d Element **elt**. **angle** is measured in an anti-clockwise direction from the horizontal axis.

The angle is returned as Real **angle**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 439

Set_4d_offset(Element elt,Real offset)

Name

Integer Set_4d_offset(Element elt,Real offset)

Description

Set the offset distance of the text to be used for each 4d point (x,y) for the 4d Element **elt**.

The offset is returned as Real **offset**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 443

Get_4d_offset(Element elt,Real &offset)

Name

Integer Get_4d_offset(Element elt,Real &offset)

Description

Get the offset distance of the text to be used for each 4d point (x,y) for the 4d Element **elt**.

The offset is returned as Real **offset**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 437

Set_4d_rise(Element elt,Real rise)

Name

Integer Set_4d_rise(Element elt,Real rise)

Description

Set the rise distance of the text to be used for each 4d point (x,y) for the 4d Element **elt**.

The rise is given as Real **rise**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully set.

ID = 444

Get_4d_rise(Element elt,Real &rise)

Name

Integer Get_4d_rise(Element elt,Real &rise)

Description

Get the rise distance of the text to be used for each 4d point (x,y) for the 4d Element **elt**.

The rise is returned as Real **rise**.

For a diagram, see Textstyle Data .

A function return value of zero indicates the data was successfully returned.

ID = 438

Set_4d_height(Element elt,Real height)

Name

Integer Set_4d_height(Element elt,Real height)

Description

Set the height of the characters of the 4d text of the Element **elt**.

The text height is given as Real **height**.

A function return value of zero indicates the data was successfully set.

ID = 648

Get_4d_height(Element elt,Real &height)

Name

Integer Get_4d_height(Element elt,Real &height)

Description

Get the height of the characters of the 4d text of the Element **elt**.

The text height is returned as Real **height**.

A function return value of zero indicates the data was successfully returned.

ID = 644

Set_4d_slant(Element elt,Real slant)

Name

Integer Set_4d_slant(Element elt,Real slant)

Description

Set the slant of the characters of the 4d text of the Element **elt**.

The text slant is given as Real **slant**.

A function return value of zero indicates the data was successfully set.

ID = 649

Get_4d_slant(Element elt,Real &slant)

Name

Integer Get_4d_slant(Element elt,Real &slant)

Description

Get the slant of the characters of the 4d text of the Element **elt**.

The text slant is returned as Real **slant**.

A function return value of zero indicates the data was successfully returned.

ID = 645

Set_4d_x_factor(Element elt,Real xfact)

Name

Integer Set_4d_x_factor(Element elt,Real xfact)

Description

Set the x factor of the characters of the 4d text of the Element **elt**.

The text x factor is given as Real **xfact**.

A function return value of zero indicates the data was successfully set.

ID = 650

Get_4d_x_factor(Element elt,Real &xfact)

Name

Integer Get_4d_x_factor(Element elt,Real &xfact)

Description

Get the x factor of the characters of the 4d text of the Element **elt**.

The text x factor is returned as Real **xfact**.

A function return value of zero indicates the data was successfully returned.

ID = 646

Set_4d_style(Element elt,Text style)

Name

Integer Set_4d_style(Element elt,Text style)

Description

Set the style of the characters of the 4d text of the Element **elt**.

The text style is given as Text **style**.

A function return value of zero indicates the data was successfully set.

ID = 651

Get_4d_style(Element elt,Text &style)

Name

Integer Get_4d_style(Element elt,Text &style)

Description

Get the style of the characters of the 4d text of the Element **elt**.

The text style is returned as Text **style**.

A function return value of zero indicates the data was successfully returned.

ID = 647

Set_4d_ttf_underline(Element elt,Integer underline)

Name

Integer Set_4d_ttf_underline(Element elt,Integer underline)

Description

For the Element **elt** of type **4d,** set the underline state to **underline**.

If **underline** = 1, then for a true type font the text will be underlined.

If **underline** = 0, then text will not be underlined.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates underlined was successfully set.

ID = 2588

Get_4d_ttf_underline(Element elt,Integer &underline)

Name

Integer Get_4d_ttf_underline(Element elt,Integer &underline)

Description

For the Element **elt** of type **4d,** get the underline state and return it in **underline**.

If **underline** = 1, then for a true type font the text will be underlined.
If **underline** = 0, then text will not be underlined.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates underlined was successfully returned.

ID = 2584

Set_4d_ttf_strikeout(Element elt,Integer strikeout)

Name

Integer Set_4d_ttf_strikeout(Element elt,Integer strikeout)

Description

For the Element **elt** of type **4d,** set the strikeout state to **strikeout**.

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates strikeout was successfully set.

ID = 2589

Get_4d_ttf_strikeout(Element elt,Integer &strikeout)

Name

Integer Get_4d_ttf_strikeout(Element elt,Integer &strikeout)

Description

For the Element **elt** of type **4d,** get the strikeout state and return it in **strikeout**.

For a diagram, see Textstyle Data .

If **strikeout** = 1, then for a true type font the text will be strikeout.
If **strikeout** = 0, then text will not be strikeout.

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates strikeout was successfully returned.

ID = 2585

Set_4d_ttf_weight(Element elt,Integer weight)

Name

Integer Set_4d_ttf_weight(Element elt,Integer weight)

Description

For the Element **elt** of type **4d,** set the font weight to **weight**.

For the list of allowable weights, go to Allowable Weights

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates weight was successfully set.

ID = 2591

Get_4d_ttf_weight(Element elt,Integer &weight)

Name

Integer Get_4d_ttf_weight(Element elt,Integer &weight)

Description

For the Element **elt** of type **4d,** get the font weight and return it in **weight**.

Allowable Weights

The allowable numbers for weight are:

0 = FW_DONTCARE
100 = FW_THIN
200 = FW_EXTRALIGHT
300 = FW_LIGHT
400 = FW_NORMAL
500 = FW_MEDIUM
600 = FW_SEMIBOLD
700 = FW_BOLD
800 = FW_EXTRABOLD

900 = FW_HEAVY

Note that in the distributed file *set_ups.h* these are defined as:

```
#define FW_DONTCARE      0
#define FW_THIN          100
#define FW_EXTRALIGHT    200
#define FW_LIGHT         300
#define FW_NORMAL        400
#define FW_MEDIUM        500
#define FW_SEMIBOLD      600
#define FW_BOLD          700
#define FW_EXTRABOLD     800
#define FW_HEAVY         900
#define FW_ULTRALIGHT    FW_EXTRALIGHT
#define FW_REGULAR       FW_NORMAL
#define FW_DEMIBOLD      FW_SEMIBOLD
#define FW_ULTRABOLD     FW_EXTRABOLD
#define FW_BLACK         FW_HEAVY
```

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates weight was successfully returned.

ID = 2587

Set_4d_ttf_italic(Element elt,Integer italic)

Name

Integer Set_4d_ttf_italic(Element elt,Integer italic)

Description

For the Element **elt** of type **4d,** set the italic state to **italic**.

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates italic was successfully set.

ID = 2590

Get_4d_ttf_italic(Element elt,Integer &italic)

Name

Integer Get_4d_ttf_italic(Element elt,Integer &italic)

Description

For the Element **elt** of type **4d,** get the italic state and return it in **italic**.

If **italic** = 1, then for a true type font the text will be italic.
If **italic** = 0, then text will not be italic.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates italic was successfully returned.

ID = 2586

Set_4d_ttf_outline(Element elt,Integer outline)

Name

Integer Set_4d_ttf_outline(Element elt,Integer outline)

Description

For the Element **elt** of type **4d,** set the outline state to **outline**.

If **outline** = 1, then for a true type font the text will be only shown in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates ***outline*** was successfully set.

ID = 2770

Get_4d_ttf_outline(Element elt,Integer &outline)

Name

Integer Get_4d_ttf_outline(Element elt,Integer &outline)

Description

For the Element **elt** of type **4d,** get the outline state and return it in **outline**.

If **outline** = 1, then for a true type font the text will be shown only in outline.
If **outline** = 0, then text will not be only shown in outline.

For a diagram, see Textstyle Data .

A non-zero function return value is returned if **elt** is not of type **4d.**

A function return value of zero indicates ***outline*** was successfully returned.

ID = 2769

Set_4d_whiteout(Element element,Integer colour)

Name

Integer Set_4d_whiteout(Element element,Integer colour)

Description

For the 4d Element **elt**, set the colour number of the colour used for the whiteout box around vertex text, to be **colour**.

If no text whiteout is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully set.

ID = 2750

Get_4d_whiteout(Element element,Integer &colour)

Name

Integer Get_4d_whiteout(Element element,Integer &colour)

Description

For the 4d Element **elt**, get the colour number that is used for the whiteout box around vertex text. The whiteout colour is returned as Integer **colour**.

NO_COLOUR is the returned as the colour number if whiteout is not being used.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully returned.

ID = 2749

Set_4d_border(Element element,Integer colour)

Name

Integer Set_4d_border(Element element,Integer colour)

Description

For the 4d Element **elt**, set the colour number of the colour used for the border of the whiteout box around vertex text, to be **colour**.

If no whiteout border is required, then set the colour number to NO_COLOUR.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff).

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully set.

ID = 2760

Get_4d_border(Element element,Integer &colour)

Name

Integer Get_4d_border(Element element,Integer &colour)

Description

For the 4d Element **elt**, get the colour number that is used for the border of the whiteout box around vertex text. The whiteout border colour is returned as Integer **colour**.

NO_COLOUR is the returned as the colour number if there is no whiteout border.

Note: The colour number for "view colour" is VIEW_COLOUR (or **2147483647** - that is 0x7fffffff)

For a diagram, see Textstyle Data .

A function return value of zero indicates the colour number was successfully returned.

ID = 2759

Pipe Strings

A pipe string consists of (x,y,z) values at each point of the string and a diameter for the entire string.

The following functions are used to create new pipe strings and make inquiries and modifications to existing pipe strings.

Note: From ***12d Model 9*** onwards, pipe strings have been replaced by Super strings.

Create_pipe(Real x[],Real y[],Real z[],Integer num_pts)

Name

Element Create_pipe(Real x[],Real y[],Real z[],Integer num_pts)

Description

Create an Element of type **pipe**.

The Element has num_pts points with (x,y,z) values given in the Real arrays **x[], y[]** and **z[]**.

The function return value gives the actual Element created.

If the pipe string could not be created, then the returned Element will be null.

ID = 676

Create_pipe(Integer num_pts)

Name

Element Create_pipe(Integer num_pts)

Description

Create an Element of type **pipe** with room for **num_pts** (x,y,z) points.

The actual x, y and z values of the pipe string are set after the string is created.

If the pipe string could not be created, then the returned Element will be null.

ID = 677

Create_pipe(Integer num_pts,Element seed)

Name

Element Create_pipe(Integer num_pts,Element seed)

Description

Create an Element of type pipe with room for **num_pts** (x,y) points, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

The actual x, y and z values of the pipe string are set after the string is created.

If the pipe string could not be created, then the returned Element will be null.

ID = 678

Get_pipe_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts)

Name

Integer Get_pipe_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z) data for the first **max_pts** points of the pipe Element elt.

The (x,y,z) values at each string point are returned in the Real arrays **x[], y[]** and **z[]**.

The maximum number of points that can be returned is given by max_pts (usually the size of the arrays). The point data returned starts at the first point and goes up to the minimum of max_pts and the number of points in the string.

The actual number of points returned is returned by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type pipe, then num_pts is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Set_pipe_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts)

Name

Integer Set_pipe_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts)

Description

Set the (x,y,z) data for the first **num_pts** points of the pipe Element **elt**.

This function allows the user to modify a large number of points of the string in one call.

The maximum number of points that can be set is given by the number of points in the string.

The (x,y,z) values for each string point are given in the Real arrays **x[], y[]** and **z[]**.

The number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type pipe, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Note

This function can not create new pipe Elements but only modify existing pipe Elements.

ID = 80

Get_pipe_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts,Integer start_pt)

Name

Integer Get_pipe_data(Element elt,Real x[],Real y[],Real z[],Integer max_pts,Integer &num_pts,Integer start_pt)

Description

For a pipe Element **elt**, get the (x,y,z) data for **max_pts** points starting at point number **start_pt**.

This routine allows the user to return the data from a pipe string in user specified chunks.

This is necessary if the number of points in the string is greater than the size of the arrays available to contain the information.

As in the previous function, the maximum number of points that can be returned is given by **max_pts** (usually the size of the arrays).

However, for this function, the point data returned starts at point number **start_pt** rather than point one.

The (x,y,z) values at each string point are returned in the Real arrays **x[], y[]** and **z[]**.

The actual number of points returned is given by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type pipe, then **num_pts** is set to zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Note

A start_pt of one gives the same result as for the previous function.

Set_pipe_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts,Integer start_pt)

Name

Integer Set_pipe_data(Element elt,Real x[],Real y[],Real z[],Integer num_pts,Integer start_pt)

Description

For the pipe Element **elt**, set the (x,y,z) data for num_pts points, starting at point number **start_pt**.

This function allows the user to modify a large number of points of the string in one call starting at point number **start_pt** rather than point one.

The maximum number of points that can be set is given by the difference between the number of points in the string and the value of start_pt.

The (x,y,z) values for the string points are given in the Real arrays **x[], y[]** and **z[]**.

The number of the first string point to be modified is **start_pt**.

The total number of points to be set is given by Integer num_pts

If the Element **elt** is not of type pipe, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Notes

(a) A start_pt of one gives the same result as the previous function.

(b) This function can not create new pipe Elements but only modify existing pipe Elements.

Get_pipe_data(Element elt,Integer i, Real &x,Real &y,Real &z)

Name

Integer Get_pipe_data(Element elt,Integer i, Real &x,Real &y,Real &z)

Description

Get the (x,y,z) data for the **i**th point of the string.

The x value is returned in Real **x**.

The y value is returned in Real **y**.

The z value is returned in Real **z**.

A function return value of zero indicates the data was successfully returned.

Set_pipe_data(Element elt,Integer i,Real x,Real y,Real z)

Name

Integer Set_pipe_data(Element elt,Integer i,Real x,Real y,Real z)

Description

Set the (x,y,z) data for the **i**th point of the string.

The x value is given in Real **x**.

The y value is given in Real **y**.

The z value is given in Real **z**.

A function return value of zero indicates the data was successfully set.

ID = 83

Get_pipe_diameter(Element elt,Real &diameter)

Name

Integer Get_pipe_diameter(Element elt,Real &diameter)

Description

Get the pipe diameter of the string Element **elt**.

The pipe diameter is returned in Real **diameter**.

A function return value of zero indicates the data was successfully returned.

ID = 681

Set_pipe_diameter(Element elt,Real diameter)

Name

Integer Set_pipe_diameter(Element elt,Real diameter)

Description

Set the pipe diameter of the string Element **elt**.

The pipe diameter is given as Real **diameter**.

A function return value of zero indicates the data was successfully set.

ID = 679

Get_pipe_justify(Element elt,Integer &justify)

Name

Integer Get_pipe_justify(Element elt,Integer &justify)

Description

Get the justification used for the pipe Element **elt**

The justification is returned as Integer **justify**.

A function return value of zero indicates the data was successfully returned.

ID = 682

Set_pipe_justify(Element elt,Integer justify)

Name

Integer Set_pipe_justify(Element elt,Integer justify)

Description

Set the justification used for the text parameter of the pipe Element **elt**.

The justification is given as Integer **justify**.

A function return value of zero indicates the data was successfully set.

ID = 680

Polyline Strings

A polyline string consists of (x,y,z,radius,bulge) values at each point of the string.

For a given point, (**x,y,z**) defines the co-ordinates of the point, and (**radius,bulge**) defines an arc of radius **radius** between the point and the and the next point.

The sign of **radius** defines which side of the line joining the consecutive points that the arc is on (positive - on the left; negative - on the right) and **bulge** specifies whether the arc is a minor or major arc (0 for a minor arc < 180 degrees; 1 for a major arc > 180 degrees). The minor/major value is given in Integer bulge.

The following functions are used to create new polyline strings and make inquiries and modifications to existing polyline strings.

Note: From ***12d Model 9*** onwards, Polyline strings have been replaced by Super strings.

For setting up a Super Polyline String rather than the superseded polyline string see 3d Super String .

Create_polyline(Real x[],Real y[],Real z[],Real r[],Integer bulge[],Integer num_pts)

Name

Element Create_polyline(Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts)

Description

Create an Element of type **polyline**.

The Element has **num_pts** points with (x,y,z) values given in the Real arrays **x[]**, **y[]** and **z[]**, and arcs between consecutive points given in the Real array **r[]** and the Integer array **bulge[]**.

The radius of the arc between the nth and the n+1 point is given by **r[n]** and the arc is on the right of the line joining the nth and n+1 point if **r[n]** is positive, and on the left if **r[n]** is negative. Hence the absolute value of **r[n]** gives the radius of the curve between the nth and n+1 point and the sign of **r[n]** defines what side the curve lies on.

The value of **bulge[n]** defines whether the arc is a minor or major arc. A value of 0 denotes a minor arc and 1 a major arc.

The function return value gives the actual Element created.

If the polyline string could not be created, then the returned Element will be null.

ID = 481

Create_polyline(Integer num_pts)

Name

Element Create_polyline(Integer num_pts)

Description

Create an Element of type **Polyline** with room for **num_pts** (x,y,z,r,bulge) points.

The actual x, y, z, r, and bulge values of the polyline string are set after the string is created.

If the polyline string could not be created, then the returned Element will be null.

ID = 482

Create_polyline(Integer num_pts,Element seed)

Name

Element Create_polyline(Integer num_pts,Element seed)

Description

Create an Element of type **Polyline** with room for **num_pts** (x,y,z,r,bulge) points, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

The actual x, y, z, r, and bulge values of the polyline string are set after the string is created.

If the polyline string could not be created, then the returned Element will be null.

ID = 669

Create_polyline(Segment seg)

Name

Element Create_polyline(Segment seg)

Description

Create an Element of type **Polyline** from the **Segment** seg. The segment may be a Line, or Arc.

The created Element will have two points with co-ordinates equal to the end points of the Segment seg.

The function return value gives the actual Element created.

If the polyline string could not be created, then the returned Element will be null.

ID = 554

Get_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer max_pts,Integer &num_pts)

Name

Integer Get_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer b[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z,r,b) data for the first **max_pts** points of the polyline Element **elt**.

The (x,y,z,r,b) values at each string point are returned in the Real arrays **x[], y[], z[], r[]** and **b[]**.

The maximum number of points that can be returned is given by max_pts (usually the size of the arrays). The point data returned starts at the first point and goes up to the minimum of max_pts and the number of points in the string.

The actual number of points returned is returned by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type Polyline, then **num_pts** is returned as zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

ID = 483

Get_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer max_pts,Integer &num_pts,Integer start_pt)

Name

Integer Get_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer max_pts,Integer &num_pts,Integer start_pt)

Description

For a polyline Element **elt**, get the (x,y,z,r,f) data for **max_pts** points starting at point number **start_pt**.

This routine allows the user to return the data from a polyline string in user specified chunks. This is necessary if the number of points in the string is greater than the size of the arrays available to contain the information.

As in the previous function, the maximum number of points that can be returned is given by **max_pts** (usually the size of the arrays).

However, for this function, the point data returned starts at point number **start_pt** rather than point one.

The (x,y,z,r,f) values at each string point are returned in the Real arrays **x[], y[], z[], r[]** and **f[]**.

The actual number of points returned is given by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** is not of type Polyline, then **num_pts** is set to zero and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully returned.

Note

A **start_pt** of one gives the same result as for the previous function.

ID = 484

Get_polyline_data(Element elt,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &f)

Name

Integer Get_polyline_data(Element elt,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &f)

Description

Get the (x,y,z,r,f) data for the ith point of the **Polyline** Element **elt**.

The x value is returned in Real **x**.

The y value is returned in Real **y**.

The z value is returned in Real **z**.

The radius value is returned in Real **r**.

The minor/major value is returned in Integer **f**.

A function return value of zero indicates the data was successfully returned.

ID = 485

Set_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts)

Name

Integer Set_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts)

Description

Set the (x,y,z,r,f) data for the first **num_pts** points of the polyline Element **elt**.

This function allows the user to modify a large number of points of the string in one call.

The maximum number of points that can be set is given by the number of points in the string.

The (x,y,z,r,f) values for each string point are given in the Real arrays **x[], y[], z[], r[]** and **f[]**.

The number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type Polyline, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Note

This function can not create new Polyline Elements but only modify existing Polyline Elements.

ID = 486

Set_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts,Integer start_pt)

Name

Integer Set_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts,Integer start_pt)

Description

For the polyline Element **elt**, set the (x,y,z,r,f) data for **num_pts** points, starting at point number **start_pt**.

This function allows the user to modify a large number of points of the string in one call starting at point number **start_pt** rather than point one.

The maximum number of points that can be set is given by the difference between the number of points in the string and the value of **start_pt**.

The (x,y,z,r,f) values for the string points are given in the Real arrays **x[], y[], z[], r[]**

and **f[]**.

The number of the first string point to be modified is **start_pt**.

The total number of points to be set is given by Integer **num_pts**

If the Element **elt** is not of type **Polyline**, then nothing is modified and the function return value is set to a non-zero value.

A function return value of zero indicates the data was successfully set.

Notes

(a) A **start_pt** of one gives the same result as the previous function.

(b) This function can not create new Polyline Elements but only modify existing Polyline Elements.

ID = 487

Set_polyline_data(Element elt,Integer i,Real x,Real y,Real z,Real r,Integer f)

Name

Integer Set_polyline_data(Element elt,Integer i,Real x,Real y,Real z,Real r,Integer f)

Description

Set the (x,y,z,r,f) data for the ith point of the string.

The x value is given in Real **x**.
The y value is given in Real **y**.
The z value is given in Real **z**.
The radius value is given in Real **r**.
The minor/major value is given in Integer **f**.

A function return value of zero indicates the data was successfully set.

ID = 488

Alignment String Element

An Alignment string holds both the horizontal and vertical information needed in defining entities such as the centre line of a road.

Horizontal intersection points (hips), arcs and spirals are used to define the plan geometry.

Vertical intersection points (vips) and parabolic and circular curves are used to define the vertical geometry.

The process to define an Alignment string is

(a) create an Alignment Element

(b) add the horizontal geometry

(c) perform a Calc_alignment on the string

(d) add the vertical geometry

(e) perform a Calc_alignment

For an existing Alignment string, there are functions to get the positions of all critical points (such as horizontal and vertical tangent points, spiral points, curve centres) for the string.

The functions used to create new Alignment strings and make inquiries and modifications to existing Alignment strings now follow.

Note: From ***12d Model 9*** onwards, Alignment strings have been replaced by Super Alignment strings.

Element Create_align()

Name

Element Create_align()

Description

Create an Element of type **Alignment**.

The function return value gives the actual Element created.

If the Alignment string could not be created, then the returned Element will be null.

ID = 92

Create_align(Element seed)

Name

Element Create_align(Element seed)

Description

Create an Element of type Alignment, and set the colour, name, style etc. of the new string to be the same as those from the Element **seed**.

If the alignment string could not be created, then the returned Element will be null.

ID = 670

Append_hip(Element elt,Real x,Real y)

Name

Integer Append_hip(Element elt,Real x,Real y)

Description

Append a horizontal intersection point (hip) with plan co-ordinates (**x**,**y**) to the Element elt

. The radius and spiral lengths are set to zero.

The order in which the hips are appended is taken as the order of the hips in the Alignment string.

The hips must be appended in order of increasing chainage along the Alignment string.

Append_hip is used to place the first hip as well as the subsequent hips.

A function return value of zero indicates that the hip was successfully appended.

ID = 93

Append_hip(Element elt,Real x,Real y,Real rad)

Name

Integer Append_hip(Element elt,Real x,Real y,Real rad)

Description

Append a horizontal intersection point (hip) with plan co-ordinates (**x**,**y**) and curve radius **rad** to the Element **elt**. The spiral lengths are set to zero.

A zero curve radius indicates that no curve is present.

A function return value of zero indicates that the hip was successfully appended.

ID = 94

Append_hip(Element elt,Real x,Real y,Real rad,Real left_spiral,Real right_spiral)

Name

Integer Append_hip(Element elt,Real x,Real y,Real rad,Real left_spiral,Real right_spiral)

Description

Append to the Element **elt** a horizontal intersection point (hip) with co-ordinates (**x**,**y**), curve radius **rad** and left and right spirals of length **left_spiral** and **right_spiral** respectively.

A zero curve radius indicates that no curve is present.

A zero spiral length indicates that a spiral is not present.

A function return value of zero indicates that the hip was successfully appended.

ID = 95

Get_hip_points(Element elt,Integer &num_pts)

Name

Integer Get_hip_points(Element elt,Integer &num_pts)

Description

Get the number of hips, **num_pts**, in the Alignment Element **elt**.

A function return value of zero indicates the number of hip points was successfully returned.

ID = 100

Get_hip_data(Element elt,Integer i,Real &x,Real &y)

Name

Integer Get_hip_data(Element elt,Integer i,Real &x,Real &y)

Description

Get the plan co-ordinates (**x**,**y**) of the **i**th hip point of the Alignment string **elt**.

A function return value of zero indicates the hip data was successfully returned.

ID = 101

Get_hip_data(Element elt,Integer i,Real &x,Real &y,Real &rad)

Name

Integer Get_hip_data(Element elt,Integer i,Real &x,Real &y,Real &rad)

Description

Get the plan co-ordinates (**x**,**y**) and the curve **radius**, rad, for the ith hip point of the Alignment string **elt**.

If the radius is:

positive,	it is a right hand curve
negative,	it is a left hand curve.
zero,	there is no curve.

A function return value of zero indicates the hip data was successfully returned.

ID = 102

Get_hip_data(Element elt,Integer i,Real &x,Real &y,Real &rad,Real &left_spiral,Real &right_spiral)

Name

Integer Get_hip_data(Element elt,Integer i,Real &x,Real &y,Real &rad,Real &left_spiral,Real &right_spiral)

Description

Get the plan co-ordinates (**x**,**y**), the curve radius **rad**, and the left and right spiral lengths, **left_spiral** and **right_spiral** for the **i**th hip point of the Alignment Element **elt**.

If the radius is:

positive,	it is a right hand curve
negative,	it is a left hand curve.
zero,	there is no curve.

A spiral length of zero indicates that there is no spiral.

A function return value of zero indicates the hip data was successfully returned.

ID = 103

Set_hip_data(Element elt,Integer i,Real x,Real y)

Name

Integer Set_hip_data(Element elt,Integer i,Real x,Real y)

Description

Modify the plan co-ordinates (**x**,**y**) of the ith hip point of the Alignment string **elt**. The existing curve radius and spiral lengths are not altered.

The **i**th hip point must already exist.

A function return value of zero indicates the hip was successfully set.

ID = 104

Set_hip_data(Element elt,Integer i,Real x,Real y,Real rad)

Name

Integer Set_hip_data(Element elt,Integer i,Real x,Real y,Real rad)

Description

Modify the plan co-ordinates (**x**,**y**) and the curve radius, **rad**, of the **i**th hip point of the Alignment string **elt**. The spiral lengths are not altered.

The **i**th hip point must already exist.

A function return value of zero indicates the hip was successfully set.

ID = 105

Set_hip_data(Element elt,Integer i,Real x,Real y,Real rad,Real left_spiral,Real right_spiral)

Name

Integer Set_hip_data(Element elt,Integer i,Real x,Real y,Real rad,Real left_spiral,Real right_spiral)

Description

Modify the plan co-ordinates (**x**,**y**), the curve radius **rad**, and the left and right spiral lengths, **left_spiral** and **right_spiral** for the **i**th hip point of the Alignment string **elt**.

The ith hip point must already exist.

A function return value of zero indicates the hip was successfully set.

ID = 106

Insert_hip(Element elt,Integer i,Real x,Real y)

Name

Integer Insert_hip(Element elt,Integer i,Real x,Real y)

Description

Insert a new hip with plan co-ordinates (**x**,**y**) **before** the existing ith hip point.

The curve radius and spiral lengths are set to zero.

The inserted hip becomes the ith hip and the position of all subsequent hip's increases by one.

If **i** is greater than number of hips, then the new hip is appended to the string.

If **i** is less than one, then the new hip is prepended to the string.

A function return value of zero indicates the hip was inserted successfully.

ID = 107

Insert_hip(Element elt,Integer i,Real x,Real y,Real rad)

Name

Integer Insert_hip(Element elt,Integer i,Real x,Real y,Real rad)

Description

Insert a new hip with plan co-ordinates (x,y) and curve radius **rad before** the existing **i**th hip point.

The spiral lengths are set to zero.

The inserted hip becomes the **i**th hip and the position of all subsequent hip's increases by one.

If **i** is greater than number of hips, then the new hip is appended to the string.

If **i** is less than one, then the new hip is prepended to the string.

A function return value of zero indicates the hip was inserted successfully.

ID = 108

Insert_hip(Element elt,Integer i, Real x,Real y,Real rad,Real left_spiral,Real right_spiral)

Name

Integer Insert_hip(Element elt,Integer i,Real x,Real y,Real rad,Real left_spiral,Real right_spiral)

Description

Insert a new hip with plan co-ordinates (**x**,**y**), curve radius **rad** and left and right spirals of length **left_spiral** and **right_spiral** respectively, **before** the existing **i**th hip point.

The inserted hip becomes the ith hip and the position of all subsequent hip's increases by one.

If **i** is greater than number of hips, then the new hip is appended to the string.

If **i** is less than one, then the new hip is prepended to the string.

A function return value of zero indicates the hip was inserted successfully.

ID = 109

Delete_hip(Element elt,Integer i)

Name

Integer Delete_hip(Element elt,Integer i)

Description

Delete the **i**th hip from the Alignment string **elt**.

The position of all subsequent hips is decreased by one.

A function return value of zero indicates the hip was successfully deleted.

ID = 110

Get_hip_type(Element elt,Integer hip_no,Text &type)

Name

Integer Get_hip_type(Element elt,Integer hip_no,Text &type)

Description

Get the type of the horizontal intersection point number **hip_no** for the Alignment string **elt**.

The Text **type** has a returned value of

Spiral	if there is spiral/s and horizontal curve at the hip.
Curve	if there is a horizontal curve with no spirals at the hip.
IP	if there are no spirals or horizontal curves at the hip.

A function return value of zero indicates the hip information was successfully returned.

ID = 397

Get_hip_geom(Element elt,Integer hip_no,Integer mode, Real &x,Real &y)

Name

Integer Get_hip_geom(Element elt,Integer hip_no,Integer mode,Real &x,Real &y)

Description

Return the (**x**,**y**) co-ordinates of the critical horizontal points around the horizontal intersection point hip_no (i.e. tangent spiral points, spiral curve points etc.) for the Alignment string **elt**.

The type of critical point (x,y) returned is specified by **mode** and depends on the type of the hip.

The following table gives the description of the returned co-ordinate (x,y) and whether or not the mode is applicable for the given HIP type (Y means applicable, N means not applicable).

			HIP Type	
Mode	**Returned co-ordinate**	**HIP**	**Curve**	**Spiral**
0	HIP co-ords	Y	Y	Y
1	start tangent	N	Y TC	Y TS
2	end tangent	N	Y CT	Y ST
3	curve centre	N	Y	Y
4	spiral-curve	N	N	Y
5	curve-spiral	N	N	Y

A function return value of zero indicates the hip information was successfully returned and that the mode was appropriate for the HIP type of the hip **hip_no**.

ID = 395

Append_vip(Element elt,Real ch,Real ht)

Name

Integer Append_vip(Element elt,Real ch,Real ht)

Description

Append a vertical intersection point (vip) with chainage-height co-ordinates (**ch**,**ht**) to the Element **elt**. The parabolic curve length is set to zero.

The order in which the vips are appended is taken as the order of the vips in the Alignment string.

The vips must be appended in order of increasing chainage along the Alignment string.

Append_vip is used to place the first vip as well as the subsequent vips.

A function return value of zero indicates the vip was appended successfully.

ID = 96

Append_vip(Element elt,Real ch,Real ht,Real parabolic)

Name

Integer Append_vip(Element elt,Real ch,Real ht,Real parabolic)

Description

Append to the Element **elt** a vertical intersection point (vip) with chainage-height co-ordinates (**ch**,**ht**) and a parabolic curve of length **parabolic**.

A parabolic curve length of zero indicates no curve is present.

A function return value of zero indicates the vip was appended successfully.

ID = 97

Append_vip(Element elt,Real ch,Real ht,Real length,Integer mode)

Name

Integer Append_vip(Element elt,Real ch,Real ht,Real length,Integer mode)

Description

Append to the Element **elt** a vertical intersection point (vip) with chainage-height co-ordinates (**ch**,**ht**) and a curve of length **length**.

If mode = 0 or 1, the curve is a parabolic vertical curve
If mode = 2, the curve is a circular vertical curve

A curve length of zero indicates no curve is present.

A function return value of zero indicates the vip was appended successfully.

ID = 98

Get_vip_points(Element elt,Integer &num_pts)

Name

Integer Get_vip_points(Element elt,Integer &num_pts)

Description

Get the number of vips, **num_pts**, in the Alignment string **elt**.

A function return value of zero indicates the number of vip points was successfully returned.

ID = 111

Get_vip_data(Element elt,Integer i,Real &ch,Real &ht)

Name

Integer Get_vip_data(Element elt,Integer i,Real &ch,Real &ht)

Description

Get the chainage-height co-ordinates (**ch**,**ht**) of the ith vip point for the Alignment string **elt**.

A function return value of zero indicates the vip data was successfully returned.

ID = 112

Get_vip_data(Element elt,Integer i,Real &ch,Real &ht,Real &parabolic)

Name

Integer Get_vip_data(Element elt,Integer i,Real &ch,Real &ht,Real &parabolic)

Description

Get the chainage-height co-ordinates (**ch**,**ht**) and the parabolic curve length **parabolic** for the **i**th vip point of the Alignment string **elt**.

A function return value of zero indicates the vip data was successfully returned.

ID = 113

Get_vip_data(Element elt,Integer i,Real &ch,Real &ht,Real &value,Integer &mode)

Name

Integer Get_vip_data(Element elt,Integer i,Real &ch,Real &ht,Real &value,Integer &mode)

Description

Get the chainage-height co-ordinates (**ch**,**ht**) and the curve length **value** for the **i**th vip point of the Alignment string **elt**.

If mode = 0 or 1, the curve is a parabolic vertical curve
If mode = 2, the curve is a circular vertical curve

A curve length of zero indicates no curve is present.

A function return value of zero indicates the vip data was successfully returned.

ID = 114

Set_vip_data(Element elt,Integer i,Real ch,Real ht)

Name

Integer Set_vip_data(Element elt,Integer i,Real ch,Real ht)

Description

Modify the chainage-height co-ordinates (**ch**,**ht**) of the **i**th vip point for the Alignment string **elt**. The existing parabolic curve length is not altered.

The **i**th vip point must already exist.

A function return value of zero indicates the vip data was successfully set.

ID = 115

Set_vip_data(Element elt,Integer i, Real ch,Real ht,Real parabolic)

Name

Integer Set_vip_data(Element elt,Integer i,Real ch,Real ht,Real parabolic)

Description

Modify the chainage-height co-ordinates (**ch**,**ht**) and the parabolic curve length **parabolic**, for the ith vip point of the Alignment string **elt**.

The **i**th vip point must already exist.

A function return value of zero indicates the vip data was successfully set.

ID = 116

Set_vip_data(Element elt,Integer i,Real ch,Real ht,Real value,Integer mode)

Name

Integer Set_vip_data(Element elt,Integer i,Real ch,Real ht,Real value,Integer mode)

Description

Modify the chainage-height co-ordinates (**ch**,**ht**) and the curve length **value**, for the **i**'th vip point of the Alignment string **elt**.

If mode = 0 or 1, the curve is set to be a parabolic vertical curve
If mode = 2, the curve is set to be a circular vertical curve

A curve length of zero indicates no curve is present.

A function return value of zero indicates the vip data was successfully returned.

ID = 117

Insert_vip(Element elt,Integer i,Real ch,Real ht)

Name

Integer Insert_vip(Element elt,Integer i,Real ch,Real ht)

Description

Insert a new vip with chainage-height co-ordinates (**ch**,**ht**) before the existing i'th vip point.

The parabolic curve length is set to zero.

The inserted vip becomes the **i**th vip and the position of all subsequent vips increases by one.

If **i** is greater than number of vips, then the new vip is appended to the string.

If **i** is less than one, then the new vip is prepended to the string.

A function return value of zero indicates that the vip was successfully inserted.

ID = 118

Insert_vip(Element elt,Integer i,Real ch,Real ht,Real parabolic)

Name

Integer Insert_vip(Element elt,Integer i,Real ch,Real ht,Real parabolic)

Description

Insert a new vip with chainage-height co-ordinates (**ch**,**ht**) and parabolic length **parabolic** before the existing ith vip point.

The inserted vip becomes the **i**th vip and the position of all subsequent vips increases by one.

If **i** is greater than number of vips, then the new vip is appended to the string.

If **i** is less than one, then the new vip is prepended to the string.

A function return value of zero indicates that the vip was successfully inserted.

ID = 119

Insert_vip(Element elt,Integer i,Real ch,Real ht,Real value,Integer mode)

Name

Integer Insert_vip(Element elt,Integer i,Real ch,Real ht,Real value,Integer mode)

Description

Insert a new vip with chainage-height co-ordinates (**ch**,**ht**) and curve length **value** before the existing **i'**th vip point.

The inserted vip becomes the **i**th vip and the position of all subsequent vips increases by one.

If **i** is greater than number of vips, then the new vip is appended to the string.

If **i** is less than one, then the new vip is prepended to the string.

If mode = 0 or 1, the curve is set to be a parabolic vertical curve
If mode = 2, the curve is set to be a circular vertical curve

A curve length of zero indicates no curve is present.

A function return value of zero indicates that the vip was successfully inserted.

ID = 120

Delete_vip(Element elt,Integer i)

Name

Integer Delete_vip(Element elt,Integer i)

Description

Delete the **i**th vip from the Alignment string **elt**.

The position of all subsequent vips is decreased by one.

A function return value of zero indicates that the vip was successfully deleted.

ID = 121

Calc_alignment(Element elt)

Name

Integer Calc_alignment(Element elt)

Description

Use all the horizontal and vertical data to calculate the full geometry for the Alignment string.

A Calc_alignment must be done before the Alignment string can be used in 12d Model.

A function return value of zero indicates the geometry of the alignment was successfully calculated.

ID = 99

Get_vip_type(Element elt,Integer vip_no,Text &type)

Name

Integer Get_vip_type(Element elt,Integer vip_no,Text &type)

Description

Get the type of the vertical intersection point number **vip_no** for the Alignment string **elt**.

The Text **type** has a returned value of

VC	if there is a parabolic curve at the vip.
Curve	if there is a circular curve at the vip.
IP	if there is no vertical curves at the vip.

A function return value of zero indicates the vip information was successfully returned.

ID = 398

Get_vip_geom(Element elt,Integer vip_no,Integer mode,Real &chainage,Real &height)

Name

Integer Get_vip_geom(Element elt,Integer vip_no,Integer mode,Real &chainage,Real &height)

Description

Return the **chainage** and **height** co-ordinates of the critical points (tangent points, curve centre) for vertical intersection point number **vip_no** of the Alignment string **elt**.

The type of critical point (chainage,height) returned is given by **mode** and depends on the type of the vip.

The following table gives the description of the returned co-ordinates (chainage,height) and states whether the mode is applicable or not for the given VIP type (Y means applicable, N means not applicable).

VIP Type

Mode	Returned co-ordinate	VIP	VC	Curve
0	VIP co-ords	Y	Y	Y
1	start tangent	N	Y TC	Y TC
2	end tangent	N	Y CT	Y CT
3	curve centre	N	N	Y

A function return value of zero indicates that the vip information was successfully returned and that the mode was appropriate for the VIP type of the vip **number vip_no**.

ID = 396

Get_hip_id(Element elt,Integer position,Integer &id)

Name

Integer Get_hip_id(Element elt,Integer position,Integer &id)

Description

<no description>

ID = 1451

Get_vip_id(Element elt,Integer position,Integer &id)

Name

Integer Get_vip_id(Element elt,Integer position,Integer &id)

Description

<no description>

ID = 1452

General Element Operations

See Selecting Strings
See Drawing Elements
See Open and Closing Strings
See Length and Area of Strings
See Position and Drop Point on Strings
See Parallel Strings
See Self Intersection of String
See Loop Clean Up for String
See Check Element Locks

Selecting Strings

Select_string(Text msg,Element &string)

Name

Integer Select_string(Text msg,Element &string)

Description

Write the message **msg** to the 12d Model ***Output Window*** and wait until a selection is made.

If a pickable Element is selected, then return the Element picked by the user in **string** and the function return value is 1.

If no pickable Element is picked and the function returns, then the function returns codes are:

-1	indicates cancel was chosen from the pick-ops menu.
0	pick unsuccessful
1	pick was successful
2	a cursor pick

ID = 29

Select_string(Text msg,Element &string,Real &x,Real &y,Real &z,Real &ch,Real &ht)

Name

Integer Select_string(Text msg,Element &string,Real &x,Real &y,Real &z,Real &ch,Real &ht)

Description

Write the message **msg** to the 12d Model ***Output Window*** and then return the Element picked by the user. The co-ordinates of the picked point are also returned.

The picked Element is returned in the Element **string**.

The co-ordinates and chainage of the picked point on the Element string are (**x,y,z**) and **ch** respectively.

The value **ht** is reserved for future use and should be ignored.

A function return value of

-1	indicates cancel was chosen from the pick-ops menu.
0	pick unsuccessful
1	pick was successful
2	a cursor pick

ID = 214

Select_string(Text msg,Element &string,Real &x,Real &y,Real &z,Real &ch,Real &ht,Integer &dir)

Name

Integer Select_string(Text msg,Element &string,Real &x,Real &y,Real &z,Real &ch,Real &ht, Integer &dir)

Description

Write the message **msg** to the 12d Model Output Window and then return the Element picked by the user. The co-ordinates of the picked point are also returned plus whether the string selecting was picked in the same direction as the string, or the opposite direction to the string.

The picked Element is returned in the Element **string**.

The co-ordinates and chainage of the picked point on the Element string are (**x,y,z**) and **ch** respectively.

The value **ht** is reserved for future use and should be ignored.

The value **dir** indicates if the picking motion was in the same direction as the selected string, or in the opposite direction.

dir = when the picking motion was in the same direction as the selected string.
dir = when the picking motion was in the opposite direction as the selected string.

A function return value of

-1	indicates cancel was chosen from the pick-ops menu.
0	pick unsuccessful
1	pick was successful
2	a cursor pick

ID = 547

Drawing Elements

Element_draw(Element elt,Integer col_num)

Name

Integer Element_draw(Element elt,Integer col_num)

Description

Draw the Element **elt** in the colour number **col_num** on all the views that **elt** is displayed on.

A function return value of zero indicates that **elt** was drawn successfully.

ID = 372

Element_draw(Element elt)

Name

Integer Element_draw(Element elt)

Description

Draw the Element **elt** in its natural colour on all the views that **elt** is displayed on.

A function return value of zero indicates that **elt** was drawn successfully.

ID = 371

Open and Closing Strings

String_closed(Element elt,Integer &closed)

Name

Integer String_closed(Element elt,Integer &closed)

Description

Checks to see if the Element elt is **closed**. That is, check if the first and the last points of the element are the same. The close status is returned as **closed**.

If **closed** is

1	then **elt** is closed
0	then **elt** is not closed (i.e. open)

A zero function return value indicates that the closure check was successful.

ID = 368

String_open(Element elt)

Name

Integer String_open(Element elt)

Description

Open the Element **elt**.

That is, if the first and the last points of the elt are the same, then delete the last point of **elt**.

A function return value of zero indicates that **elt** was successfully opened.

ID = 366

String_close(Element elt)

Name

Integer String_close(Element elt)

Description

Close the Element elt.

That is, if the first and the last points of **elt** are not the same, then add a point to the end of **elt** which is the same as the first point of **elt**.

A function return value of zero indicates that **elt** was successfully closed.

ID = 367

Length and Area of Strings

Get_length(Element string,Real &length)

Name

Integer Get_length(Element string,Real &length)

Description

Get the **plan** length of the Element **string** (which equals the end chainage minus the start chainage) and return the plan length in **length**.

A function return value of zero indicates the plan length was successfully returned.

ID = 122

Get_length_3d(Element string,Real &length)

Name

Integer Get_length_3d(Element string,Real &length)

Description

Get the 3d length of the Element **string** and return the 3d length in **length**.

A function return value of zero indicates the 3d length was successfully returned.

ID = 359

Get_length_3d(Element string,Real ch,Real &length)

Name

Integer Get_length_3d(Element string,Real ch,Real &length)

Description

Get the 3d length of the Element **string** from the start of the string up the given chainage **ch**. Return the 3d length in **length**.

A function return value of zero indicates the 3d length was successfully returned.

ID = 2681

Plan_area(Element string, Real &plan_area)

Name

Integer Plan_area(Element string,Real &plan_area)

Description

Calculate the plan area of the Element **string**. If the Element is not closed, then the first and last points are joined before calculating the area. For an arc, the plan area of the sector is returned.

The plan area is returned in the Real **plan_area**.

A function return value of zero indicates the plan area was successfully returned.

ID = 221

Position and Drop Point on Strings

Get_position(Element elt,Real ch,Real &x,Real &y,Real &z,Real &inst_dir)

Name

Integer Get_position(Element elt,Real ch,Real &x,Real &y,Real &z,Real &inst_dir)

Description

For the Element **elt**, get the (**x,y,z**) position and instantaneous direction (**inst_dir** - as an angle, measured in radians) of the point at chainage **ch** on **elt**.

A function return value of zero indicates success.

ID = 190

Get_position(Element elt,Real ch,Real &x,Real &y,Real &z,Real &inst_dir,Real &rad, Real &inst_grade)

Name

Integer Get_position(Element elt,Real ch,Real &x,Real &y,Real &z,Real &inst_dir,Real &rad,Real &inst_grade)

Description

For a Element, **elt**, of type **Alignment** only, get the (**x,y,z**) position, radius **rad**, instantaneous direction (**inst_dir** - as an angle, measured in radians) and instantaneous grade (**inst_grade**) of a point on **elt** at chainage **ch**.

A function return value of zero indicates success.

ID = 471

Drop_point(Element elt,Real xd,Real yd,Real zd,Real &xf,Real &yf, Real &zf,Real &ch,Real &inst_dir,Real &off)

Name

Integer Drop_point(Element elt,Real xd,Real yd,Real zd,Real &xf,Real &yf,Real &zf,Real &ch,Real &inst_dir,Real &off)

Description

In plan, drop the point (xd,yd) perpendicularly onto the Element **elt**. If the point cannot be dropped onto any segment of the Element, then the point is dropped onto the closest end point. A z-value for the dropped point is created by interpolation.

The position of the dropped point on the Element is returned in **xf**, **yf** and **zf**. The chainage of the dropped point on the string is **ch** and **inst_dir** the instantaneous direction (as an angle, measured in radians) at the dropped point.

Off is the plan distance from the original point to the dropped point on the string.

A function return value of zero indicates that the drop was successful.

ID = 191

Drop_point(Element elt,Real xd,Real yd,Real zd,Real &xf,Real &yf, Real &zf,Real &ch,Real &inst_dir,Real &off,Segment &segment)

Name

Integer Drop_point(Element elt,Real xd,Real yd,Real zd,Real &xf,Real &yf,Real &zf,Real &ch,Real &inst_dir,Real &off,Segment &segment)

Description

In plan, drop the point (xd,yd) perpendicularly onto the Element elt. If the point cannot be dropped onto any segment of the Element, then the point is dropped onto the closest end point. A z-value for the dropped point is created by interpolation.

The position of the dropped point on the Element is returned in **xf, yf** and **zf**. The chainage of the dropped point on the string is **ch** and **inst_dir** the instantaneous direction (as an angle, measured in radians) at the dropped point.

Off is the plan distance from the original point to the dropped point on the string.

Segment **segment** is the link of the string that the point drops onto.

A function return value of zero indicates that the drop was successful.

ID = 302

Parallel Strings

The parallel command is a plan parallel and is used for all Elements except Tin and Text.

The sign of the distance to parallel the object is used to indicate whether the object is parallelled to the left or to the right.

A **positive** distance means to parallel the object to the **right**.

A **negative** distance means to parallel the object to the **left**.

Parallel(Element elt,Real distance,Element &parallelled)

Name

Integer Parallel(Element elt,Real distance,Element &parallelled)

Description

Plan parallel the Element elt by the distance distance.

The parallelled Element is returned as the Element **parallelled**. The z-values are not modified, i.e. they are the same as for **elt**.

A function return value of zero indicates the parallel was successful.

ID = 365

Self Intersection of String

String_self_intersects(Element elt,Integer &intersects)

Name

Integer String_self_intersects(Element elt,Integer &intersects)

Description

Find the number of self intersections for the Element **elt**.

The number of self intersections is returned as **intersects**.

A function return value of zero indicates that there were no errors in the function.

Note

For Elements of type Alignment, Arc, Circle and Text the number of intersects is set to negative.

ID = 328

Loop Clean Up for String

Loop_clean(Element elt,Point ok_pt,Element &new_elt)

Name

Integer Loop_clean(Element elt,Point ok_pt,Element &new_elt)

Description

This routine tries to remove any plan loops in the Element **elt**.

If **elt** is closed, then the function assumes that the Point **ok_pt** is near a segment of the string that will also be in the cleaned string.

If **elt** is open, then the function starts cleaning from the end of the string closest to the Point **ok_pt**.

The cleaned Element is returned as Element **new_elt**.

A function return value of zero indicates the clean was successful.

Note

Loop_clean is not defined for the Elements of type Alignment, Arc, Circle and Text

ID = 329

Check Element Locks

Get_read_locks(Element elt,Integer &num_locks)

Name

Integer Get_read_locks(Element elt,Integer &num_locks)

Description

For a valid Element **elt**, return the number of read locks on **elt** in **num_locks**.

Note: There are no 12dPL functions that a macro programmer can use to set read locks. They are automatically assigned and removed as required by various 12dPL functions.

A function return value of zero indicates the number of read locks was successfully returned.

ID = 1453

Get_write_locks(Element elt,Integer &num_locks)

Name

Integer Get_write_locks(Element elt,Integer &num_locks)

Description

For a valid Element **elt**, return the number of write locks on **elt** in **num_locks**.

Note: There are no 12dPL functions that a macro programmer can use to set write locks. They are automatically assigned and removed as required by various 12dPL functions.

A function return value of zero indicates the number of write locks was successfully returned.

ID = 1454

Miscellaneous Element Functions

String_replace(Element from,Element &to)

Name

Integer String_replace(Element from,Element &to)

Description

Copy the *contents* of the Element **from** and use them to replace the contents of the Element **to.**

The id/Uid of **to** is **not** replaced.

The Elements **to** and **from** must be **strings** and also be the same string types. For example, both of type Super.

Note: this will not work for Elements of type Tin.

A function return value of zero indicates the replace was successful.

ID = 1176

Creating Valid Names

Valid_string_name(Text old_name,Text &valid_name)

Name

Integer Valid_string_name(Text old_name,Text &valid_name)

Description

Convert the Text *old_name* to a valid string name by substituting spaces for any illegal characters in *old_name*. The new name is returned in *valid_name*.

A function return value of zero indicates the function was successful.

ID = 2277

Valid_model_name(Text old_name,Text &valid_name)

Name

Integer Valid_model_name(Text old_name,Text &valid_name)

Description

Convert the Text *old_name* to a valid model name by substituting spaces for any illegal characters in *old_name*. The new name is returned in *valid_name*.

A function return value of zero indicates the function was successful.

ID = 2278

Valid_tin_name(Text old_name,Text &valid_name)

Name

Integer Valid_tin_name(Text old_name,Text &valid_name)

Description

Convert the Text *old_name* to a valid tin name by substituting spaces for any illegal characters in *old_name*. The new name is returned in *valid_name*.

A function return value of zero indicates the function was successful.

ID = 2279

Valid_attribute_name(Text old_name,Text &valid_name)

Name

Integer Valid_attribute_name(Text old_name,Text &valid_name)

Description

Convert the Text *old_name* to a valid attribute name by substituting spaces for any illegal characters in *old_name*. The new name is returned in *valid_name*.

A function return value of zero indicates the function was successful.

ID = 2280

Valid_linestyle_name(Text old_name,Text &valid_name)

Name

Integer Valid_linestyle_name(Text old_name,Text &valid_name)

Description

Convert the Text *old_name* to a valid linestyle name by substituting spaces for any illegal characters in *old_name*. The new name is returned in *valid_name*.

A function return value of zero indicates the function was successful.

ID = 2281

Valid_symbol_name(Text old_name,Text &valid_name)

Name

Integer Valid_symbol_name(Text old_name,Text &valid_name)

Description

Convert the Text *old_name* to a valid symbol name by substituting spaces for any illegal characters in *old_name*. The new name is returned in *valid_name*.

A function return value of zero indicates the function was successful.

ID = 2282

XML

The XML macro calls allow the user to read or write xml files from 12dPL in a DOM based manner. This will be effective for small to mid size XML files, but very large XML files may not be supported.

For more information on the XML standard, see http://www.w3.org/XML/

Create_XML_document()

Name

XML_Document Create_XML_document()

Description

This call creates a new XML document. This is the entry point for all macro code that works with XML. Existing files can then be read into the document, or the code may start to build up nodes into the document.

ID = 2436

Read_XML_document(XML_Document doc,Text file)

Name

Integer Read_XML_document(XML_Document doc,Text file)

Description

Reads the supplied file and loads the nodes into the supplied XML Document object.

Returns 0 if successful.

ID = 2419

Write_XML_document(XML_Document doc,Text file)

Name

Integer Write_XML_document(XML_Document doc,Text file)

Description

Writes the supplied XML Document to the given file name.

Returns 0 if successful.

ID = 2420

Get_XML_declaration(XML_Document doc,Text &version,Text &encoding, Integer &standalone)

Name

Integer Get_XML_declaration(XML_Document doc,Text &version,Text &encoding,Integer &standalone)

Description

Finds and returns the values from the XML declaration in the given document. Not all documents may contain XML declarations.

Returns 0 if successful.

ID = 2437

Set_XML_declaration(XML_Document doc,Text version,Text encoding, Integer standalone)

Name

Integer Set_XML_declaration(XML_Document doc,Text version,Text encoding,Integer standalone)

Description

This call sets the details for the XML declaration. If the document does not already contain an XML declaration, one will be added to the top of the document.

Returns 0 if successful.

ID = 2438

Create_node(Text name)

Name

XML_Node Create_node(Text name)

Description

This call creates a new XML node. This node can have its value set, or have other children nodes appended to it. It must also be either set as the root node (see **Set_Root_Node**) or appended to another node (see **Append_Node**) to become part of a document.

ID = 2435

Get_root_node(XML_Document doc,XML_Node &node)

Name

Integer Get_root_node(XML_Document doc,XML_Node &node)

Description

This call finds and retrieves the node at the root of the document. This is the top level node. If there is no root node, the call will return non 0.

Returns 0 if successful.

ID = 2421

Set_root_node(XML_Document,XML_Node &node)

Name

Integer Set_root_node(XML_Document,XML_Node &node)

Description

This call sets the root node (the top level node) for the given document. There must be at most one root node in a document.

ID = 2422

Get_number_of_nodes(XML_Node node)

Name

Integer Get_number_of_nodes(XML_Node node)

Description

This call returns the number of children nodes for the given nodes. A node may contain 0 or more children.

ID = 2423

Get_child_node(XML_Node node,Integer index,XML_Node &child_node)

Name

Integer Get_child_node(XML_Node node,Integer index,XML_Node &child_node)

Description

This call retrieves the n'th child, as specified by index, of a parent node and stores it in the child_node argument.

Returns 0 if successful.

ID = 2424

Get_child_node(XML_Node node,Text name,XML_Node &child_node)

Name

Integer Get_child_node(XML_Node node,Text name,XML_Node &child_node)

Description

This call retrieves the first instance of a child of a parent node, by its name. If there is more than one element of the same name, this call will only return the first. The retrieved node will be stored in the child_node argument.

This call will return 0 if successful.

ID = 2439

Append_node(XML_Node parent,XML_Node new_node)

Name

Integer Append_node(XML_Node parent,XML_Node new_node)

Description

This call appends a child node to a parent node. A parent node may contain 0 or more children nodes.

This call will return 0 if successful.

ID = 2425

Remove_node(XML_Node parent,Integer index)

Name

Integer Remove_node(XML_Node parent,Integer index)

Description

This call removes the n'th child node, as given by index, from the supplied parent node.

This call will return 0 if successful.

ID = 2426

Get_parent_node(XML_Node child,XML_Node &parent)

Name

Integer Get_parent_node(XML_Node child,XML_Node &parent)

Description

This call will find the parent node of the supplied child and store it in the parent argument.

This call will return 0 if successful.

ID = 2427

Get_next_sibling_node(XML_Node node,XML_Node &sibling)

Name

Integer Get_next_sibling_node(XML_Node node,XML_Node &sibling)

Description

Given a node, this call will retrieve the next sibling, or same level node.

In the following example, **Child2** is the next sibling of **Child1**.

```
<Parent>
   <Child1/>
   <Child2/>
</Parent>
```

This call will return 0 if successful.

ID = 2428

Get_prev_sibling_node(XML_Node node,XML_Node &sibling)

Name

Integer Get_prev_sibling_node(XML_Node node,XML_Node &sibling)

Description

Given a node, this call will retrieve the previous sibling, or same level node.

In the following example, **Child1** is the previous sibling of **Child2.**

```
<Parent>
   <Child1/>
   <Child2/>
</Parent>
```

This call will return 0 if successful.

ID = 2429

Get_node_name(XML_Node node,Text &name)

Name

Integer Get_node_name(XML_Node node,Text &name)

Description

This call will retrieve the name of a supplied node and store it in the name argument.

The name of a node is the value within the brackets or tags. In the following example, **MyNode** is the name of the node.

```
<MyNode>1234</MyNode>
```

This call will return 0 if successful.

ID = 2433

Get_node_attribute(XML_Node node,Text name,Text &value)

Name

Integer Get_node_attribute(XML_Node node,Text name,Text &value)

Description

This call will try find an attribute of given name belonging to the supplied node, and will store the value in the value attribute.

In the following example, the data stored in value will be: **MyAttributeData**

```
<MyNode MyAttribute="MyAttributeData" />
```

This call will return 0 if successful.

ID = 2440

Set_node_attribute(XML_Node node,Text name,Text value)

Name

Integer Set_node_attribute(XML_Node node,Text name,Text value)

Description

This call will set the value of an attribute attached to a node. If it does not exist, the attribute will be created.

This call will return 0 if successful.

ID = 2441

Remove_node_attribute(XML_Node node,Text name)

Name

Integer Remove_node_attribute(XML_Node node,Text name)

Description

This call will attempt to remove a node of a given name from the supplied node.

This call will return 0 if successful.

ID = 2442

Is_text_node(XML_node &node)

Name

Integer Is_text_node(XML_node &node)

Description

This call will attempt to determine if a node is a text only node or not.

A text node is one that contains only text, and no other child nodes.

This call will return 1 if the node is a text node.

ID = 2430

Get_node_text(XML_Node &node,Text &text)

Name

Integer Get_node_text(XML_Node &node,Text &text)

Description

This call will attempt to retrieve the internal text value of a node and store it in text.

Not all nodes may contain text.

In the following example, the value of text will be set to **MyText**

```
<MyNode>MyText</MyNode>
```

This call will return 0 if successful.

ID = 2431

Set_node_text(XML_Node &node,Text value)

Name

Integer Set_node_text(XML_Node &node,Text value)

Description

This call will set the internal text of node to the value.

This call will return 0 if successful.

ID = 2432

Create_text_node(Text name,Text value)

Name

XML_Node Create_text_node(Text name,Text value)

Description

This call will create a new text node of the given name and set the internal text to the given value.

This call will return the created node.

ID = 2434

Map File

Map_file_create(Map_File &file)

Name

Integer Map_file_create(Map_File &file)

Description

Create a mapping file. The file unit is returned as Map_file **file**.

A function return value of zero indicates the file was opened successfully.

ID = 864

Map_file_open(Text file_name, Text prefix, Integer use_ptline,Map_File &file)

Name

Integer Map_file_open(Text file_name, Text prefix, Integer use_ptline,Map_File &file)

Description

Open up a mapping file to read.

The file unit is returned as Map_file **file**.

The prefix of models is given as Text **prefix**.

The string type is given as Integer **use_ptline**,

0 – point string

1 – line sting.

A function return value of zero indicates the file was opened successfully.

ID = 865

Map_file_close(Map_File file)

Name

Integer Map_file_close(Map_File file)

Description

Close a mapping file. The file being closed is Map_file **file**.

A function return value of zero indicates the file was closed successfully.

ID = 866

Map_file_number_of_keys(Map_File file,Integer &number)

Name

Integer Map_file_number_of_keys(Map_File file,Integer &number)

Description

Get the number of keys in a mapping file.

The file is given as Map_file **file**.

The number of keys is returned in Integer **number**.

A function return value of zero indicates the number was returned successfully.

ID = 868

Map_file_add_key(Map_File file,Text key,Text name,Text model,Integer colour,Integer ptln,Text style)

Name

Integer Map_file_add_key(Map_File file,Text key,Text name,Text model,Integer colour,Integer ptln,Text style)

Description

Add key to a mapping file.

The file is given in Map_file **file**.

The key is given in Text **key**.

The string name is given in Text **name**.

The model name is given in Text **model**.

The string colour is given in Integer **colour**.

The string type is given in Integer **ptln**.

The string style is given in Text **style**.

A function return value of zero indicates the key was added successfully.

ID = 869

Map_file_get_key(Map_File file,Integer n,Text &key,Text &name,Text &model, Integer &colour,Integer &ptln,Text &style)

Name

Integer Map_file_get_key(Map_File file,Integer n,Text &key,Text &name,Text &model, Integer &colour,Integer &ptln,Text &style)

Description

Get **n**th key's data from a mapping file.

The file is given in Map_file **file**.

The key is returned in Text **key**.

The string name is returned in Text **name**.

The model name is returned in Text **model**.

The string colour is returned in Integer **colour**.

The string type is returned in Integer **ptln**.

The string style is returned in Text **style**.

A function return value of zero indicates the key was returned successfully.

ID = 870

Map_file_find_key(Map_File file,Text key, Integer &number)

Name

Integer Map_file_find_key(Map_File file,Text key,Integer &number)

Description

Find the record number from a mapping file that contains the given **key**.

The file unit is given in Map_file **file**.

The record number is returned in Integer **number**.

A function return value of zero indicates the key was find successfully.

ID = 871

Macro Console

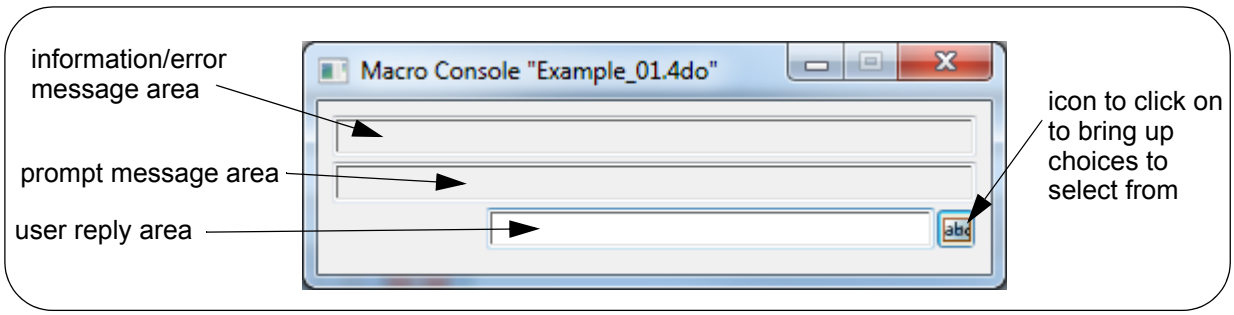
Before *Panels* where introduced into the *12d Model Programming Language*, a **Macro Console** was the only method for writing information to the user, and soliciting answers from the user.

Note: the **Macro Console** is rarely used in newer macros.

When a macro is invoked, a Macro Console is placed on the screen.

The Macro Console has three distinct areas
information/error message area (or just information message area or error message area)
prompt message area
user reply area.

and optionally, three buttons, **restart**, **abort** and **finish**.



Using **Macro Console** functions, information can be written to the **information/error message area** and the **prompt message area**, and user input read in from the **user reply area** of the Macro Console.

Some of the functions have pop-ups defined (of models, tins etc.) so that information can be selected from pop-ups displayed by clicking LB on the icon at the right hand end of the **user reply area** rather than being typed in by the user. **Note** that the icon at the right hand end of the user reply area changes depending on the type of Prompt.

The reply, either typed or selected from the icon popup, must be terminated by pressing the <Enter> key for the macro to continue.

Also the **information/error message area** is used to display progress information. This information can be standard 12dPL messages or user defined messages.

Note: Some functions also write information to the **12d Model** Output Window.

WARNING: Because the Macro Console functions all use the same three areas for messages and input, messages from one Macro Console may be overwritten by the messages from the next Macro Console function before the user has a chance to see the message.

Set_message_mode(Integer mode)

Name

Integer Set_message_mode(Integer mode)

Description

When macros are running, progress information can be displayed in the **information/error message area**. Most 12dPL computational intensive functions have standard messages that can be displayed. For example, when triangulating, regular messages showing the number of points triangulated can be displayed. Or the message ***running*** with the ticker character **"/"** rotating through 360 degrees.

The user can have the standard 12dPL messages displayed, or replace them at any time by a user defined message (set using the function Set_message_text).

If **mode** is set to

0	the user defined message
1	the standard 12dPL message

is displayed in the information/error message area.

A function return value of zero indicates the mode was successfully set.

ID = 427

Set_message_text(Text msg)

Name

void Set_message_text(Text msg)

Description

Set the user defined information message to **msg**. This is a prefix for the ticker "/".

When the message mode is set to 0 (using the function Set_message_mode), **msg** is displayed in the **information/error message area**. The message **msg** is followed by a rotating ticker (/) to indicate to the user that the macro is running.

A function return value of zero indicates the message was successfully set.

ID = 426

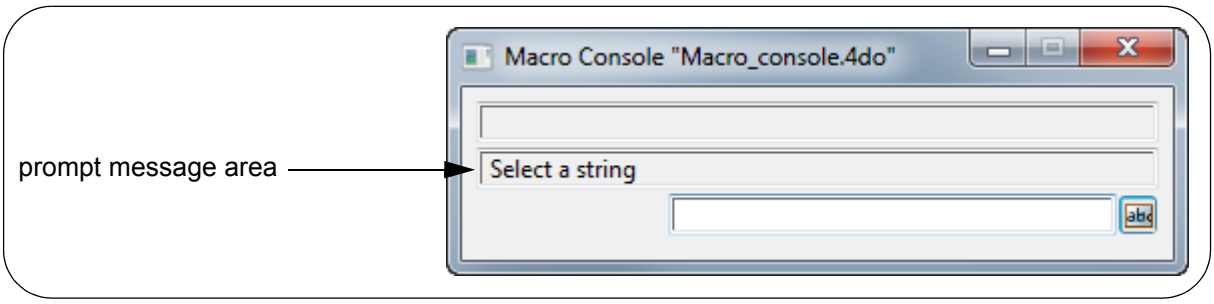
Prompt(Text msg)

Name

void Prompt(Text msg)

Description

Print the message **msg** to the **prompt message area** of the macro console.



If another message is written to the prompt message area then the previous message will be

overwritten by the new message.

ID = 34

Prompt(Text msg,Text &ret)

Name

Integer Prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then wait for the user to type text into the **user reply area** of the Macro Console. When <enter> is pressed then the text in the **user reply area** is returned in **ret**.

That is, write out the message **msg** and get a Text **ret** from the Macro Console when the text is terminated by pressing <enter>.

The reply is returned in Text **ret**.

A function return value of zero indicates the text is returned successfully.

ID = 28

Prompt(Text msg,Integer &ret)

Name

Integer Prompt(Text msg,Integer &ret)

Description

Print the message **msg** to the **prompt message area** and then read back an Integer from the user reply area of the Macro Console.

That is, write out the message **msg** and wait for an integer reply from the Macro Console. The reply is terminated by pressing <enter>.

The reply is returned in Integer **ret**.

A function return value of zero indicates that the Integer was returned successfully.

ID = 26

Prompt(Text msg,Real &ret)

Name

Integer Prompt(Text msg,Real &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Real from the **user reply area** of the Macro Console. The reply is terminated by pressing <enter>.

The reply is returned in Real **ret**.

A function return value of zero indicates that the Real was returned successfully.

ID = 27

Colour_prompt(Text msg,Text &ret)

Name

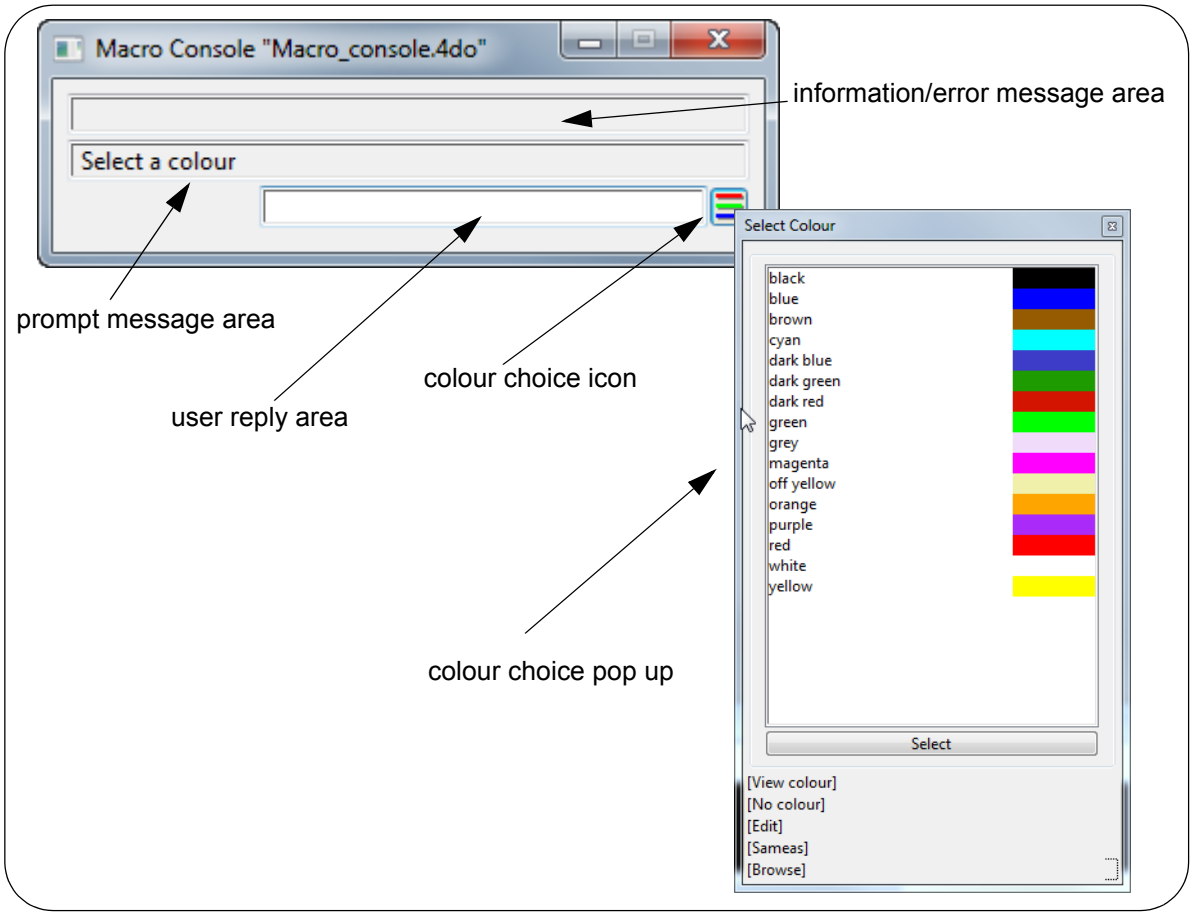
Integer Colour_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** of the Macro Console and then read back text from the **user reply area** of the Macro Console as the name of a **12d Model** colour.

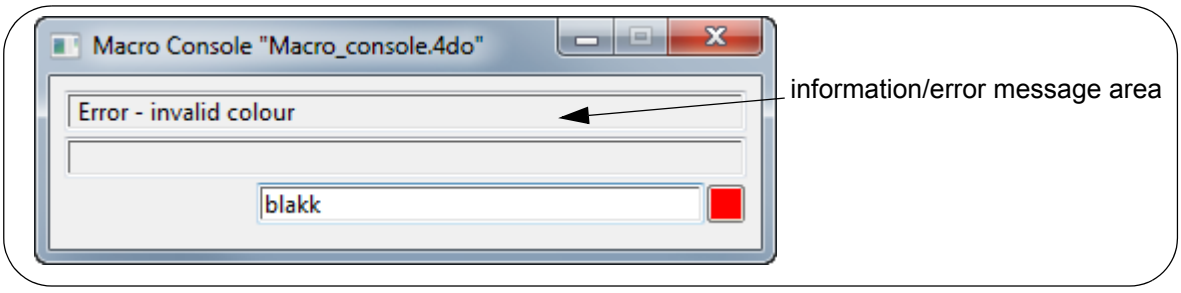
If LB is clicked on the colour choice icon at the right hand end of the **user reply area**, a list of all existing colours is placed in a pop-up. If a colour is selected from the pop -up (using LB), the colour name is written to the **user reply area**.

The reply, either typed or selected from the colour pop-up, is then terminated by pressing <Enter>.



If the text is a valid colour then a function return value of zero is returned and the colour name is returned in **ret**.

If the text is **not** a valid colour name, then the message **Error - invalid colour** is written to the **information message area** and a non-zero function return value is returned.



A function return value of zero indicates the Text **ret** is a valid colour name and is successfully

returned.

ID = 404

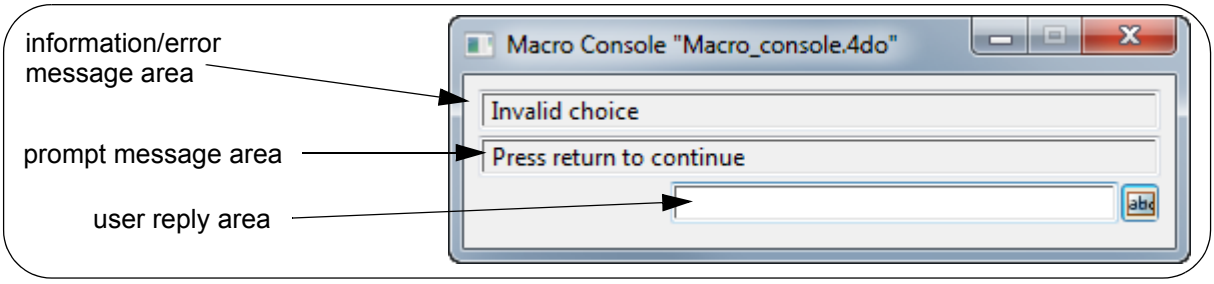
Error_prompt(Text msg)

Name

Integer Error_prompt(Text msg)

Description

Print the message **msg** to the **information/error message area** of the Macro Console, and writes *Press return to continue* to the **prompt message area** and then waits for an <enter> in the **user reply area** before the macro continues.



A function return value of zero indicates the function terminated successfully.

ID = 419

Choice_prompt(Text msg,Integer no_choices,Text choices[],Text &ret)

Name

Integer Choice_prompt(Text msg,Integer no_choices,Text choices[],Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the choice icon at the right hand end of the **user reply area**, **user reply area**, the list of text given in the Text array **choices** is placed in a pop-up. If one of the choices is selected from the pop-up (using LB), the choice is placed in the **user reply area**.

The reply, either typed or selected from the choice pop-up, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the text is returned successfully.

ID = 421

File_prompt(Text msg,Text wild_card_key,Text &ret)

Name

Integer File_prompt(Text msg,Text wild_card_key,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the folder icon at the right hand end of the **user reply area**, a list of all files in

the current area which match the **wild_card_key** (for example, *.dat) is placed in a pop-up. If a file is selected from the pop-up (using LB), the file name is placed in the **user reply area**.

If a name is entered without a dot ending (e.g. fred and not fred.csv say) then the ending after the dot in the **wild_card_key** is automatically added to the name.

For example, if **wild_card_key** = "*.rpt" and "fred" is type in as the file name, then **ret** will be returned as **ret** = "fred.rpt".

The reply, either typed or selected from the file pop-up, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 405

Model_prompt(Text msg,Text &ret)

Name

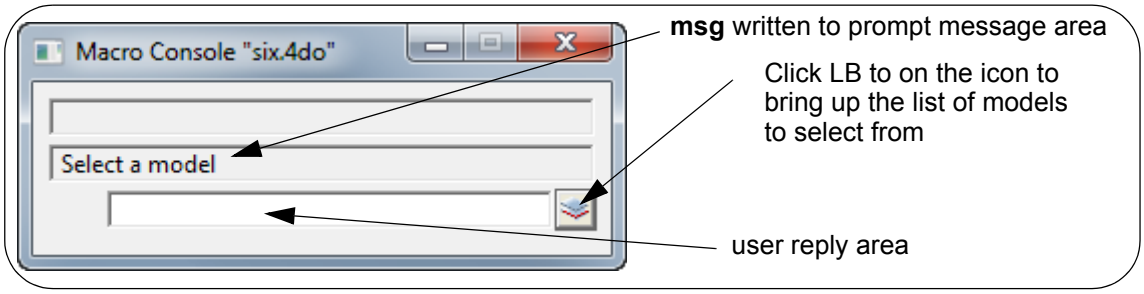
Integer Model_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the icon at the right hand end of the **user reply area**, a list of all existing models is placed in a pop-up. If a model is selected from the pop-up (using LB), the model name is placed in the **user reply area**.

MB for "Same As" also applies. That is, If MB is clicked in the **user reply area** and then a string from a model on a view is selected, then the name of the model containing the selected string is written to the **user reply area**.



The reply, either typed or selected from the model pop-up or Same As, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text **ret** is returned successfully.

ID = 401

Template_prompt(Text msg,Text &ret)

Name

Integer Template_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is pressed on the icon at the right hand end of the **user reply area**, a list of all existing templates is placed in a pop-up. If a template is selected from the pop-up (using LB), the template name is placed in the **user reply area**.

The reply, either typed or selected from the template popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the text is returned successfully.

ID = 403

Tin_prompt(Text msg,Text &ret)

Name

Integer Tin_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the tin icon at the right hand end of the **user reply area**, a list of all existing tins is placed in a pop-up. If a tin is selected from the pop-up (using LB), the Tin name is placed in the user reply area.

The reply, either typed or selected from the Tin popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 402

Tin_prompt(Text msg,Integer mode,Text &ret)

Name

Integer Tin_prompt(Text msg,Integer mode,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the tin icon at the right hand end of the **user reply area**, a list of all existing tins is placed in a pop-up. If a tin is selected from the pop-up (using LB), the Tin name is placed in the **user reply area**.

The value of mode determines whether Super Tins are listed in the pop-up.

Mode	Description
0	Don't list SuperTin.
1	List SuperTin.

The reply, either typed or selected from the Tin pop-up, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 684

View_prompt(Text msg,Text &ret)

Name

Integer View_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the view icon at the right hand end of the **user reply area**, a list of all existing views is placed in a pop-up. If a view is selected from the pop-up (using LB), the view name is placed in the **user reply area**.

The reply, either typed or selected from the view popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 406

Yes_no_prompt(Text msg,Text &ret)

Name

Integer Yes_no_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the choice icon at the right hand end of the **user reply area**, a yes/no pop-up is placed on the screen. If **yes** or **no** is selected from the pop-up (using LB), the selected test is placed in the **user reply area**.

The reply, either typed or selected from the yes/no popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 420

Plotter_prompt(Text msg,Text &ret)

Name

Integer Plotter_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the plotter icon at the right hand end of the **user reply area**, a list of all existing plotters is placed in a pop-up. If a plotter is selected from the pop-up (using LB), the plotter name is placed in the **user reply area**.

The reply, either typed or selected from the plotter popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 817

Sheet_size_prompt(Text msg,Text &ret)

Name

Integer Sheet_size_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the user reply area of the Macro Console.

If LB is clicked on the choice icon at the right hand end of the **user reply area**, a list of all existing sheet sizes is placed in a pop-up. If a sheet size is selected from the pop-up (using LB), the sheet size name is placed in the **user reply area**.

The reply, either typed or selected from the sheet_size popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 818

Linestyle_prompt(Text msg,Text &ret)

Name

Integer Linestyle_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the linestyle icon at the right hand end of the **user reply area**, a list of all existing linestyles is placed in a pop-up. If a linestyle is selected from the pop-up (using LB), the linestyle name is placed in the **user reply area**.

The reply, either typed or selected from the linestyle popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 819

Textstyle_prompt(Text msg,Text &ret)

Name

Integer Textstyle_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the textstyle icon at the right hand end of the **user reply area**, a list of all existing textstyles is placed in a pop-up. If a textstyle is selected from the pop-up (using LB), the textstyle name is placed in the **user reply area**.

The reply, either typed or selected from the textstyle popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 820

Justify_prompt(Text msg,Text &ret)

Name

Integer Justify_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the user reply area of the Macro Console.

If LB is clicked on the choice icon at the right hand end of the **user reply area**, a list of all existing justifications is placed in a pop-up. If a Justify is selected from the pop-up (using LB), the Justify name is placed in the **user reply area**.

The reply, either typed or selected from the Justify popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 821

Angle_prompt(Text msg,Text &ret)

Name

Integer Angle_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the user reply area of the Macro Console.

If LB is clicked on the angle icon at the right hand end of the **user reply area**, a list of Angle measure options is placed in a pop-up. If a Angle is selected from the pop-up (using LB), the Angle name is placed in the **user reply area**.

The reply, either typed or selected from the Angle popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 822

Function_prompt(Text msg,Text &ret)

Name

Integer Function_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the function icon at the right hand end of the **user reply area**, a list of all existing 12d Model Functions is placed in a pop-up. If a Function is selected from the pop-up (using LB), the Function name is placed in the **user reply area**.

The reply, either typed or selected from the Function popup, must be terminated by pressing

<Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 823

Project_prompt(Text msg,Text &ret)

Name

Integer Project_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the icon at the right hand end of the **user reply area**, a list of all existing Projects in the folder is placed in a pop-up. If a Project is selected from the pop-up (using LB), the Project name is placed in the **user reply area**.

The reply, either typed or selected from the Project popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 824

Directory_prompt(Text msg,Text &ret)

Name

Integer Directory_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the folder icon at the right hand end of the **user reply area**, the Select Folder dialogue is opened. If a Folder is selected by clicking on it with LB and then clicking on the Select Folder button, the Folder name is placed in the **user reply area**.

The reply, either typed or selected from the Select Folder dialogue, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 825

Text_units_prompt(Text msg,Text &ret)

Name

Integer Text_units_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the choice icon at the right hand end of the **user reply area**, a list of all existing Text units is placed in a pop-up. If a Text_units is selected from the pop-up (using LB), the Text

units name is placed in the **user reply area**.

The reply, either typed or selected from the Text_units popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 826

XYZ_prompt(Text msg,Real &x,Real &y,Real &z)

Name

Integer XYZ_prompt(Text msg,Real &x,Real &y,Real &z)

Description

Print the message **msg** to the **prompt message area** and then read back what must be x-value y-value z- value with the values separated by one or more spaces.

If LB is clicked on the pick icon at the right hand end of the **user reply area**, an XYZ pick is started and when a pick is made, the coordinates of the pick, separated by spaces, are written in the **user reply area**.

The reply, either typed or selected from the Pick, must be terminated by pressing <Enter> for the macro to continue.

The values are returned in **x, y** and **z**.

A function return value of zero indicates values x, y and z are successfully returned.

ID = 827

Name_prompt(Text msg,Text &ret)

Name

Integer Name_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the Name icon at the right hand end of the **user reply area**, a list of all existing Names is placed in a pop-up. If a Name is selected from the pop-up (using LB), the Name is placed in the user reply area.

The reply, either typed or selected from the Name popup, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

ID = 828

Panel_prompt(Text panel_name, Integer interactive, Integer no_field,Text field_name[], Text field_value[])

Name

Integer Panel_prompt(Text panel_name,Integer interactive,Integer no_field,Text field_name[],Text field_value[])

Description

Pop up a panel of the name **panel_name**.

No_field specifies how many fields you wish to fill in for the panel.

The name of each field is specified in **Field_name** array.

The value of each field is specified in **field_value** array.

If **interactive** is 1, the panel is displayed and remains until the finish button is selected.
If **interactive** is 0, the panel is displayed, runs the option and then closes.

A function return value of zero indicates success.

See example Defining and Using Panel_prompt

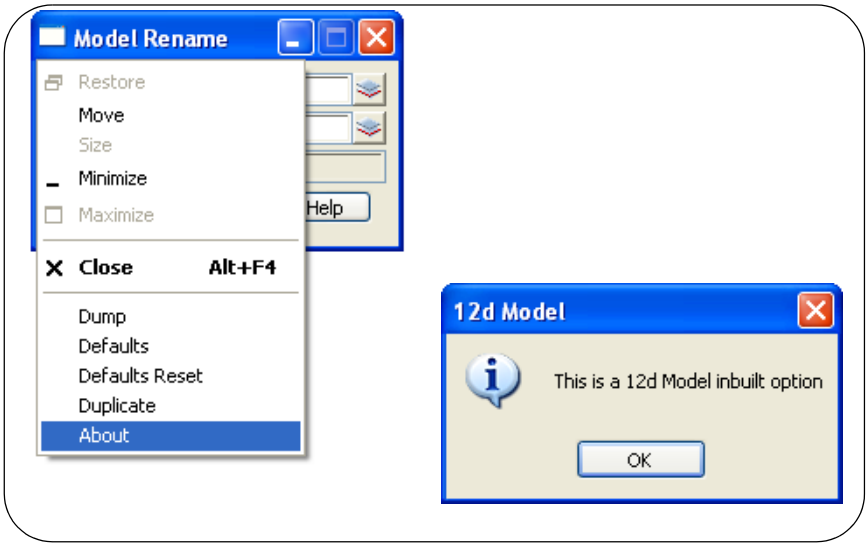
ID = 685

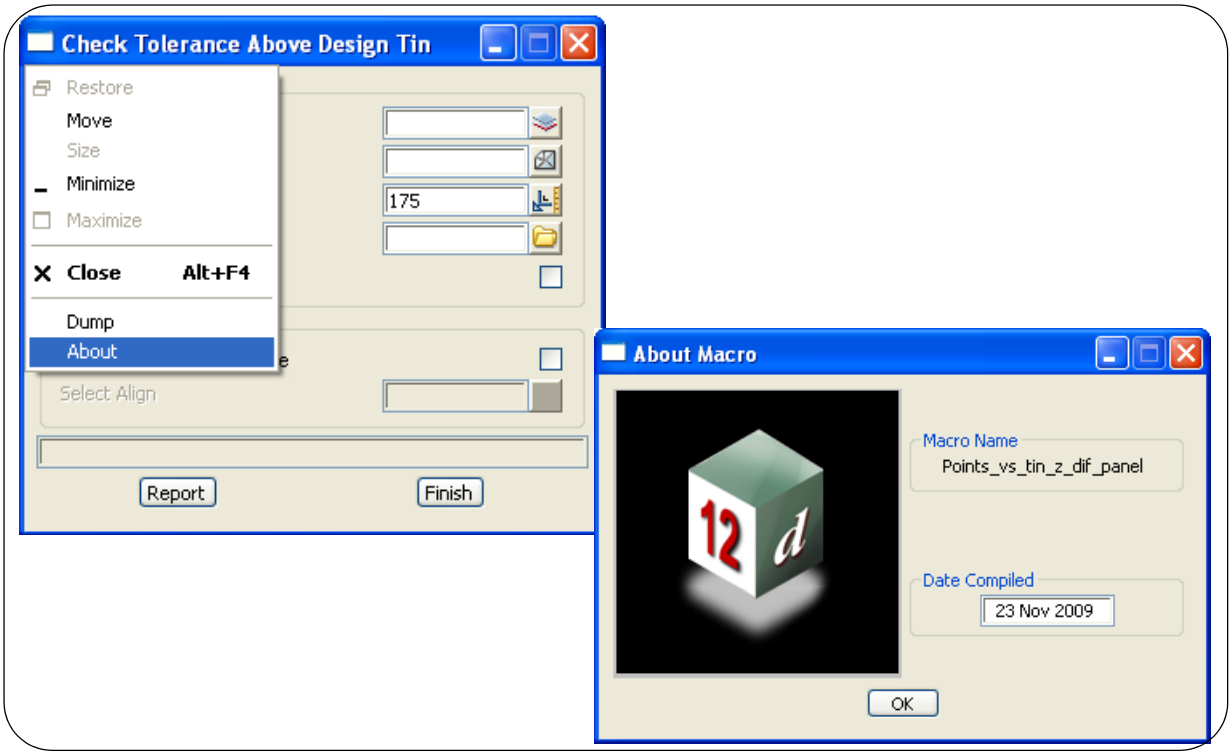
Defining and Using Panel_prompt

```
Text    panel_name;
Integer interactive = 1;
Integer no_fields;
Integer code;
Text    field_name [20];
Text    field_value[20];

panel_name = "Contour a Tin";
no_fields = 0;
no_fields++;  field_name[no_fields] = "Tin to contour";
field_value[no_fields] = "terrain";
no_fields++;  field_name[no_fields] = "Model for conts";
field_value[no_fields] = "terrain contours";
no_fields++;  field_name[no_fields] = "Cont min";
field_value[no_fields] = "";
no_fields++;  field_name[no_fields] = "Cont max";
field_value[no_fields] = "";
no_fields++;  field_name[no_fields] = "Cont inc";
field_value[no_fields] = "0.5";
no_fields++;  field_name[no_fields] = "Cont ref";
field_value[no_fields] = "0.0";
no_fields++;  field_name[no_fields] = "Cont colour";
field_value[no_fields] = "purple";
no_fields++;  field_name[no_fields] = "Model for bolds";
field_value[no_fields] = "terrain bold contours";
no_fields++;  field_name[no_fields] = "Bold inc";
field_value[no_fields] = "2.5";
no_fields++;  field_name[no_fields] = "Bold colour";
field_value[no_fields] = "orange";
Prompt("Contouring");

code = Panel_prompt(panel_name,interactive,no_fields,field_name,field_value);
```

Panels and Widgets

The user can build panels in the *12d Model* Programming Language (12dPL) that replicates the look and feel, and much of the functionality, of standard *12d Model* panels. Even in *12d Model* there are many options that are written in 12dPL and in most cases, the only way to tell if a panel is an inbuilt *12d Model* panel or is a 12dPL panel is by clicking on the Windows button on the top left hand side of a panel and then selecting **About**.





Panels are made up of **Widgets** and most panels have:

(a) Panel title

(a) Simple Input/Output widgets such a *Tin_Box*, *Model_Box and Named_Tick_Box*. These Widgets usually have their own validation methods and are often linked to special *12d Model* objects such as *Tins*, *Models* and *Linestyles* so that lists of pop-ups to choose from,

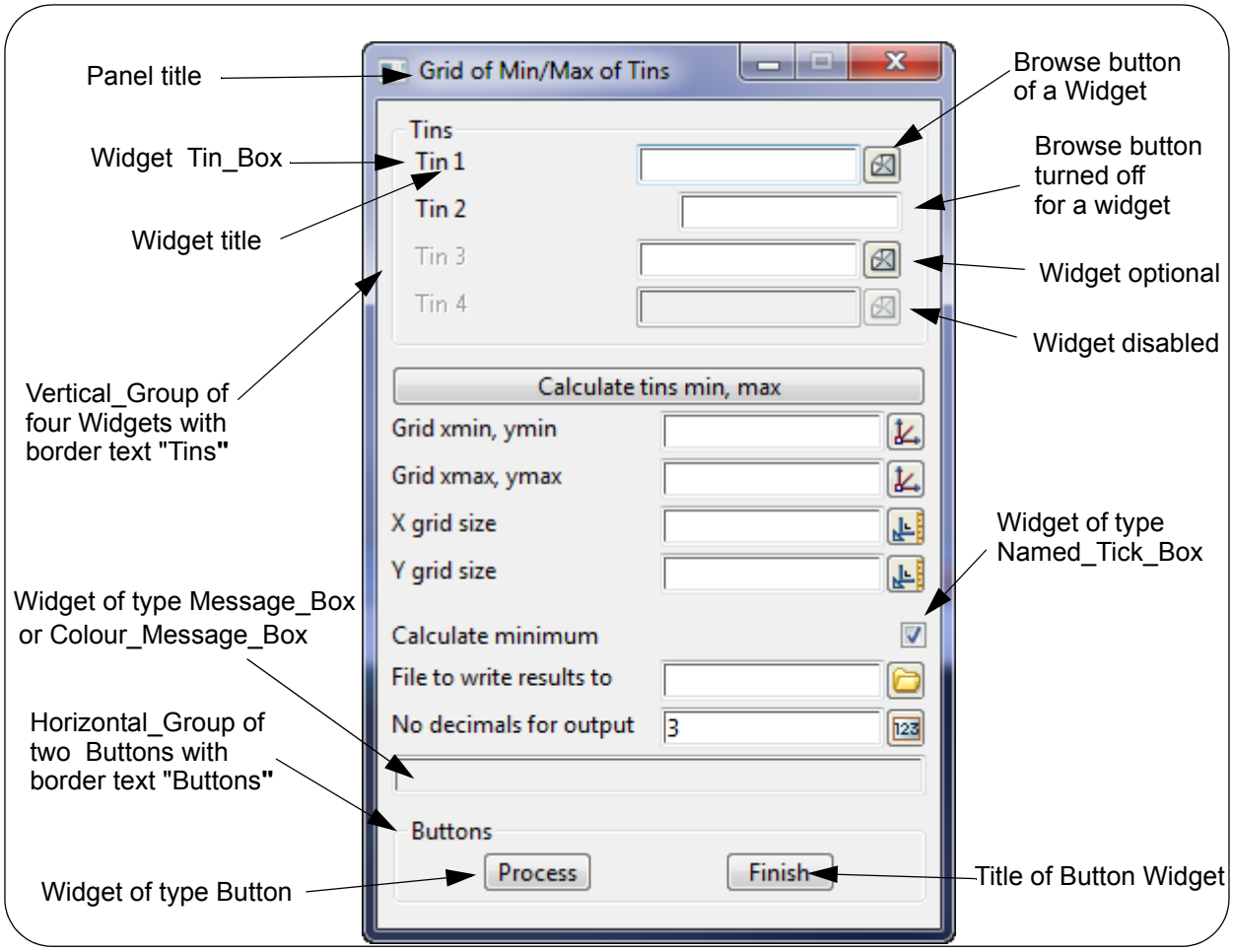
and special validations can be done by *12d Model* rather than having to be done in the macro.

(b) More complex Widgets such as Draw Boxes, Sliders, Log Boxes, Trees and Grids.

(c) A panel Message Area. Usually one Message_Box for writing messages for the user.

(d) Buttons such as *Process* or *Finish.* Unlike Input Widget, or Trees, or Grids, Buttons usually consist of just their Title and a Reply message that it sent back to the macro when the Button is pressed.

The Widgets can be built up in horizontal or vertical groups. Widgets inside a Group are automatically spaced out by 12d Model.



Once the Panel is constructed, it is displayed on screen by calling *Show_widget(Panel panel)*.

Programming for panels is more complicated than for simple sequential programs using say a Console because for panels the program is ***event driven***.

That is, once the panel is displayed, the user is not very constrained and can fill in Input boxes in any order, click on any Buttons in any order.

The programmer's code has to watch and cover all these possibilities.

The Widgets in the Panel have to be checked and validated whenever a user works with one of them.

And when the Button to start the processing of the Panel is finally pushed, all the Widgets have to be checked/validated again because you can't be sure which ones have been filled in/not filled in correctly.

Once the panel is constructed and displayed using *Show_widget,* the program normally has to sit and wait, watching what events the user triggers.

This is achieved in the macro by calling the *Wait_on_widgets(Integer &id,Text &cmd,Text &msg)*. The macro then sits and waits until an activated Widget returns control back to the macro and passes information about what has happened via the id, cmd and msg arguments of *Wait_on_widgets*. See Wait_on_widgets(Integer &id,Text &cmd,Text &msg) .

What messages are returned through *Wait_on_widgets* depends on each Widget in the panel.

The Screen_Text sends no messages at all.

Widgets such as the Integer_Box and Real_Box send keystrokes when each character is typed into their information area.

Other Widgets, such as the Tin_Box, control what characters can be typed into their information area and only valid characters are passed back via Wait_on_widgets.

For example, for a Tin_Box, only valid tin name characters are passed back. Invalid tin name characters are rejected by the Tin_Box itself and typing them does not even display anything but just produces a warning bell.

Some Widgets such as the Draw_Box and Select_Box can be very chatty.

For a Draw_Box: as the mouse is moved around the Draw_Box, a "mouse_move" command with a message containing the Draw_Box coordinates are returned via

Wait_on_widgets(draw_box_id,"mouse_move",draw_box coordinates of mouse as text)

plus "hover" commands when the mouse is in the Draw_Box and not moving, and a "mouse_leave" command when the mouse leaves the Draw_Box.

For New_Select_Box and Select_Box: after the Pick button is selected, whenever the mouse moves around a view, a "motion select" command with view coordinates of the mouse as part of the text message, are passed back via *Wait_on_widgets*.

These evens are returned in case the macro wants to use the coordinates to do something.

Buttons just sit there and only return the command (that is supplied by the programmer) via *Wait_on_widgets* when the button is pressed.

So the process for monitoring a panel is very chatty and normally is controlled why setting a *While* loop watching a variable to stop the loop.

A snippet of code to watch *Wait_on_widgets* is:

```
 Integer doit = 1;
 while(doit) {
// Process events from any of the Widgets on the panel
   Integer ret = Wait_on_widgets(id,cmd,msg);
            . . .
//       somewhere in here doit must be set to 0 (or a jump made to outside the loop)
//       or the loop will go on forever
 }
```

After the *Wait_on_widgets(id,cmd,msg)* call, the *id* of the Widget, and/or the command *cmd*, and/or the message *msg* can be interrogated to see what action is required by the program.

For example, a more of the code could be:

```
 Integer doit = 1;
 while(doit) {
 // Process events from any of the Widgets on the panel
    Integer ret = Wait_on_widgets(id,cmd,msg);
    if(cmd == "keystroke") continue;      // only a keystroke; go back and wait for more
```

```
switch(id) {                    // check which Widget was activated by checking the Widget id
case Get_id(panel) : {          // the case when the id belongs to the Widget panel
   if(cmd == "Panel Quit") doit = 0;     // case when click on X on top right of the panel
//                                       // set doit to 0 so the While loop will terminate
 break;
case Get_id(finish) : {         // the id belongs to the Button finish
   if(cmd == "finish") doit = 0;
} break;
case Get_id(process) : {   // the id belongs to the Button process. Start doing the work
                           // but first check the validity of all the relevant data in the panel
. . .
```

The important commands and messages for each Widget are given in the introductory section for each Widget.

Note: To quickly see what, and how many, commands and messages are generated whilst in a macro panel, insert a print line after Wait_on_widgets(id,cmd, msg). For example:

```
Wait_on_widgets(id,cmd,msg);
Print("id= " + To_text(id) +" cmd=<" + cmd + ">" +" msg=<" + msg + ">\n");
```

The best way to get an understanding of the event driven process is to look at examples of working macros that have panels in them. For example, see Examples 11 to 15 in the examples section Examples .

For information on creating Panels and the Widgets that make up panels:

See Cursor Controls
See Panel Functions
See Widget Controls
See Widget Information Area Menu
See Horizontal Group
See Vertical Group
See Widget Tooltip and Help Calls
See Panel Page
See Input Widgets
See Message Boxes
See Log_Box and Log_Lines
See Buttons
See GridCtrl_Box
See Tree Box Calls

Cursor Controls

Get_cursor_position(Integer &x,Integer &y)

Name

Integer Get_cursor_position(Integer &x,Integer &y)

Description

Get the cursor position (x,y).

The units of x and y are screen units (pixels).

The type of x and y must be **Integer**.

A function return value of zero indicates the position was returned successfully.

ID = 1329

Set_cursor_position(Integer x,Integer y)

Name

Integer Set_cursor_position(Integer x,Integer y)

Description

Set the cursor position with the coordinates (**x, y**).

The units of x and y are screen units (pixels).

A function return value of zero indicates the position was successfully set.

ID = 1330

Panel Functions

Create_panel(Text title_text)

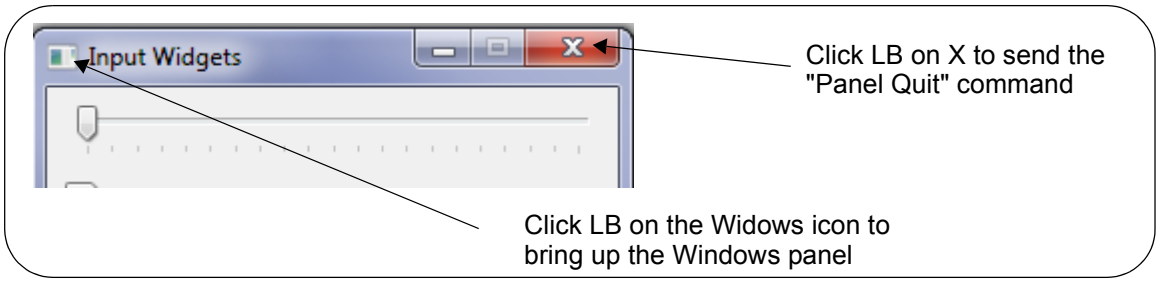
Name

Panel Create_panel(Text title_text)

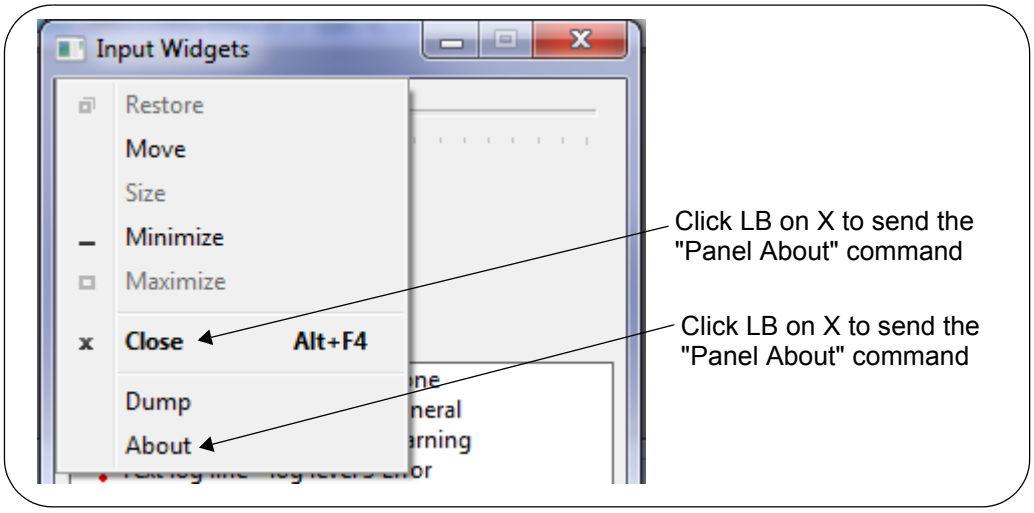
Description

Create a panel with the title **title_text**.

If LB is clicked on the X on the top right corner of the panel, the text "Panel Quit" is returned as the *cmd* argument to *Wait_on_widgets*.



If LB is clicked on the Windows icon on the top left hand corner of the panel,



See Wait_on_widgets(Integer &id,Text &cmd,Text &msg) .

For an example of a panel with Widgets Tin_Box, Buttons, Message_Box and Horizontal and Vertical Groups etc, see Panel Example:

The function return value is the created Panel.

Note: the *Show_widget(Panel panel) call must be made to* display the panel on the screen - see Panel Example: .

ID = 843

Append(Widget widget,Panel panel)

Name

Integer Append(Widget widget,Panel panel)

Description

Append the Widget **widget** to the Panel **panel**.

The Panel displays the Widgets from the top in the *order* that the Widgets are Appended to the Panel. That is, the first Widget appended is at the top of the Panel. The last Widget appended is at the bottom of the Widget.

Rather than a Panel having just a simple structure of a number of Widgets appended to the Panel, Horizontal and Vertical grouping can be used to collect the Widgets together in logical fashions and then the Horizontal and Vertical groups are Appended to the Panel using this *Append(Widget widget, Panel panel)* call. There are even more complicated groupings allowed including Panel pages, Grid Controls and Trees.

See Horizontal Group , Vertical Group , Panel Page , GridCtrl_Box ,Tree Box Calls

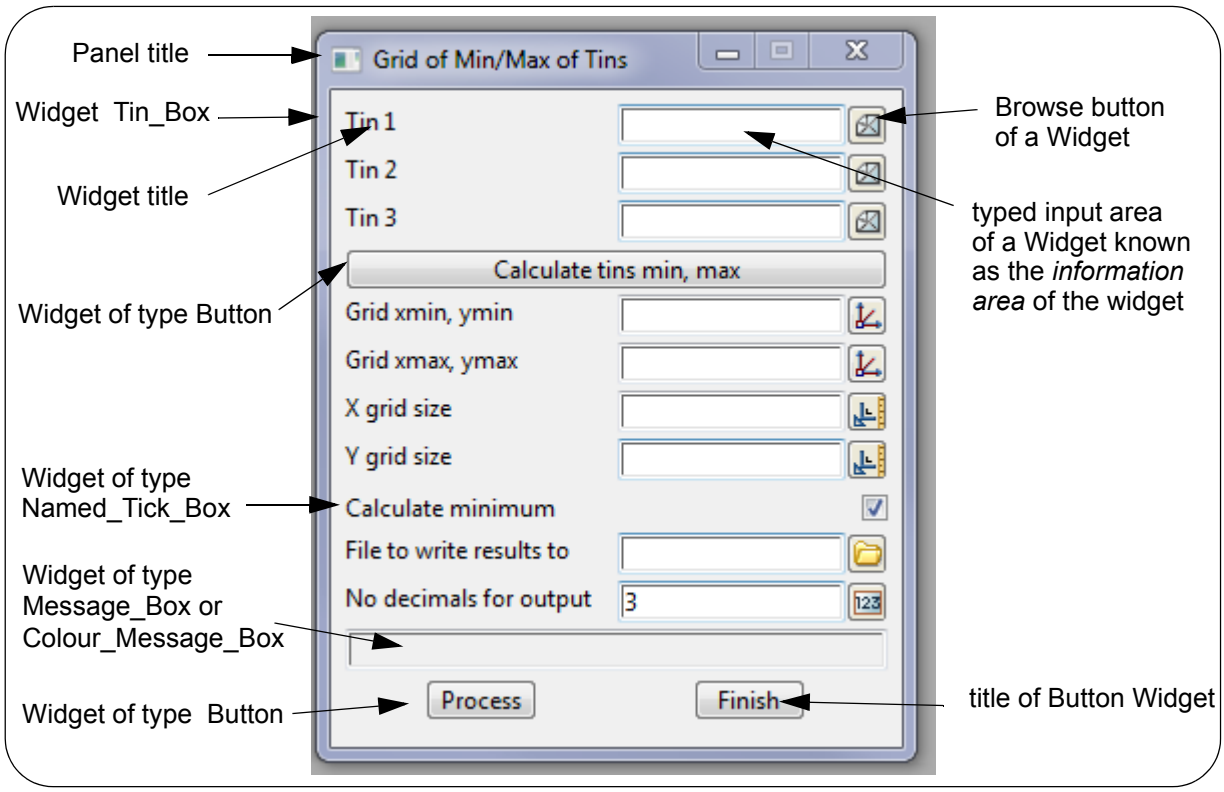
A function return value of zero indicates the widget was appended successfully.

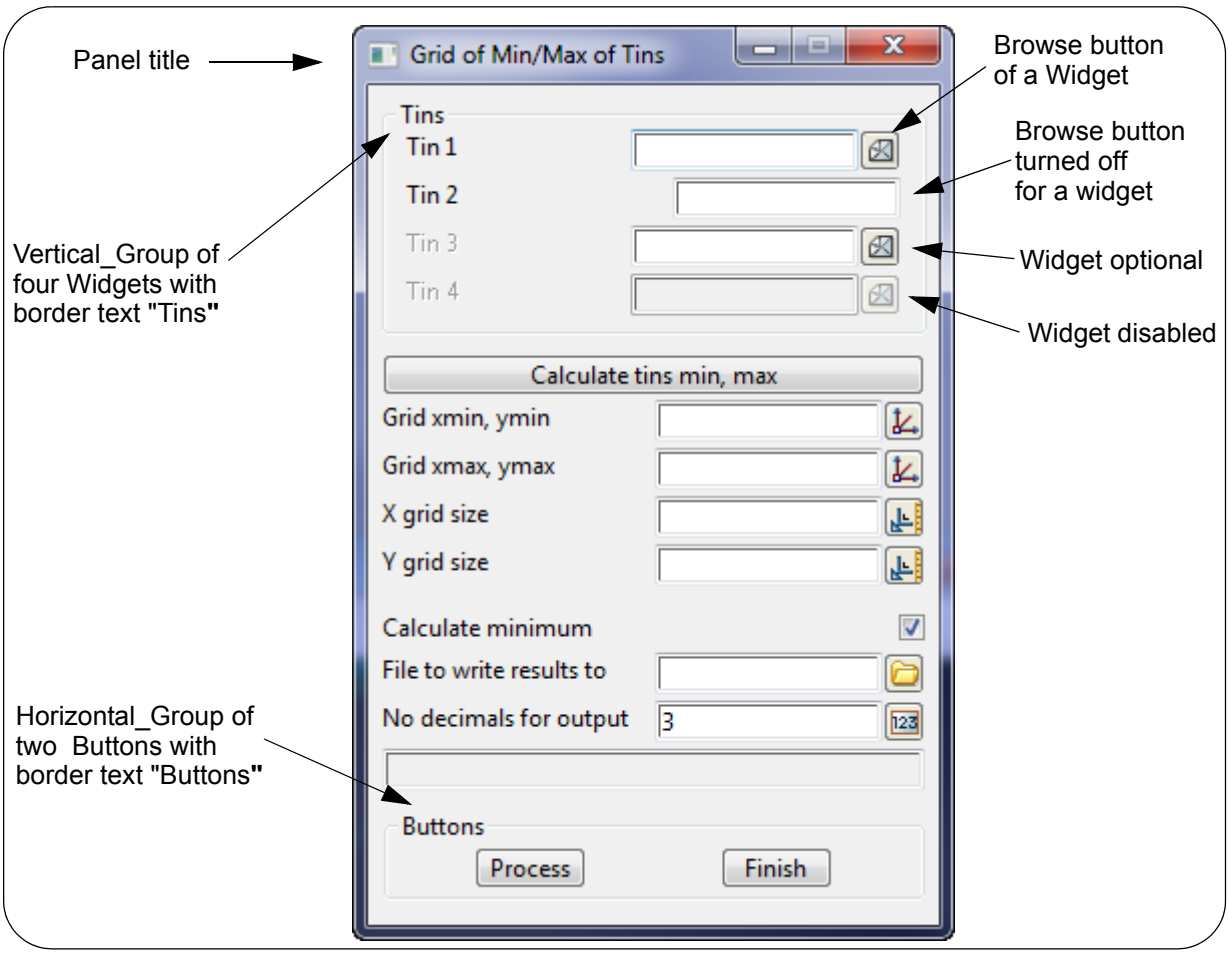
For an example of a panel with Widgets Tin_Box, Buttons, Message_Box and Horizontal and Vertical Groups etc, see Panel Example:

ID = 852

Panel Example:

```
Panel     panel    = Create_panel("Grid of Min/Max of Tins");
Show_widget(panel);
```




Panel title
Grid of Min/Max of Tins
Browse button of a Widget
Browse button turned off for a widget
Widget optional
Widget disabled
Vertical_Group of four Widgets with border text "Tins"
Tins
Tin 1
Tin 2
Tin 3
Tin 4
Calculate tins min, max
Grid xmin, ymin
Grid xmax, ymax
X grid size
Y grid size
Calculate minimum
File to write results to
No decimals for output
3
Horizontal_Group of two Buttons with border text "Buttons"
Buttons
Process
Finish

Horizontal Group

A Horizontal_Group is used to collect a number of Widgets together.

The Widgets are added to the Horizontal_Group using the *Append(Widget widget,Horizontal_Group group)* call.

The Widgets are automatically spaced horizontally in the order that they are appended.

Horizontal_Group Create_horizontal_group(Integer mode)

Name

Horizontal_Group Create_horizontal_group(Integer mode)

Description

Create a Widget of type **Horizontal_Group**.

A Horizontal_Group is used to collect a number of Widgets together. The Widgets are added to the Horizontal_Group using the *Append(Widget widget,Horizontal_Group group)* call. The Widgets are automatically spaced horizontally in the order that they are appended.

mode has the values (defined in set_ups.h)

// modes for Horizontal_Group (note -1 is also allowed)

For BALANCE_WIDGETS_OVER_WIDTH = 1

the widgets in the horizontal group are all given the same width and are evenly spaced horizontally. So the widgets all have the size of what the largest widget needed.

For ALL_WIDGETS_OWN_WIDTH = 2

the widgets in the horizontal group are all their own size all.

For COMPRESS_WIDGETS_OVER_WIDTH = 4

.

The function return value is the created **Horizontal_Group**.

ID = 845

Horizontal_Group Create_button_group()

Name

Horizontal_Group Create_button_group()

Description

Create a Widget of type Horizontal_Group to hold Widgets of type Button.

A Horizontal_Group is used to collect a number of Widgets together. The Widgets are added to the Horizontal_Group using the *Append(Widget widget,Horizontal_Group group)* call. The Widgets are automatically spaced horizontally in the order that they are appended.

The *Create_button_group* goes a bit further than *Create_horizontal_group* in making the button spacing more even.

The function return value is the created **Horizontal_Group**.

ID = 846

Append(Widget widget,Horizontal_Group group)

Name

Integer Append(Widget widget,Horizontal_Group group)

Description

Append the Widget **widget** to the Horizontal_Group **group**.

A Horizontal_Group is used to collect a number of Widgets together and the Widgets are added to the Horizontal_Group using this call. The Widgets are automatically spaced horizontally in the order that they are appended.

A function return value of zero indicates the Widget was appended successfully.

ID = 853

Set_border(Horizontal_Group group,Text text)

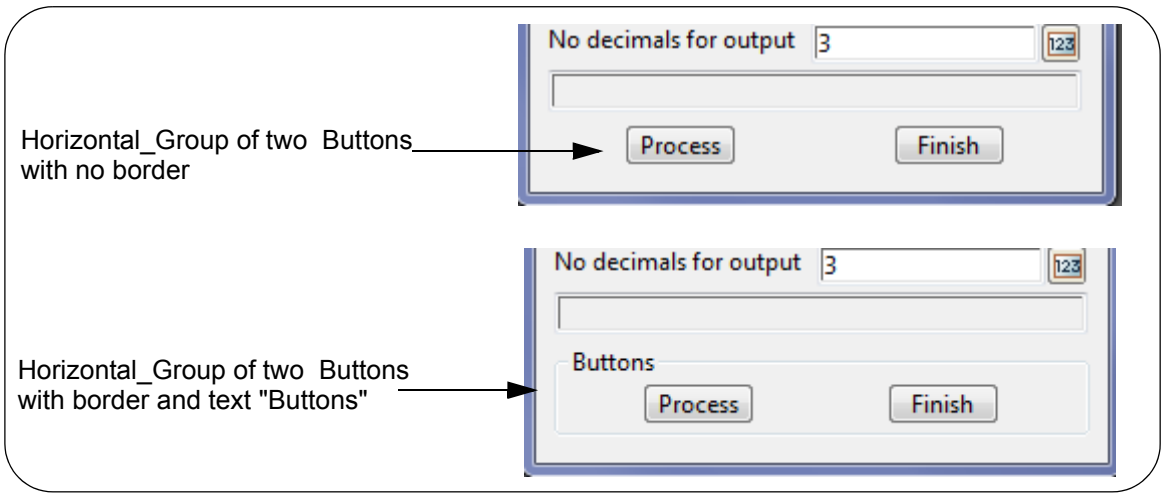
Name

Integer Set_border(Horizontal_Group group,Text text)

Description

Set a border for the Horizontal_Group **group** with Text **text**.on the top left side of the border.

If text is blank, the border is removed.



A function return value of zero indicates the border was successfully set.

ID = 1098

Set_border(Horizontal_Group group,Integer bx,Integer by)

Name

Integer Set_border(Horizontal_Group group,Integer bx,Integer by)

Description

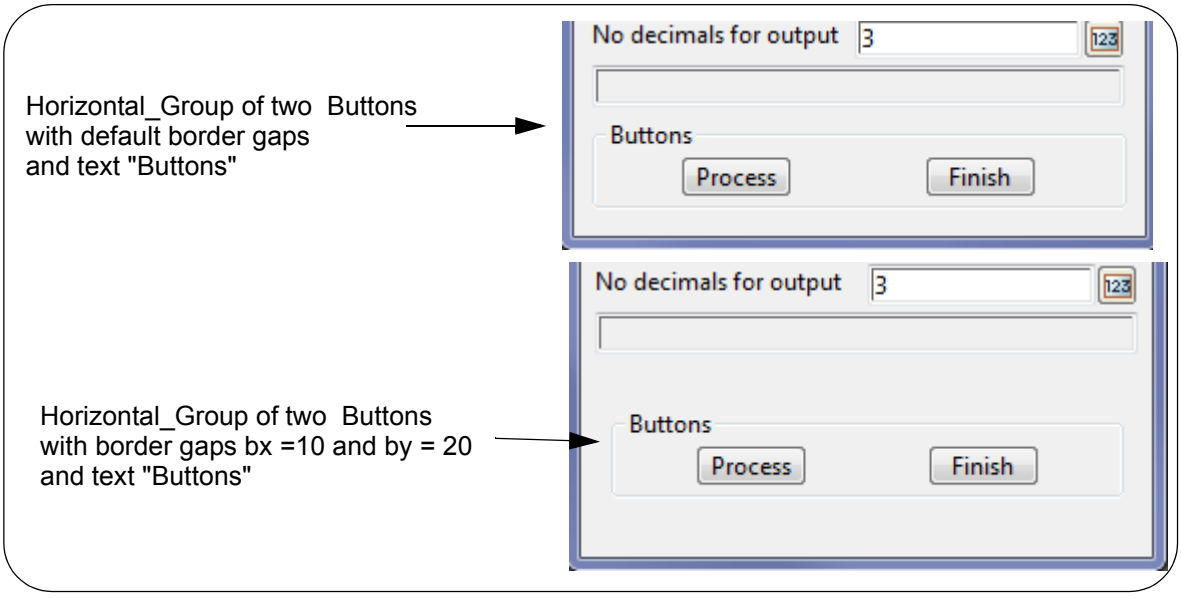
Set a gap around the border of the Horizontal_Group **group.**

bx sets the left and right side gap around the border.

by sets the top and bottom side gap around of the border.

The units of bx and by are screen units (pixels).

A function return value of zero indicates the border gap was successfully set.



ID = 858

Set_gap(Horizontal_Group group,Integer gap)

Name

Integer Set_gap(Horizontal_Group group,Integer gap)

Description

Set a horizontal gap of at least **gap** screen units (pixels) between the Widgets of the Horizontal_Group **group.**

A function return value of zero indicates the vertical gap was successfully set.

ID = 1506

Vertical Group

A Vertical_Group is used to collect a number of Widgets together.

The Widgets are added to the Vertical_Group using the *Append(Widget widget,Vertical_Group group)* call.

All the Widgets appended to the Vertical_Group are given the same width. The Widgets are automatically spaced vertically in the order that they are appended to the Vertical_Group.

Vertical_Group Create_vertical_group(Integer mode)

Name

Vertical_Group Create_vertical_group(Integer mode)

Description

Create a widget of type Vertical_Group.

A Vertical_Group is used to collect a number of Widgets together. The Widgets are added to the Vertical_Group using the *Append(Widget widget,Vertical_Group group)* call. All the Widgets appended to the Vertical_Group are given the same width. The Widgets are automatically spaced vertically in the order that they are appended to the Vertical_Group.

mode has the values (defined in set_ups.h)

// modes for Vertical_Group (note -1 is also allowed)

For BALANCE_WIDGETS_OVER_HEIGHT = 1
the widgets in the vertical group are evenly spaced vertically.

For ALL_WIDGETS_OWN_HEIGHT = 2

For ALL_WIDGETS_OWN_LENGTH = 4

The function return value is the created Vertical_Group.

ID = 844

Append(Widget widget,Vertical_Group group)

Name

Integer Append(Widget widget,Vertical_Group group)

Description

Append the Widget widget to the Vertical_Group group.

A function return value of zero indicates the widget was appended successfully.

ID = 854

Set_border(Vertical_Group group,Text text)

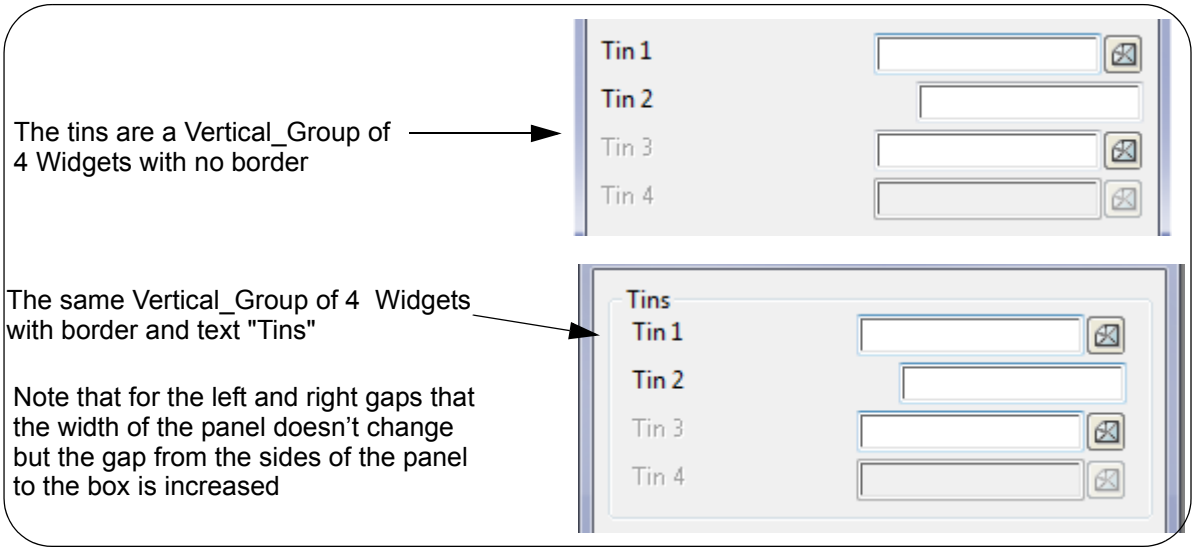
Name

Integer Set_border(Vertical_Group group,Text text)

Description

Set a border of the Vertical_Group **group** with Text text.on the top left side of the border. If text is blank, the border is removed.

A function return value of zero indicates the border was successfully set.



ID = 1099

Set_border(Vertical_Group group,Integer bx,Integer by)

Name

Integer Set_border(Vertical_Group group,Integer bx,Integer by)

Description

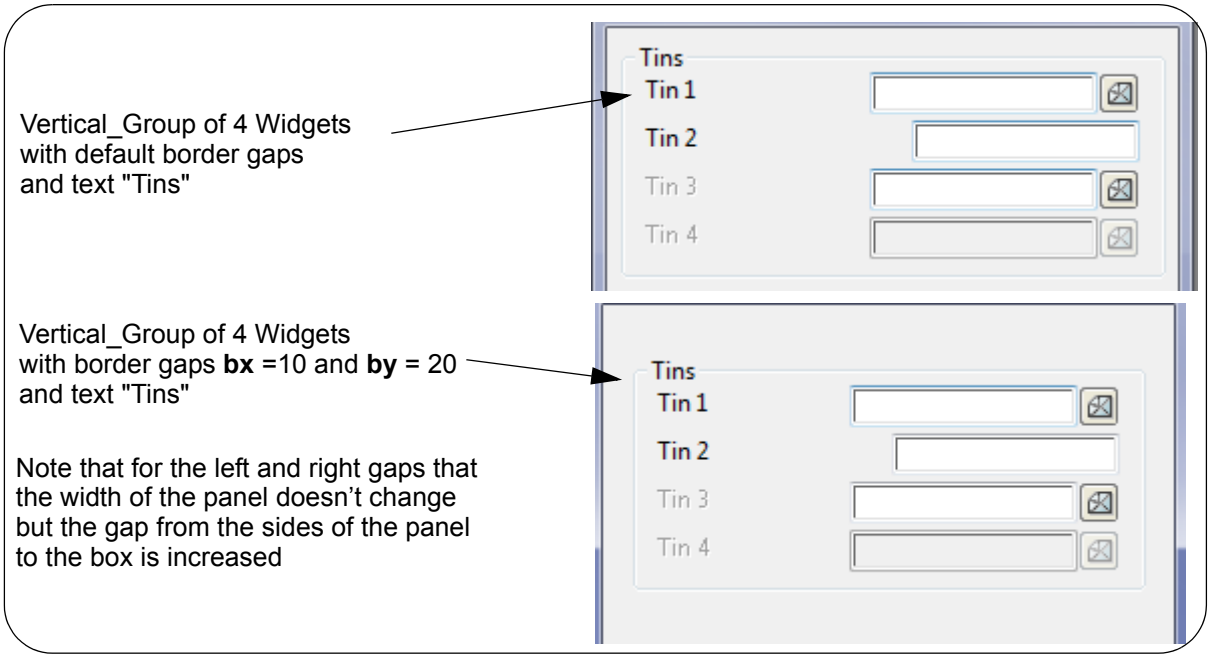
Set a gap around the border of the Vertical_Group **group.**

bx sets the left and right side gap around the border.

by sets the top and bottom side gap around of the border.

The units of **bx** and **by** are screen units (pixels).

A function return value of zero indicates the border gap was successfully set.



ID = 859

Set_gap(Vertical_Group group,Integer gap)

Name

Integer Set_gap(Vertical_Group group,Integer gap)

Description

Set a vertical gap of at least **gap** screen units (pixels) between the Widgets of the Vertical_Group **group.**

A function return value of zero indicates the vertical gap was successfully set.

ID = 1507

Widget Controls

Wait_on_widgets(Integer &id,Text &cmd,Text &msg)

Name

Integer Wait_on_widgets(Integer &id,Text &cmd,Text &msg)

Description

When the user activates a Widget displayed on the screen (for example by clicking on a Button Widget), the **id**, **cmd** and **msg** from the widget is passed back to *Wait_on_widgets*.

id is the id of the Widget that has been activated.

cmd is the command text that is returned from the Widget.

msg is the message text that is returned from the Widget.

A function return value of zero indicates the data was successfully returned.

Note: for a Button, the returned **cmd** is the Text **reply** given when the Button was created. See Create_button(Text title_text,Text reply) .

ID = 857

Use_browse_button(Widget widget,Integer mode)

Name

Integer Use_browse_button(Widget widget,Integer mode)

Description

Set whether the browse button is available for Widget **widget**.

If **mode** = 1 use the browse button
if **mode** = 0 don't use the browse button.

The default value for a Widget is mode = 1.

If the browse button is not used, the space where the button would be, is removed.

Note: This call must be made **before** the Panel that contains the widget is shown.

A function return value of zero indicates the value was valid.

ID = 1095

Show_browse_button(Widget widget,Integer mode)

Name

Integer Show_browse_button(Widget widget,Integer mode)

Description

This calls you to show or hide the browse button for the Widget **widget.**

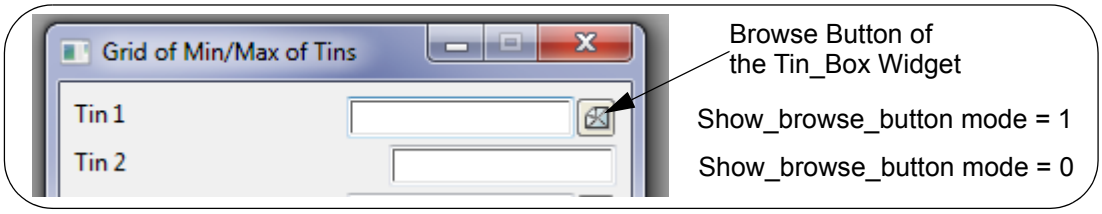
If **mode** = 1 show the browse button
if **mode** = 0 don't show the browse button.

The default value for a Widget is mode = 1.

This call can be made after the Widget has been added to a panel and allows the Browse button of the Widget to be turned on and off under the programmers control.

Note if Use_browse_button was called with a mode of 0 then this call is ineffective. *See* Use_browse_button(Widget widget,Integer mode)

A function return value of zero indicates the mode was successfully set.



ID = 1096

Set_enable(Widget widget,Integer mode)

Name

Integer Set_enable(Widget widget,Integer mode)

Description

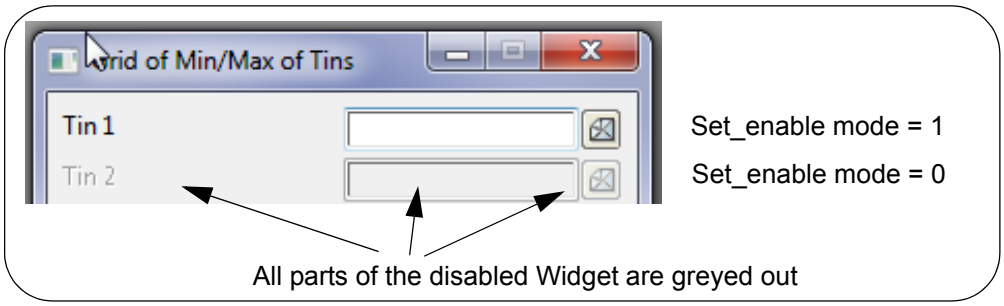
Set the enabled **mode** for the Widget **widget**.

If **mode** = 1 the Widget is to be enabled
mode = 0 the Widget is not to be enabled.

The default value for a Widget is mode = 1.

Note If the widget is not enabled, it will be greyed out in the standard Windows fashion and no interaction with the Widget is possible.

A function return value of zero indicates the **mode** was successfully set.



ID = 1101

Get_enable(Widget widget,Integer &mode)

Name

Integer Get_enable(Widget widget,Integer &mode)

Description

Check if the Widget **widget** is enabled or disabled. *See* Set_enable(Widget widget,Integer

mode)

Return the Integer **mode** where

mode = 1 if the Widget is enabled
mode = 0 if the Widget is not enabled.

A function return value of zero indicates the **mode** was returned successfully.

ID = 1100

Set_optional(Widget widget,Integer mode)

Name

Integer Set_optional(Widget widget,Integer mode)

Description

Set the optional **mode** for the Widget **widget**.

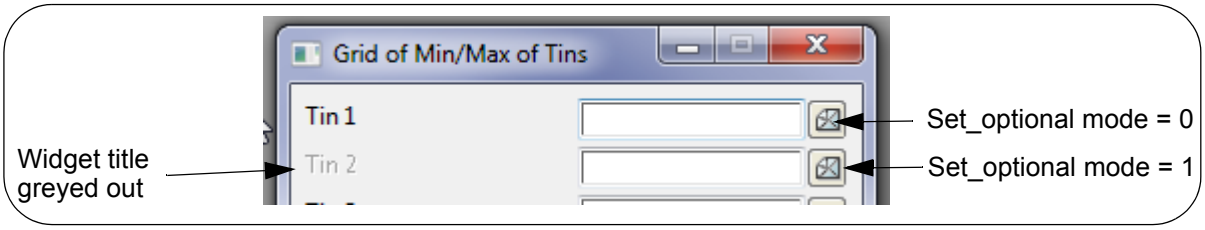
That is, if the Widget field is blank, the title text to the left is greyed out, signifying that this Widget is optional.

If **mode** = 1 the widget is optional
mode = 0 the widget is not optional.

The default value for a Widget is mode = 0.

If this mode is used (i.e. 1), the widget must be able to accept a blank response for the field, or assume a reasonable value.

A function return value of zero indicates the **mode** was successfully set.



Note: not all Widgets can be set to be optional.
For example Choice_Box, Named_Tick_Box, Source_Box,

ID = 1324

Get_optional(Widget widget,Integer &mode)

Name

Integer Get_optional(Widget widget,Integer &mode)

Description

Check if the Widget **widget** is optional. That is, the Widget does not have to be answered.*See* Set_optional(Widget widget,Integer mode)

Return the Integer **mode** where

mode = 1 if the Widget is optional
mode = 0 if the Widget is not optional.

A function return value of zero indicates the **mode** was returned successfully.

ID = 1325

Set_visible(Widget widget,Integer mode)

Name

Integer Set_visible(Widget widget,Integer mode)

Description

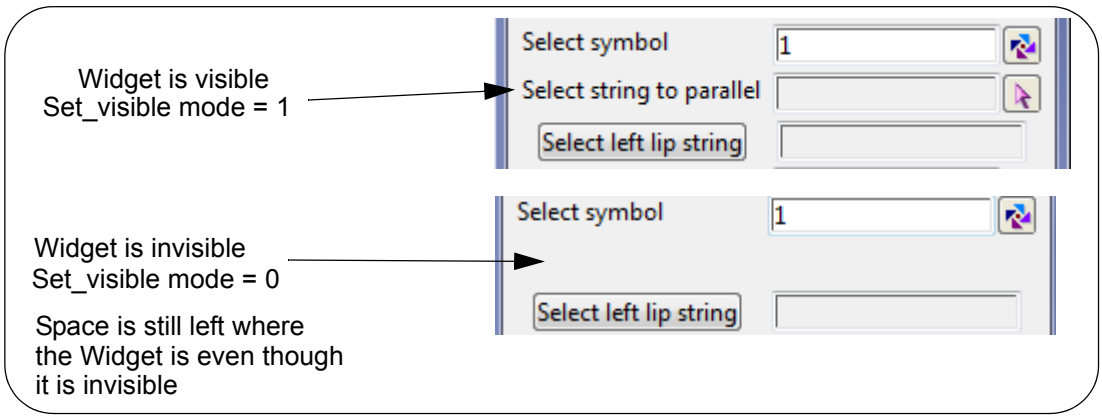
Set the visible **mode** for the Widget **widget**.

If **mode** = 1 the widget is visible, and not displayed on the panel
mode = 0 the widget is not visible and not displayed.

Even if the widget is invisible, it still takes the same space on a panel.

The default value for a Widget is visible. That is, mode = 1.

A function return value of zero indicates the visibility was successfully set.



ID = 1614

Get_visible(Widget widget,Integer &mode)

Name

Integer Get_visible(Widget widget,Integer &mode)

Description

Get the visibility **mode** for the Widget **widget**.

Return the Integer **mode** where

mode = 1 if the Widget is visible
mode = 0 if the Widget is not visible.

A function return value of zero indicates the visibility was returned successfully.

ID = 1615

Set_name(Widget widget,Text text)

Name

Integer Set_name(Widget widget,Text text)

Description

Set the title **text** of the Widget **widget**.

A Widget is usually given a title when it is first created This call can be made after the Widget has been added to a panel and allows the title of the Widget to be changed under the programmers

control.

A function return value of zero indicates the title was successfully set.

ID = 1326

Get_name(Widget widget,Text &text)

Name

Integer Get_name(Widget widget,Text &text)

Description

Get the title **text** from the Widget **widget**.

A function return value of zero indicates the **text** was returned successfully.

ID = 1327

Set_error_message(Widget widget,Text text)

Name

Integer Set_error_message(Widget widget,Text text)

Description

This call is used to set the error message for a Widget if it is validated and there is an error.

LJG ?

When there is an error, **text** is sent to the associated Message_Box of the **widget**, the focus is set to the widget and the cursor is moved to the widget.

A function return value of zero indicates the text was successfully set.

ID = 1437

Set_width_in_chars(Widget widget,Integer num_char)

Name

Integer Set_width_in_chars(Widget widget,Integer num_char)

Description

Set the Widget **widget** to be num_char characters wide.

A function return value of zero indicates the width was set successful.

ID = 1042

Show_widget(Widget widget)

Name

Integer Show_widget(Widget widget)

Description

Show the Widget **widget** at the **cursor's** current position.

Note: The call *Show_widget(Widget widget,Integer x,Integer y)* allows you to give the screen coordinates to position the Widget. *See* Show_widget(Widget widget,Integer x,Integer y) .

A function return value of zero indicates the **widget** was shown successfully.

ID = 855

Show_widget(Widget widget,Integer x,Integer y)

Name

Integer Show_widget(Widget widget,Integer x,Integer y)

Description

Show the Widget **widget** at the screen coordinates x, y. The units for x and y are pixels.

A function return value of zero indicates the **widget** was shown successfully.

ID = 1039

Hide_widget(Widget widget)

Name

Integer Hide_widget(Widget widget)

Description

Hide the Widget **widget**. That is, don't display the Widget on the screen.

Note the Widget still exists but it is not visible on the screen. The Widget will appear again by calling Show_widget. *See* Show_widget(Widget widget) .

A function return value of zero indicates the **widget** was hidden successfully.

ID = 856

Set_size(Widget widget,Integer x,Integer y)

Name

Integer Set_size(Widget widget,Integer x,Integer y)

Description

Set the size in screen units (pixels) of the Widget widget with the width x and height y.

The type of x and y must be **Integer**.

A function return value of zero indicates the size was successfully set.

ID = 1365

Get_size(Widget widget,Integer &x,Integer &y)

Name

Integer Get_size(Widget widget,Integer &x,Integer &y)

Description

Get the size in screen units (pixels) of the Widget **widget** in **x** and **y**.

The type of x and y must be **Integer**.

A function return value of zero indicates the size was returned successfully.

ID = 1331

Get_widget_size(Widget widget,Integer &w,Integer &h)

Name

Integer Get_widget_size(Widget widget,Integer &w,Integer &h)

Description

Get the size of the Widget **widget** in screen units (pixels)

The width of **widget** is returned in **w** and the height of **widget** is returned in **h**.

A function return value of zero indicates the size was successfully returned.

ID = 1041

Set_cursor_position(Widget widget)

Name

Integer Set_cursor_position(Widget widget)

Description

Move the cursor position to the Widget **widget**.

A function return value of zero indicates the position was successfully set.

ID = 1059

Get_widget_position(Widget widget,Integer &x,Integer &y)

Name

Integer Get_widget_position(Widget widget,Integer &x,Integer &y)

Description

Get the screen position of the Widget **widget**.

The position of the **widget** is returned in **x, y**. The units of x and y are screen units (pixels).

A function return value of zero indicates the position was successfully returned.

ID = 1040

Get_position(Widget widget,Integer &x,Integer &y)

Name

Integer Get_position(Widget widget,Integer &x,Integer &y)

Description

Get the screen position of the Widget **widget**.

The position of the **widget** is returned in **x, y**. The units of x and y are screen units (pixels).

A function return value of zero indicates the position was successfully returned.

ID = 1366

Get_id(Widget widget)

Name

Integer Get_id(Widget widget)

Description

When a Widget is created, it is given a unique identifying number (id) in the project.

This function get the *id* of the Widget **widget** and returns id as the function return value.

That is, the Integer function return value is the Widget **id**.

D = 879

Set_focus(Widget widget)

Name

Integer Set_focus(Widget widget)

Description

Set the focus to the typed input area for an Input Widget **widget**, or on the button for a Button Widget **widget**.

After this call all *typed input* will go to this widget.

A function return value of zero indicates the focus was successfully set.

ID = 1097

General Widget Commands and Messages

accept select
message: view_name

cancel select
message: blank

cut
message: blank

kill_focus
message: blank

keystroke
message: character typed in

left_button_up
message: blank

middle_button_up
message: blank

motion select
message: x y z a b view_name

This is returned whenever the cursor is over the exposed area of a ***12d Model*** View.

Panel Quit
message: blank

paste
message: information to be pasted

pick select
message: view_name

right_button_up
message: blank

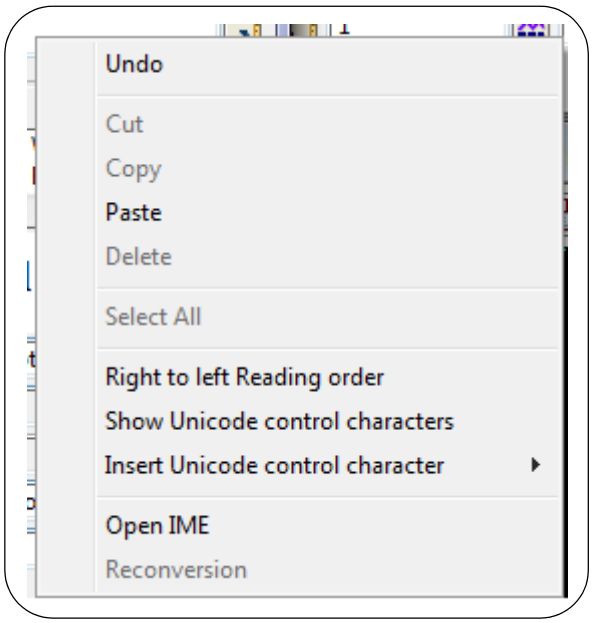
set_focus
message: blank

start select
message: blank

text selected
message: text typed in

Widget Information Area Menu

Clicking RB in the i***nformation area*** of most Widgets brings up the menu:



Picking *Cut* from the menu cuts the highlighted characters, and sends a "**cut**" command and nothing in message via *Wait_on_widgets*.

Picking *Copy* from the menu copies the highlighted characters into the paste buffer, and sends a "**copy**" command and the copied text in message via *Wait_on_widgets*.

Picking *Paste* from the menu pastes the paste buffer into the information area, and sends a "**paste**" command and the paste buffer in message via *Wait_on_widgets*.

Widget Tooltip and Help Calls

Set_tooltip(Widget widget,Text tip)

Name

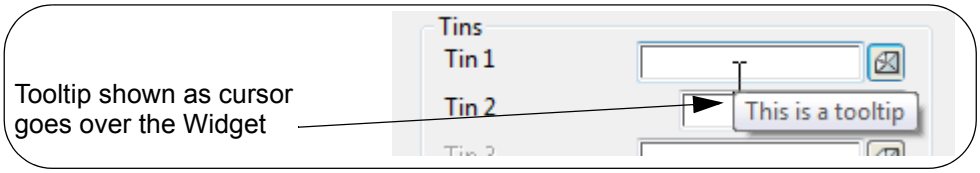
Integer Set_tooltip(Widget widget,Text tip)

Description

Sets the tool tip message for the Widget **widget** to **tip**.

When the user hovers over **widget**, this message **tip** will be displayed as a Windows tooltip.

A function return value of zero indicates the tooltip was successfully set.



ID = 1363

Get_tooltip(Widget widget,Text &tip)

Name

Integer Get_tooltip(Widget widget,Text &tip)

Description

Queries the current tool tip message and returns the message in **tip**.

A function return value of zero indicates the tooltip was successfully returned.

ID = 1364

Set_help(Widget widget,Integer help_num)

Name

Integer Set_help(Widget widget,Integer help_num)

Description

For the Widget **widget**, the help number for **widget** is set to **help_num**.

This is currently not used.

A function return value of zero indicates the help number was successfully set.

Note: See Help Button for creating a **Help** button that allows the macro to access the **12d Model** *Extra Help* system.

ID = 1312

Get_help(Widget widget,Integer &help_num)

Name

Integer Get_help(Widget widget,Integer &help_num)

Description

Get the help number for Widget **widget** and return it in **help_num**.

The type of **help** must be **integer**.

A function return value of zero indicates the help number was successfully returned.

Note: See Help Button for creating a **Help** button that allows the macro to access the **12d Model** *Extra Help* system.

ID = 1313

Set_help(Widget widget,Text help_message)

Name

Integer Set_help(Widget widget,Text help_message)

Description

For the Widget **widget**, the help message for **widget** is set to **help_message**.

This help message will be sent back to *12d Model* via *Wait_on_widgets(Integer &id,Text &cmd,Text &msg)* with command **cmd** equal to "Help", and **msg** equal to **help_message**.

So a sample bit of code to handle help is

```
Wait_on_widgets(id,cmd,msg);
if (cmd == "Help") {;
   Winhelp(panel,"12d.hlp",'a',msg);        // in the Winhelp file 12d.hlp,
                                            // find and display the a table entry msg
   continue;
}
```

A function return value of zero indicates the **text** was successfully set.

ID = 1314

Get_help(Widget widget,Text &help_message)

Name

Integer Get_help(Widget widget,Text &help_message)

Description

Queries the current help message for a widget and returns the message in **help_mesage**.

A function return value of zero indicates the message was successfully returned.

ID = 1315

Winhelp(Widget widget,Text help_file,Text key)

Name

Integer Winhelp(Widget widget,Text help_file,Text key)

Description

Calls the Windows help system to display the key from the k table of the Windows help file **help_file**. The Windows help file **help_file** must exist and be in a location that can be found.

A function return value of zero indicates the function was successful.

ID = 1316

Winhelp(Widget widget,Text help_file,Integer table,Text key)

Name

Integer Winhelp(Widget widget,Text help_file,Integer table,Text key)

Description

Calls the Windows help system to display the **key** from the named **table** of the help file **help_file**. **table** takes the form 'a', 'k' etc. The Windows help file **help_file** must exist and be in a location that can be found.

A function return value of zero indicates the function was successful.

ID = 1317

Winhelp(Widget widget,Text help_file,Integer help_id)

Name

Integer Winhelp(Widget widget,Text help_file,Integer help_id)

Description

Calls the Windows help system to display the **key** from the k table of the help file **help_file**. The Windows help file **help_file** must exist and be in a location that can be found.

A function return value of zero indicates the function was successful.

ID = 1318

Winhelp(Widget widget,Text help_file,Integer help_id,Integer popup)

Name

Integer Winhelp(Widget widget,Text help_file,Integer helpid,Integer popup)

Description

Calls the Windows help system to display the help with help number **help_id** from the k table of the help file help_file. The Windows help file **help_file** must exist and be in a location that can be found. The value **popup** is used to determine whether the help information appears as a popup style help or normal help.

LJG? what are the values for popup

A function return value of zero indicates the function was successful.

ID = 1319

Panel Page

Widget_Pages Create_widget_pages()

Name

Widget_Pages Create_widget_pages()

Description

A Widget_Pages object allows a number of controls to exist in the same physical location on a dialog. This is very handy if you want a field to change between a Model_Box, View_Box or the like.

A bit of sample code might look like,

```
Vertical_Group vgroup1 = Create_vertical_group(0);
Model_Box mbox = Create_model_box(…);
Append(mbox,vgroup1);

Vertical_Group vgroup2 = Create_vertical_group(0);
View_Box vbox = Create_view_box(…);
Append(vbox,vgroup2);
Widget_Pages pages = Create_widget_pages();
Append(vgroup1,pages);
Append(vgroup2,pages);
Set_page(page,1)                // this shows the 1st page - vgroup1
```

The function return value is the created **Widget_pages**.

ID = 1243

Append(Widget widget,Widget_Pages pages)

Name

Integer Append(Widget widget,Widget_Pages pages)

Description

Append Widget **widget** into the Widget_Pages **pages**.

For each item appended, another page is created.

If you want more than 1 item on a page, add each item to a Horizontal_Group, Vertical_Group.

A function return value of zero indicates the **widget** was appended successfully.

ID = 1244

Set_page(Widget_Pages pages,Integer n)

Name

Integer Set_page(Widget_Pages pages,Integer n)

Description

Show (display on the screen) the **n**'th page of the Widget_Pages **pages**.

Note the "**n**'th page" is the n'th widget appended to the Widget_Pages **pages**.

All the controls associated with the **n**'th page_no are shown.

A function return value of zero indicates the **page** was successfully set.

ID = 1245

Set_page(Widget_Pages pages,Widget widget)

Name

Integer Set_page(Widget_Pages pages,Widget widget)

Description

Show (display on the screen) the page of **pages** containing the Widget **widget**.

All the controls associated with the **widget** are shown.

A function return value of zero indicates the page was successfully set.

ID = 1606

Get_page(Widget_Pages pages,Widget widget,Integer &page_no)

Name

Integer Get_page(Widget_Pages pages,Widget widget,Integer &page_no)

Description

For the Widget_Pages **pages**, get the page number of the page containing the Widget **widget**.

Note the "**n**'th page" of a Widget_Pages is the n'th widget appended to the Widget_Pages.

The page n umber is returned as **page_no**.

A function return value of zero indicates the page number was successfully returned.

ID = 1607

Input Widgets

See Angle_Box
See Attributes_Box
See Texture_Box
See Bitmap_Fill_Box
See Chainage_Box
See Choice_Box
See Colour_Box
See Date_Time_Box
See Directory_Box
See Draw_Box
See File_Box
See Function_Box
See HyperLink_Box
See Input_Box
See Integer_Box
See Justify_Box
See Linestyle_Box
See List_Box
See Map_File_Box
See Model_Box
See Name_Box
See Named_Tick_Box
See New_Select_Box
See New_XYZ_Box
See Plotter_Box
See Polygon_Box
See Real_Box
See Report_Box
See Screen_Text
See Select_Box
See Select_Boxes
See Sheet_Size_Box
See Slider_Box
See Source_Box
See Symbol_Box
See Target_Box
See Template_Box
See Text_Style_Box
See Text_Units_Box
See Textstyle_Data_Box
See Text_Edit_Box
See Texture_Box
See Tick_Box
See Tin_Box
See View_Box
See XYZ_Box

Angle_Box

The ***Angle_Box*** is a panel field designed to take angle data and display it in degrees, minutes and seconds. If data is typed into the box, then it will be validated when <enter> is pressed.

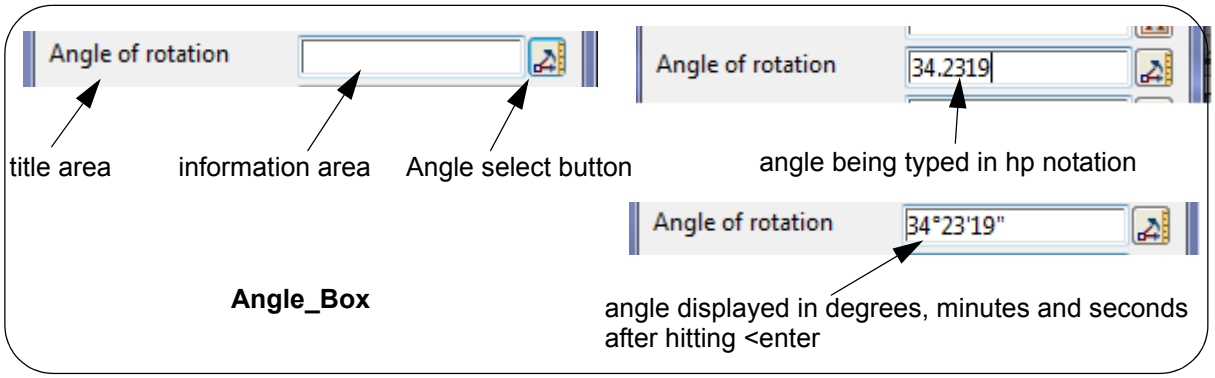
An ***Angle_Box*** is a made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in an angle or to display the angle if it is selected by the angle select button. This information area is in the middle

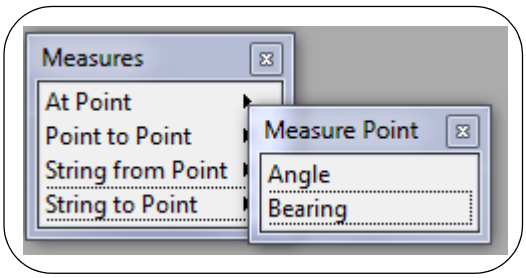
and

(c) an Angle select button on the right.



An angle can be typed into the ***information area*** in hp notation (ddd.mmss). Hitting the <enter> key will validate the angle and then display it in degree, minutes and seconds in the information area.

Clicking **LB** or **RB** on the Angle select button brings up the *Measure* pop-up menu in *Angle* mode. Selecting an option from the *Measure* menu and making a measure displays the angle in the information area.



Clicking **MB** on the Angle select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**real selected**" command and nothing in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu

are documented in the section Widget Information Area Menu .

Picking a value with the Angle Select button sends a "**real_selected**" command.

Create_angle_box(Text title_text,Message_Box message)

Name

Angle_Box Create_angle_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Angle_Box** for inputting and validating angles. See Angle_Box .

An angle is typed into the Angle_Box in hp notation (i.e. ddd.mmssss) but after it is validated it is displayed in degrees, minutes and seconds. However the validated angle is stored in the Angle_Box as a Real in **radians**.

The **Angle_Box** is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Angle_Box validation messages.

The function return value is the created **Angle_Box**.

ID = 886

Set_data(Angle_Box box,Real angle)

Name

Integer Set_data(Angle_Box box,Real angle)

Description

Set the data for the Angle_Box **box** to the Real value **angle**.

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

A function return value of zero indicates the data was successfully set.

ID = 888

Set_data(Angle_Box box,Text text_data)

Name

Integer Set_data(Angle_Box box,Text text_data)

Description

Set the text displayed in the Angle_Box **box** to the Text **text_data**.

Note that **text_data** should be in degrees, minutes and seconds using the hp notation (i.e. ddd.mmssss) BUT the text_data can be any text at all and may not even be a valid angle (in degrees in hp notation). This may lead to an error when the Angle_Box is validated.

A function return value of zero indicates the data was successfully set, even if the **text_data** will not validate.

ID = 1515

Get_data(Angle_Box box,Text &text_data)

Name

Integer Get_data(Angle_Box box,Text &text_data)

Get the actual text displayed in the Angle_Box **box** and return it in **text_data**.

Note that this is just the text in the Angle_Box. It may be any text at all and may not even be a valid angle (in degrees in hp notation). To get the validated data from the Angle_box, use *Validate*. See Validate(Angle_Box box,Real &angle) .

A function return value of zero indicates the data was successfully returned.

ID = 889

Validate(Angle_Box box,Real &angle)

Name

Integer Validate(Angle_Box box,Real &angle)

Description

Validate the contents of the Angle_Box **box** and return the angle in radians **angle.**

angle is in radians and is measured in a counterclockwise direction from the positive x-axis.

The function returns the value of:

NO_NAME if the Widget Angle_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 887

For information on the other Input Widgets, go to Input Widgets

Attributes_Box

Attributes_Box Create_attributes_box(Text title_text,Message_Box message)

Name

Attributes_Box Create_attributes_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Attributes_Box**. *See* Attributes_Box .

The Attributes_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Attribute_Box validation messages.

The function return value is the created Attributes_Box.

ID = 2210

Set_data(Attributes_Box box,Attributes &data)

Name

Integer Set_data(Attributes_Box box,Attributes &data)

Description

Set the data of type Attributes for the Attributes_Box **box** to **data**.

A function return value of zero indicates the data was successfully set.

ID = 2213

Set_data(Attributes_Box box,Text text_data)

Name

Integer Set_data(Attributes_Box box,Text text_data)

Description

Set the data of type Text for the Attributes_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 2214

Get_data(Attributes_Box box,Text &text_data)

Name

Integer Get_data(Attributes_Box box,Text &text_data)

Description

Get the data of type Text from the Attributes_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 2212

Validate(Attributes_Box box,Attributes &result)

Name

Integer Validate(Attributes_Box box,Attributes &result)

Description

Validate the contents of Attributes_Box **box** and return the Attributes in **result.**

The function returns the value of:

NO_NAME if the Widget Attributes_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 2211

For information on the other Input Widgets, go to Input Widgets

Billboard_Box

Billboard_Box Create_billboard_box(Text title_text,Message_Box message)

Name

Billboard_Box Create_billboard_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Billboard_Box**. *See* Billboard_Box .

The Billboard_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Billboard_Box validation messages.

The function return value is the created Billboard_Box.

ID = 1871

Set_data(Billboard_Box box,Text text_data)

Name

Integer Set_data(Billboard_Box box,Text text_data)

Description

Set the data of type Text for the Billboard_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1873

Get_data(Billboard_Box box,Text &text_data)

Name

Integer Get_data(Billboard_Box box,Text &text_data)

Description

Get the data of type Text from the Billboard_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1874

Validate(Billboard_Box box,Text &result)

Name

Integer Validate(Billboard_Box box,Text &result)

Description

Validate the contents of Billboard_Box **box** and return the name of the billboard in Text **result.**

The function returns the value of:

NO_NAME if the Widget Billboard_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1872

For information on the other Input Widgets, go to Input Widgets

Bitmap_Fill_Box

Create_bitmap_fill_box(Text title_text,Message_Box message)

Name

Bitmap_Fill_Box Create_bitmap_fill_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Bitmap_Fill_Box**. *See* Bitmap_Fill_Box .

The Bitmap_Fill_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Bitmap_Fill_Box validation messages.

The function return value is the created Bitmap_Fill_Box.

ID = 1879

Validate(Bitmap_Fill_Box box,Text &result)

Name

Integer Validate(Bitmap_Fill_Box box,Text &result)

Description

Validate the contents of Bitmap_Fill_Box **box** and return the name of the bitmap in Text **result.**

The function returns the value of:

NO_NAME if the Widget Bitmap_Fill_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1880

Set_data(Bitmap_Fill_Box box,Text text_data)

Name

Integer Set_data(Bitmap_Fill_Box box,Text text_data)

Description

Set the data of type Text for the Bitmap_Fill_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1881

Get_data(Bitmap_Fill_Box box,Text &text_data)

Name

Integer Get_data(Bitmap_Fill_Box box,Text &text_data)

Description

Get the data of type Text from the Bitmap_Fill_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1882

For information on the other Input Widgets, go to Input Widgets

Chainage_Box

The ***Chainage_Box*** is a panel field designed to enter chainages which normally just have to be Real numbers. If data is typed into the box, then it will be validated when <enter> is pressed.

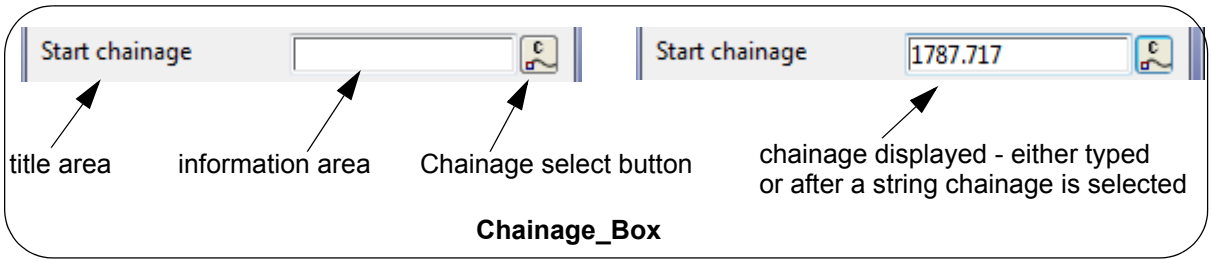
The ***Chainage_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area in the middle where the chainage is displayed

and

(c) a Chainage select button on the right.

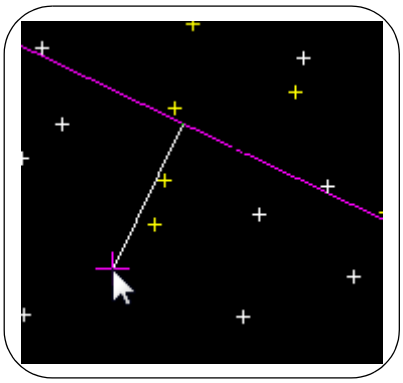


Chainage_Box

A chainage can be typed into the ***information area***. Then hitting the <enter> key will validate the chainage.

MB clicked in the ***information area*** starts a "Same As" selection. A string is then selected but at the moment, nothing else is done with it.

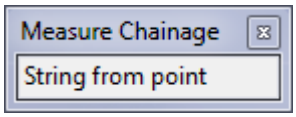
Clicking **LB** on the ***chainage select button*** starts a Measure chainage selection in the *String from point* mode. A string is then selected, and as the cursor is moved around the perpendicular drop to the selected string is displayed.



And when a final position selected, the chainage of that position dropped onto the selected string is then displayed in the information box.

Clicking **RB** on the ***chainage select button*** brings up the ***Measure Chainage*** pop-up with only the *String from point* choice available.

.



After selecting *String from point,* the action is the same as for **LB** described above.

Clicking **MB** on the ***Chainage select button*** does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.
Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**real selected**" command and nothing in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.
Pressing and releasing MB in the information area sends a "**middle_button_up**" command.
Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a value with the Chainage Select button sends a "**real_selected**" command.

Chainage_Box Create_chainage_box(Text title_text,Message_Box message)

Name

Chainage_Box Create_chainage_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Chainage_Box**. *See* Chainage_Box .

The Chainage_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Chainage_Box validation messages.

The function return value is the created Chainage_Box.

ID = 2203

Validate(Chainage_Box box,Real &result)

Name

Integer Validate(Chainage_Box box,Real &result)

Description

Validate the contents of Chainage_Box **box** and return the chainage in Real **result.**

The function returns the value of:

NO_NAME if the Widget Chainage_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 2204

Get_data(Chainage_Box box,Text &text_data)

Name

Integer Get_data(Chainage_Box box,Text &text_data)

Description

Get the data of type Text from the Chainage_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 2205

Set_data(Chainage_Box box,Real real_data)

Name

Integer Set_data(Chainage_Box box,Real real_data)

Description

Set the data of type Real for the Chainage_Box **box** to **real_data**.

A function return value of zero indicates the data was successfully set.

ID = 2206

Set_data(Chainage_Box box,Text text_data)

Name

Integer Set_data(Chainage_Box box,Text text_data)

Description

Set the data of type Text for the Chainage_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 2207

For information on the other Input Widgets, go to Input Widgets

Choice_Box

The ***Choice_Box*** is a panel field designed to select one item from a list of choices. If data is typed into the box, then it will be validated when <enter> is pressed.

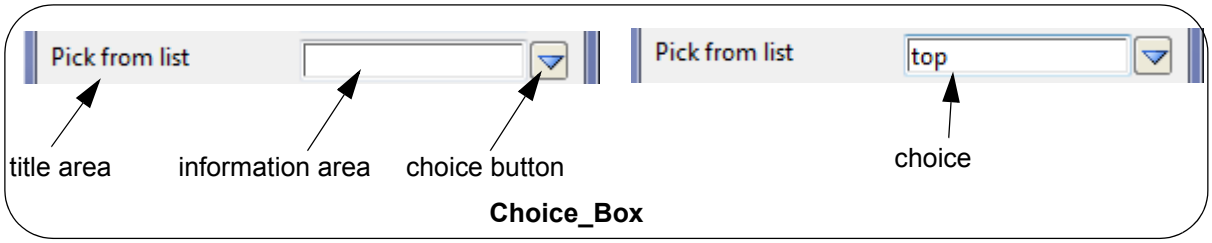
A ***Choice_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a choice name or to display a choice if it is selected by the choice select button. This information area is in the middle

and

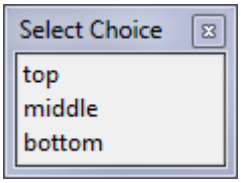
(c) a Choice button on the right.



Choice_Box

A choice can be typed into the ***information area*** and hitting the <enter> key will validate the choice. Note that to be valid, the typed in choice must exist in the Choice pop-up list.

Clicking **LB** or **RB** on the Choice button brings up the *Select Choice* pop-up list. Selecting a choice from the pop-up list writes the choice to the information area.



Clicking **MB** on the Choice button does nothing.

Note: the list of choices is defined by the call Set_data(Choice_Box box,Integer nc,Text choices[]) .

Note: A Choice_Box cannot be made optional

Create_choice_box(Text title_text,Message_Box message)

Name

Choice_Box Create_choice_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Choice_Box**. *See* Choice_Box .

The **Choice_Box** is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Choice_Box validation messages.

The function return value is the created **Choice_Box**.

ID = 890

Validate(Choice_Box box,Text &result)

Name

Integer Validate(Choice_Box box,Text &result)

Description

Validate the contents of Choice_Box **box** and return the Text **result.**

The function returns the value of:

NO_NAME if the Widget Choice_Box is optional and the box is left empty

1 if no other return code is needed and *result* is valid.

-1 if there is an invalid choice.

zero if there is a drastic error.

So a function return value of zero indicates that there is an error as well as other values.

Warning this is the opposite of most 12dPL function return values

Double Warning: most times the function return code is not zero even when you think it should be. The actual value of the function return code must be checked to see what is going on. For example, when there is an incorrect choice, the function return value is -2.

ID = 891

Get_data(Choice_Box box,Text &text_data)

Name

Integer Get_data(Choice_Box box,Text &text_data)

Description

Get the data of type Text from the Choice_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 893

Set_data(Choice_Box box,Text text_data)

Name

Integer Set_data(Choice_Box box,Text text_data)

Description

Set the data of type Text for the Choice_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 892

Set_data(Choice_Box box,Integer nc,Text choices[])

Name

Integer Set_data(Choice_Box box,Integer nc,Text choices[])

Description

Set the values available in the choice list. There are **nc** items in the **choices** list for the Choice_Box **box**.

For example

```
Text choices[3];
choices[1] = "top";
choices[2] = "middle";
choices[3] = "bottom";

Choice_Box choice_box = Create_choice_box("Pick from list",message);
Set_data(choice_box,3,choices);
```

Note: To be valid, any data typed into the Choice_Box information area must be from the **choices** list.

A function return value of zero indicates the **nc**'th data in the **choices** list was successfully set.

ID = 997

For information on the other Input Widgets, go to Input Widgets

Colour_Box

The ***Colour_Box*** is a panel field designed to select a *12d Model* colour. If data is typed into the box, then it will be validated when <enter> is pressed.

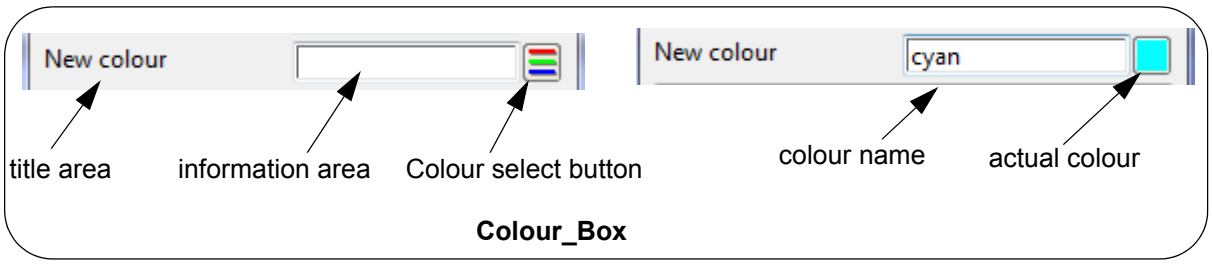
The ***Colour_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in the colour name or to display the colour name if it is selected by the colour select button. This information area is in the middle

and

(c) a Colour select button on the right.

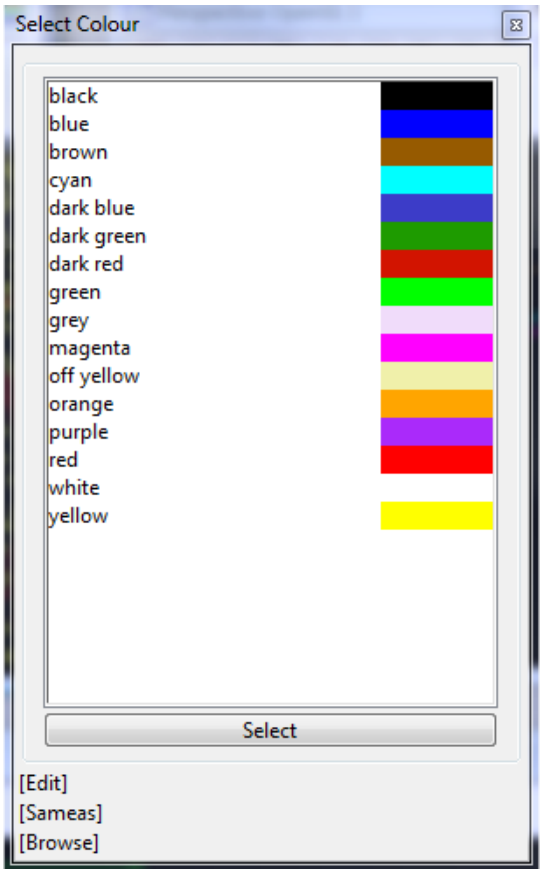


Colour_Box

A colour name can be typed into the ***information area***. Then hitting the <enter> key will validate the colour name and if it is a valid colour name, the actual colour is shown on the colour select button.

MB clicked in the ***information area*** starts a "Same As" selection. A string is then selected and the colour of the selected string is placed in the information area and the actual colour shown on the Colour select button.

Clicking **LB** or **RB** on the colour select button brings up the *Select Colour* pop-up. Selecting the colour from the pop-up list writes the colour in the information area and the actual colour is shown on the Colour select button.



Clicking **MB** on the colour select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a colour with the Colour Select button sends a "**text selected**" command and the colour name in *message*.

Create_colour_box(Text title_text,Message_Box message)

Name

Colour_Box Create_colour_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Colour_Box**. *See* Colour_Box .

The **Colour_Box** is created with the title **title_text**.

The Message_Box message is normally the message box for the panel and is used to display Colour_Box validation messages.

The function return value is the created **Colour_Box**.

ID = 894

Validate(Colour_Box box,Integer &col_num)

Name

Integer Validate(Colour_Box box,Integer &col_num)

Description

Validate the contents of Colour_Box **box** and return the Integer colour number I in **col_num.**

The function returns the value of:

NO_NAME if the Widget Colour_Box is optional and the box is left empty

-1 if the text in the Colour_Box is not a valid colour number or colour name.

TRUE (1) if no other return code is needed and *col_num* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error. For example, the Colour_Box is not optional and is left blank.

Warning this is the opposite of most 12dPL function return values

Double Warning the function return can be non zero but the col_num is unusable.

ID = 895

Set_data(Colour_Box box,Integer colour_num)

Name

Integer Set_data(Colour_Box box,Integer colour_num)

Description

Set the data for the Colour_Box **box** to be the colour number **colour_num**.

This is the colour number that will be first displayed in the Colour_Box.

colour_num must be **Integer**.

A function return value of zero indicates the colour number was successfully set.

ID = 896

Set_data(Colour_Box box,Text text_data)

Name

Integer Set_data(Colour_Box box,Text text_data)

Description

Set the data of type Text for the Colour_Box **box** to **text_data**.

This is the colour name that will be first displayed in the Colour_Box.

A function return value of zero indicates the data was successfully set.

ID = 1328

Get_data(Colour_Box box,Text &text_data)

Name

Integer Get_data(Colour_Box box,Text &text_data)

Description

Get the data of type Text from the Colour_Box **box** and return it in **text_data**.

This is the colour name entered into the Colour_Box.

A function return value of zero indicates the data was successfully returned.

ID = 897

For information on the other Input Widgets, go to Input Widgets

Date_Time_Box

Date_Time_Box Create_date_time_box(Text title_text,Message_Box message)

Name

Date_Time_Box Create_date_time_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Date_Time_Box**. *See* Date_Time_Box .

The Date_Time_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Date_Time_Box validation messages.

The function return value is the created Date_Time_Box.

ID = 1883

Validate(Date_Time_Box box,Text &data)

Name

Integer Validate(Date_Time_Box box,Text &data)

Description

Validate the contents of Date_Time_Box **box** and return the time in Text **data.**

The function returns the value of:

NO_NAME if the Widget Date_Time_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *data* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1884

Set_data(Date_Time_Box box,Text text_data)

Name

Integer Set_data(Date_Time_Box box,Text text_data)

Description

Set the data of type Text for the Date_Time_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1885

Get_data(Date_Time_Box box,Text &text_data)

Name

Integer Get_data(Date_Time_Box box,Text &text_data)

Description

Get the data of type Text from the Date_Time_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1886

Get_data(Date_Time_Box box,Integer &integer_data)

Name

Integer Get_data(Date_Time_Box box,Integer &integer_data)

Description

Get the data of type Integer from the Date_Time_Box **box** and return it in **integer_data**.

A function return value of zero indicates the data was successfully returned.

ID = 2284

Get_data(Date_Time_Box box,Real &real_data)

Name

Integer Get_data(Date_Time_Box box,Real &real_data)

Description

Get the data of type Real from the Date_Time_Box **box** and return it in **real_data**.

A function return value of zero indicates the data was successfully returned.

ID = 2286

For information on the other Input Widgets, go to Input Widgets

Directory_Box

The ***Directory_Box*** is a panel field designed to select or create, *disk folder*. If a folder name is typed into the box, then it will be validated when <enter> is pressed.

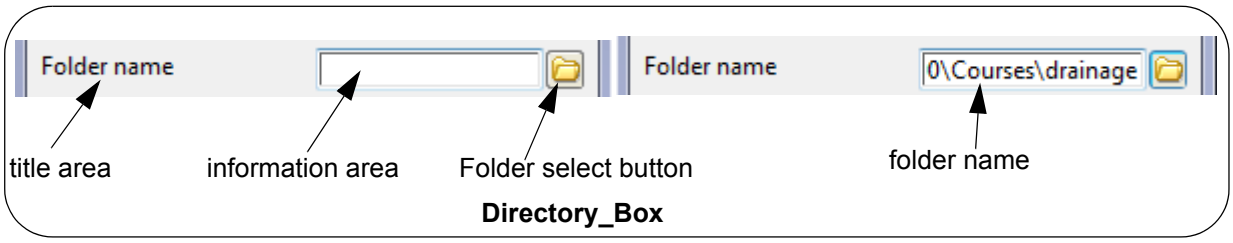
A ***Directory_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a folder name or to display the folder name if it is selected by the Folder select button. This information area is in the middle

and

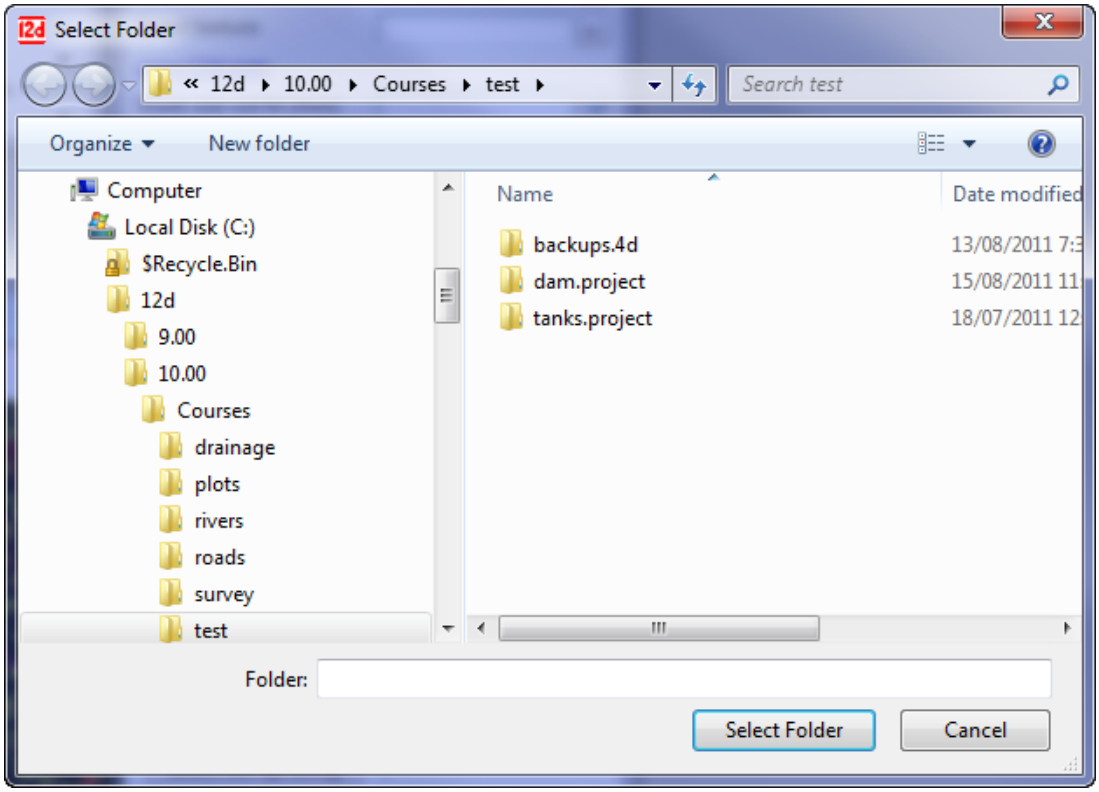
(c) a Folder select button on the right.



Directory_Box

A folder name can be typed into the ***information area***. Then hitting the <enter> key will validate the folder name.

Clicking **LB** or **RB** on the Folder select button brings up the *Select Folder* pop-up. Selecting a folder from the pop-up writes the folder name to the ***information area***.



Clicking **MB** on the Folder select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text

of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a folder with the Folder Select button sends three events:

a "**start_browse**" command with a blank *message*.
a " **text selected**" command and the full path name of the folder in *message*.
a "**finish_browse**" command with a blank *message*.

Create_directory_box(Text title_text,Message_Box message,Integer mode)

Name

Directory_Box Create_directory_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Directory_Box**. *See* Directory_Box .

The Directory_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Directory_Box validation messages.

The value of **mode** is listed in the Appendix A - Directory mode

The function return value is the created Directory_Box.

ID = 898

Validate(Directory_Box box,Integer mode,Text &result)

Name

Integer Validate(Directory_Box box,Integer mode,Text &result)

Description

Validate the contents of Directory_Box **box** and return the Text **result.**

The value of **mode** is listed in the Appendix A - Directory mode. *See* Directory Mode

The function returns the value of:

NO_NAME if the Widget Directory_Box is optional and the box is left empty

NO_DIRECTORY, DIRECTORY_EXISTS, or NEW_DIRECTORY.

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 899

Get_data(Directory_Box box,Text &text_data)

Name

Integer Get_data(Directory_Box box,Text &text_data)

Description

Get the data of type Text from the Directory_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 901

Set_data(Directory_Box box,Text text_data)

Name

Integer Set_data(Directory_Box box,Text text_data)

Description

Set the data of type Text for the Directory_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 900

For information on the other Input Widgets, go to Input Widgets

Draw_Box

The ***Draw_Box*** is a panel field designed to create an area for drawing by supplying the parameters **box_width** and **box_height**. The units of box_width and box_height are screen units (pixels).

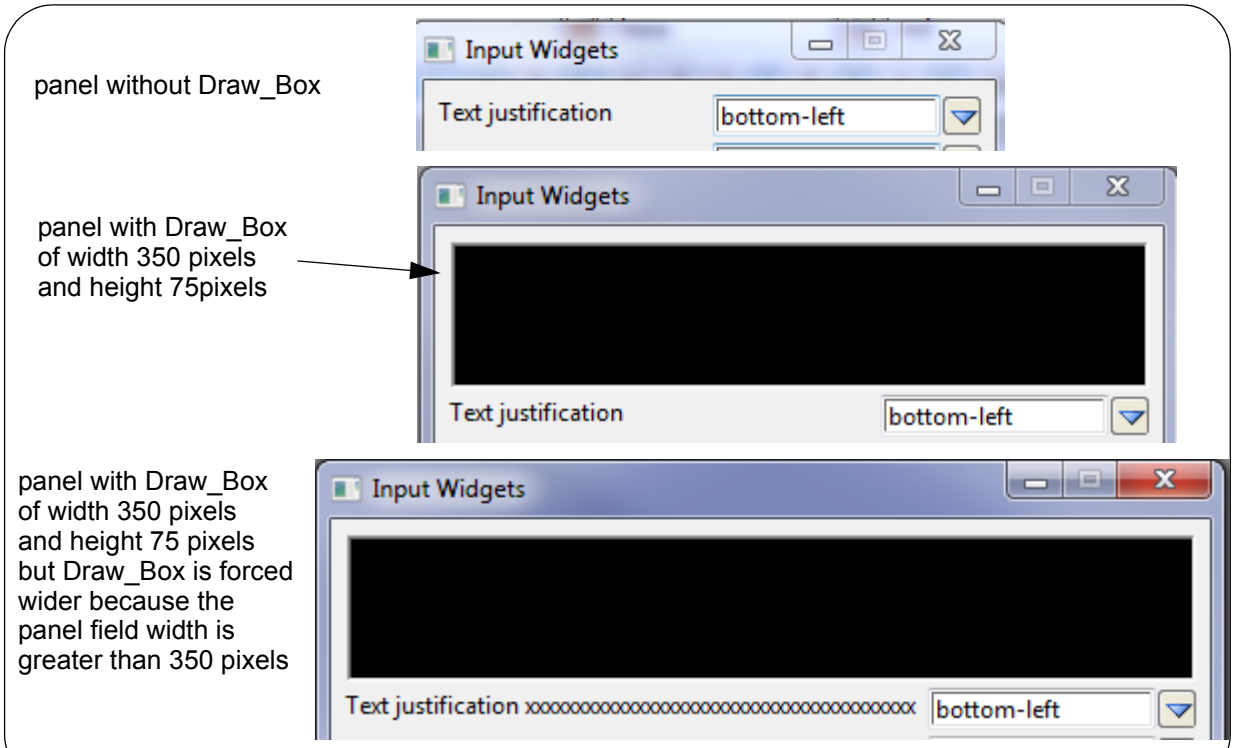
The actual size of the drawing area is actual width and actual height pixels where:

the actual width of the drawing area is the maximum of the width of the panel without the Draw_Box, and **box_width**.

and

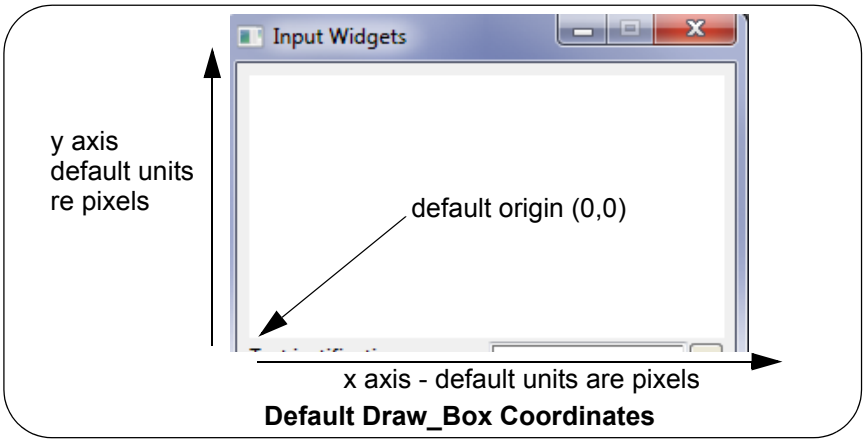
the height of the box is **box_height**.

LJG? **border** seems to be ignored.



Draw_Box

The default coordinate system for the Draw_Box is a Cartesian coordinate system with the origin (0,0) in the bottom left hand corner of the Draw_Box. That is, the x-axis is along the bottom of the Draw_Box and the y-axis goes up the side of the draw box.



Default Draw_Box Coordinates

The coordinates of the bottom left hand corner can be modified by a *Set_origin* call (*see* Set_origin(Draw_Box box,Real x,Real y)), and the units for the x-axis and the y-axis can be scaled by a *Set_scale* call (*see* Set_scale(Draw_Box box,Real xs,Real ys)).

IMPORTANT NOTE

Before making any calls to draw anything in a Draw_Box, the *Start_batch_draw* must be called (see Start_batch_draw(Draw_Box box)) otherwise the drawing calls will return an error.

Commands and Messages for Wait_on_Widgets

Moving the mouse around in the Draw_Box sends a "**mouse_move**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.

When the mouse is not moving in the Draw_Box, a "hover" command with a blank *message is sent*.

When the mouse leaves the Draw_Box, a "mouse_leave" command with a blank *message is sent*.

Pressing LB in the Draw_Box sends a "**click_lb_down**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.
Releasing LB in the Draw_Box sends a "**click_lb**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.
Double clicking LB in the Draw_Box sends a "**double_click_lb**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.

Pressing MB in the Draw_Box sends a "**click_mb_down**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.
Releasing MB in the Draw_Box sends a "**click_mb**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.
Double clicking MB in the Draw_Box sends a "**double_click_mb**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.

Pressing RB in the Draw_Box sends a "**click_rb_down**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.
Releasing RB in the Draw_Box sends a "**click_rb**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.
Double clicking RB in the Draw_Box sends a "**double_click_rb**" command with the Draw_Box coordinates in *message*. The coordinates are in Draw_Box units and are given as x and y separated by a space.

Create_draw_box(Integer box_width,Integer box_height,Integer border)

Name

Draw_Box Create_draw_box(Integer box_width,Integer box_height,Integer border)

Description

Create an input Widget of type **Draw_Box** with the drawing area defined by the parameters **box_width, box_height** and **border** which are all in screen units (pixels).*See* Draw_Box .

The function return value is the created **Draw_Box**.

ID = 1337

Get_size(Draw_Box,Integer &actual_width,Integer &actual_height)

Name

Integer Get_size(Draw_Box,Integer &actual_width,Integer &actual_height)

Description

Get the width and height in pixels of the Draw_Box drawing area on the panel and return the values in **actual_width** and **actual_height**. *See* Draw_Box for the calculations of width and height.

A function return value of zero indicates the width and height were successfully returned.

ID = 1352

Set_origin(Draw_Box box,Real x,Real y)

Name

Integer Set_origin(Draw_Box box,Real x,Real y)

Description

Set the coordinates of the left hand bottom corner of the Draw_Box box to (**x**,**y**) where **x** and **y** are given in the units of the Draw_Box.

A function return value of zero indicates the origin was successfully set.

ID = 1340

Set_scale(Draw_Box box,Real xs,Real ys)

Name

Integer Set_scale(Draw_Box box,Real xs,Real ys)

Description

Change the units for the x-axis and the y-axis of the Draw_Box **box**.

The new length of one unit in the x-direction is **xs** times the previous unit length on the x-axis. For example, if **xs** = 0.5, then the new unit length along the x-axis is half the size of the previous unit length.

Similarly, the new length of one unit in the y-direction is **ys** times the previous unit length on the y-axis.

A function return value of zero indicates the scales were successfully set.

ID = 1341

Start_batch_draw(Draw_Box box)

Name

Integer Start_batch_draw(Draw_Box box)

Description

The Start_batch_draw command must be given before any drawing calls for the Draw_Box **box** are made.

Any drawing calls made before Start_batch_draw is called will do nothing and return a non-zero function return code (that is, the call was not successful).

A function return value of zero indicates the batch draw call was successful.

ID = 1361

End_batch_draw(Draw_Box box)

Name

Integer End_batch_draw(Draw_Box box)

Description

<no description>

ID = 1362

Clear(Draw_Box box,Integer r,Integer g,Integer b)

Name

Integer Clear(Draw_Box box,Integer r,Integer g,Integer b)

Description

Clear the Draw_Box **box** and then fill **box** with a colour given by **r**, **g** and **b**.

The colour is given in rgb which requires three Integers with values between 0 and 255, one each for red, green and blue. The red, green and blue values are given in **r**, **g** and **b** respectively.

If *Clear* is called before a *Start_batch_draw (**box**)* call is made, then the *Clear* fails and a non-zero function return value is returned.

A function return value of zero indicates the clear was successful.

ID = 1344

Set_colour(Draw_Box box,Integer colour_num)

Name

Integer Set_colour(Draw_Box box,Integer colour_num)

Description

For the Draw_Box **box**, set the drawing colour for following line work to have the 12d Model colour **colour_num**.

A function return value of zero indicates the set was successful.

ID = 1342

Set_colour(Draw_Box box,Integer r,Integer g,Integer b)

Name

Integer Set_colour(Draw_Box box,Integer r,Integer g,Integer b)

Description

For the Draw_Box **box**, set the drawing colour for following line work to have the an rgb colour.

The colour is given in rgb which requires three Integers with values between 0 and 255, one

each for red, green and blue.

The red, green and blue values are given in **r**, **g** and **b** respectively.

A function return value of zero indicates the set was successful.

ID = 1343

Move_to(Draw_Box box,Real x,Real y)

Name

Integer Move_to(Draw_Box box,Real x,Real y)

Description

For the Draw_Box **box**, move the current position of the drawing nib to (**x**, **y)** where **x** and **y** are given in the units of the Draw_Box.

If *Move_to* is called before a *Start_batch_draw (**box**)* call is made, then the *Move_to* fails and a non-zero function return value is returned.

A function return value of zero indicates the move was successful.

ID = 1338

Draw_to(Draw_Box box,Real x,Real y)

Name

Integer Draw_to(Draw_Box box,Real x,Real y)

Description

For the Draw_Box **box**, draw from the current position to (**x**, **y**) where **x** and **y** are given in the units of the Draw_Box.

If *Draw_to* is called before a *Start_batch_draw (**box**)* call is made, then the *Draw_to* fails and a non-zero function return value is returned.

A function return value of zero indicates the draw was successful.

ID = 1339

Draw_polyline(Draw_Box box,Integer num_pts,Real x[],Real y[])

Name

Integer Draw_polyline(Draw_Box box,Integer num_pts,Real x[],Real y[])

Description

For the Draw_Box **box**, draw the polyline of **num_pts** points with the x-coordinates given in the array **x[]**, and the y-coordinates in the array **y[]**.

If *Draw_polyline* is called before a *Start_batch_draw (**box**)* call is made, then the *Draw_polyline* fails and a non-zero function return value is returned.

A function return value of zero indicates the draw was successful.

ID = 1355

Set_text_colour(Draw_Box box,Integer r,Integer g,Integer b)

Name

Integer Set_text_colour(Draw_Box box,Integer r,Integer g,Integer b)

Description

Set the colour used for the drawing text in the Draw_Box **box**.

The colour is given in rgb which requires three Integers with values between 0 and 255, one each for red, green and blue.

The red, green and blue values are given in **r**, **g** and **b** respectively.

A function return value of zero indicates the colour was successfully set.

ID = 1346

Set_text_font(Draw_Box box,Text font)

Name

Integer Set_text_font(Draw_Box box,Text font)

Description

For the Draw_Box **box**, set the font for the following text calls to be the True Type Font **font**.

A function return value of zero indicates the text font was successfully set.

ID = 1349

Set_text_weight(Draw_Box box,Integer weight)

Name

Integer Set_text_weight(Draw_Box box,Integer weight)

Description

Set the text weight **weight** for the Draw_Box **box**.

A function return value of zero indicates the weight was successfully set.

ID = 1350

Set_text_align(Draw_Box box,Integer mode)

Name

Integer Set_text_align(Draw_Box box,Integer mode)

Description

Set the text alignment to **mode** for any text drawn in the Draw_Box **box** after the Set_text_align call.

The values for **mode** are given in Text Alignment Modes for Draw_Box . The file set_ups.h needs to be included for the modes to be defined.

The default mode is that the coordinates of the text are for the top left of the bounding box surrounding the text.

A function return value of zero indicates the text alignment was successfully set.

ID = 1351

Draw_text(Draw_Box box,Real x,Real y,Real size,Real angle,Text txt)

Name

Integer Draw_text(Draw_Box box,Real x,Real y,Real size,Real angle,Text txt)

Description

In the Draw_Box **box**, draw the text **txt** at the position (x,y) where the coordinates (x,y) are in the Draw_Box's coordinate system.

The text has size **size** (in pixels), and the rotation angle of **angle** radians.

If *Draw_text* is called before a *Start_batch_draw (**box**)* call is made, then the *Draw_text* fails and a non-zero function return value is returned.

A function return value of zero indicates the text was successfully drawn.

ID = 1345

For information on the other Input Widgets, go to Input Widgets

File_Box

The ***File_Box*** is a panel field designed to select or create, *disk* files. If a file name is typed into the box, then it will be validated when <enter> is pressed.

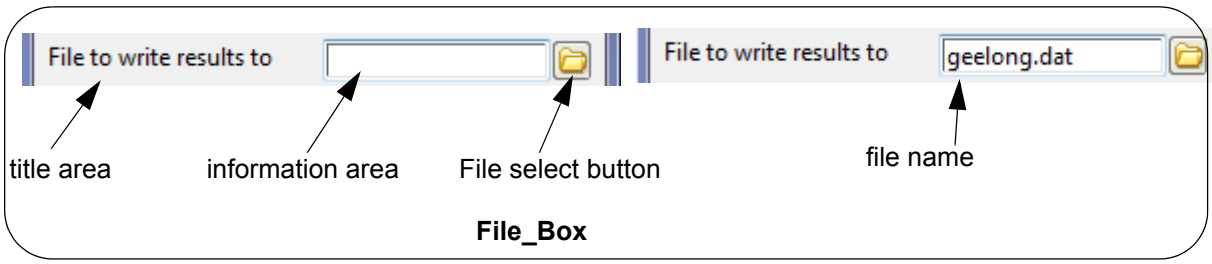
A ***File_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a file name or to display the file name if it is selected by the file select button. This information area is in the middle

and

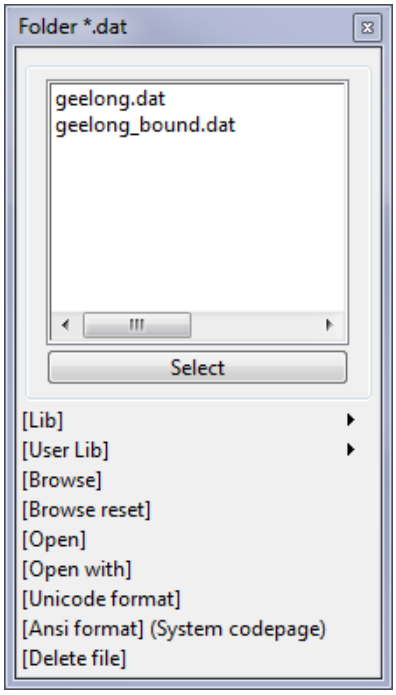
(c) a File select button on the right.



File_Box

A file name can be typed into the ***information area***. Then hitting the <enter> key will validate the file name.

Clicking **LB** or **RB** on the File select button brings up the *Folder* pop-up. Selecting a file from the pop-up list writes the file name to the ***information area***.



Clicking **MB** on the File select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**file**

selected" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a file with the Folder Select button sends a " **file selected**" command and the full path name of the file in *message*.

Create_file_box(Text title_text,Message_Box message,Integer mode,Text wild)

Name

File_Box Create_file_box(Text title_text,Message_Box message,Integer mode,Text wild)

Description

Create an input Widget of type **File_Box** for inputting and validating files.

The **File_Box** is created with the title **title_text** (*see* File_Box).

The Message_Box **message** is normally the message box for the panel and is used to display File_Box validation messages.

If <enter> is typed into the File_Box, automatic validation is performed by the File_Box according to **mode**. What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

For example,

```
CHECK_FILE_NEW      20   // if the file doesn't exists, the message says "will be created"
                         // if it exist, the messages says "ERROR"
```

The values for **mode** and their actions are listed in Appendix A (*see* File Mode).

If LB is clicked on the icon at the right hand end of the **File_Box**, a list of the files in the current area which match the wild card text **wild** (for example, *.dat) Is placed in a pop-up. If a file is selected from the pop-up (using LB), the file name is placed in the **information area** of the File_Box and validation performed according to **mode**.

The function return value is the created **File_Box**.

Special Note:

#include "set_ups.h" must be in the macro code to define CHECK_FILE_NEW etc.

ID = 906

Validate(File_Box box,Integer mode,Text &result)

Name

Integer Validate(File_Box box,Integer mode,Text &result)

Description

Validate the contents of File_Box **box** and return the text typed into the File_Box in **result.**

The value of **mode** is listed in the Appendix A - File mode. *See* File Mode .

The function returns the value of:

NO_NAME if the Widget File_Box is optional and the box is left empty

NO_FILE, FILE_EXISTS, or NO_FILE_ACCESS.

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 907

Get_data(File_Box box,Text &text_data)

Name

Integer Get_data(File_Box box,Text &text_data)

Description

Get the data of type Text from the File_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 909

Set_data(File_Box box,Text text_data)

Name

Integer Set_data(File_Box box,Text text_data)

Description

Set the data of type Text for the File_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 908

Get_wildcard(File_Box box,Text &data)

Name

Integer Get_wildcard(File_Box box,Text &data)

Description

Get the wildcard from the File_Box **box**.

The type of data must be **Text**.

A function return value of zero indicates the wildcard **data** was returned successfully.

ID = 1321

Set_wildcard(File_Box box,Text text_data)

Name

Integer Set_wildcard(File_Box box,Text text_data)

Description

Set the wildcard to the File_Box **box**.

The type of data must be **Text**.

A function return value of zero indicates the wildcard data was successfully set.

ID = 1320

Get_directory(File_Box box,Text &data)

Name

Integer Get_directory(File_Box box,Text &data)

Description

Get folder for the file from the File_Box **box** and return the folder in **data**.

A function return value of zero indicates the directory **data** was returned successfully.

ID = 1323

Set_directory(File_Box box,Text text_data)

Name

Integer Set_directory(File_Box box,Text text_data)

Description

Set the folder to the file in the File_Box **box** to the Text **data**.

A function return value of zero indicates the directory **data** was successfully set.

ID = 1322

For information on the other Input Widgets, go to Input Widgets

Function_Box

The ***Function_Box*** is a panel field designed to select, or create, Macro_Functions. If data is typed into the box, then it will be validated when <enter> is pressed.

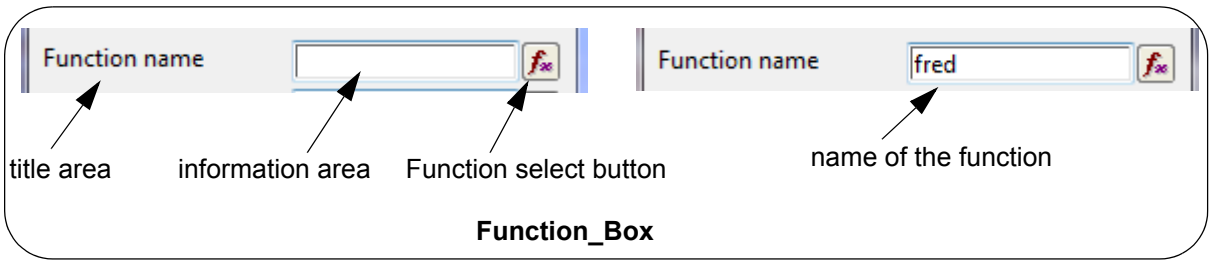
The ***Function_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in the function name or to display the function name if it is selected by the function select button. This information area is in the middle.

and

(c) a Function select button on the right.

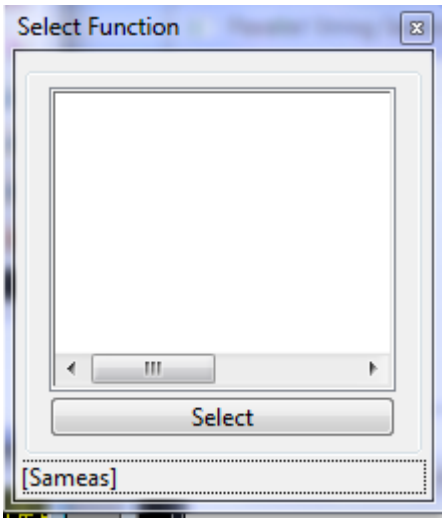


Function_Box

A function name can be typed into the ***information area***. Then hitting the <enter> key will validate the function name.

MB clicked in the ***information area*** starts a "Same As" selection. A string is then selected and if the string comes from a function of the same function type, the function name is placed in the information area.

Clicking **LB** or **RB** on the Function select button brings up the *Select Function* pop-up. Selecting the function from the pop-up list writes the function name in the information area.



Clicking **MB** on the Function select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**function selected**" command and nothing in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.
Pressing and releasing MB in the information area sends a "**middle_button_up**" command.
Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a function with the Function Select button sends a "**function selected**" command and nothing in *message*.

Function_Box Create_function_box(Text title_text,Message_Box message,Integer mode,Integer type)

Name

Function_Box Create_function_box(Text title_text,Message_Box message,Integer mode,Integer type)

Description

Create an input Widget of type **Function_Box** for inputting and validating Functions. *See* Function_Box .

The Function_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Function_Box validation messages.

The value of **mode** is listed in the Appendix A - Function mode. See Function Mode .

LJG? What is type? It also needs to be in Appendix A.

The function return value is the created **Function_Box**.

ID = 1183

Validate(Function_Box box,Integer mode,Function &result)

Name

Integer Validate(Function_Box box,Integer mode,Function &result)

Description

Validate the contents of Function_Box **box** and return the Function **result.**

The value of **mode** is listed in the Appendix A - Function mode. See Function Mode

The function returns the value of:

NO_NAME if the Widget Function_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1184

Get_data(Function_Box box,Text &text_data)

Name

Integer Get_data(Function_Box box,Text &text_data)

Description

Get the data of type Text from the Function_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1185

Set_data(Function_Box box,Text text_data)

Name

Integer Set_data(Function_Box box,Text text_data)

Description

Set the data of type Text for the Function_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1186

Get_type(Function_Box box,Integer &type)

Name

Integer Get_type(Function_Box box,Integer &type)

Description

Get the function Integer type from the Function_Box box and return it in **type**.

A function return value of zero indicates the type was returned successfully.

ID = 1334

Set_type(Function_Box box,Integer type)

Name

Integer Set_type(Function_Box box,Integer type)

Description

Set the function Integer type for the Function_Box **box** to **type**.

The type of **type** must be **Integer**.

A function return value of zero indicates the type was successfully set.

ID = 1333

Get_type(Function_Box box,Text &type)

Name

Integer Get_type(Function_Box box,Text &type)

Description

Get the function Text type from the Function_Box **box** and return it in **type**.

A function return value of zero indicates the type was returned successfully.

ID = 1336

Set_type(Function_Box box,Text type)

Name

Integer Set_type(Function_Box box,Text type)

Description

Set the function Text type for the Function_Box **box** to **type**.

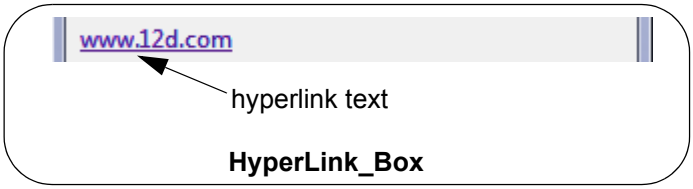
A function return value of zero indicates the type was successfully set.

ID = 1335

For information on the other Input Widgets, go to Input Widgets

HyperLink_Box

The ***HyperLink_Box*** is a panel field designed to display a hyperlink on the panel.



Commands and Messages for Wait_on_Widgets

No commands or messages are sent from the Hyperlink_Box.

HyperLink_Box Create_hyperlink_box(Text hyperlink,Message_Box message)

Name

HyperLink_Box Create_hyperlink_box(Text hyperlink,Message_Box message)

Description

Create an input Widget of type **HyperLink_Box**. *See* HyperLink_Box .

The Hyperlink_Box is created with the Text in **hyperlink**. This text should be a hyperlink.

When the user clicks on the HyperLink then the HyperLink will be activated,

The Message_Box **message** is normally the message box for the panel and is used to display Hyperlink_Box validation messages.

The function return value is the created Hyperlink_Box.

ID = 1887

Validate(HyperLink_Box box,Text &result)

Name

Integer Validate(HyperLink_Box box,Text &result)

Description

Validate the contents of HyperLink_Box **box** and return the name of the hyperlink in Text **result.**

The function returns the value of:

NO_NAME if the Widget HyperLink_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1888

Set_data(HyperLink_Box box,Text text_data)

Name

Integer Set_data(HyperLink_Box box,Text text_data)

Description

Set the data of type Text for the Hyperlink_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1889

Get_data(HyperLink_Box box,Text &text_data)

Name

Integer Get_data(HyperLink_Box box,Text &text_data)

Description

Get the data of type Text from the Hyperlink_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1890

For information on the other Input Widgets, go to Input Widgets

Input_Box

The ***Input_Box*** is a panel field designed to accept typed input, and there is no restrictions on what data can be typed into it.

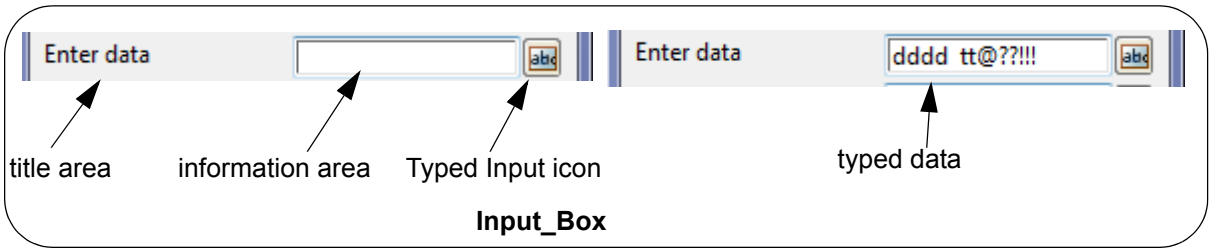
An ***Input_Box*** is a panel field that is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type text into. This information area is in the middle

and

(c) a Typed Input icon on the right.



Input_Box

Data is typed into the ***information area*** and hitting the <enter> key will validate the typed data.

Clicking **LB**, **MB** or **RB** on the typed input icon does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.
Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.
Pressing and releasing MB in the information area sends a "**middle_button_up**" command.
Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Clicking LB or RB on the Typed Input icon sends a "**text selected**" command and "[Browse]" in *message*.

Create_input_box(Text title_text,Message_Box message)

Name

Input_Box Create_input_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Input_Box**. *See* Input_Box .

The Input_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Input_Box validation messages.

The function return value is the created Input_Box.

ID = 910

Validate(Input_Box box,Text &result)

Name

Integer Validate(Input_Box box,Text &result)

Description

Validate the contents of Input_Box **box** and return the Text **result.**

This call is almost not required as the box either has text or it does not but it is required to know if the Input_Box was optional and nothing was typed in.

The function returns the value of:

NO_NAME if the Widget Input_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 911

Get_data(Input_Box box,Text &text_data)

Name

Integer Get_data(Input_Box box,Text &text_data)

Description

Get the data of type Text from the Input_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 913

Set_data(Input_Box box,Text text_data)

Name

Integer Set_data(Input_Box box,Text text_data)

Description

Set the data of type Text for the Input_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 912

For information on the other Input Widgets, go to Input Widgets

Integer_Box

The ***Integer_Box*** is a panel field designed to enter an integer (or whole number). That is, it takes typed input of optionally + or a -, followed by one or more of the numbers 0 to 9. No other characters can be typed into the *Integer_Box*.

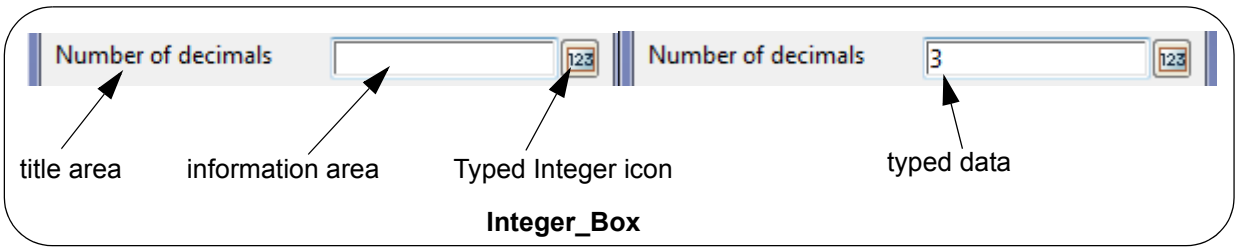
An ***Integer_Box*** is a panel field that is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in the number text. This information area is in the middle

and

(c) a Typed Integer icon on the right.



Integer_Box

Data is typed into the ***information area*** and hitting the <enter> key will validate the typed data. Only +, - and the number 0 to 9 can be typed into the ***information area***.

Clicking **LB**, **MB** or **RB** on the Typed Integer icon does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.
Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**integer selected**" command and nothing in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.
Pressing and releasing MB in the information area sends a "**middle_button_up**" command.
Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Clicking LB or RB on the Typed Integer icon sends a "**integer selected**" command and nothing in *message*.

Create_integer_box(Text title_text,Message_Box message)

Name

Integer_Box Create_integer_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Integer_Box**. *See* Integer_Box .

The Integer_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Integer_Box validation messages.

The function return value is the created Integer_Box.

ID = 914

Validate(Integer_Box box,Integer &result)

Name

Integer Validate(Integer_Box box,Integer &result)

Description

Validate **result** (of type **Integer**) in the Integer_Box **box**.

Validate the contents of Integer_Box **box** and return the Integer **result.**

The function returns the value of:

NO_NAME if the Widget Integer_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 915

Get_data(Integer_Box box,Text &text_data)

Name

Integer Get_data(Integer_Box box,Text &text_data)

Description

Get the data of type Text from the Input_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 917

Set_data(Integer_Box box,Integer integer_data)

Name

Integer Set_data(Integer_Box box,Integer integer_data)

Description

Set the data of type Integer for the Integer_Box **box** to **integer_data**.

A function return value of zero indicates the data was successfully set.

ID = 916

Set_data(Integer_Box box,Text text_data)

Name

Integer Set_data(Integer_Box box,Text text_data)

Description

Set the data of type Text for the Integer_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1517

For information on the other Input Widgets, go to Input Widgets

Justify_Box

The ***Justify_Box*** is a panel field designed to select one item from a list of text justifications. If data is typed into the box, then it will be validated when <enter> is pressed.

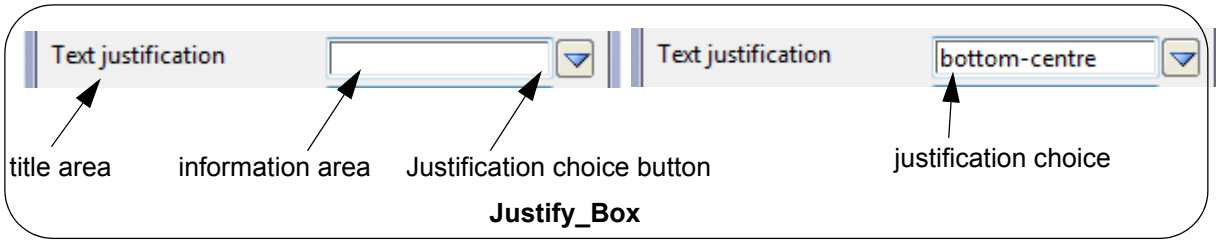
A ***Justify_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a justification or to display a justification choice if it is selected by the justification choice button. This information area is in the middle

and

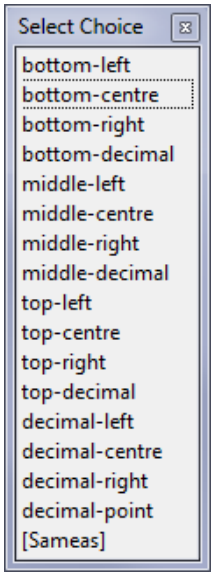
(c) a Justification choice button on the right.



Justify_Box

A justification can be typed into the ***information area*** and hitting the <enter> key will validate the justification. Note that to be valid, the typed in justification must exist in the Justification choice pop-up list.

Clicking **LB** or **RB** on the Justification choice button brings up the *Select Choice* pop-up list. Selecting a justification choice from the pop-up list writes the justification to the information area.



Clicking **MB** on the Justification choice button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command with the justification choice in *message*, or blank if it is not a valid justification.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command. Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a justification after clicking on the Justification Choice button sends a "**text selected**" command and the justification choice in *message*.

Create_justify_box(Text title_text,Message_Box message)

Name

Justify_Box Create_justify_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Justify_Box**. *See* Justify_Box .

The Justify_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Justify_Box validation messages.

The function return value is the created Justify_Box.

ID = 918

Validate(Justify_Box box,Integer &result)

Name

Integer Validate(Justify_Box box,Integer &result)

Description

Validate the contents of Justify_Box **box** and return the Integer **result.**

The function returns the value of:

NO_NAME if the Widget Justify_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 919

Get_data(Justify_Box box,Text &text_data)

Name

Integer Get_data(Justify_Box box,Text &text_data)

Description

Get the data of type Text from the Justify_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 921

Set_data(Justify_Box box,Integer integer_data)

Name

Integer Set_data(Justify_Box box,Integer integer_data)

Description

Set the data of type Integer for the Justify_Box **box** to **integer_data**.

integer_data represents the text justification and can have the values 1 to 9.

A function return value of zero indicates the data was successfully set.

ID = 920

Set_data(Justify_Box box,Text text_data)

Name

Integer Set_data(Justify_Box box,Text text_data)

Description

Set the data of type Text for the Justify_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1518

For information on the other Input Widgets, go to Input Widgets

Linestyle_Box

The ***Linestyle_Box*** is a panel field designed to select *12d Model* linestyles. If a linestyle name is typed into the box, then the linestyle name will be validated when <enter> is pressed.

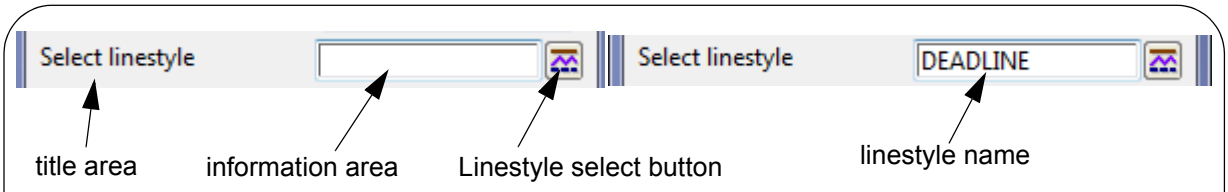
A ***Linestyle_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a linestyle name or to display the linestyle name if it is selected by the linestyle select button. This information area is in the middle

and

(c) a Linestyle select button on the right.

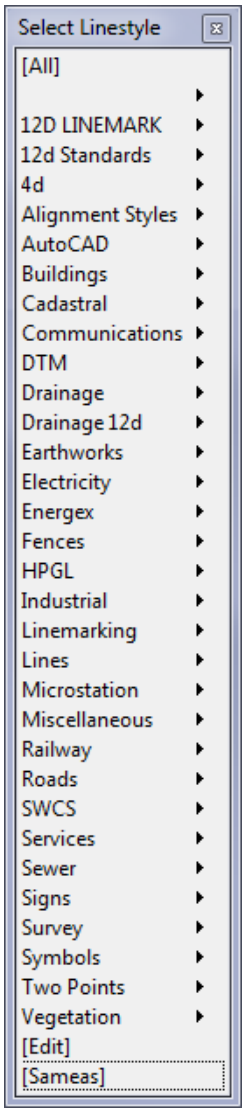


Linestyle_Box

A linestyle name can be typed into the ***information area***. Then hitting the <enter> key will validate the linestyle name.

MB clicked in the ***information area*** starts a "Same As" selection. A string is then selected and the linestyle of the string is written in the information area.

Clicking **LB** or **RB** on the Linestyle select button brings up the *Select Linestyle* pop-up. Selecting a linestyle from the pop-up list writes the linestyle name in the information area.



Clicking **MB** on the Linestyle select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a linestyle after clicking on the Linestyle Select button sends a "**text selected**" command and the linestyle name in *message*.

Create_linestyle_box(Text title_text,Message_Box message,Integer mode)

Name

Linestyle_Box Create_linestyle_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Linestyle_Box**. *See* Linestyle_Box .

The Linestyle_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Linestyle_Box validation messages.

The value of **mode** is listed in the Appendix A - Linestyle mode. *See* Linestyle Mode .

The function return value is the created Linestyle_Box.

ID = 922

Validate(Linestyle_Box box,Integer mode,Text &result)

Name

Integer Validate(Linestyle_Box box,Integer mode,Text &result)

Description

Validate the contents of Linestyle_Box **box** and return the name of the linestyle in Text **result.**

The value of **mode** is listed in the Appendix A - Linestyle mode. *See* Linestyle Mode

The function returns the value of:

NO_NAME if the Widget Linestyle_Box is optional and the box is left empty

LINESTYLE_EXISTS or NO_LINESTYLE.

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 923

Get_data(Linestyle_Box box,Text &text_data)

Name

Integer Get_data(Linestyle_Box box,Text &text_data)

Description

Get the data of type Text from the Linestyle_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 925

Set_data(Linestyle_Box box,Text text_data)

Name

Integer Set_data(Linestyle_Box box,Text text_data)

Description

Set the data of type Text for the Linestyle_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 924

For information on the other Input Widgets, go to Input Widgets

List_Box

Create_list_box(Text title_text,Message_Box message,Integer nlines)

Name

List_Box Create_list_box(Text title_text,Message_Box message,Integer nlines)

Description

Create an input Widget of type **List_Box**. *See* List_Box .

The List_Box is created with the title **title_text**.

The number of lines **nline** will be created in the List_Box.

The Message_Box **message** is normally the message box for the panel and is used to display List_Box validation messages.

The function return value is the created List_Box.

ID = 1278

Get_number_of_items(List_Box box,Integer &count)

Name

Integer Get_number_of_items(List_Box box,Integer &count)

Description

For the List_Box **box**, get the number of items in the list and return the number in **count**.

A function return value of zero indicates that count is successfully returned.

ID = 1546

Set_sort(List_Box box,Integer mode)

Name

Integer Set_sort(List_Box box,Integer mode)

Description

Set the sort mode for the List_Box **box** depending on the Integer **mode**.

If **mode** is 0 then the sort is ascending,

If **mode** is 1 then the sort is descending.

A function return value of zero indicates the sort was successfully set.

ID = 1279

Get_sort(List_Box box,Integer &mode)

Name

Integer Get_sort(List_Box box,Integer &mode)

Description

Get the sort mode from the List_Box **box** and return it in **mode**.

If **mode** is 0 then the sort is ascending,

If **mode** is 1 then the sort is descending.

A function return value of zero indicates the mode was returned successfully.

ID = 1280

For information on the other Input Widgets, go to Input Widgets

Map_File_Box

Create_map_file_box(Text title_text,Message_Box message,Integer mode)

Name

Map_File_Box Create_map_file_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Map_File_Box**. *See* Map_File_Box .

The Map_File_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Map_File_Box validation messages.

The value of **mode** is listed in the Appendix A - File mode. See LJG? Map File Modes need to be added to Appendix.

The function return value is the created Map_File_Box.

ID = 926

Validate(Map_File_Box box,Integer mode,Text &result)

Name

Integer Validate(Map_File_Box box,Integer mode,Text &result)

Description

Validate the contents of Map_File_Box **box** and return the Text **result.**

The value of **mode** is listed in the Appendix A - File mode. *See* File Mode

The function returns the value of:

NO_NAME if the Widget Map_File_Box is optional and the box is left empty

NO_FILE, FILE_EXISTS or NO_FILE_ACCESS

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 927

Get_data(Map_File_Box box,Text &text_data)

Name

Integer Get_data(Map_File_Box box,Text &text_data)

Description

Get the data of type Text from the Map_File_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 929

Set_data(Map_File_Box box,Text text_data)

Name

Integer Set_data(Map_File_Box box,Text text_data)

Description

Set the data of type Text for the Map_File_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 928

For information on the other Input Widgets, go to Input Widgets

Model_Box

The ***Model_Box*** is a panel field designed to select *12d Model* models. If a model name is typed into the model box and <enter> pressed or a model selected from the model pop-up list, then the text in the Model_Box is validated.

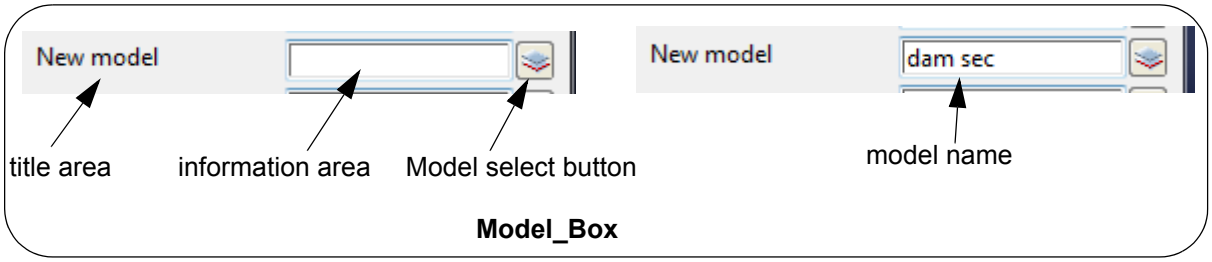
A ***Model_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a model name or to display the model name if it is selected by the model select button. This information area is in the middle

and

(c) a Model select button on the right.

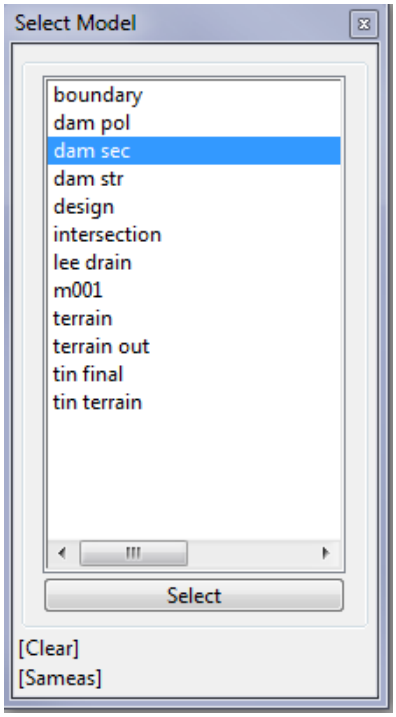


Model_Box

A model name can be typed into the ***information area***. Then hitting the <enter> key validates the model name.

MB clicked in the ***information area*** starts a "Same As" selection. A string is then selected and the model name of the selected string name is placed in the information area.

Clicking **LB** or **RB** on the Model select button brings up the *Select Model* pop-up. Selecting a model from the pop-up list writes the model name in the information area and validation occurs.



Clicking **MB** on the Model select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.
Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**model selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.
Pressing and releasing MB in the information area sends a "**middle_button_up**" command.
Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a model with the Model Select button sends a "**model selected**" command and the model name in *message*.

Create_model_box(Text title_text,Message_Box message,Integer mode)

Name

Model_Box Create_model_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Model_Box** for inputting and validating Models.

The **Model_Box** is created with the title **title_text** (*see* Model_Box).

The Message_Box **message** is normally the message box for the panel and is used to display Model_Box validation messages.

If <enter> is typed into the Model_Box automatic validation is performed by the Model_Box according to **mode**. What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

For example,

```
CHECK_MODEL_MUST_EXIST          7   // if the model exists, the message says "exists".
                                    // if it doesn't exist, the messages says "ERROR"
```

The values for **mode** and their actions are listed in Appendix A (*see* Model Mode).

If LB is clicked on the icon at the right hand end of the **Model_Box**, a list of all existing models is placed in a pop-up. If a model is selected from the pop-up (using LB), the model name is placed in the **information area** of the Model_Box and validation performed according to **mode**.

MB for "Same As" also applies. That is, If MB is clicked in the **information area** and then a string from a model on a view is selected, then the name of the model containing the selected string is written to the **information area** and validation performed according to **mode**.

The function return value is the created **Model_Box**.

Special Note:

#include "set_ups.h" must be in the macro code to define CHECK_MODEL_MUST_EXIST etc.

ID = 848

Validate(Model_Box box,Integer mode,Model &result)

Name

Integer Validate(Model_Box box,Integer mode,Model &result)

Description

Validate the contents of the Model_Box **box** and return the Model **result.**

The value of **mode** will determine what validation occurs, what messages are written to the Message_Box, what actions are taken and what the function return value is.

The values for **mode** and the actions are listed in Appendix A (*see* Model Mode).

The function return value depends on mode and are given in Appendix A (*see* Model Mode).

A function return value of zero indicates that there is a drastic error.

Warning this is the opposite of most 12dPL function return values

Double Warning: most times the function return code is not zero even when you think it should be. The actual value of the function return code must be checked to see what is going on. For example, when **mode** = CHECK_MODEL_MUST_EXIST will return NO_MODEL if the model name is not blank and no model of that name exist (NO_MODEL does not equal zero).

ID = 880

Get_data(Model_Box box,Text &text_data)

Name

Integer Get_data(Model_Box box,Text &text_data)

Description

Get the data of type Text from the Model_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 885

Set_data(Model_Box box,Text text_data)

Name

Integer Set_data(Model_Box box,Text text_data)

Description

Set the data of type Text for the Model_Box **box** as the Text **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 884

For information on the other Input Widgets, go to Input Widgets

Name_Box

The ***Name_Box*** is a panel field designed to type in, or display, string names. If data is typed into the box, then it will be validated when <enter> is pressed.

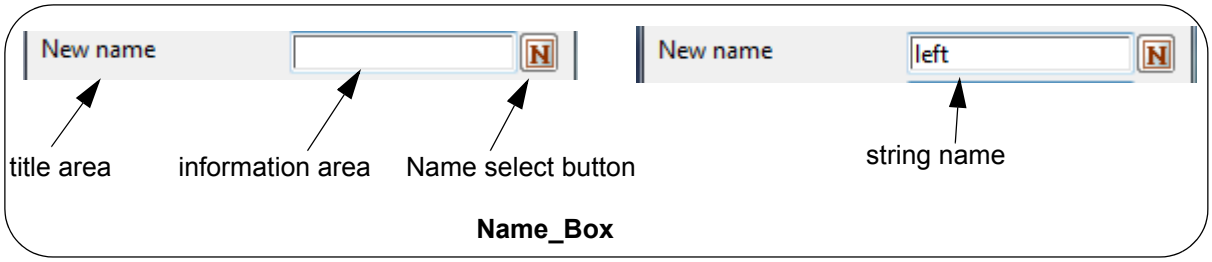
A ***Name_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a string name or to display the string name if it is selected by the name select button. This information area is in the middle

and

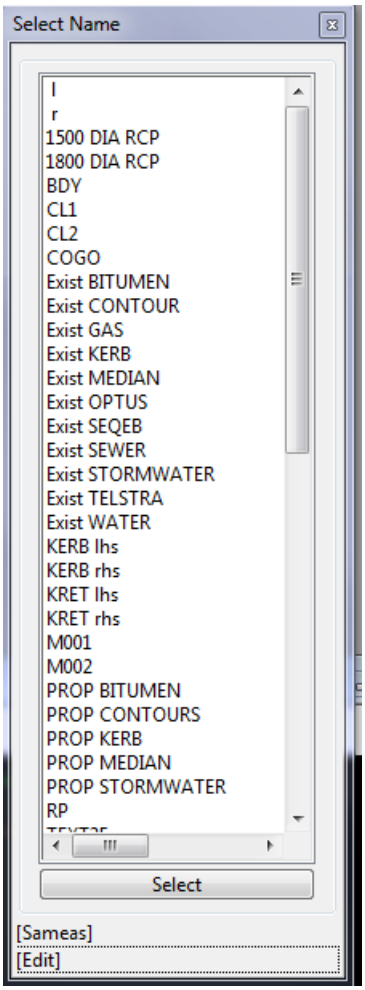
(c) a Name select button on the right.



A string name can be typed into the ***information area***. Then hitting the <enter> key will validate the string name.

MB clicked in the ***information area*** starts a "Same As" selection. A string is then selected and the name of the selected string name is placed in the information area.

Clicking **LB** or **RB** on the Name select button brings up the *Select Name* pop-up. Selecting the name from the pop-up list writes the name in the information area.



Clicking **MB** on the Name select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a Name with the Name Select button sends a "**text selected**" command and the Name in *message*.

Create_name_box(Text title_text,Message_Box message)

Name

Name_Box Create_name_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Name_Box**. *See* Name_Box .

The Name_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Name_Box validation messages.

The function return value is the created Name_Box.

ID = 930

Validate(Name_Box box,Text &result)

Name

Integer Validate(Name_Box box,Text &result)

Description

Validate the contents of Name_Box **box** and return the Text **result.**

The function returns the value of:

NO_NAME if the Widget Name_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (0) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 931

Get_data(Name_Box box,Text &text_data)

Name

Integer Get_data(Name_Box box,Text &text_data)

Description

Get the data of type Text from the Name_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 933

Set_data(Name_Box box,Text text_data)

Name

Integer Set_data(Name_Box box,Text text_data)

Description

Set the data of type Text for the Name_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 932

For information on the other Input Widgets, go to Input Widgets

Named_Tick_Box

The ***Named_Tick_Box*** is a panel field designed to be in only two states:

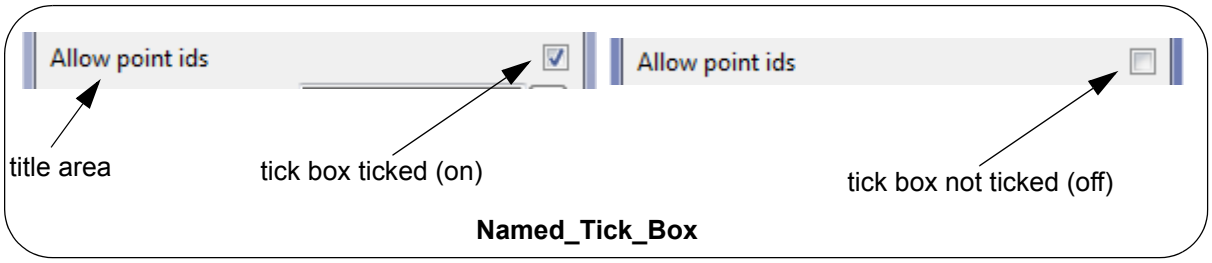
ticked (on) or not ticked (off).

A ***Named_Tick_Box*** is made up of two items:

(a) a title area on the left with the user supplied title on it

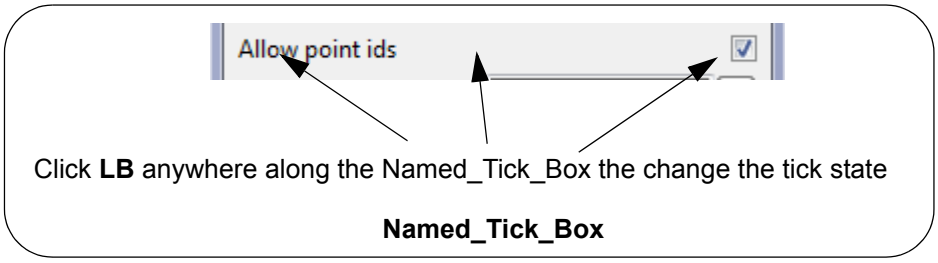
and

(b) a box that can display, or not display, a tick.



Named_Tick_Box

Clicking **LB** anywhere along the length of the Named_Tick_Box from the title area to the tick box, will reverse the state of the tick. That is, a tick will go to no tick, and no tick will go to tick.



Named_Tick_Box

Clicking **MB** or **RB** anywhere along the Named_Tick_Box does nothing.

Note: A Named_Tick_Box cannot be made optional

Commands and Messages for Wait_on_Widgets

Clicking LB anywhere in the Named_Tick_Box sends a "**toggle tick**" command and a blank message.
Nothing else sends any commands or messages.

Create_named_tick_box(Text title_text,Integer state,Text response)

Name

Named_Tick_Box Create_named_tick_box(Text title_text,Integer state,Text response)

Description

Create an input Widget of type **Named_Tick_Box**. *See* Named_Tick_Box .

The Named_Tick_Box is created with the Text **title_text**.

The Integer **state** specifies the ticked/unticked state of the box:

state = 0 set the box as unticked

state = 1 set the box as ticked

The Text **response** returns the **msg** when calling the Wait_on_widgets function.

The function return value is the created Named_Tick_Box.

ID = 849

Validate(Named_Tick_Box box,Integer &result)

Name

Integer Validate(Named_Tick_Box box,Integer &result)

Description

Validate the contents of Named_Tick_Box **box** and return the Integer **result.**

result = 0 if the tick box is unticked
result = 1 if the tick box is ticked

A function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 974

Set_data(Named_Tick_Box box,Integer state)

Name

Integer Set_data(Named_Tick_Box box,Integer state)

Description

Set the state of the Named_Tick_Box to

ticked if **state** = 1
unticked if **state** = 0

A function return value of zero indicates the data was successfully set.

ID = 2239

Get_data(Named_Tick_Box box,Text &text_data)

Name

Integer Get_data(Named_Tick_Box box,Text &text_data)

Description

Get the data of type Text from the Named_Tick_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 976

Set_data(Named_Tick_Box box,Text text_data)

Name

Integer Set_data(Named_Tick_Box box,Text text_data)

Description

Set the data of type Text for the Named_Tick_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 975

For information on the other Input Widgets, go to Input Widgets

New_Select_Box

The ***New_Select_Box*** is a panel field designed to select *12d Model* strings.

Note that the *New_Select_Box only picks strings* and does not return information if a cursor pick is made. The Select_Box allows for cursor picks.

The ***New_Select_Box*** is made up of three items:

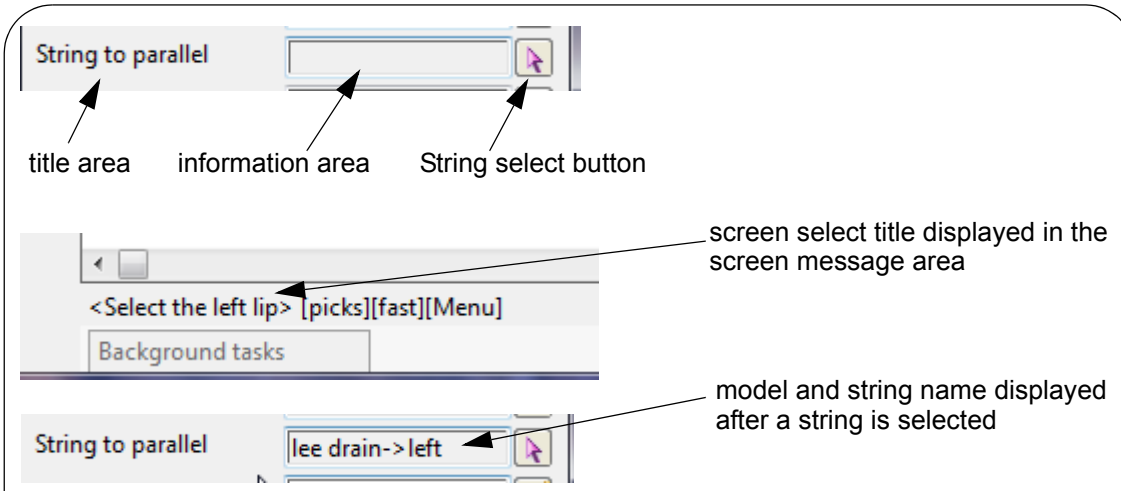
(a) a title area on the left with the user supplied title on it

(b) an information area in the middle where the name and model of the selected string are displayed

(c) a String select button on the right.

plus

(d) a screen select title that is displayed in the screen message area after the select button is selected.



New_Select_Box

Nothing can be typed into the ***information area*** but if **MB** clicked in the ***information area*** starts a "Same As" selection. A string is then selected and the model and name of the selected string are displayed in the information area.

Clicking **LB** on the ***string select button*** and then selecting the string. The model and name of the string are then displayed in the information area.

Clicking **RB** on the ***String select button*** brings up the string select *Choice* box.

Clicking **MB** on the ***String select button*** does nothing.

Commands and Messages for Wait_on_Widgets

Clicking LB on the String Select button:

As the mouse is moved over a view, a "**motion select**" command is sent with the view coordinates and view name as text in *message*.

Once in the select:

if a string is clicked on with LB, a "**pick select**" command is sent with the name of the view that the string was selected in, in *message.* if the string is accepted (MB), an "**accept select**" command is sent with the view name (in quotes) in *message,* or if RB is clicked and *Cancel* selected from the *Pick Ops* menu, then a "**cancel select**" command is sent with nothing in *message.*

if a string is clicked on with MB (the pick and accept in one click method), a "**pick select**" command is sent with the name of the view that the string was selected in, in *message,* followed by an "**accept select**" command with the view name (in quotes) in *message.*

Nothing else sends any commands or messages.

Create_new_select_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Name

New_Select_Box Create_new_select_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Description

Create an input Widget of type **New_Select_Box**. *See* New_Select_Box .

The New_Select_Box is created with the title **title_text**.

The Select title displayed in the screen message area is **select_title**.

The value of mode is listed in the Appendix A - Select mode. *See* Select Mode .

The Message_Box **message** is normally the message box for the panel and is used to display New_Select_Box validation messages.

Note that the *New_Select_Box only picks strings* and does not return information if a cursor pick is made. The Select_Box allows for cursor picks.

The function return value is the created New_Select_Box.

ID = 2240

Validate(New_Select_Box select,Element &string)

Name

Integer Validate(New_Select_Box select,Element &string)

Description

Validate the contents of New_Select_Box **select** and return the selected Element in **string.**

The function returns the value of:

NO_NAME if the Widget New_Select_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 2241

Validate(New_Select_Box select,Element &string,Integer silent)

Name

Integer Validate(New_Select_Box select,Element &string,Integer silent)

Description

Validate the contents of New_Select_Box **select** and return the selected Element in **string.**

If **silent** = 0, and there is an error, a message is written and the cursor goes back to the box.
If **silent** = 1 and there is an error, no message or movement of cursor is done.

The function returns the value of:

NO_NAME if the Widget New_Select_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 2242

Set_data(New_Select_Box select,Element string)

Name

Integer Set_data(New_Select_Box select,Element string)

Description

Set the data of for the New_Select_Box **select** to **string**.

A function return value of zero indicates the data was successfully set.

ID = 2243

Set_data(New_Select_Box select,Text model_string)

Name

Integer Set_data(New_Select_Box select,Text model_string)

Description

Set the Element of the New_Select_Box **box** by giving the model name and string name as a Text **model_string** in the form "model_name->string_name".

A function return value of zero indicates the data was successfully set.

ID = 2244

Get_data(New_Select_Box select,Text &model_string)

Name

Integer Get_data(New_Select_Box select,Text &model_string)

Description

Get the model and string name of the Element in the New_Select_Box **box** and return it in Text **model_string**.

Note: the model and string name is in the form "model_name->string_name" so only one Text is required.

A function return value of zero indicates the data was successfully returned.

ID = 2245

For information on the other Input Widgets, go to Input Widgets

New_XYZ_Box

The ***New_XYZ_Box*** is a panel field designed to get x, y and z coordinates and the X Y and Z coordinates are each displayed in their own information areas.

Also see XYZ_Box where the XYZ values are displayed in the one information area, separated by spaces.

The ***New_XYZ_Box*** is made up of:

(a) a title area on the left with the user supplied title on it

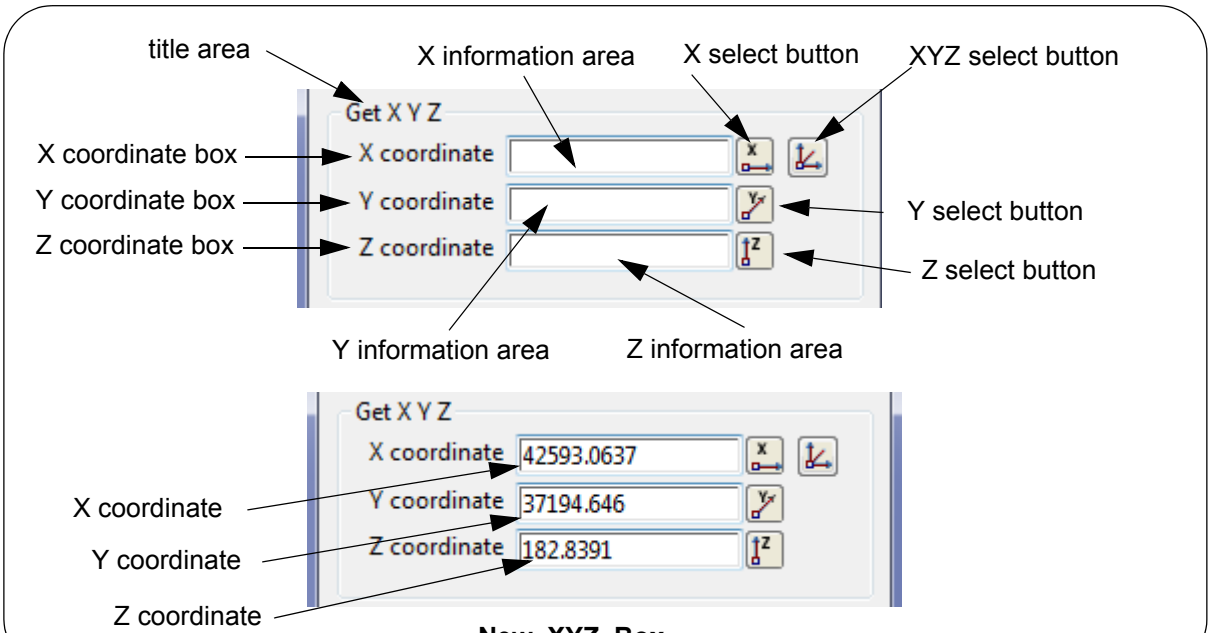
(b) a X coordinate box consisting of the title **X coordinate,** a **X information area** and a X select button.

(c) a Y coordinate box consisting of the title **Y coordinate,** a **Y information area** and a Y select button.

(d) a Z coordinate box consisting of the title **Z coordinate,** a **Z information area** and a Z select button.

and

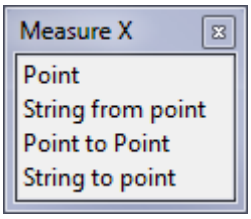
(e) a XYZ select button on the right.



New_XYZ_Box

A X coordinate can be typed into the ***X information area***. Then hitting the <enter> key will validate that the value is a Real number.

Clicking **LB** or **RB** on the X select button brings up the *Measure X* pop-up menu. Selecting an option from the *Measure X* menu and making a measure displays the X coordinate in the X information area.

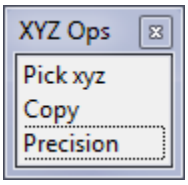


Clicking **MB** on the X select button does nothing.

Similarly for Y and Z coordinates.

Clicking **LB** on the XYZ select button starts the XYZ Pick option and after selecting a position, the X, Y and Z are displayed in the X, Y and Z information areas respectively.

Clicking **RB** on the XYZ select button brings up the *XYZ Ops* pop-up menu. Selecting the *Pick xyz* option starts the XYZ Pick option and after selecting a position, the X, Y and Z are displayed in the X, Y and Z information areas respectively.



Clicking **MB** on the XYZ select button does nothing.

Commands and Messages for Wait_on_Widgets

LJG? The New_XYZ_Box is actually made up of 4 widgets. So how do you know the ids?. The id of the New_XYZ_Box returns he id of the Select XYZ button.

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.
Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking an X coordinate with the X Select button sends a "**real selected**" command and nothing in *message*.
Picking an Y coordinate with the Y Select button sends a "**real selected**" command and nothing in *message*.
Picking an Z coordinate with the Z Select button sends a "**real selected**" command and nothing in *message*.
Picking a coordinate with the XYZ Select button sends a "**coordinate accepted**" command with nothing in *message*.

Create_new_xyz_box(Text title_text,Message_Box message)

Name

New_XYZ_Box Create_new_xyz_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **New_XYZ_Box**. *See* New_XYZ_Box .

The New_XYZ_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display New_XYZ_Box validation messages.

The function return value is the created New_XYZ_Box.

ID = 2252

Validate(New_XYZ_Box box,Real &x,Real &y,Real &z)

Name

Integer Validate(New_XYZ_Box box,Real &x,Real &y,Real &z)

Description

Validate the contents of the New_XYZ_Box **box** and check that it decodes to three Reals.

The three Reals are returned in **x, y,** and **z**.

The function returns the value of:

NO_NAME if the Widget New_XYZ_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *x, y* and *z* are valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 2253

Get_data(New_XYZ_Box box,Text &text_data)

Name

Integer Get_data(New_XYZ_Box box,Text &text_data)

Description

Get the data of type Text from the New_XYZ_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 2254

Set_data(New_XYZ_Box box,Real x,Real y,Real z)

Name

Integer Set_data(New_XYZ_Box box,Real x,Real y,Real z)

Description

Set the x y z data (all of type Real) for the New_XYZ_Box **box** to the values **x**, **y** and **z**.

A function return value of zero indicates the data was successfully set.

ID = 2255

Set_data(New_XYZ_Box box,Text text_data)

Name

Integer Set_data(New_XYZ_Box box,Text text_data)

Description

Set the data of type Text for the New_XYZ_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 2256

Plotter_Box

Create_plotter_box(Text title_text,Message_Box message)

Name

Plotter_Box Create_plotter_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Plotter_Box**. *See* Plotter_Box .

The Plotter_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Plotter_Box validation messages.

The function return value is the created Plotter_Box.

ID = 934

Validate(Plotter_Box box,Text &result)

Name

Integer Validate(Plotter_Box box,Text &result)

Description

Validate the contents of Plotter_Box **box** and return the Text **result.**

The function returns the value of:

NO_NAME if the Widget Plotter_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (0) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 935

Get_data(Plotter_Box box,Text &text_data)

Name

Integer Get_data(Plotter_Box box,Text &text_data)

Description

Get the data of type Text from the Plotter_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 937

Set_data(Plotter_Box box,Text text_data)

Name

Integer Set_data(Plotter_Box box,Text text_data)

Description

Set the data of type Text for the Plotter_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 936

Validate(Plotter_Box box,Text &plotter_mode,Text &plotter_names,Text &plotter_type)

Name

Integer Validate(Plotter_Box box,Text &plotter_mode,Text &plotter_names,Text &plotter_type)

Description

<no description>

ID = 2465

Set_data(Plotter_Box box,Text plotter_mode,Text plotter_names,Text plotter_type)

Name

Integer Set_data(Plotter_Box box,Text plotter_mode,Text plotter_names,Text plotter_type)

Description

<no description>

ID = 2466

Get_data(Plotter_Box box,Text &plotter_mode,Text &plotter_names,Text &plotter_type)

Name

Integer Get_data(Plotter_Box box,Text &plotter_mode,Text &plotter_names,Text &plotter_type)

Description

<no description>

ID = 2467

For information on the other Input Widgets, go to Input Widgets

Polygon_Box

Polygon_Box Create_polygon_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Name

Polygon_Box Create_polygon_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Description

Create an input Widget of type **Polygon_Box**. *See* Polygon_Box .

The Polygon_Box is created with the title **title_text**.

LJG? select_title

LJG? mode

The Message_Box **message** is normally the message box for the panel and is used to display Polygon_Box validation messages.

The function return value is the created Polygon_Box.

ID = 2246

Validate(Polygon_Box select,Element &string)

Name

Integer Validate(Polygon_Box select,Element &string)

Description

Validate the contents of Polygon_Box **select** and return the selected Element in **string.**

If there is an error, a message is written and the cursor goes back to the Polygon_Box.

The function returns the value of:

NO_NAME if the Widget Polygon_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 2247

Validate(Polygon_Box select,Element &string,Integer silent)

Name

Integer Validate(Polygon_Box select,Element &string,Integer silent)

Description

Validate the contents of Polygon_Box **select** and return the selected Element in **string.**

If **silent** = 0, and there is an error, a message is written and the cursor goes back to the Polygon_Box.

If **silent** = 1 and there is an error, no message or movement of cursor is done.

The function returns the value of:

NO_NAME if the Widget Polygon_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 2248

Set_data(Polygon_Box select,Element string)

Name

Integer Set_data(Polygon_Box select,Element string)

Description

Set the data of type Element for the Polygon_Box **select** to **string**.

A function return value of zero indicates the data was successfully set.

ID = 2249

Set_data(Polygon_Box select,Text string_name)

Name

Integer Set_data(Polygon_Box select,Text string_name)

Description

Set the data of type Text for the Polygon_Box **select** to **string_name**.

A function return value of zero indicates the data was successfully set.

ID = 2250

Get_data(Polygon_Box select,Text &string)

Name

Integer Get_data(Polygon_Box select,Text &string)

Description

Get the data of type Text from the Polygon_Box **select** and return it in **string**.

A function return value of zero indicates the data was successfully returned.

ID = 2251

For information on the other Input Widgets, go to Input Widgets

Real_Box

The ***Real_Box*** is a panel field designed to enter real numbers where a real value may be given as a decimal, or in exponential format such as 1.3e10 or 1.3d3. So the real number can only contain +, -, decimal point, e, d and the numbers 0 to 9. No other characters can be typed into the *Real_Box*.

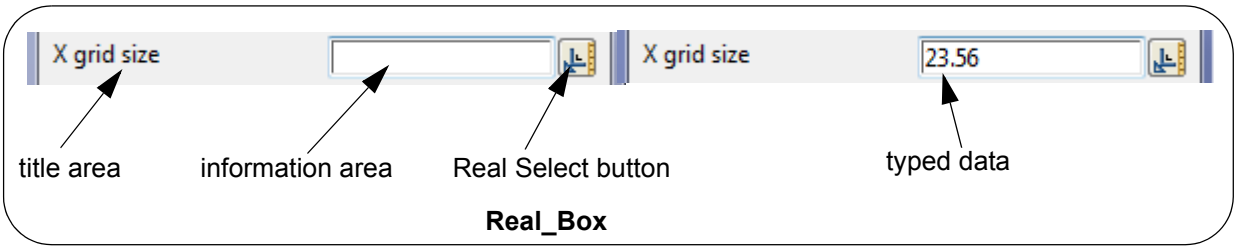
A ***Real_Box*** is a panel field that is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) a information area to type in the real number. This information area is in the middle

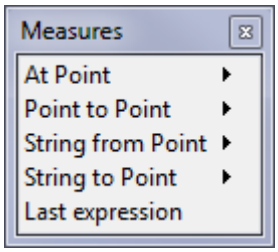
and

(c) a Real select button on the right.



Data is typed into the ***information area*** and hitting the <enter> key will validate the typed data. Only real values can be typed into the ***information area*** (that is, the real number can only contain +, -, decimal point, e, d and the numbers 0 to 9).

Clicking **LB** or **RB** on the Real Select button brings up the *Measure* pop-up menu. Selecting an option from the *Measure* menu and making a measure displays the real number in the information area.



Clicking **MB** on the Real select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**real selected**" command and nothing in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Clicking LB or RB on the Real Select button and accepting a value sends a "**real selected**" command and nothing in *message*.

Create_real_box(Text title_text,Message_Box message)

Name

Real_Box Create_real_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Real_Box**. *See* Real_Box .

The Real_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Real_Box validation messages.

The function return value is the created Real_Box.

ID = 902

Validate(Real_Box box,Real &result)

Name

Integer Validate(Real_Box box,Real &result)

Description

Validate the contents of Real_Box **box** and return the Real **result.**

A function return value of zero indicates the value was valid.

The function returns the value of:

NO_NAME if the Widget Real_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 903

Get_data(Real_Box box,Text &text_data)

Name

Integer Get_data(Real_Box box,Text &text_data)

Description

Get the data of type Text from the Real_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 905

Set_data(Real_Box box,Real real_data)

Name

Integer Set_data(Real_Box box,Real real_data)

Description

Set the data of type Real for the Real_Box **box** to **real_data**.

A function return value of zero indicates the data was successfully set.

ID = 904

Set_data(Real_Box box,Text text_data)

Name

Integer Set_data(Real_Box box,Text text_data)

Description

Set the data of type Text for the Real_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1516

For information on the other Input Widgets, go to Input Widgets

Report_Box

The ***Report_Box*** is a panel field designed to select or create, *disk report* files. If a file name is typed into the box, then it will be validated when <enter> is pressed.

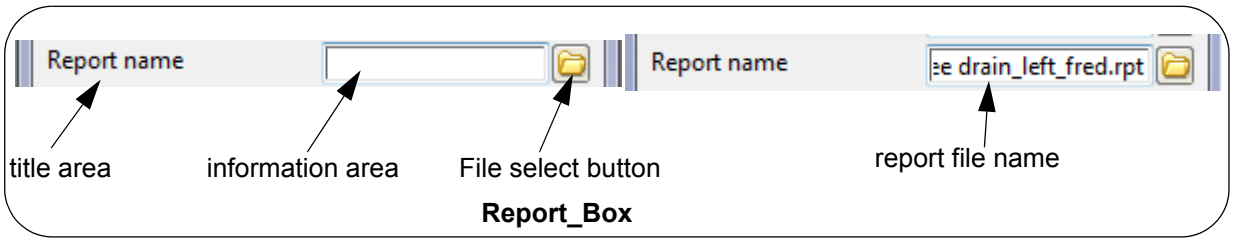
A ***Report_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a file name or to display the file name if it is selected by the File select button. This information area is in the middle

and

(c) a File select button on the right.

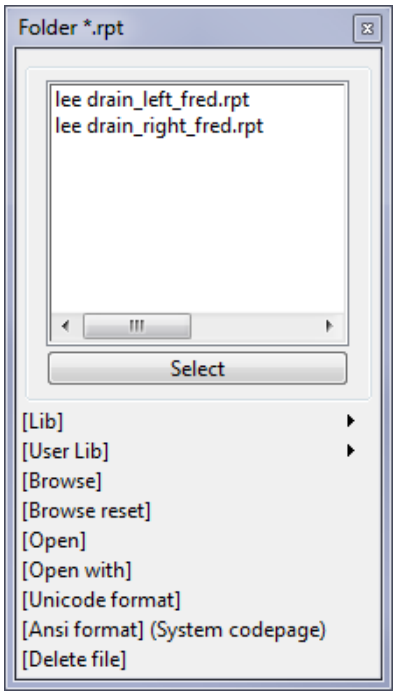


Report_Box

A file name can be typed into the ***information area***. Then hitting the <enter> key will validate the file name.

Clicking **LB** or **RB** on the File select button brings up the *Folder* pop-up with the wild card for showing files set to *.rpt. Files with other ending can be created/selected but the default for a Report_Box is "*.rpt".

Selecting a file from the pop-up list writes the file name to the ***information area***.



Clicking **MB** on the File select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**file selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a file with the Folder Select button sends a " **file selected**" command and the full path name of the file in *message*.

Create_report_box(Text title_text,Message_Box message,Integer mode)

Name

Report_Box Create_report_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Report_Box**. *See* Report_Box .

The Report_Box is created with the title **title_text**.

The Message_Box **message** is normally the message box for the panel and is used to display Report_Box validation messages.

The value of **mode** is listed in the Appendix A - File mode.

The function return value is the created Report_Box.

ID = 938

Validate(Report_Box box,Integer mode,Text &result)

Name

Integer Validate(Report_Box box,Integer mode,Text &result)

Description

Validate the contents of Report_Box **box** and return the Text **result.**

The value of **mode** is listed in the Appendix A - File mode. See File Mode

The function returns the value of:

NO_NAME if the Widget Report_Box is optional and the box is left empty

NO_FILE, FILE_EXISTS or NO_FILE_ACCESS

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 939

Get_data(Report_Box box,Text &text_data)

Name

Integer Get_data(Report_Box box,Text &text_data)

Description

Get the data of type Text from the Report_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 941

Set_data(Report_Box box,Text text_data)

Name

Integer Set_data(Report_Box box,Text text_data)

Description

Set the data of type Text for the Report_Box **box** to **text_data**.

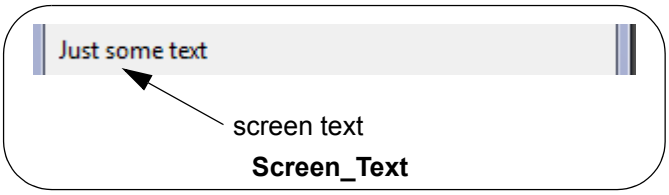
A function return value of zero indicates the data was successfully set.

ID = 940

For information on the other Input Widgets, go to Input Widgets

Screen_Text

The ***Screen_Text*** is a panel field designed to simply place some text on the panel.



Commands and Messages for Wait_on_Widgets

No commands or messages are send from the Screen_Text Widget.

Create_screen_text(Text text)

Name

Screen_Text Create_screen_text(Text text)

Description

Create a **Screen_Text** with the Text **text**. *See* Screen_Text .

The function return value is the created Screen_Text.

ID = 1369

Set_data(Screen_Text widget,Text text_data)

Name

Integer Set_data(Screen_Text widget,Text text_data)

Description

Set the data of type Text for the Screen_Text **widget** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1371

Get_data(Screen_Text widget,Text &text_data)

Name

Integer Get_data(Screen_Text widget,Text &text_data)

Description

Get the data of type Text from the Screen_Text **widget** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1370

For information on the other Input Widgets, go to Input Widgets

Select_Box

The ***Select_Box*** is a panel field designed to select *12d Model* strings and also cursor picks.

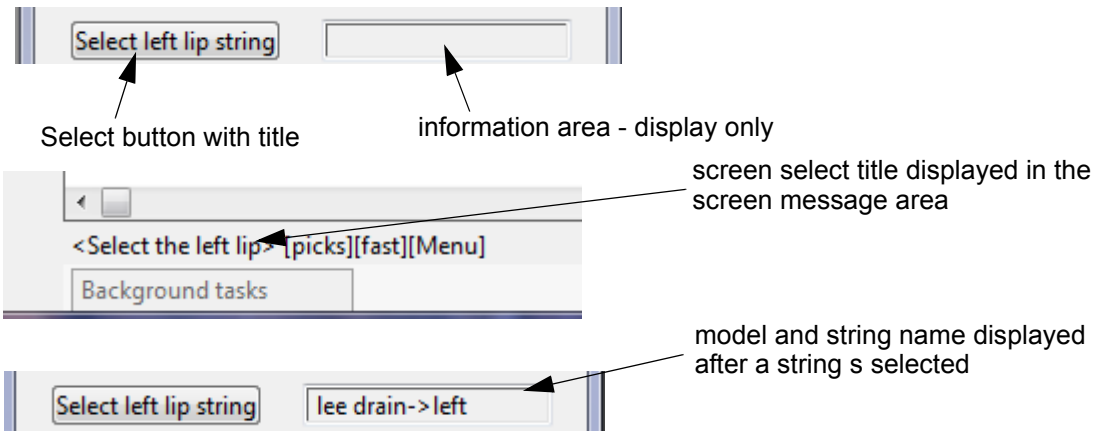
The ***Select_Box*** creates a panel field which is made up two items:

(a) a Select button on the left with the user supplied title on it

(b) an information area on the right where the name and model of the selected string are displayed

plus

(c) a screen select title that is displayed in the screen message area after the select button is selected.



Select_Box

A string is selected by first clicking **LB** on the button and then selecting the string (with **MB** or *accept* from the **Pick Ops** menu). The model and name of the selected string is then displayed in the information area.

A cursor pick can also be made first clicking **LB** on the button and then MB when at the required cursor position. For a cursor pick, nothing is displayed in the information area.

After the select is started, the screen select title for the button is displayed in the screen message area.

Clicking **MB** and **RB** on the select button does nothing.

Note: The *New_Select_Box* is normally used instead of the Select_Box. *See* New_Select_Box

Commands and Messages for Wait_on_Widgets

Clicking LB on the String Select button:

sends a "**start select**" command with nothing in *message*, then as the mouse is moved over a view, a "**motion select**" command is sent and the view coordinates and view name in *message*.

Once in the select:

if a string is clicked on with LB, or a cursor pick is made, a "**pick select**" command is sent with the name of the view that the string was selected in, in *message.* if the string or cursor pick is accepted (MB), an "**accept select**" command is sent with the view name (in quotes) in *message,* or if RB is clicked and *Cancel* selected from the *Pick Ops* menu, then a "**cancel select**" command is sent with nothing in *message.*

if a string, or cursor pick, is clicked on with MB (the pick and accept in one click method), a "**pick select**" command is sent with the name of the view that the string or cursor pick was selected in, in *message,* followed by an "**accept select**" command with the view name (in

quotes) in *message.*

Nothing else sends any commands or messages.

Create_select_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Name

Select_Box Create_select_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Description

Create an input Widget of type **Select_Box**.

The Select_Box is created with the title **title_text**.

The Select title displayed in the screen message area is **select_title**.

The value of **mode** is listed in the Appendix A - Select mode. *See* Select Mode .

The Message_Box **message** is normally the message box for the panel and is used to display string select validation messages.

The function return value is the created Select_Box.

ID = 882

Validate(Select_Box select,Element &string)

Name

Integer Validate(Select_Box select,Element &string)

Description

Validate the Element **string** in the Select_Box **select**.

The function returns the value of:

NO_NAME if the Widget Select_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 981

Validate(Select_Box select,Element &string,Integer silent)

Name

Integer Validate(Select_Box select,Element &string,Integer silent)

Description

Validate the Element **string** in the Select_Box **select**.

If **silent** = 0, and there is an error, a message is written and the cursor goes back to the box.
If **silent** = 1 and there is an error, no message or movement of cursor is done.

The function returns the value of SELECT_STRING indicates the string is selected successfully.

ID = 1376

Set_data(Select_Box select,Text model_string)

Name

Integer Set_data(Select_Box select,Text model_string)

Description

Set the Element in the Select_Box **select** by giving the model name and string name as a Text **model_string** in the form "model_name->string_name"

.A function return value of zero indicates the data was successfully set.

ID = 982

Set_data(Select_Box select,Element string)

Name

Integer Set_data(Select_Box select,Element string)

Description

Set the Element for the Select_Box **select** to **string**.

A function return value of zero indicates the data was successfully set.

ID = 1174

Get_data(Select_Box select,Text &string)

Name

Integer Get_data(Select_Box select,Text &string)

Description

Get the model and string name of the Element in Select_Box **select** and return it in the Text **model_string**,

Note: the model and string name is in the form "model_name->string_name" so only one Text is required.

A function return value of zero indicates the data was successfully returned.

ID = 983

Select_start(Select_Box select)

Name

Integer Select_start(Select_Box select)

Description

Starts the string selection for the Select_Box **select**. This is the same as if the button on the Select_Box had been clicked.

A function return value of zero indicates the start was successful.

ID = 1169

Select_end(Select_Box select)

Name

Integer Select_end(Select_Box select)

Description

Cancels the string selection that is running for the Select_Box **select**. This is the same as if *Cancel* had been selected from the *Pick Ops* menu.

A function return value of zero indicates the end was successful.

ID = 1170

Set_select_type(Select_Box select,Text type)

Name

Integer Set_select_type(Select_Box select,Text type)

Description

Set the string selection type **type** for the Select_Box **select**. For example “Alignment”, “3d”.

A function return value of zero indicates the type was successfully set.

ID = 1048

Set_select_snap_mode(Select_Box select,Integer snap_control)

Name

Integer Set_select_snap_mode(Select_Box select,Integer snap_control)

Description

Set the snap control for the Select_Box **select** to **snap_control**.

snap_control	**control value**
Ignore_Snap	= 0
User_Snap	= 1
Program_Snap	= 2

A function return value of zero indicates the snap control was successfully set.

ID = 1049

Set_select_snap_mode(Select_Box select,Integer snap_mode,Integer snap_control,Text snap_text)

Name

Integer Set_select_snap_mode(Select_Box select,Integer snap_mode,Integer snap_control,Text snap_text)

Description

Set the snap mode **snap_mode** and snap control **snap_control** for the Select_Box **select**.

Where **snap_mode** is:

Failed_Snap = -1
No_Snap = 0
Point_Snap = 1
Line_Snap = 2
Grid_Snap = 3
Intersection_Snap = 4
Cursor_Snap = 5
Name_Snap = 6

Tin_Snap = 7
Model_Snap = 8
Height_Snap = 9
Segment_Snap = 11
Text_Snap = 12
Fast_Snap = 13
Fast_Accept = 14

and **snap_control** is

Ignore_Snap = 0
User_Snap = 1
Program_Snap = 2

The **snap_text** must be *string name*; *tin name*, *model name* respectively, otherwise, leave the **snap_text** blank (“”).

A function return value of zero indicates the snap mode was successfully set.

ID = 1045

Get_select_direction(Select_Box select,Integer &dir)

Name

Integer Get_select_direction(Select_Box select,Integer &dir)

Description

Get the selection direction **dir** from the string selected for the Select_Box **select**.

The returned **dir** type must be **Integer**.

If select without direction, the returned **dir** is 1, otherwise, the returned dir is:

Dir Value	Pick direction
1	the direction of the string
-1	against the direction of the string

A function return value of zero indicates the direction was successfully returned.

ID = 1051

Get_select_coordinate(Select_Box select,Real &x,Real &y,Real &z,Real &ch,Real &ht)

Name

Integer Get_select_coordinate(Select_Box select,Real &x,Real &y,Real &z,Real &ch,Real &ht)

Description

Get the coordinates, chainage and height of the selected snap point of the string for the Select_Box **select**.

The return values of **x, y, z, ch**, and **ht** are of type **Real**.

A function return value of zero indicates the values were successfully returned.

ID = 1052

For information on the other Input Widgets, go to Input Widgets

Select_Boxes

The ***Select_Boxes*** is a panel item that contains a *number* of selection boxes.

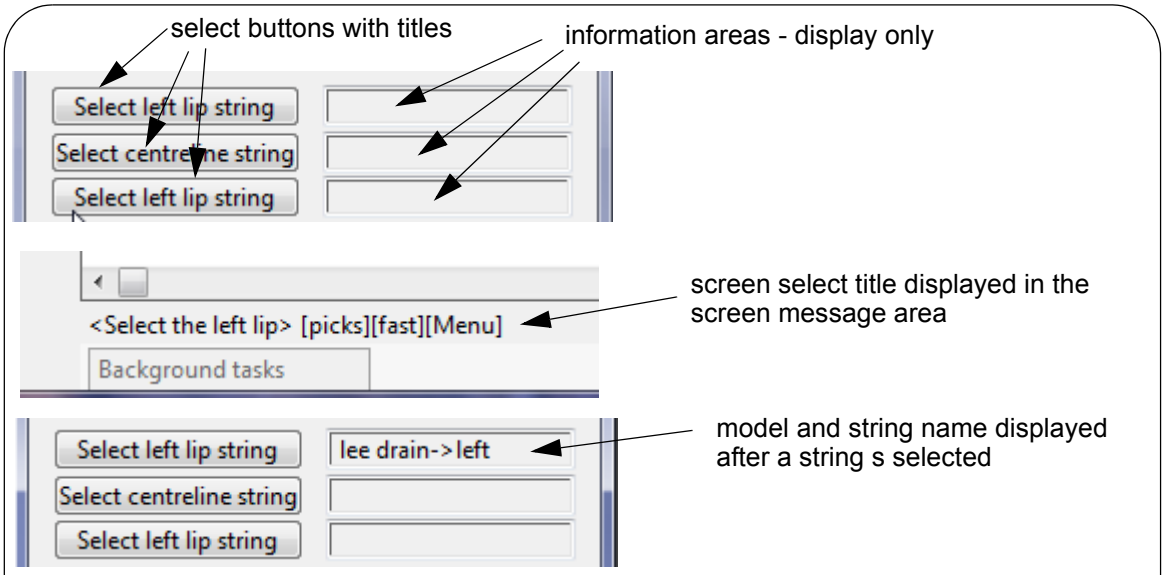
Each of the selection boxes is made up two items:

(a) a select button on the left with the user supplied title on it

(b) an information area on the right where the name and model of the selected string are displayed

plus

(c) a screen select title that is displayed in the screen message area after the select button is selected.



Select_Boxes

A string is selected by first clicking **LB** on one of the buttons and then selecting the string. The model and name of the selected string is then displayed in the information area for that button.

After the select is started, the screen select title for that button is displayed in the screen message area.

Clicking **MB** and **RB** on the select buttons does nothing.

Commands and Messages for Wait_on_Widgets

Select_Boxes consists of a number of selection boxes.

For the i'th selection box of the Select_Boxes:

Clicking LB on the i'th Select button:

sends a "**start select i**" command with nothing in *message*, then as the mouse is moved over a view, a "**motion select i**" command is sent and the view coordinates and view name in *message*.

Once in the select:

if a string is clicked on with LB, a "**pick select i**" command is sent with the name of the view that the string was selected in, in *message.* if the string is accepted (MB), an "**accept select i**" command is sent with the view name (in quotes) in *message,* or if RB is clicked and *Cancel* selected from the *Pick Ops* menu, then a "**cancel select i**" command is sent with nothing in *message.*

if a string is clicked on with MB (the pick and accept in one click method), a "**pick select i**"

command is sent with the name of the view that the string was selected in, in *message,* followed by an "**accept select i**" command with the view name (in quotes) in *message.*

Nothing else sends any commands or messages.

Create_select_boxes(Integer no_boxes,Text title_text[],Text select_title[],Integer mode[],Message_Box message)

Name

Select_Boxes Create_select_boxes(Integer no_boxes,Text title_text[],Text select_title[],Integer mode[],Message_Box message)

Description

Create an input Widget of type **Select_Boxes** which is actually a collection of 0 or more boxes that each acts like a Select_Box. *See* Select_Boxes .

no_boxes indicates the number of boxes in the boxes array.

The Select_Boxes are created with the titles given in the array **title_text[]**.

The Screen select titles displayed in the screen message area are given in the array **select_title[]**.

The value of **mode[]** is listed in the Appendix A - Select mode.

The Message_Box **message** is used to display the select information.

The function return value is the created **Select_Boxes**.

ID = 883

Validate(Select_Boxes select,Integer n,Element &string)

Name

Integer Validate(Select_Boxes select,Integer n,Element &string)

Description

Validate the **n**th Element **string** in the Select_Box **select**.

The function returns the value of:

NO_NAME if the **n**'th box of the New_Select_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 984

Validate(Select_Boxes select,Integer n,Element &string,Integer silent)

Name

Integer Validate(Select_Boxes select,Integer n,Element &string,Integer silent)

Description

Validate the **n**th Element **string** in the Select_Box **select**.

If **silent** = 0, and there is an error, a message is written and the cursor goes back to the box.
If **silent** = 1 and there is an error, no message or movement of cursor is done.

The function returns the value of:

NO_NAME if the **n**'th box of the New_Select_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1377

Set_data(Select_Boxes select,Integer n,Text model_string)

Name

Integer Set_data(Select_Boxes select,Integer n,Text model_string)

Description

Set the Element of the **n'**th box in the Select_Boxes **select** by giving the model name and string name as a Text **model_string** in the form "model_name->string_name".

A function return value of zero indicates the data was successfully set.

ID = 985

Set_data(Select_Boxes select,Integer n,Element string)

Name

Integer Set_data(Select_Boxes select,Integer n,Element string)

Description

Set the data of type Element for the **n'**th box in the Select_Boxes **select** to **string**.

A function return value of zero indicates the data was successfully set.

ID = 1175

Get_data(Select_Boxes select,Integer n,Text &model_string)

Name

Integer Get_data(Select_Boxes select,Integer n,Text &model_string)

Description

Get the model and string name of the Element in the **n'**th box of the Select_Boxes **select**. and return it in the Text **model_string**,

Note: the model and string name is in the form "model_name->string_name" so only one Text is required.

A function return value of zero indicates the data was successfully returned.

ID = 986

Select_start(Select_Boxes select,Integer n)

Name

Integer Select_start(Select_Boxes select,Integer n)

Description

Starts the string selection for the **n**'th box of the Select_Boxes **select**. This is the same as if the button on the **n**'th box of Select_Boxes had been clicked.

A function return value of zero indicates the start was successful.

ID = 1171

Select_end(Select_Boxes select,Integer n)

Name

Integer Select_end(Select_Boxes select,Integer n)

Description

Cancels the string selection that is running for the **n**'th box of the Select_Boxes **n**'th box of the Select_Boxes **select**. This is the same as if *Cancel* had been selected from the *Pick Ops* menu.

A function return value of zero indicates the end was successful.

ID = 1172

Set_select_type(Select_Boxes select,Integer n,Text type)

Name

Integer Set_select_type(Select_Boxes select,Integer n,Text type)

Description

Set the string selection for the **n**'th box of the Select_Boxes **select** to **type**. For example "Alignment", "3d".

A function return value of zero indicates the type was successfully set.

ID = 1053

Set_select_snap_mode(Select_Boxes select,Integer n,Integer control)

Name

Integer Set_select_snap_mode(Select_Boxes select,Integer n,Integer control)

Description

Set the snap control for **n**'th box of the Select_Boxes **select** to **control**.

snap control	control value
Ignore_Snap	0
User_Snap	
Program_Snap	2

A function return value of zero indicates the snap control was successfully set.

ID = 1054

Set_select_snap_mode(Select_Boxes select,Integer n,Integer snap_mode,Integer snap_control,Text snap_text)

Name

Integer Set_select_snap_mode(Select_Boxes select,Integer n,Integer snap_mode,Integer snap_control,Text

snap_text)

Description

Set the snap mode **mode** and snap control **snap_control** for the **n**th box of the Select_Boxes **select**.

When snap mode is:

Name_Snap	6
Tin_Snap	7
Model_Snap	8

the **snap_text** must be *string name; tin name*, *model name* respectively, otherwise, leave the **snap_text** blank (“”).

A function return value of zero indicates the snap mode was successfully set.

ID = 1055

Get_select_direction(Select_Boxes select,Integer n,Integer &dir)

Name

Integer Get_select_direction(Select_Boxes select,Integer n,Integer &dir)

Description

Get the selection direction **dir** of the string selected for the **n**’th box of the Select_Boxes **select**.

The returned **dir** type must be **Integer**.

If select without direction, the returned **dir** is 1, otherwise, the returned **dir** is:

Dir Value	Pick direction
1	the direction of the string
-1	against the direction of the string

A function return value of zero indicates the direction was successfully returned.

ID = 1056

Get_select_coordinate(Select_Boxes select,Integer n,Real &x,Real &y,Real &z,Real &ch,Real &ht)

Name

Integer Get_select_coordinate(Select_Boxes select,Integer n,Real &x,Real &y,Real &z,Real &ch,Real &ht)

Description

Get the coordinate, chainage and height of the snap point of the string selected for the **n’**th box of the Select_Boxes **select**.

The return value of **x, y, z, ch**, and **ht** are of type of **Real**.

A function return value of zero indicates the coordinate was successfully returned.

ID = 1057

For information on the other Input Widgets, go to Input Widgets

Sheet_Size_Box

The ***Sheet_Size_Box*** is a panel field designed to select a sheet size name, or type in a sheet size by giving width and height separate by spaces. The units for width and height are millimetres. If a sheet size name, or a width and height is typed into the box, then the sheet size name, or the width and height, will be validated when <enter> is pressed.

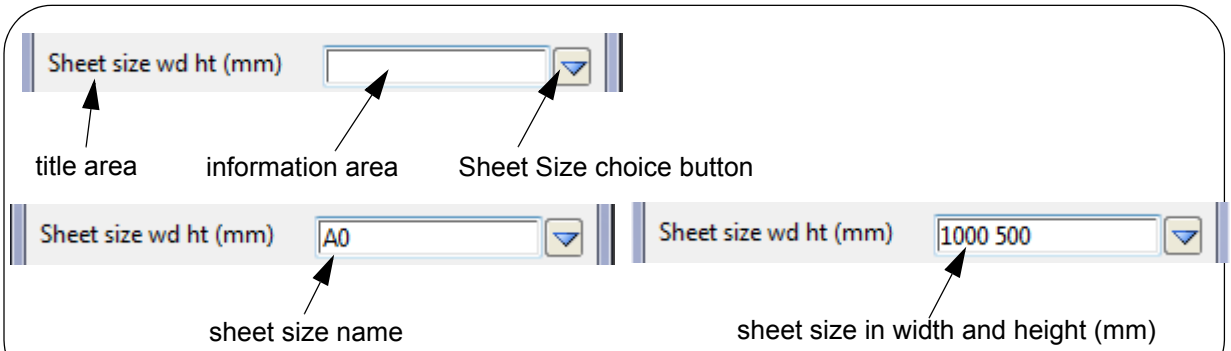
A ***Sheet_Size_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a sheet size name, or widths and heights of a sheet (where width and height are separated by spaces and the units are millimetres), or to display the sheet size name if it is selected by the Sheet Size select button. This information area is in the middle

and

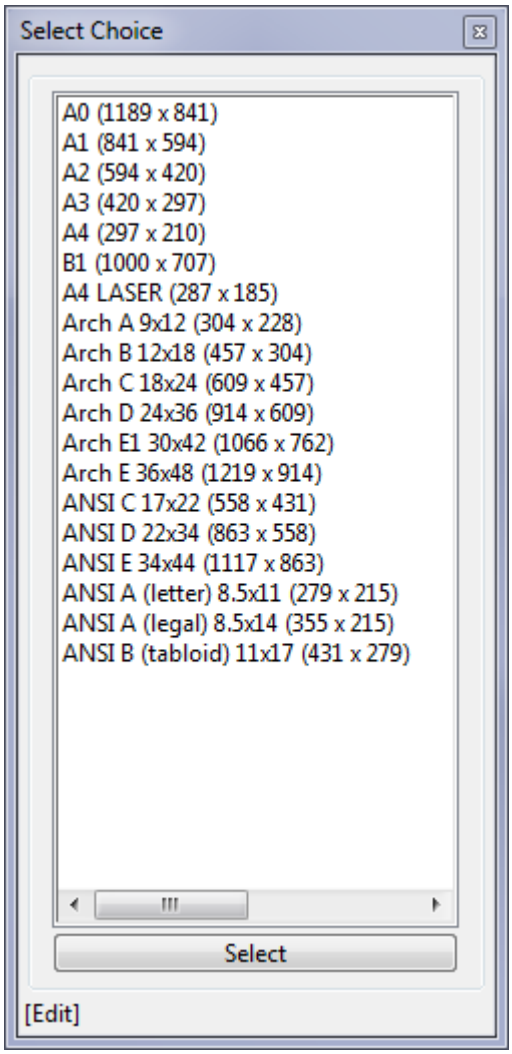
(c) a Sheet Size choice button on the right.



Sheet_Size_Box

A sheet size name can be typed into the ***information area,*** or widths and heights of a sheet (where width and height are separated by spaces and the units are millimetres). Then hitting the <enter> key will validate the sheet size.

Clicking **LB** or **RB** on the Sheet Size choice button brings up the *Select Sheet Size Choice* pop-up. Selecting a sheet size from the pop-up list writes the sheet size name in the information area.



Clicking **MB** on the Sheet Size choice button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and if

(a) the text in the information area is a valid sheet size choice, then a "**sheet selected**" command is sent with the sheet size choice in *message*

(b) if the text is made up of two words then a "**sheet selected**" command is sent with nothing in *message* (this could be a typed *width height*)

(c) if the text is not two words and is not a valid sheet size, then nothing is sent.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a justification after clicking on the Sheet Size Choice button sends a "**sheet selected**" command and the sheet size choice in *message*.

Create_sheet_size_box(Text title_text,Message_Box message)

Name

Sheet_Size_Box Create_sheet_size_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Sheet_Size_Box**. *See* Sheet_Size_Box .

The Sheet_Size_Box is created with the title **title_text**.

The Message_Box **message** is used to display sheet size information.

The function return value is the created Sheet_Size_Box.

ID = 946

Validate(Sheet_Size_Box box,Real &w,Real &h,Text &sheet)

Name

Integer Validate(Sheet_Size_Box box,Real &w,Real &h,Text &sheet)

Description

Validate the contents of Sheet_Size_Box **box** and return the width of the sheet as **w**, the height of the sheet as **h** and the sheet size as Text **sheet** or blank if it is not a standard size.

The function returns the value of:

NO_NAME if the Widget Sheet_Size_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *w, h, sheet* are valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 947

Get_data(Sheet_Size_Box box,Text &text_data)

Name

Integer Get_data(Sheet_Size_Box box,Text &text_data)

Description

Get the data of type Text from the Sheet_Size_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 949

Set_data(Sheet_Size_Box box,Text text_data)

Name

Integer Set_data(Sheet_Size_Box box,Text text_data)

Description

Set the data of type Text for the Sheet_Size_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 948

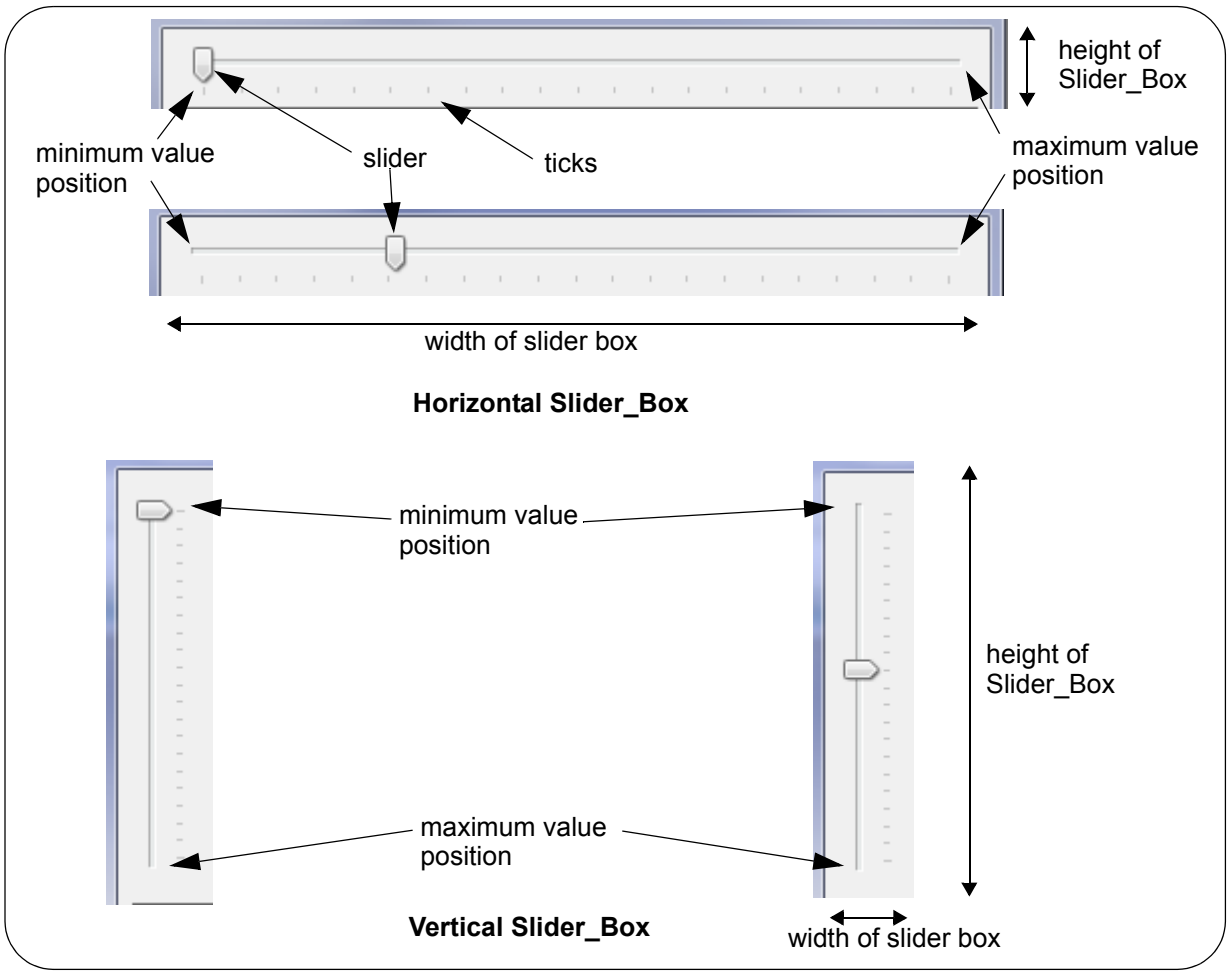
For information on the other Input Widgets, go to Input Widgets

Slider_Box

The ***Slider_Box*** is a panel field designed to display a slider (or bar) that the user is able to move along the Slider_Box.

The programmer supplies a minimum and maximum value for the Slider_Box and as the slider is moved in the Slider_Box, values are sent back to the macro indicating the position of the slider between the minimum and maximum values.

The **Slider_Box** can be horizontal or vertical.



Commands and Messages for Wait_on_Widgets

Moving the slider will send a "slider_updated" command back to the macro via the *Wait_on_widgets(id,cmd,msg)* call with the id of the Slider_Box. The actual value of the slider position is then given by the call *Get_slider_position*. *See* Get_slider_position(Slider_Box box,Integer &value) .

"**slider_updated**" - generated by holding the cursor on the slider and moving it to the left/right for a horizontal slider, or down/up for a vertical slider.

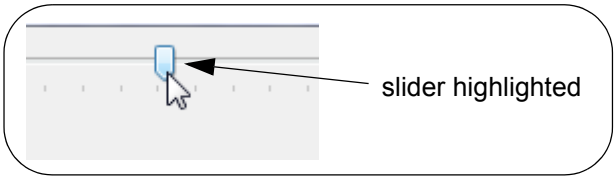
Moving the horizontal slider to the right increases the units
Moves the vertical slider down increases the units.

Moving the horizontal slider to the left decreases the units
Moves the vertical slider up decreases the units.

When the slider is finally released after moving it by the cursor, the "**slider_end_tracking**" command is returned via *Wait_on_widgets.*

When the slider is not being moved but the cursor is clicked on the slider and highlights it:

,



then other keystrokes are recognised and return the following text commands via the *Wait_on_widgets(id,cmd,msg)* call with the id of the Slider_Box.

"**slider_down**" - generated by pressing the right arrow (->) key or the down arrow key.

Moves the horizontal slider to the right by one unit
Moves the vertical slider down by one unit.

"**slider_up**" - generated by pressing the up arrow key or the left arrow (<-) key.

Moves the vertical slider up by one unit.
Moves the horizontal slider to the left by one unit

"**slider_top**" - generated by pressing the Home key.

Moves the vertical slider up to the top, and hence to the minimum value.
Moves the horizontal slider to the far left, and hence to the minimum value.

"**slider_bottom**" - generated by pressing the End key.

Moves the vertical slider down to the bottom, and hence to the maximum value.
Moves the horizontal slider to the far right, and hence to the maximum value.

"**slider_page_up**" - generated by pressing the Page Up key.

Moves the vertical slider up by a number of units.
Moves the horizontal slider to the left by a number of units.

"**slider_page_down**" - generated by pressing the Page Down key.

Moves the vertical slider down by a number of units.
Moves the horizontal slider to the right by a number of units.

After any of the above keystrokes, the "**slider_end_tracking**" command is returned via *Wait_on_widgets.*

After each of the commands, the value of the slider position is given by the call *Get_slider_position*. *See* Get_slider_position(Slider_Box box,Integer &value) .

Create_slider_box(Text name,Integer width,Integer height,Integer min_value,Integer max_value,Integer tick_interval,Integer horizontal)

Name

Slider_Box Create_slider_box(Text name,Integer width,Integer height,Integer min_value,Integer max_value,Integer tick_interval,Integer horizontal)

Description

Create an input Widget of type **Slider_Box**. *See* Slider_Box .

The Slider_Box can be horizontal or vertical.

If **horizontal** = 1 then the Slider_Box is horizontal.
If **horizontal** = 0 then the Slider_Box is vertical.

The range of values returned by the Slider_Box are specified by a minimum value (**min_val**) which is when the slider is at the left of a horizontal Slider_Box, or the top for a vertical Slider_Box, and a maximum value (**max_range**) which is reached when the slider is at the right of a horizontal Slider_Box, or at the bottom of a vertical Slider_Box.

min_value must be less than **max_val**.

Tick marks are drawn at the interval given by **tick_interval** on the bottom of a horizontal slider, of to the right of a vertical slider.

The slider box is created with a width **width** and height **height** where the width and height are given in screen units (pixels).

The function return value is the created **Slider_Box**.

Note: the height for a horizontal Slider_Box or the width for a vertical Slider_Box should be at least 30 or there will be no room to display the slider and tick marks.

ID = 2706

Set_slider_position(Slider_Box box,Integer value)

Name

Integer Set_slider_position(Slider_Box box,Integer value)

Description

Move the slider of Slider_Box **box** to the position given by **value** units of the Slider_Box.

A function return value of zero indicates the set was successful.

ID = 2707

Get_slider_position(Slider_Box box,Integer &value)

Name

Integer Get_slider_position(Slider_Box box,Integer &value)

Description

For the Slider_Box **box**, get the position of the slider in units of the Slider_Box and return the number of units in **value**.

A function return value of zero indicates the get was successful.

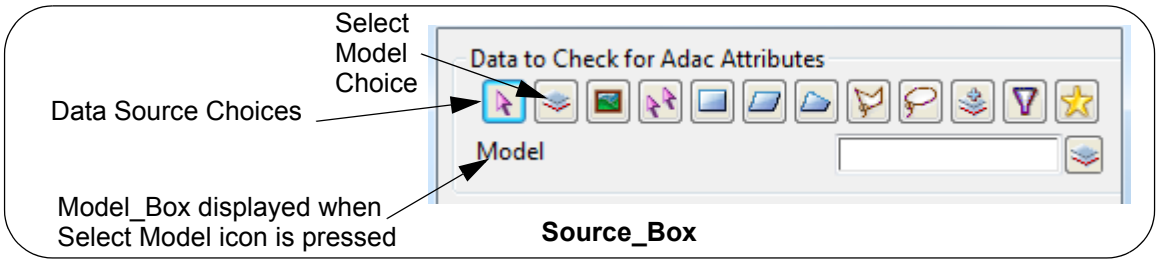
ID = 2708

Source_Box

The ***Source_Box*** is a panel field designed to allow the user to define how to select data.

The Source_Box consists of a row of Data Source Choices for the user to select one from, and when a Data Source Choice is selected, depending on the choice one or more additional fields will be presented to fully define/refine what date the user wishes to select.

For example, if the user selects the Select Model Choice, a Model_Box is then displayed for the user to enter a Model name.

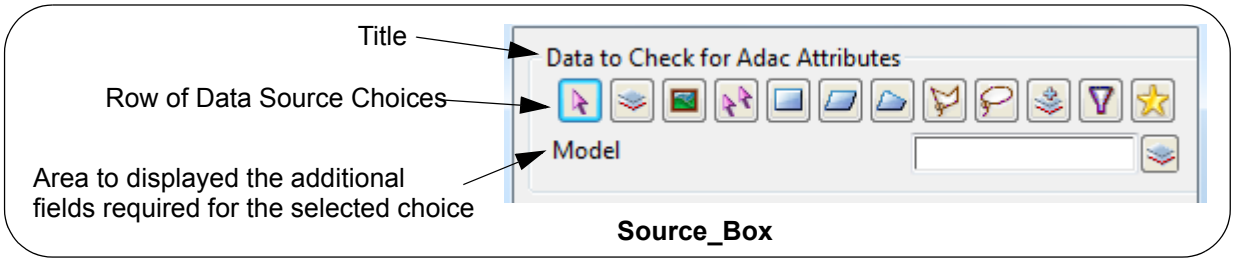


Hence a ***Source_Box*** is made up of three items:

(a) a title area above the row of Data Source Choices with the user supplied title on it

(b) the row of Data Source Choices to pick from

(c) an area under the row of Data Source Choices to display the extra panel fields required to fully define the users data selection method.



Note: If the panel appears to be sizing weirdly when there is a Source_Box involved, try putting all the Input Widgets into a Vertical_Group and then append the Vertical_Group to the Panel.

Note: A Source_Box cannot be made optional

Source_Box Create_source_box(Text title_text,Message_Box box,Integer flags)

Name

Source_Box Create_source_box(Text title_text,Message_Box box,Integer flags)

Description

Create an input Widget of type **Source_Box** which is used to define how to select data. *See* Source_Box .

The Source_Box is created with the title "Data " followed by **title_text**.

What Data Source Choices are displayed and hence available to select, is controlled by **flags**. i

If **flags** = 0, then all the choices are displayed.

Model	Source_Box_Model =	0x001 = 1
View	Source_Box_View =	0x002 = 2
String	Source_Box_String =	0x004 = 4
Rectangle	Source_Box_Rectangle =	0x008 = 8
Trapezoid	Source_Box_Trapezoid =	0x010 = 16
Polygon	Source_Box_Polygon =	0x020
Lasso	Source_Box_Lasso =	0x040
Filter	Source_Box_Filter =	0x080
Models	Source_Box_Models =	0x100
Favourites	Source_Box_Favorites =	0x200
All	Source_Box_All =	0xfff
Fence inside	Source_Box_Fence_Inside =	0x01000
Fence cross	Source_Box_Fence_Cross =	0x02000
Fence outside	Source_Box_Fence_Outside =	0x04000
Fence string	Source_Box_Fence_String =	0x08000
Fence points	Source_Box_Fence_Points =	0x10000
Fence all	Source_Box_Fence_All =	0xff000

Source_Box_Standard = Source_Box_All | Source_Box_Fence_Inside | Source_Box_Fence_Outside | Source_Box_Fence_Cross | Source_Box_Fence_String

You can have just some of them by combining the ones you want with |.
For example Source_Box_Model | Source_Box_View

The Message_Box message is used to display information.

The function return value is the created Source_Box.

ID = 1675

Validate(Source_Box box,Dynamic_Element &de_results)

Name

Integer Validate(Source_Box box,Dynamic_Element &elements)

Description

Validate the contents of Source_Box **box** and return the Dynamic_Element **de_results.**

The function returns the value of:

NO_NAME if the Widget Source_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *elements* is valid.

-2 if there is something wrong with the choices. For example the panel field is blank.

FALSE (zero) if there is a drastic error.

Having no Elements returned in **de_results** is NOT an error.

Always check the number of Elements in **de_results** and make your decisions based on that.

ierr = Get_number_of_items(de_results,no_elts);

So a function return value of zero indicates that there is a drastic error.

Warning this is the opposite of most 12dPL function return values

Double Warning: most times the function return code is non zero even when you think it should be. For example, when nothing is entered into the box, the return code is -2, not 0.

ID = 1676

Set_data(Source_Box box,Text text_data)

Name

Integer Set_data(Source_Box box,Text text_data)

Description

Set the data of type Text for the Source_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 2156

Get_data(Source_Box box,Text &text_data)

Name

Integer Get_data(Source_Box box,Text &text_data)

Description

Get the data of type Text from the Source_Edit_Box **box** and return it in **text_data**.

text_data describes what has been selected in the Source_Box. Because of all the choices it is very complicated looking.

A function return value of zero indicates the data was successfully returned.

ID = 2157

Read_favorite(Source_Box box,Text filename)

Name

Integer Read_favorite(Source_Box box,Text filename)

Description

For the Source_Box **box**, read in and set the Source_Box selection from the file named **filename**.

Note: the *Read_favourite* and *Write_favourite* calls allow Source_Box selection settings to be saved, and passed around between different Source_Box's.

A function return value of zero indicates filename was read and the Source_Box was successfully set.

ID = 2158

Write_favorite(Source_Box box,Text filename)

Name

Integer Write_favorite(Source_Box box,Text filename)

Description

For the Source_Box **box**, write out the Source_Box selection information to the file named **filename**.

Note: the *Read_favourite* and *Write_favourite* calls allow Source_Box selection settings to be saved, and passed around between different Source_Box's.

A function return value of zero indicates the file was successfully written.

ID = 2159

For information on the other Input Widgets, go to Input Widgets

Symbol_Box

The ***Symbol_Box*** is a panel field designed to select *12d Model* symbols. If a symbol name is typed into the box, then the symbol name will be validated when <enter> is pressed.

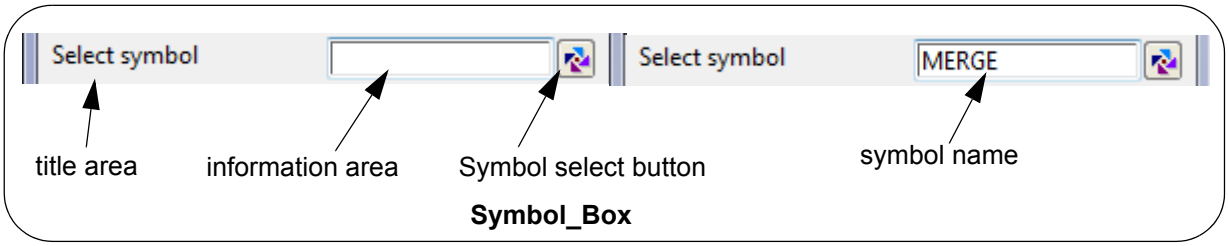
A ***Symbol_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a symbol name or to display the symbol name if it is selected by the Symbol select button. This information area is in the middle

and

(c) a Symbol select button on the right.

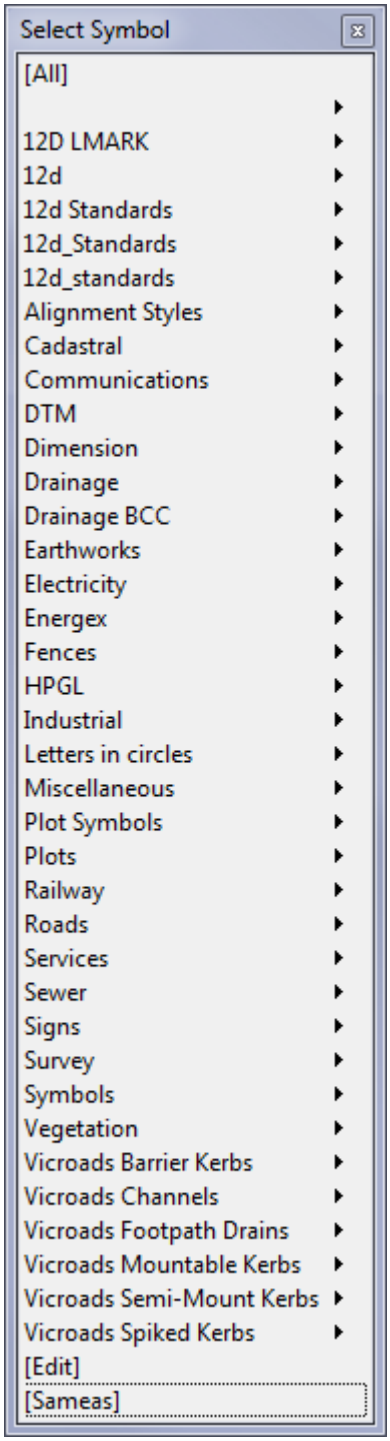


Symbol_Box

A symbol name can be typed into the ***information area***. Then hitting the <enter> key will validate the symbol name.

MB clicked in the ***information area*** starts a "Same As" selection. A symbol is then selected and the symbol name is written in the information area.

Clicking **LB** or **RB** on the Symbol select button brings up the *Select Symbol* pop-up. Selecting a symbol from the pop-up list writes the symbol name in the information area.



Clicking **MB** on the Symbol select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command with the symbol choice in *message*, or blank if it is not a valid symbol choice (that is, it is not in the Symbol list).

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command. Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a justification after clicking on the Symbol Select button sends a "**text selected**" command and the symbol choice in *message*.

Symbol_Box Create_symbol_box(Text title_text,Message_Box message,Integer mode)

Name

Symbol_Box Create_symbol_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Symbol_Box**. *See* Symbol_Box .

The Symbol_Box is created with the title **title_text**.

The Message_Box message is used to display information.

LJG? **mode**

The function return value is the created Symbol_Box.

ID = 2170

Validate(Symbol_Box box,Integer mode,Text &result)

Name

Integer Validate(Symbol_Box box,Integer mode,Text &result)

Description

Validate the contents of Symbol_Box **box** and return the name of the symbol in Text **result.**

LJG? The value of **mode** is listed in the Appendix A - Symbol mode. See Symbol Mode

The function returns the value of:

NO_NAME if the Widget Symbol_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 2171

Get_data(Symbol_Box box,Text &text_data)

Name

Integer Get_data(Symbol_Box box,Text &text_data)

Description

Get the data of type Text from the Symbol_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 2172

Set_data(Symbol_Box box,Text text_data)

Name

Integer Set_data(Symbol_Box box,Text text_data)

Description

Set the data of type Text for the Symbol_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 2173

For information on the other Input Widgets, go to Input Widgets

Target_Box

Target_Box Create_target_box(Text title_text,Message_Box box,Integer flags)

Name

Target_Box Create_target_box(Text title_text,Message_Box box,Integer flags)

Description

Create an input Widget of type **Target_Box**. *See* Target_Box .

The Target_Box is created with the title **title_text**.

The Message_Box message is used to display information.

LJG?flags

The function return value is the created Target_Box.

ID = 1677

Validate(Target_Box box)

Name

Integer Validate(Target_Box box)

Description

<no description>

ID = 1678

Validate(Target_Box box,Integer &mode,Text &text_data) For V10 only

Name

Integer Validate(Target_Box box,Integer &mode,Text &text_data)

Description

<no description>

ID = 2653

For information on the other Input Widgets, go to Input Widgets

Template_Box

The ***Template_Box*** is a panel field designed to select, or create *12d Model* templates. If a template name is typed into the box, then the template name will be validated when <enter> is pressed.

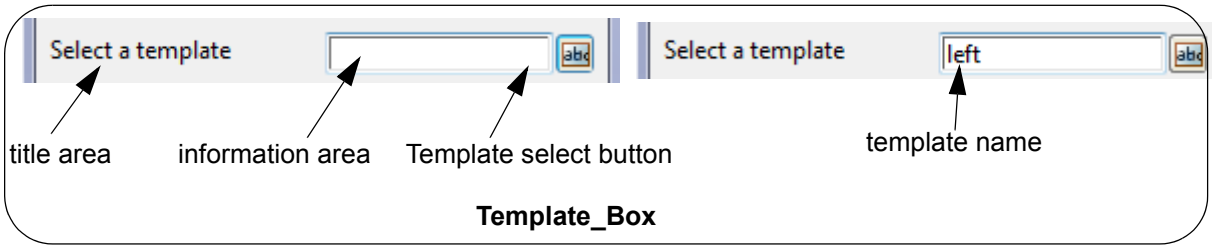
A ***Template_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a template name or to display the template name if it is selected by the template select button. This information area is in the middle

and

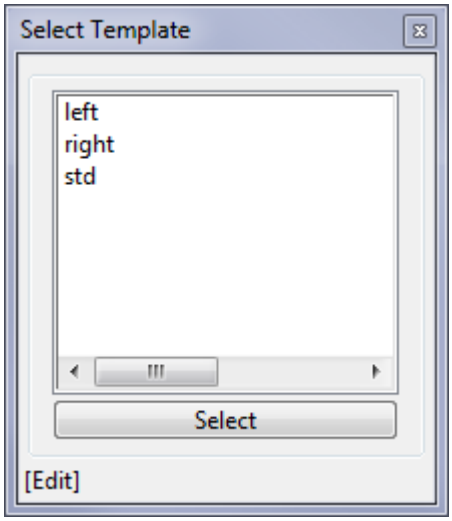
(c) a Template select button on the right.



Template_Box

A template name can be typed into the ***information area***. Then hitting the <enter> key will validate the template name.

Clicking **LB** or **RB** on the Template select button brings up the *Select Template* pop-up. Selecting a template from the pop-up list writes the template name in the information area.



Clicking **MB** on the template select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command with the text in *message.*

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu

are documented in the section Widget Information Area Menu .

Picking a template after clicking on the Justification Choice button sends a "**text selected**" command and the template choice in *message*.

Create_template_box(Text title_text,Message_Box message,Integer mode)

Name

Template_Box Create_template_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Template_Box**. *See* Template_Box .

The Template_Box is created with the title **title_text**.

The Message_Box **message** is used to display template information.

The value of **mode** is listed in the Appendix A - Template mode.

The function return value is the created Template_Box.

ID = 942

Validate(Template_Box box,Integer mode,Text &result)

Name

Integer Validate(Template_Box box,Integer mode,Text &result)

Description

Validate the contents of Template_Box **box** and return the Text **result.**

The value of **mode** is listed in the Appendix A - Template mode. See Template Mode

The value **result** must be type of **Text**.

The function returns the value of:

NO_NAME if the Widget Template_Box is optional and the box is left empty

NO_TEMPLATE, TEMPLATE_EXISTS, DISK_TEMPLATE_EXISTS or NEW_TEMPLATE

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 943

Get_data(Template_Box box,Text &text_data)

Name

Integer Get_data(Template_Box box,Text &text_data)

Description

A function return value of zero indicates the data was successfully returned.

Get the data of type Text from the Template_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 945

Set_data(Template_Box box,Text text_data)

Name

Integer Set_data(Template_Box box,Text text_data)

Description

Set the data of type Text for the Template_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 944

For information on the other Input Widgets, go to Input Widgets

Text_Style_Box

The ***Text_Style_Box*** is a panel field designed to select *12d Model* text styles. If a text style name is typed into the box, then the text style name will be validated when <enter> is pressed.

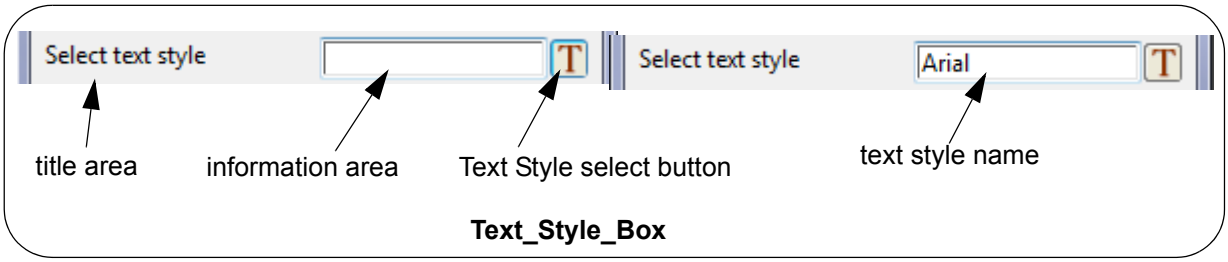
A ***Text_Style_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a text style name or to display the text style name if it is selected by the text style select button. This information area is in the middle

and

(c) a text style select button on the right.

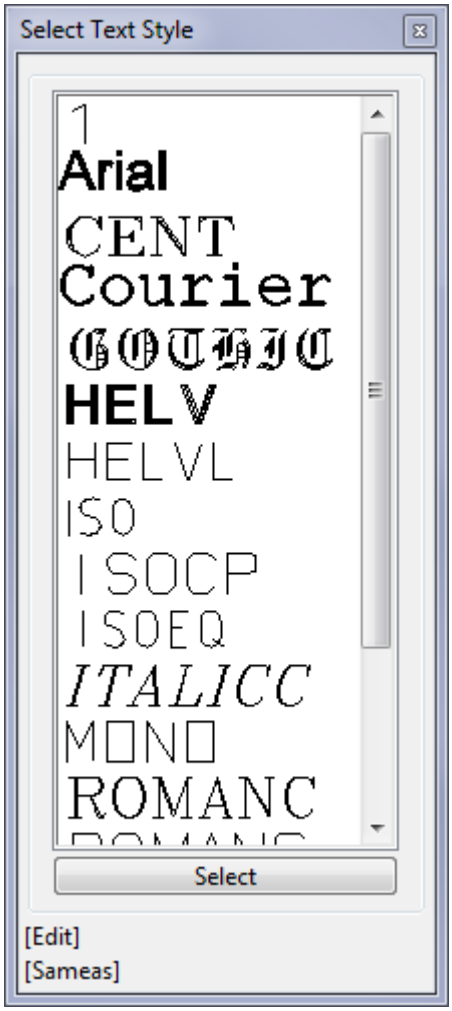


Text_Style_Box

A text style name can be typed into the ***information area***. Then hitting the <enter> key will validate the text style name.

MB clicked in the ***information area*** starts a "Same As" selection. A text string is then selected and the text style of the string is written in the information area.

Clicking **LB** or **RB** on the Text Style select button brings up the *Select Text Style* pop-up. Selecting a text style from the pop-up list writes the text style name in the information area.



Clicking **MB** on the Text Style select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command with the text in *message.*

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a text style after clicking on the Text Style select button sends a "**text selected**" command and the text style choice in *message*.

Create_text_style_box(Text title_text,Message_Box message)

Name

Text_Style_Box Create_text_style_box(Text title_text,Message_Box message)

Description

Create an input of type **Text_Style_Box**. *See* Text_Style_Box .

The Text_Style_Box is created with the title **title_text**.

The Message_Box **message** is used to display the text style information.

The function return value is the created Text_Style_Box.

ID = 950

Validate(Text_Style_Box box,Text &result)

Name

Integer Validate(Text_Style_Box box,Text &result)

Description

Validate the contents of Text_Style_Box **box** and return name of the textstyle as the Text **result.**

The function returns the value of:

NO_NAME if the Widget Text_Style_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 951

Get_data(Text_Style_Box box,Text &text_data)

Name

Integer Get_data(Text_Style_Box box,Text &text_data)

Description

Get the data of type Text from the Text_Style_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 953

Set_data(Text_Style_Box box,Text text_data)

Name

Integer Set_data(Text_Style_Box box,Text text_data)

Description

Set the data of type Text for the Text_Style_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 952

For information on the other Input Widgets, go to Input Widgets

Text_Units_Box

The ***Text_Units_Box*** is a panel field designed to select one item from a list of text units. If data is typed into the box, then it will be validated when <enter> is pressed.

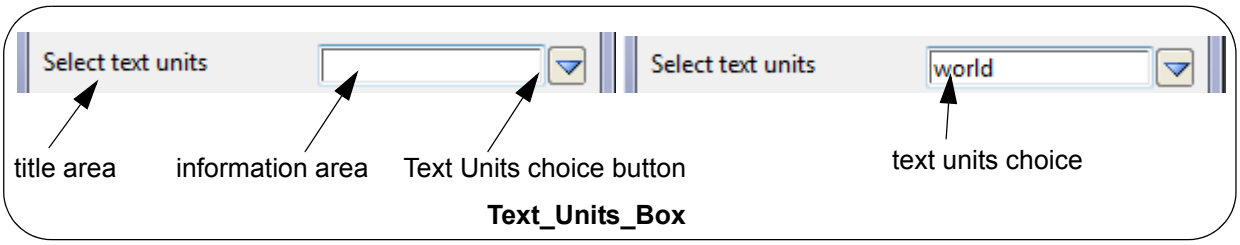
A ***Text_Units_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in text units or to display a units choice if it is selected by the text units choice button. This information area is in the middle

and

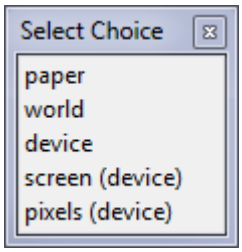
(c) a Text Units choice button on the right.



Text_Units_Box

A text units can be typed into the ***information area*** and hitting the <enter> key will validate the text units. Note that to be valid, the typed in text units must exist in the Text Units choice pop-up list.

Clicking **LB** or **RB** on the Text Units choice button brings up the *Select Choice* pop-up list. Selecting a Text Units choice from the pop-up list writes the text units to the information area.



Clicking **MB** on the Text Units choice button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command with the text units choice in *message*, or blank if it is not a valid text unit.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a text unit after clicking on the Text Units Choice button sends a "**text selected**" command and the text unit choice in *message*.

Create_text_units_box(Text title_text,Message_Box message)

Name

Text_Units_Box Create_text_units_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Text_Units_Box**. *See* Text_Units_Box .

The Text_Units_Box is created with the title **title_text**.

The Message_Box **message** is used to display the text units information.

The function return value is the created Text_Units_Box.

ID = 954

Validate(Text_Units_Box box,Integer &result)

Name

Integer Validate(Text_Units_Box box,Integer &result)

Description

Validate the contents of Text_Units_Box **box** and return the Integer **result.**

The function returns the value of:

NO_NAME if the Widget Text_Units_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 955

Get_data(Text_Units_Box box,Text &text_data)

Name

Integer Get_data(Text_Units_Box box,Text &text_data)

Description

Get the data of type Text from the Text_Units_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 957

Set_data(Text_Units_Box box,Integer integer_data)

Name

Integer Set_data(Text_Units_Box box,Integer integer_data)

Description

Set the data of type Integer for the Text_Units_Box **box** to **integer_data**.

A function return value of zero indicates the data was successfully set.

ID = 956

Set_data(Text_Units_Box box,Text text_data)

Name

Integer Set_data(Text_Units_Box box,Text text_data)

Description

Set the data of type Text for the Text_Units_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1519

For information on the other Input Widgets, go to Input Widgets

Textstyle_Data_Box

Textstyle_Data_Box Create_textstyle_data_box(Text text,Message_Box box,Integer flags)

Name

Textstyle_Data_Box Create_textstyle_data_box(Text text,Message_Box box,Integer flags)

Description

Create an input Widget of type **Textstyle_Data_Box**. *See* Textstyle_Data_Box .

The Textstyle_Data_Box is created with the title **title_text**.

The Message_Box message is used to display the information.

LJG?flags

The function return value is the created Textstyle_Data_Box.

ID = 1671

Validate(Textstyle_Data_Box box,Textstyle_Data &data)

Name

Integer Validate(Textstyle_Data_Box box,Textstyle_Data &data)

Description

Validate the contents of Textstyle_Data_Box **box** and return the Textstyle_Data **data.**

The function returns the value of:

NO_NAME if the Widget Textstyle_Data_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *data* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1672

Set_data(Textstyle_Data_Box box,Textstyle_Data data)

Name

Integer Set_data(Textstyle_Data_Box box,Textstyle_Data data)

Description

Set the data of type Textstyle_Data for the Textstyle_Data_Box **box** to **data**.

A function return value of zero indicates the data was successfully set.

ID = 1673

Set_data(Textstyle_Data_Box box,Text text_data)

Name

Integer Set_data(Textstyle_Data_Box box,Text text_data)

Description

Set the data of type Text for the Texstyle_Data_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 2161

Get_data(Textstyle_Data_Box box,Textstyle_Data &data)

Name

Integer Get_data(Textstyle_Data_Box box,Textstyle_Data &data)

Description

Get the data of type Textstyle_Data from the Textstyle_Data_Box **box** and return it in **data**.

A function return value of zero indicates the data was successfully returned.

ID = 1674

Get_data(Textstyle_Data_Box box,Text &text_data)

Name

Integer Get_data(Textstyle_Data_Box box,Text &text_data)

Description

Get the data of type Text from the Textstyle_Data_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 2160

For information on the other Input Widgets, go to Input Widgets

Text_Edit_Box

Create_text_edit_box(Text title_text,Message_Box box,Integer no_lines)

Name

Text_Edit_Box Create_text_edit_box(Text title_text,Message_Box box,Integer no_lines)

Description

Create an input Widget of type **Text_Edit_Box**. *See* Text_Edit_Box .

The Text_Edit_Box is created with the title **title_text**.

The **Message_Box** box is used to display information.

The number of lines allowed is **no_lines**.

The function return value is the created Text_Edit_Box.

ID = 1372

Set_data(Text_Edit_Box box,Text text_data)

Name

Integer Set_data(Text_Edit_Box box,Text text_data)

Description

Set the data of type Text for the Text_Edit_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1374

Set_data(Text_Edit_Box widget,Dynamic_Text dt_data)

Name

Integer Set_data(Text_Edit_Box widget,Dynamic_Text dt_data)

Description

Set the data of type Dynamic_Text for the Text_Edit_Box **widget** to **dt_data**.

A function return value of zero indicates the data was successfully set.

ID = 1617

Get_data(Text_Edit_Box widget,Text &text_data)

Name

Integer Get_data(Text_Edit_Box widget,Text &text_data)

Description

Get the data of type Text from the Text_Edit_Box **widget** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1373

Get_data(Text_Edit_Box widget,Dynamic_Text &dt_data)

Name

Integer Get_data(Text_Edit_Box widget,Dynamic_Text &dt_data)

Description

Get the data of type Dynamic_Text from the Text_Edit_Box **widget** and return it in **dt_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1616

For information on the other Input Widgets, go to Input Widgets

Texture_Box

The ***Texture_Box*** is a panel field designed to select *12d Model* linestyles. If a texture name is typed into the box, then the texture name will be validated when <enter> is pressed.

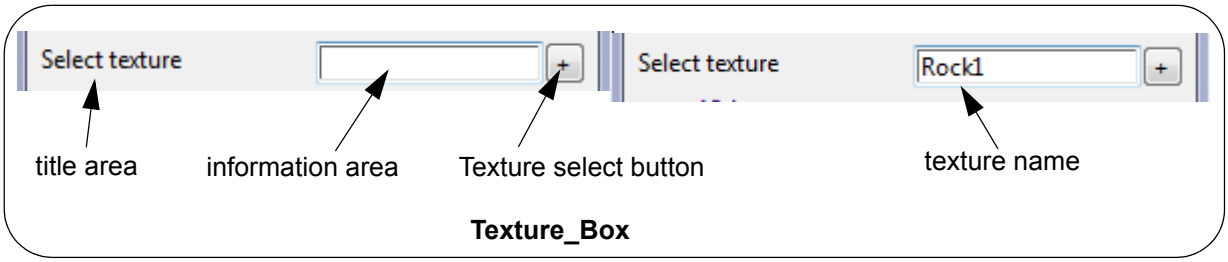
A ***Texture_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a texture name or to display the texture name if it is selected by the Textstyle select button. This information area is in the middle

and

(c) a Texture select button on the right.

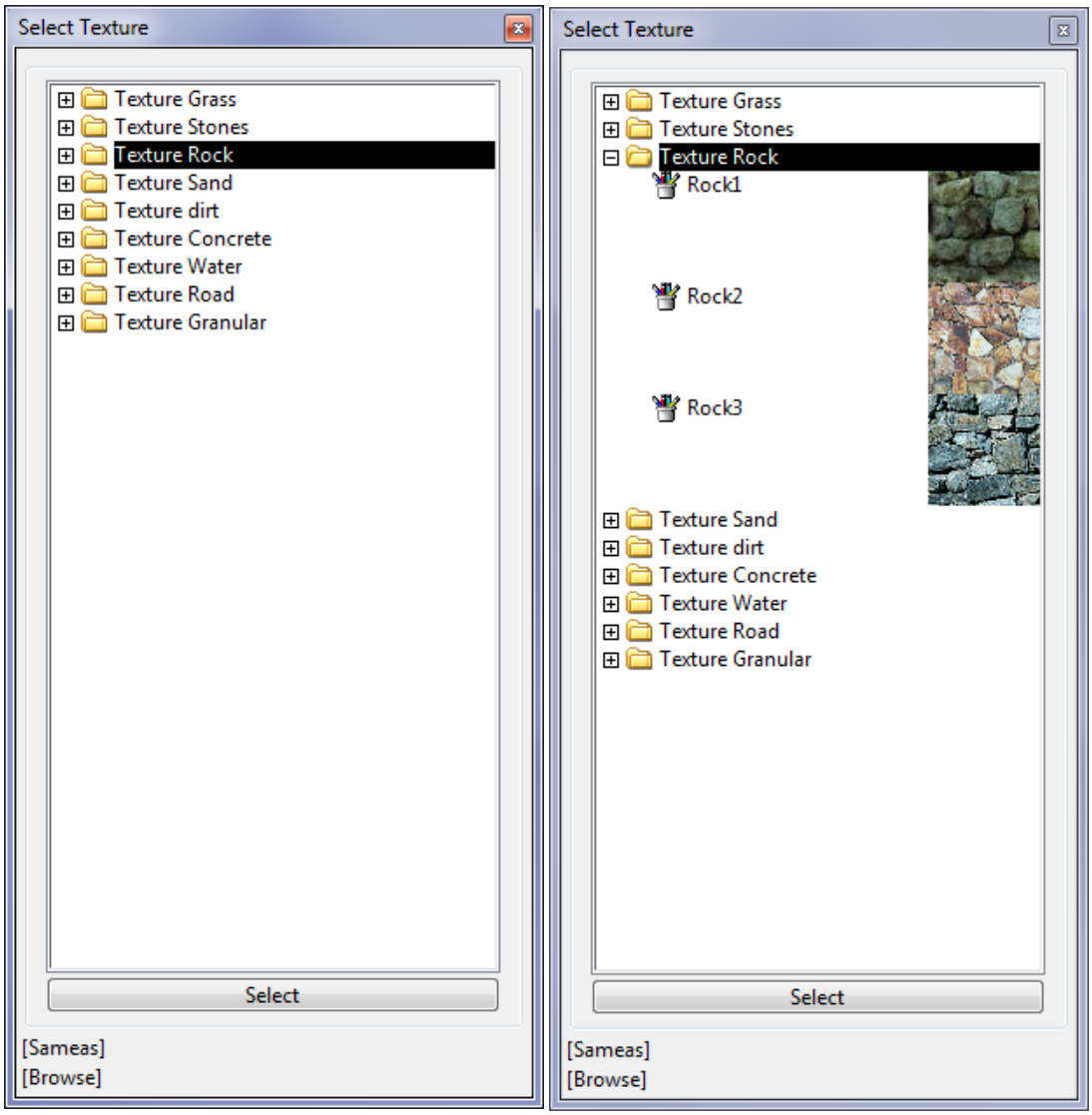


Texture_Box

A texture name can be typed into the ***information area***. Then hitting the <enter> key will validate the texture name.

MB clicked in the ***information area*** starts a "Same As" selection. A string with a texture is then selected and the texture of the string is written in the information area.

Clicking **LB** or **RB** on the Texture select button brings up the *Select Texture* pop-up. Selecting a texture from the pop-up list writes the texture name in the information area.



Clicking **MB** on the Textures select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**text selected**" command with the text in *message.*

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a texture after clicking on the Texture select button sends a "**text selected**" command and the texture choice in *message*.

Texture_Box Create_texture_box(Text title_text,Message_Box message)

Name

Texture_Box Create_texture_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Texture_Box**. *See* Texture_Box .

The Texture_Box is created with the title **title_text**.

The Message_Box message is used to display information.

The function return value is the created Texture_Box.

ID = 1875

Validate(Texture_Box box,Text &result)

Name

Integer Validate(Texture_Box box,Text &result)

Description

Validate the contents of Texture_Box **box** and return the name of the texture in Text **result.**

The function returns the value of:

NO_NAME if the Widget Texture_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1876

Set_data(Texture_Box box,Text text_data)

Name

Integer Set_data(Texture_Box box,Text text_data)

Description

Set the data of type Text for the Texture_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1877

Get_data(Texture_Box box,Text &text_data)

Name

Integer Get_data(Texture_Box box,Text &text_data)

Description

Get the data of type Text from the Texture_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1878

For information on the other Input Widgets, go to Input Widgets

Tick_Box

The Tick_Box has been superseded by the Named_Tick_Box .

Create_tick_box(Message_Box message)

Name

Tick_Box Create_tick_box(Message_Box message)

Description

Create an input Widget of type **Tick_Box**. *See* Tick_Box .

The Message_Box message is used to display the tick information.

The function return value is the created Tick_Box.

ID = 958

Validate(Tick_Box box,Integer &result)

Name

Integer Validate(Tick_Box box,Integer &result)

Description

Validate **result** (of type **Integer**) in the Tick_Box **box**.

Validate the contents of Tick_Box **box** and return the Integer **result.**

result = 0	if the tick box is unticked
result = 1	if the tick box is ticked

A function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 959

Get_data(Tick_Box box,Text &text_data)

Name

Integer Get_data(Tick_Box box,Text &text_data)

Description

Get the data of type Text from the Tick_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 961

Set_data(Tick_Box box,Text text_data)

Name

Integer Set_data(Tick_Box box,Text text_data)

Description

Set the data of type Text for the Tick_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 960

For information on the other Input Widgets, go to Input Widgets

Tin_Box

The ***Tin_Box*** is a panel field designed to select *12d Model* tins. If a tins name is typed into the tins box and <enter> pressed or a tins selected from the tins pop-up list, then the text in the Tin_Box is validated.

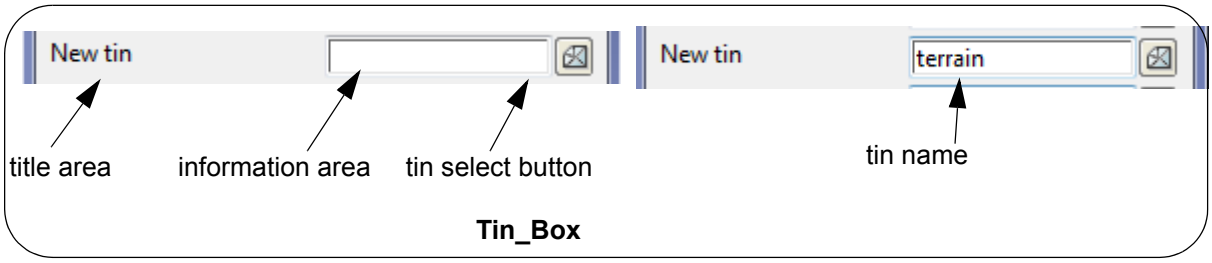
A ***Tin_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a tin name or to display the tin name if it is selected by the tin select button. This information area is in the middle

and

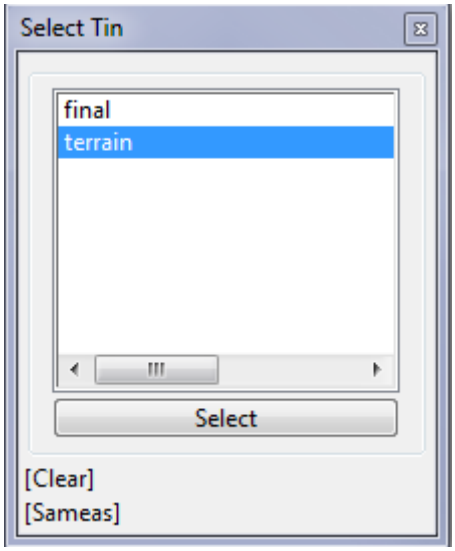
(c) a tin select button on the right.



A tin name can be typed into the ***information area***. Then hitting the <enter> key validate the tin name.

MB clicked in the ***information area*** starts a "Same As" selection. LJG This does nothing useful.

Clicking **LB** or **RB** on the tin select button brings up the *Select Model* pop-up. Selecting a tin from the pop-up list writes the tin name in the information area and validation occurs.



Clicking **MB** on the tin select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text

of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**tin selected**" command and the text in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a tin with the Tin Select button sends a "**tin selected**" command and the tin name in *message*.

Create_tin_box(Text title_text,Message_Box message,Integer mode)

Name

Tin_Box Create_tin_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Tin_Box** for inputting and validating Tins.

The Tin_Box is created with the title **title_text** (*see* Tin_Box).

The Message_Box **message** is normally the message box for the panel and is used to display Model_Box validation messages.

If <enter> is typed into the Tin_Box or a tin selected from the tin pop-up list, automatic validation is performed by the Tin_Box according to **mode**. What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

For example,

```
CHECK_TIN_MUST_EXIST        // if the tins exists, the message says "exists"
                            // if it doesn't exist, the messages says "ERROR"
```

The values for **mode** and their actions are listed in Appendix A (see Tin Mode).

The function return value is the created Tin_Box.

ID = 962

Validate(Tin_Box box,Integer mode,Tin &result)

Name

Integer Validate(Tin_Box box,Integer mode,Tin &result)

Description

Validate the contents of Tin_Box **box** and return the Tin **result.**

The value of **mode** will determine what validation occurs, what messages are written to the Message_Box, what actions are taken and what the function return value is.

The values for **mode** and the actions are listed in Appendix A (see Tin Mode).

The function return values depends on **mode** and are given in Appendix A (see Tin Mode).

A function return value of zero indicates that there is a drastic error.

Warning this is the opposite of most 12dPL function return values

Double Warning: most times the function return code is not zero even when you think it should be. The actual value of the function return code must be checked to see what is going on. For

example, when **mode** = CHECK_TIN_MUST_EXIST will return NO_TIN if the tin name is not blank and no tin of that name exist (NO_TIN does not equal zero).

ID = 963

Get_data(Tin_Box box,Text &text_data)

Name

Integer Get_data(Tin_Box box,Text &text_data)

Description

Get the data of type Text from the Tin_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 965

Set_data(Tin_Box box,Text text_data)

Name

Integer Set_data(Tin_Box box,Text text_data)

Description

Set the data of type Text for the Tin_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 964

For information on the other Input Widgets, go to Input Widgets

View_Box

The ***View_Box*** is a panel field designed to select *12d Model* views. If a view name is typed into the view box and <enter> pressed or a view selected from the view pop-up list, then the text in the View_Box is validated.

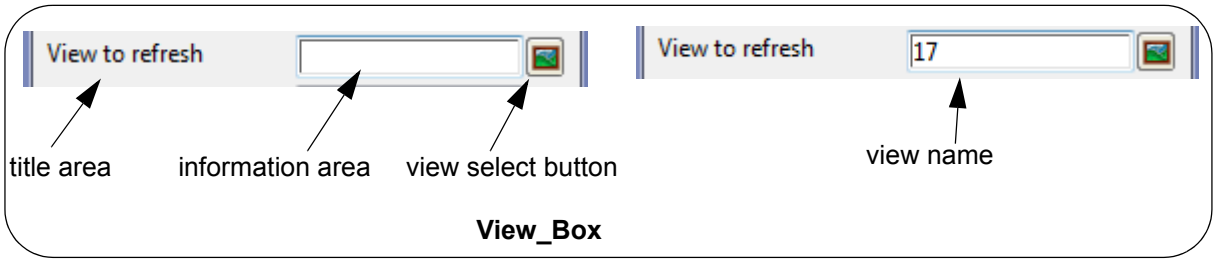
A ***View_Box*** is made up of three items:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in a view name or to display the view name if it is selected by the view select button. This information area is in the middle

and

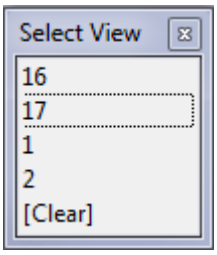
(c) a view select button on the right.



View_Box

A view name can be typed into the ***information area***. Then hitting the <enter> key validates the view name.

Clicking **LB** or **RB** on the view select button brings up the *Select View* pop-up. Selecting a view from the pop-up list writes the view name in the information area and validation occurs.



Clicking **MB** on the view select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and if it is an existing view, then a "**view selected**" command is sent with the view name in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a view with the View Select button sends a "**view selected**" command and the view name in *message*.

Create_view_box(Text title_text,Message_Box message,Integer mode)

Name

View_Box Create_view_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **View_Box** for inputting and validating Views.

The View_Box is created with the title **title_text** (*see* View_Box).

The Message_Box **message** is normally the message box of the panel and is used to display the View_Box validation messages.

If an <enter> is typed in the View_Box or a view selected from the view pop-up list, automatic validation is performed by the View_Box according to **mode** - what the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

For example,

CHECK_TIN_MUST_EXIST // if the model exists, the message says "exists" and
// if it doesn't exist, the messages says "ERROR"

The value of **mode** and their actions are listed in Appendix A (*see* View Mode).

The function return value is the created **View_Box**.

ID = 966

Validate(View_Box box,Integer mode,View &result)

Name

Integer Validate(View_Box box,Integer mode,View &result)

Description

Validate the contents of View_Box **box** and return the View **result.**

The value of **mode** will determine what validation occurs, what messages are written to the Message_Box, what actions are taken and what the function return value is.

The values for **mode** and the actions are listed in Appendix A (see View Mode).

The function return value depends on **mode** and are given in Appendix A (see View Mode).

A function return value of zero indicates that there is a drastic error.

Warning this is the opposite of most 12dPL function return values

Double Warning: most times the function return code is not zero even when you think it should

be. The actual value of the function return code must be checked to see what is going on. For example, when mode = CHECK_TIN_MUST_EXIST will return NO_TIN if the tin name is not blank and no tin of that name exist (NO_TIN does not equal zero).

ID = 967

Get_data(View_Box box,Text &text_data)

Name

Integer Get_data(View_Box box,Text &text_data)

Description

Get the data of type Text from the View_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 969

Set_data(View_Box box,Text text_data)

Name

Integer Set_data(View_Box box,Text text_data)

Description

Set the data of type Text for the View_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 968

For information on the other Input Widgets, go to Input Widgets

XYZ_Box

The ***XYZ_Box*** is a panel field designed to get X, Y and Z coordinates which are displayed in the one information area, separated by spaces.

Also see New_XYZ_Box where each of X, Y and Z are each displayed in their own information areas.

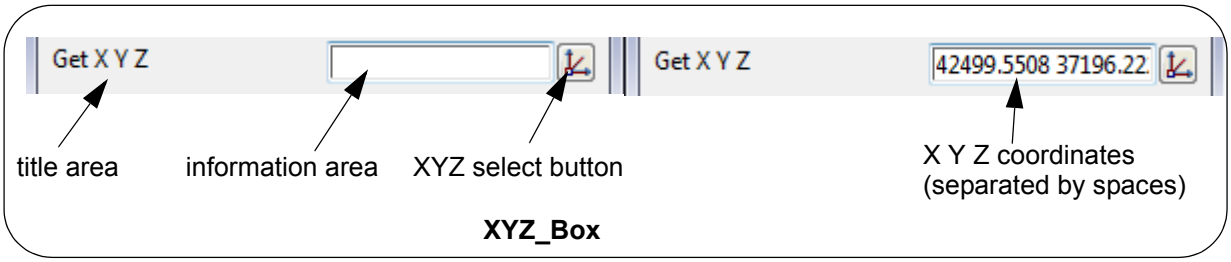
The ***XYZ_Box*** is made up of:

(a) a title area on the left with the user supplied title on it

(b) an information area to type in the X Y and Z values, each value separated by one or more spaces, or to display the X Y Z coordinates if a position is selected by the XYZ select button. This information area is in the middle

and

(c) a XYZ select button on the right.

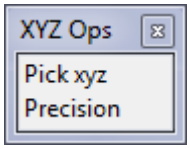


XYZ_Box

XYZ coordinates can be typed into the ***information area***, each value separated by one or more spaces. Then hitting the <enter> key will validate that the three values are all Real numbers.

Clicking **LB** on the XYZ select button starts the XYZ Pick option and after selecting a position, the X, Y and Z values are displayed information area separated by spaces.

Clicking **RB** on the XYZ select button brings up the *XYZ Ops* pop-up menu. Selecting Pick xyz option starts the XYZ Pick option and after a position, the X, Y and Z values are displayed in the information area separated by spaces.



Clicking **MB** on the XYZ select button does nothing.

Commands and Messages for Wait_on_Widgets

Typing in the information area will send a "**keystroke**" command and message which is the text of the character typed in.

Pressing the Enter key in the information area sends a "**keystroke**" command and then a "**coordinate accepted**" command and nothing in *message*.

Pressing and releasing LB in the information area sends a "**left_button_up**" command.

Pressing and releasing MB in the information area sends a "**middle_button_up**" command.

Pressing and releasing MB also starts a "Same As" and if a XYZ is selected then a "**coordinate accepted**" command is sent with nothing in *message*.

Pressing and releasing RB in the information area sends a "**right_button_up**" command and also brings up an options panel. The commands/messages send by items selected in the menu are documented in the section Widget Information Area Menu .

Picking a coordinate with the XYZ Select button sends a "**coordinate accepted**" command with nothing in *message*.

Create_xyz_box(Text title_text,Message_Box message)

Name

XYZ_Box Create_xyz_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **XYZ_Box**. *See* XYZ_Box .

The XYZ_Box is created with the title **title_text**.

The Message_Box message is used to display the XYZ information.

The function return value is the created XYZ_Box.

ID = 970

Validate(XYZ_Box box,Real &x,Real &y,Real &z)

Name

Integer Validate(XYZ_Box box,Real &x,Real &y,Real &z)

Description

Validate the contents of the XYZ_Box **box** and check it decodes to three Reals.

The three Reals are returned in **x, y,** and **z**.

The function returns the value of:

NO_NAME if the Widget XYZ_Box is optional and the box is left empty

TRUE (1) if no other return code is needed and *x, y* and *z* are valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 971

Get_data(XYZ_Box box,Text &text_data)

Name

Integer Get_data(XYZ_Box box,Text &text_data)

Description

Get the data of type Text from the XYZ_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 973

Set_data(XYZ_Box box,Real x,Real y,Real z)

Name

Integer Set_data(XYZ_Box box,Real x,Real y,Real z)

Description

Set the x y z data (all of type Real) for the XYZ_Box **box** to the values **x**, **y** and **z**.

A function return value of zero indicates the data was successfully set.

ID = 972

Set_data(XYZ_Box box,Text text_data)

Name

Integer Set_data(XYZ_Box box,Text text_data)

Description

Set the data of type Text for the XYZ_Box **box** to **text_data**.

A function return value of zero indicates the data was successfully set.

ID = 1520

For information on the other Input Widgets, go to Input Widgets

Message Boxes

See Colour_Message_Box
See Message_Box

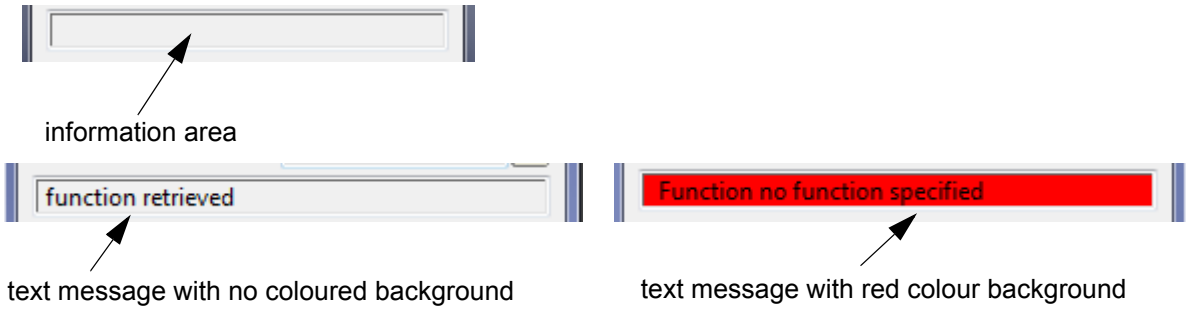
Colour_Message_Box

The ***Colour_Message_Box*** is a panel field designed to display text messages. The background colour for the text messages is under the programmers control and can vary between red, green, yellow or no colour.

This is useful for differentiating between different types of messages such as errors, warnings and successful.

The ***Colour_Message_Box*** consists of just an information area to display the text messages.

i



Colour_Message_Box

Data can not be typed into the Colour_Message_Box information area.

Note: The Colour_Message_Box is similar to a Message_Box (*see* Message_Box) except that a Message_Box has no coloured background.

When most other Input Widgets are created, a ***Colour_Message_Box*** or ***Message_Box*** needs to be supplied and that Colour_Message_Box or Message_Box is used by the Widget to display validation messages for the Widget.

Create_colour_message_box(Text message_text)

Name

Colour_Message_Box Create_colour_message_box(Text message_text)

Description

Create a box of type **Colour_Message_Box** for writing out messages. *See* Colour_Message_Box .

The Colour_Message_Box is created with the text **message_text** displayed in it.

The background colour of the display area is set using *Set_level (Colour_Message_Box, level),* or can be set with the message using *Set_data(Colour_Message_Box box,Text text_data,Integer level)).*

The function return value is the created Colour_Message_Box.

ID = 2629

Set_data(Colour_Message_Box box,Text text_data,Integer level)

Name

Integer Set_data(Colour_Message_Box box,Text text_data,Integer level)

Description

Set the data of type Text for the Colour_Message_Box **box** as the Text **text_data**.

If the Colour_Message_Box **box** is on a panel then the message text_data will be displayed in the information area of **box** with the background colour of the box set by **level**.

A function return value of zero indicates the data was successfully set.

ID = 2632

Set_data(Colour_Message_Box box,Text text_data)

Name

Integer Set_data(Colour_Message_Box box,Text text_data)

Description

Set the data of type Text for the Colour_Message_Box **box** as the Text **text_data**.

If the Colour_Message_Box **box** is on a panel then the message text_data will be displayed in the information area of **box** with the background colour previously defined by the *Set_level* call.

A function return value of zero indicates the data was successfully set.

ID = 2631

Set_level(Colour_Message_Box box,Integer level)

Name

Integer Set_level(Colour_Message_Box box,Integer level)

Description

Setting **level** defines the background colour to use when text messages are displayed in the information area of **box**. This level will be over ridden if the *Set_data(Colour_Message_Box box,Text text_data,Integer level)* call is used.

For **level** = 1, the colour is normal.
For **level** = 2, the colour is yellow (for Warning)
For **level** = 3, the colour is red (for Error)
For **level** = 4, the colour is green (for Good)

If no *Set_level* call is made then the default level is 1.

A function return value of zero indicates the level was successfully set.

ID = 2630

For information on the other Message Boxes go to Message Boxes *or for Input Widgets, go to* Input Widgets

Message_Box

The ***Message_Box*** is a panel field designed to display text messages.

The ***Message_Box*** consists of just an information area to display the text messages.

i

Message_Box

Data can **not** be typed into the Message_Box information area.

Note: The Message_Box is similar to a Colour_Message_Box (*see* Colour_Message_Box) except that a Message_Box can not have a coloured background.

When most other Input Widgets are created, a ***Colour_Message_Box*** or ***Message_Box*** needs to be supplied and that Colour_Message_Box or Message_Box is used by the Widget to display validation messages for the Widget.

Create_message_box(Text message_text)

Name

Message_Box Create_message_box(Text message_text)

Description

Create a box of type **Message_Box** for writing out messages. *See* Message_Box .

The Message_Box is created with the text **message_text** displayed in it.

The function return value is the created Message_Box.

ID = 847

Get_data(Message_Box box,Text &text_data)

Name

Integer Get_data(Message_Box box,Text &text_data)

Description

Get the data of type Text from the Message_Box **box** and return it in **text_data**.

A function return value of zero indicates the data was successfully returned.

ID = 1037

Set_data(Message_Box box,Text text_data)

Name

Integer Set_data(Message_Box box,Text text_data)

Description

Set the data of type Text for the Message_Box **box** as the Text **text_data**.

If the Message_Box **box** is on a panel then the message text_data will be displayed in the

information area of **box**.

A function return value of zero indicates the data was successfully set.

ID = 1038

For information on the other Message Boxes go to Message Boxes *or for Input Widgets, go to* Input Widgets

Log_Box and Log_Lines

A ***Log_Box*** is a panel field that behaves like the standard *12d Model* Output Window but may be added to a Panel or a Vertical or Horizontal group.

The Log_Box covers and area for messages by supplying the parameters **box_width** and **box_height**. The units of box_width and box_height are screen units (pixels).

The actual size of the Log_Box area is actual width and actual height pixels where:

the actual width of the area is the maximum of the width of the panel without the Draw_Box, and **box_width**.

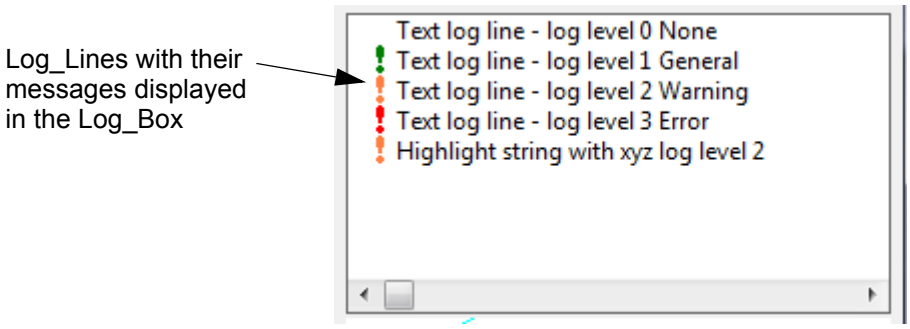
and

the height of the box is **box_height**.

Log_Box

Log_Lines are the method of passing information to the Log_Box, and unlike a message box which just takes text messages, Log_Lines can contain extra information for the user such as a link to a string that can be highlighted or edited by clicking on the Log_Line.

The ***Log_Box*** consists of just an information area to display the text messages.



Log_Box with Log_Line

Data can **not** be typed into the Log_Box information area.

After a log line is highlighted in the Log_Box, the

up arrow key moves the cursor **up one** log line
down arrow key moves the cursor **down one** log line
Home will go to the **top** log line in the Log_Box
End will go to the **bottom** log line in the Log_Box

Commands and Messages for Wait_on_Widgets

Pressing and releasing LB in the Log_Box with send a "**click_lb**" command and the line number of the log line in *message*.

Create_log_box(Text name,Integer box_width,Integer box_height)

Name

Log_Box Create_log_box(Text name,Integer box_width,Integer box_height)

Description

Create an input Widget of type **Log_Box** with the message area defined by the parameters **box_width, box_height** which are in screen units (pixels).*See* Log_Box and Log_Lines .

A Log_Box behaves like the standard *12d Model* Output Window but may be added to a Panel or Vertical / Horizontal group.

Log_Lines are the method of passing messages to the Log_Box.

The function return value is the created **Log_Box**.

ID = 2671

Create_text_log_line(Text message,Integer log_level)

Name

Log_Line Create_text_log_line(Text message,Integer log_level)

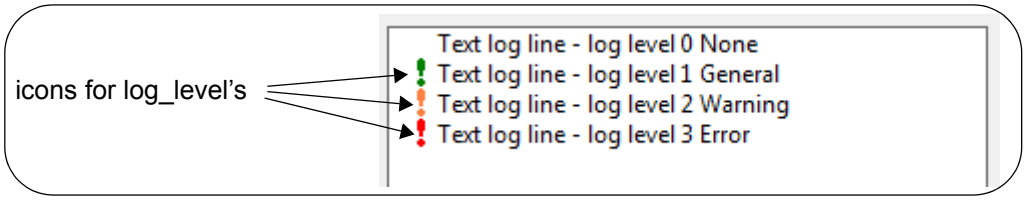
Description

Create a Text Log_Line with the message **message** and a log level **log_level**.

The text **message** is displayed in a Log_Box with the log level **log_level** when the Log_Line is added to the Log_Box.

Available log levels are
0 for none,
1 for General,
2 for Warning
3 for Error.

Log levels other than 0 will display a small icon to indicate their status.



WARNING
To be visible, the created Log_Line is added to a Log_Box using the call *Add_log_line(Log_Box box,Log_Line line)* **BUT** this call can only be made after the Log_Box is displayed in a panel using the *Show_panel* call.

The function return code is the created **Log_Line**.

ID = 2663

Create_highlight_string_log_line(Text message,Integer log_level,Uid model_id,Uid

string_id)

Name

Log_Line Create_highlight_string_log_line(Text message,Integer log_level,Uid model_id,Uid string_id)

Description

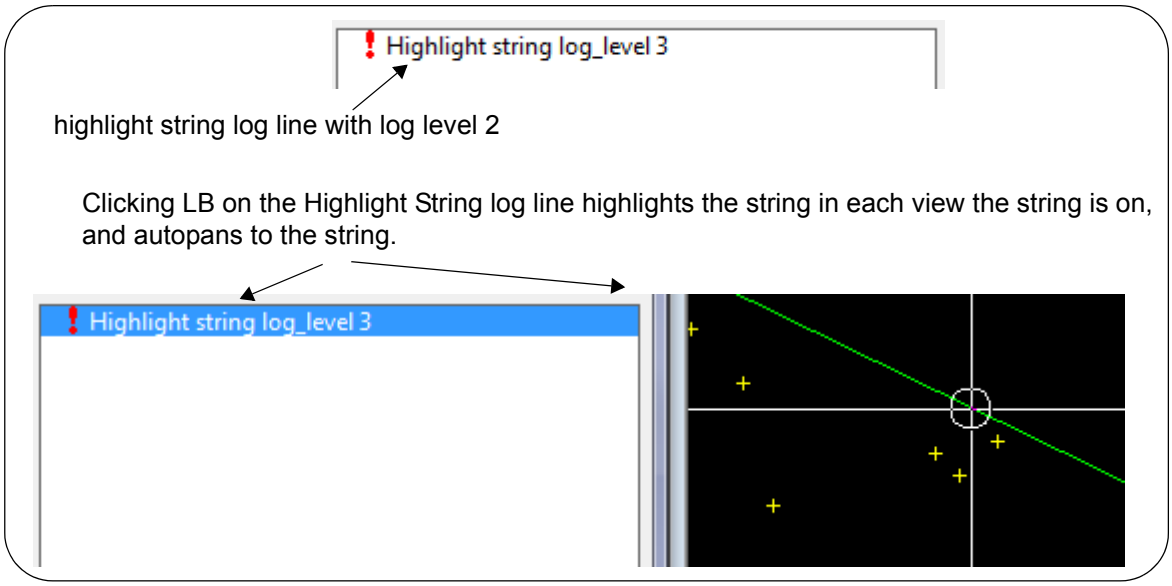
Create a Highlight String Log_Line giving a string by its model Uid **model_id** and string Uid **string_id**, a text **message** and a log level **log_level.**

The text **message** is displayed in a Log_Box with the log level **log_level** when the Log_Line is added to the Log_Box.

If LB is clicked on the log line, the string will be highlighted.

Available log levels are
0 for none,
1 for General,
2 for Warning
3 for Error.

Log levels other than 0 will display a small icon to indicate their status.



WARNING
To be visible, the created Log_Line is added to a Log_Box using the call *Add_log_line(Log_Box box,Log_Line line)* **BUT** this call can only be made after the Log_Box is displayed in a panel using the *Show_panel* call.

The function return code is the created **Log_Line**.

ID = 2664

Create_highlight_string_log_line(Text message,Integer log_level,Uid model_id,Uid string_id,Real x,Real y,Real z)

Name

Log_Line Create_highlight_string_log_line(Text message,Integer log_level,Uid model_id,Uid string_id,Real x,Real y,Real z)

Description

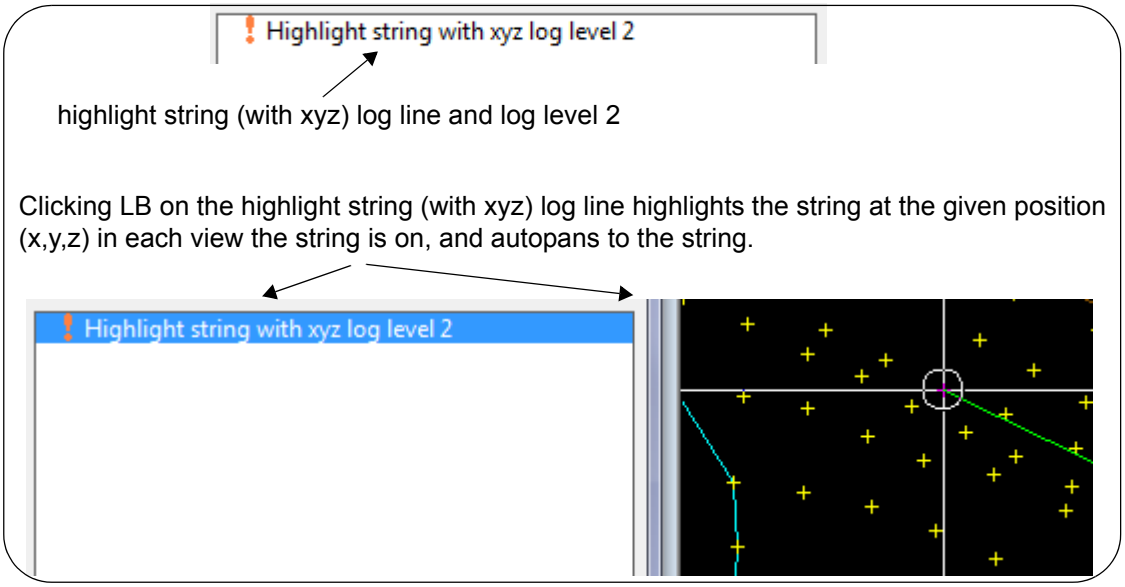
Create a Highlight String Log_Line giving a string by its model Uid **model_id** and string Uid **string_id**, a coordinate (**x**,**y**,**z**) on the string, a text **message** and a log level **log_level.**

The text **message** is displayed in a Log_Box with the log level **log_level** when the Log_Line is added to the Log_Box.

If LB is clicked on the log line, the coordinate (**x**,**y**,**z**) on the string, and the string, will be highlighted.

Available log levels are
0 for none,
1 for General,
2 for Warning
3 for Error.

Log levels other than 0 will display a small icon to indicate their status.



WARNING
To be visible, the created Log_Line is added to a Log_Box using the call *Add_log_line(Log_Box box,Log_Line line)* **BUT** this call can only be made after the Log_Box is displayed in a panel using the *Show_panel* call.

The function return code is the created **Log_Line**.

ID = 2665

Create_highlight_point_log_line(Text message,Integer log_level,Real x,Real y,Real z)

Name

Log_Line Create_highlight_point_log_line(Text message,Integer log_level,Real x,Real y,Real z)

Description

Create a Log_Line giving a coordinate (**x**,**y**,**z**).

If LB is clicked on the log line, the coordinate (**x**,**y**,**z**) will be highlighted.

LJG? on which views?

It also displays the text message **message** and has a log level **log_level**.

Available log levels are

0 for none,
1 for General,
2 for Warning

3 for Error.

Log levels other than 0 will display a small icon to indicate their status.

WARNING
To be visible, the created Log_Line is added to a Log_Box using the call *Add_log_line(Log_Box box,Log_Line line)* **BUT** this call can only be made after the Log_Box is displayed in a panel using the *Show_panel* call.

The function return code is the created **Log_Line**.

ID = 2666

Create_edit_string_log_line(Text message,Integer log_level,Uid model_id,Uid string_id)

Name

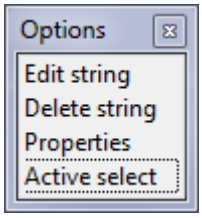
Log_Line Create_edit_string_log_line(Text message,Integer log_level,Uid model_id,Uid string_id)

Description

Create an Edit Log_Line giving a string by its model Uid **model_id** and string Uid **string_id**, a text **message** and a log level **log_level**.

The text **message** is displayed in a Log_Box with the log level **log_level** when the Log_Line is added to the Log_Box.

If LB is clicked on the log line, the string will be highlighted.
If LB is double clicked on the log line, the string is edited.
If RB is clicked on the log line then an *Options* menu is displayed with the choices:

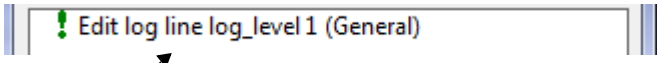


Edit the string
Delete the string
Show the string properties
Not applicable

It also displays the text message **message** and has a log level **log_level**.

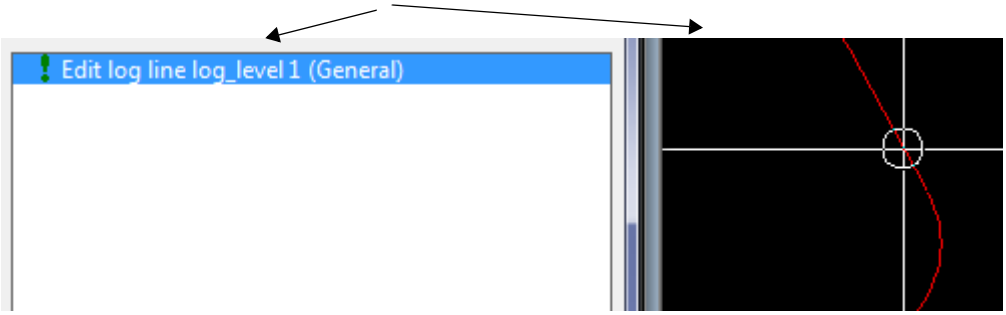
Available log levels are
0 for none,
1 for General,
2 for Warning
3 for Error.

Log levels other than 0 will display a small icon to indicate their status.

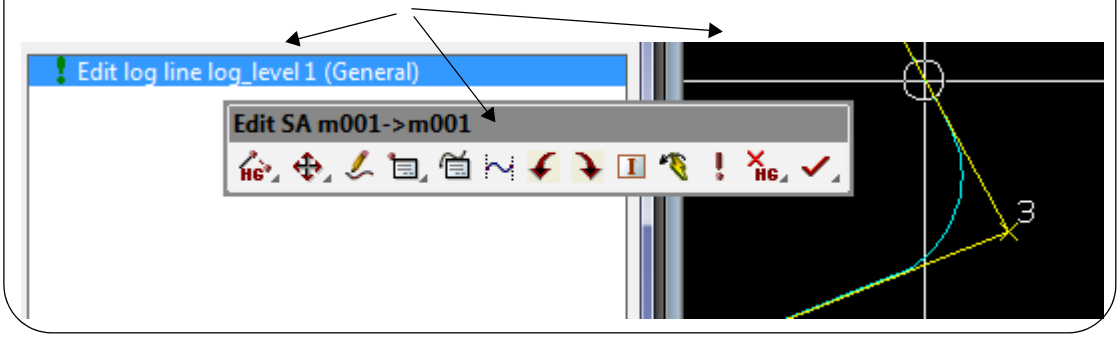


edit string log line with log level 1

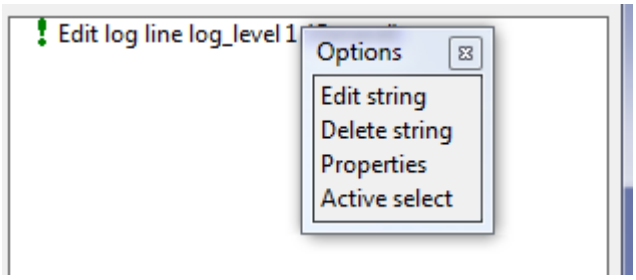
Clicking LB on the edit string log line highlights the string in each view the string is on, and autopans to the string.



Double clicking LB on the edit string log line highlights and edits the string. That is, it highlights the string and also brings the string up in its editor



Clicking RB on the edit string log line brings up the Options for the string



WARNING
To be visible, the created Log_Line is added to a Log_Box using the call *Add_log_line(Log_Box box,Log_Line line)* **BUT** this call can only be made after the Log_Box is displayed in a panel using the *Show_panel* call.

The function return code is the created **Log_Line**.

ID = 2667

Create_macro_log_line(Text message,Integer log_level,Text macro,Text select_cmd_line)

Name

Log_Line Create_macro_log_line(Text message,Integer log_level,Text macro,Text select_cmd_line)

Description

This call creates a log line that will allow the user to run a macro when the log line is double clicked. The macro is specified by the parameter **macro** and any optional arguments to be passed to it are specified by **cmd_line**.

It also displays the text message **message** and has a log level **log_level**.

Available log levels are
0 for none
1 for General,
2 for Warning
3 for Error.

Log levels other than 0 will display a small icon to indicate their status.

WARNING
To be visible, the created Log_Line is added to a Log_Box using the call *Add_log_line(Log_Box box,Log_Line line)* **BUT** this call can only be made after the Log_Box is displayed in a panel using the *Show_panel* call.

The function return code is the created **Log_Line**.

ID = 2668

Create_macro_log_line(Text message,Integer log_level,Text macro,Text select_cmd_line,Dynamic_Text menu_names,Dynamic_Text menu_command_lines)

Name

Log_Line Create_macro_log_line(Text message,Integer log_level,Text macro,Text select_cmd_line,Dynamic_Text menu_names,Dynamic_Text menu_command_lines)

Description

This call creates a log line that will allow the user to run a macro when the log line is double clicked. The macro is specified by the parameter **macro** and any optional arguments to be passed to it are specified by **cmd_line**.

This log line also provides options in a context menu when the user right clicks it. There are two parameters required; a list of all the names to be displayed in the menu, stored in a Dynamic_Text object called **menu_names** and the list of arguments to be passed down to the macro when the menu item is selected, stored in **menu_command_lines**.

It also displays the text message **message** and has a log level **log_level**.

Available log levels are

0 for none,
1 for General,
2 for Warning
3 for Error.

Log levels other than 0 will display a small icon to indicate their status.

WARNING
To be visible, the created Log_Line is added to a Log_Box using the call *Add_log_line(Log_Box box,Log_Line line)* **BUT** this call can only be made after the Log_Box is displayed in a panel using the *Show_panel* call.

The function return code is the created **Log_Line**.

ID = 2669

Add_log_line(Log_Box box,Log_Line line)

Name

Integer Add_log_line(Log_Box box,Log_Line line)

Description

Add the Log_Line **line** to the existing Log_Box **box**.

WARNING
To be visible, a Log_Line is added to a Log_Box using the call *Add_log_line(Log_Box box,Log_Line line)* **BUT** this call can only be made after the Log_Box is displayed in a panel using the *Show_panel* call.

A function return value of zero indicates the Log_Line was successfully added.

ID = 2672

Clear(Log_Box box)

Name

Integer Clear(Log_Box box)

Description

Clear any text and log lines from a Log_Box **box**.

A function return value of zero indicates the Log_Box was successfully cleared.

ID = 2673

Print_log_line(Log_Line line,Integer is_error)

Name

Integer Print_log_line(Log_Line line,Integer is_error)

Description

Print the Log_Line **line** to the *12d Model* Output window.

If **is_error** = 1, the Output window will treat the Log_line as an error message and the Output window will flash and/or pop up).

A function return value of zero indicates the Log_Line was successfully printed.

ID = 2670

Buttons

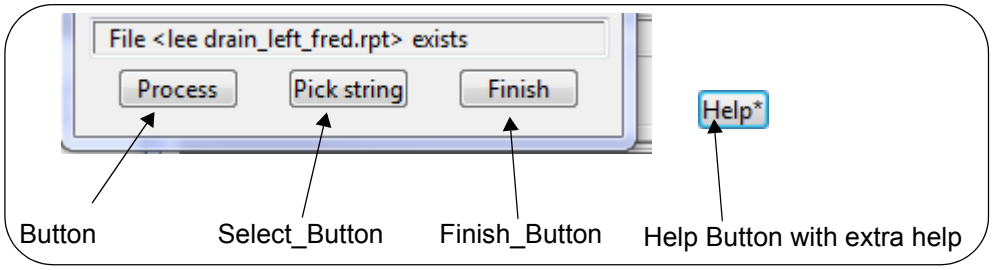
There are four types of Buttons - the Button, Finish_Button, Select_Button and a special Help button.

The **Button** and **Finish_Button** consist of just a Title, and a Text **reply**. When clicked the **reply** is send as a command via Wait_on_widgets.

The **Select_Button** is used to select strings. This has now been superseded by the Select_Box or the New_Select_Box.

The **Help Button** is created by a special call that allows the macro to hook into the Extra Help system for **12d Model.**

To the eye, the four types of buttons look identical but their behaviour is different.



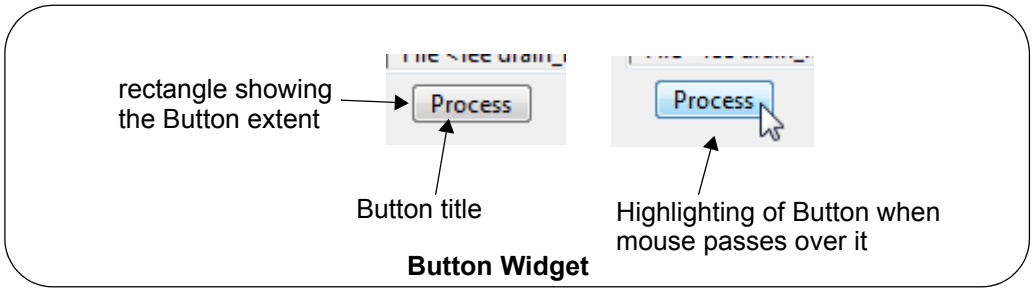
See Button
See Finish Button
See Select_Button
See Help Button

Button

A **Button** consists of a title, and a Text **reply**.

The **Button** is shown on the screen with title text surrounded by a rectangle to delineate the area on the screen associated with the Button.

Whenever the mouse is moved over the Button area, it will highlight and if LB or RB is clicked on the highlighted button, the Buttons sends the **reply** back to the macro as a command via *Wait_on_Widgets*.



Button Widget

Commands and Messages for Wait_on_Widgets

Pressing and releasing LB or RB whilst highlighting the Button sends the Text **reply** as a command with nothing in *message*.

Pressing and releasing MB does nothing.

Create_button(Text title_text,Text reply)

Name

Button Create_button(Text title_text,Text reply)

Description

Create a Widget of type **Button**.

The Button is created with **title_text** a the text on the Button.

The Text **reply** is the command that is sent by the Button back to the macro via *Wait_on_widgets* when the Button is clicked on. See Wait_on_widgets(Integer &id,Text &cmd,Text &msg) .

The function return value is the created **Button**.

ID = 850

Set_raised_button(Button button,Integer mode)

Name

Integer Set_raised_button(Button button,Integer mode)

Description

Set the **button** raised or sank depending on the **mode** value.

mode	value
-3	Raise
0	Flat
3	Sink

A function return value of zero indicates the button was successfully raised.

ID = 1058

Create_child_button(Text title_text)

Name

Button Create_child_button(Text title_text)

Description

Not implemented.

ID = 851

For information on the other Buttons, go to Buttons

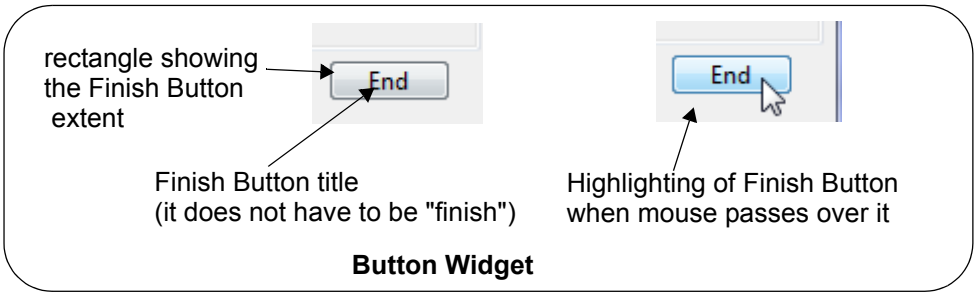
Finish Button

The **Finish Button** is a special **Button** and there should only be one per panel.

A **Finish Button** consists of a title, and a Text **reply**.

Like a standard **Button**, the **Finish Button** is shown on the screen with title text surrounded by a rectangle to delineate the area on the screen associated with the *Finish Button*.

Whenever the mouse is moved over the *Finish Button* area, it will highlight and if LB or RB is clicked on the highlighted button, the *Finish Button* sends the **reply** back to the macro as a command via *Wait_on_Widgets*.



Commands and Messages for Wait_on_Widgets

Pressing and releasing LB or RB whilst on the Button sends the Text **reply** as a command with nothing in *message*.
Pressing and releasing MB does nothing.

Create_finish_button(Text title_text,Text reply)

Name

Button Create_finish_button(Text title_text,Text reply)

Description

Creates a *Finish* Button with **title_text** the text on the Button.

The Text **reply** is the command that is sent by the Button back to the macro via *Wait_on_widgets* when the Button is clicked on. See Wait_on_widgets(Integer &id,Text &cmd,Text &msg) .

This is a special button and there should only be one per panel. The title_text is normally "Finish"

At the end of the processing in the macro, *Set_finish_button* (*see* Set_finish_button(Widget panel,Integer move_cursor)) should be called to put the cursor on the *Finish* button.

Set_finish_button needs to be called so that chains know that the macro has terminated correctly.

The function return value is the created **Button**.

ID = 1367

Set_finish_button(Widget panel,Integer move_cursor)

Name

Integer Set_finish_button(Widget panel,Integer move_cursor)

Description

If *move_cursor* = 1 then the cursor is moved onto the finish button.

ID = 1368

For information on the other Buttons, go to Buttons

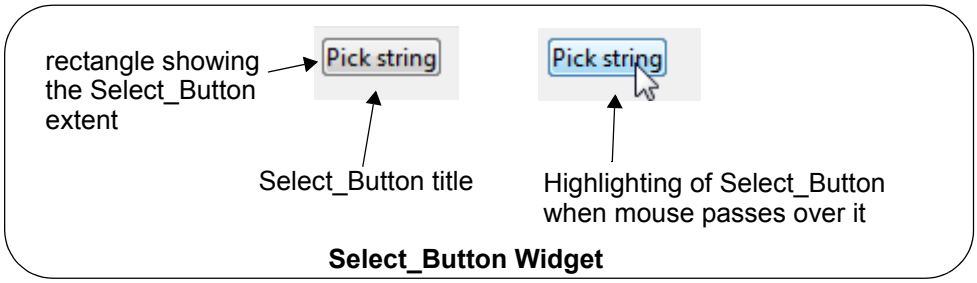
Select_Button

A **Select_Button** consists of a title, and a Text **reply**.

Like a standard **Button**, the **Select_Button** is shown on the screen with the title text surrounded by a rectangle to delineate the area on the screen associated with the Button.

Whenever the mouse is moved over the Button area, it will highlight.

However unlike a Button, clicking LB or RB on the Select_Button will start a String Select, and the selected string is recorded so that it can be used by the macro.



Commands and Messages for Wait_on_Widgets

Clicking LB or RB on the Select_Botton:

sends a "**start select**" command with nothing in *message*, then as the mouse is moved over a view, a "**motion select**" command is sent with the view coordinates and view name as text in *message*.

Once in the select:

if a string is clicked on with LB, a "**pick select**" command is sent with the name of the view that the string was selected in, in *message.* if the string is accepted (MB), an "**accept select**" command is sent with the view name (in quotes) in *message,* or if RB is clicked and *Cancel* selected from the *Pick Ops* menu, then a "**cancel select**" command is sent with nothing in *message.*

if a string is clicked on with MB (the pick and accept in one click method), a "**pick select**" command is sent with the name of the view that the string was selected in, in *message,* followed by an "**accept select**" command with the view name (in quotes) in *message.*

Nothing else typed over the Select_Button sends any commands or messages.

Create_select_button(Text title_text,Integer mode,Message_Box box)

Name

Select_Button Create_select_button(Text title_text,Integer mode,Message_Box box)

Description

Create a button of type **Select_Button**.

This is a special Button that when clicked, allows the user to select a string.

The button is created with the label text **title_text**.

The Message_Box **box** is selected to display the select information.

The value of **mode** is:

mode	**value**	
SELECT_STRING	5509	
SELECT_STRINGS	5510	not implemented!

Refer to the list in the Appendix A.

The function return value is the created **Select_Button**.

Note The Select_Button is now rarely used and has been replaced by the New_Select_Box or the Select_Box. See New_Select_Box and Select_Box

ID = 881

Validate(Select_Button select,Element &string)

Name

Integer Validate(Select_Button select,Element &string)

Description

Validate the Element **string** that is selected via the Select_Button **select**.

The function returns the value of:

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 978

Validate(Select_Button select,Element &string,Integer silent)

Name

Integer Validate(Select_Button select,Element &string,Integer silent)

Description

Validate the contents of Select_Button **select** and return the selected Element in **string.**

If **silent** = 0, and there is an error, a message is written and the cursor goes back to the button.
If **silent** = 1 and there is an error, no message or movement of cursor is done.

The function returns the value of:

TRUE (1) if no other return code is needed and *string* is valid.

FALSE (zero) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 1375

Set_data(Select_Button select,Element string)

Name

Integer Set_data(Select_Button select,Element string)

Description

Sets the Element for the Select_Button **select** to **string**.

A function return value of zero indicates the data was successfully set.

ID = 1173

Set_data(Select_Button select,Text string)

Name

Integer Set_data(Select_Button select,Text string)

Description

Set the model and string name as a Text **string** in the form "model_name->string_name"

A function return value of zero indicates the data was successfully set.

ID = 979

Get_data(Select_Button select,Text &string)

Name

Integer Get_data(Select_Button select,Text &string)

Description

Get the model and string name for the selected string in the form "model_name->string_name". Return the Text in **string**.

The returned string type must be **Text**.

A function return value of zero indicates the data was successfully returned.

ID = 980

Select_start(Select_Button select)

Name

Integer Select_start(Select_Button select)

Description

Starts the string selection for the Select_Button **select**. This is the same as if the button had been clicked.

A function return value of zero indicates the start was successful.

ID = 1167

Select_end(Select_Button select)

Name

Integer Select_end(Select_Button select)

Description

Cancels the string selection that is running for the Select_Button **select**. This is the same as if *Cancel* had been selected from the *Pick Ops* menu.

A function return value of zero indicates the end was successful.

ID = 1168

Set_select_type(Select_Button select,Text type)

Name

Integer Set_select_type(Select_Button select,Text type)

Description

Set the type of the string that can be selected to **type** for Select_Botton **select**. For example "Alignment", "3d".

A function return value of zero indicates the type was successfully set.

ID = 1043

Set_select_snap_mode(Select_Button select,Integer snap_control)

Name

Integer Set_select_snap_mode(Select_Button select,Integer snap_control)

Description

Set the snap control **snap_control** for the Select_Button **select**.

mode	value
Ignore_Snap	0
User_Snap	1
Program_Snap	2

A function return value of zero indicates the type was successfully set.

ID = 1044

Get_select_direction(Select_Button select,Integer &dir)

Name

Integer Get_select_direction(Select_Button select,Integer &dir)

Description

Get the select_direction **dir** from the selected string.

The returned **dir** type must be Integer.

If select without direction, the returned **dir** is 1, otherwise, the returned dir:

Value	Pick direction
1	the direction of the string
-1	against the direction of the string

A function return value of zero indicates the direction was successfully returned.

ID = 1046

Set_select_snap_mode(Select_Button select,Integer mode,Integer control,Text text)

Name

Integer Set_select_snap_mode(Select_Button select,Integer mode,Integer control,Text text)

Description

Set the snap mode **mode** and snap control **control**

for the Select_Button **select**.

When snap mode is:

Name_Snap	6
Tin_Snap	7
Model_Snap	8

the **snap_text** must be string name; tin name, model name accordingly, otherwise, leave the snap_text blank “”.

A function return value of zero indicates the type was successfully set.

Get_select_coordinate(Select_Button select,Real &x,Real &y,Real &z,Real &ch,Real &ht)

Name

Integer Get_select_coordinate(Select_Button select,Real &x,Real &y,Real &z,Real &ch,Real &ht)

Description

Get the coordinate of the selected snap point.

The return value of **x, y, z, ch** and **ht** must be type of **Real**.

A function return value of zero indicates the coordinate was successfully returned.

ID = 1047

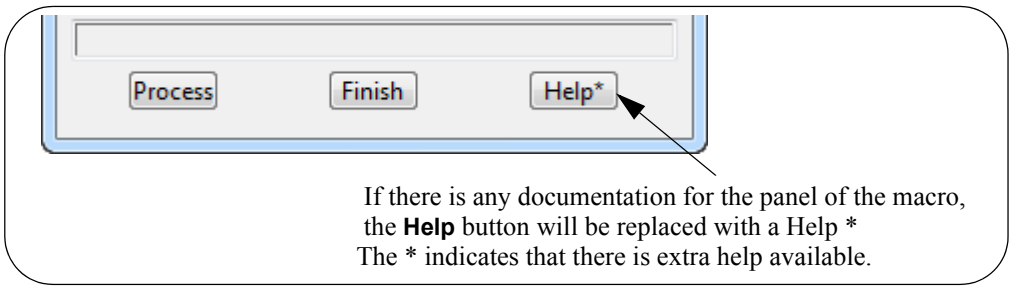
For information on the other Buttons, go to Buttons

Help Button

In **12d Model** every inbuilt panel (that is ones not created by macros) can have a **Help** button which when selected goes to the *topic* describing that panel. The default *12d Model* **Help** is all in one *Help* file but a method for displaying additional help information exists so 12d Solutions, 12d Distributors and Users can supply additional (extra) **Help** information.

In the macro language there is also a method of creating one **Help** button that is used for all the panels created in that macro and that **Help** button provides access to the context sensitive help provides by 12d Solutions (only of use to 12d Solutions programmers) AND also access to the Extra Help system that is available for all Users to supply their own, or additional (extra) **Help** information.

If there is *Extra Help* available for an option, then **Help*** will appear instead of **Help** on the panel button.



Create_help_button(Panel panel,Text title_txt)

Name

Button Create_help_button(Panel panel,Text title_txt)

Description

Create a button with the title **title_text** and return it as the function return value.

To set up the file for extra help, see How to Set Up Extra Help .

ID = 2633

How to Set Up Extra Help

Any extra help for a macro is placed in a folder with the same name as the macro but without the ending "4do" after the "." and with any blanks or non alphanumeric characters replaced by a underscore ("_").

For example, the extra help files for the macro called "testing help (3) system.4do" go in a folder called testing_help__3__system. Note there is an underscore for the blanks and the "(" and ")" in the macro name.

The extra help files for the macro that are placed in that folder can be a pdf, wmv, avi. txt etc.

The folder of Extra Help for the macro, is then placed in any one of the three places:

(a) in the *Help* folder in the 12d Model installation area: For example, for version 10,

c:\Program Files\12d\12d Model\10.00\Help

(b) in a folder called *Help* inside the *Set_ups* folder in the 12d Model installation area. For example

c:\Program Files\12d\12d Model\10.00\Set_ups\Help

or

(c) in a folder called *Help* inside the *User* folder in the 12d User area. For example

c:\12d\10.00\User\Help

For a macro, each of these areas is searched and if any extra help is found, it is listed with the full path to each extra help file.

If there is any extra help for a macro, the **Help** button on the panel will be replaced with a **Help** * button. The * indicates that there is extra help available.

When you click on the **Help** * button, you will get a list of all the extra help files for that panel with the full pathname to the extra help. Clicking on the file name will bring up that extra help.

Special Note:

Users can also have their own extra help files for macros (and also 12d Model panels) and the files are simply placed in the correctly named folder under **User\Help**. For information on Help information for 12d Model panels, see the ***12d Model Help*** section in the **12d Model** Reference manual.

For information on the other Buttons, go to Buttons

GridCtrl_Box

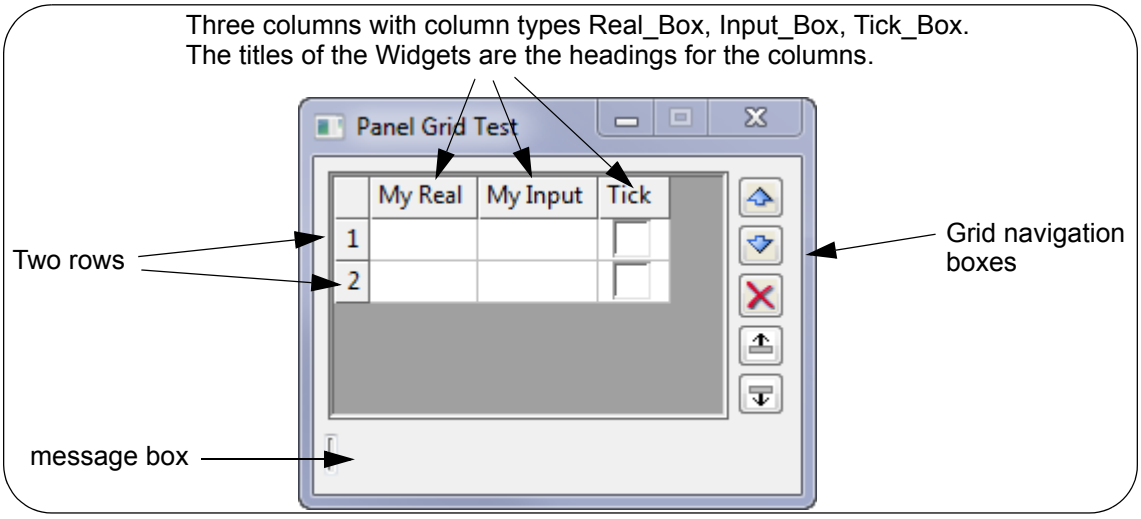
A GridCtrl_Box is made up of columns and rows of Widgets.

Each column must have a fixed Widget type, which is defined by supplying an array of Widgets of the correct type, one for each column, in column order. The title for each Widget becomes the title for the column of the GridCtrl_Box.

The only thing to be careful of is that if the variable types are not defined as actual Widget but are derived from Widgets (for example the input boxes Real_Box, Input_Box, Named_Tick_Box etc) then they must be cast to Widget before they can be loaded into the array to create the GridCtrl_Box.

As an example, a section of code required to create a GridCtrl_Box, defined the columns for the GridCtrl_Box using the array column_widgets[] and display it on the screen is:

```
Widget cast(Widget w)                    // this small routine cast needs to be in the macro code.
{
  return w;
}
void main()
{
  Panel panel   = Create_panel("Panel Grid Test");

  Widget  column_widgets[3];

  Message_Box message_box    = Create_message_box("");
  Real_Box   col_1_box       = Create_real_box("My Real", message_box);
  Input_Box   col_2_box      = Create_input_box("My Input", message_box);
  Named_Tick_Box  col_3_box  = Create_named_tick_box("Tick", 1, "resp");

  column_widgets[1] = cast(col_1_box);
  column_widgets[2] = cast(col_2_box);
  column_widgets[3] = cast(col_3_box);

  GridCtrl_Box grid_box = Create_gridctrl_box("MyGrid", 2, 3, column_widgets,1,
                                                  message_box, 100, 200);
  Append(grid_box, panel);

  Show_widget(panel);
```



Important note: Loading data into the GridCtrl_Box can only be done **after** the *Show_widget* call is made.

Create_gridctrl_box(Text name,Integer num_rows,Integer num_columns,Widget column_widgets[],Integer show_nav,Message_Box messages,Integer width,Integer height)

Name

GridCtrl_Box Create_gridctrl_box(Text name,Integer num_rows,Integer num_columns,Widget column_widgets[],Integer show_nav,Message_Box messages,Integer width,Integer height)

Description

This call creates a new **GridCtrl_Box** object which can be added to Panels.

name is the name of the GridCtrl_Box and the number of rows that the grid initially has is **num_rows** and the number of columns is **num_columns** (rows can also be added or deleted after the GridCtrl_Box has been displayed).

column_widgets[] is an array of Widgets in column order, and each Widget is of the type for that column. For an example see GridCtrl_Box .

If **show_nav** is 1 then there are navigation boxes on the side of the GridCtrl_Box.
If **show_nav** is 0 then there are no navigation boxes.

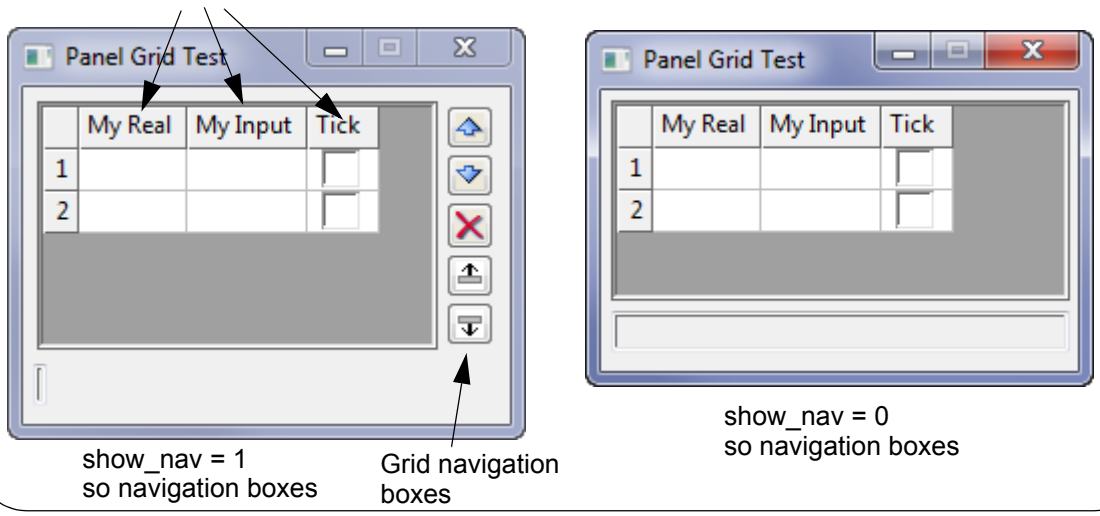
The width of the grid cell is **width** and the height of the grid cell is **height**, The units for width and height are screen units (pixels).

Important note: All Boxes, even through they have names like Real_Box and Input_Box, derived from Widgets and can be used in many options that take a Widget. For example Show_widget. However for the array of widgets **column_widgets[ ]** defining the GridCtrl_Box columns, the array values need to be Widget and so the other types derived from Widget have to be cast to a Widget before they can be used to fill the c**olumn_widgets[]** array. The cast is easily done by simply having the following *cast* function defined and in your macro code.

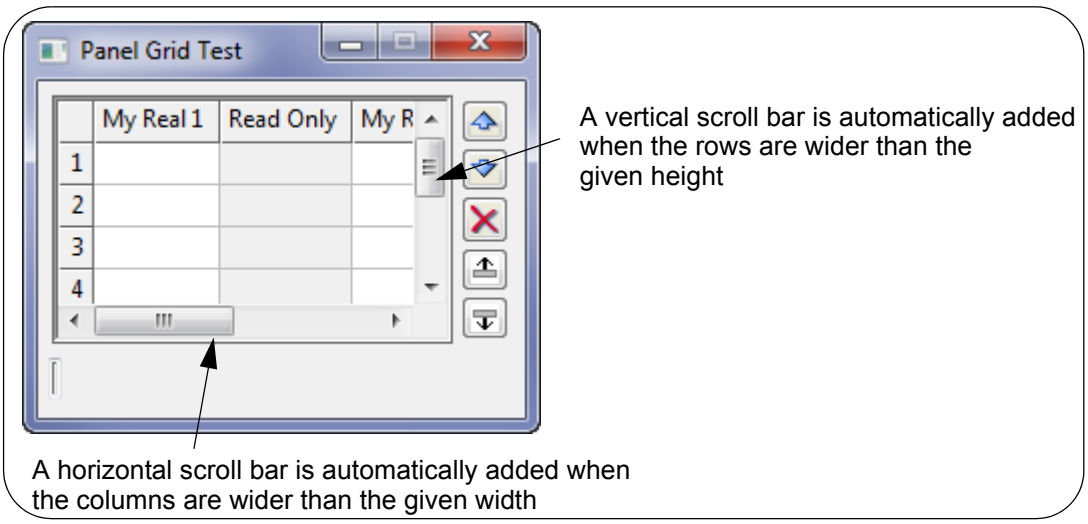
```
Widget cast(Widget w)
{
   return w;
}
```

See GridCtrl_Box for an example of using *cast* when defining values for **column_widgets[]**.

GridCtrl_Box with two row and three columns with column types Real_Box, Input_Box, Tick_Box
The titles of the Widgets are the headings for the columns



If the rows and columns are too large to fit inside the area defined by width and height, scroll bars are automatically created so that all cells can be reached.



The created GridCtrl_Box is returned as the function return value.

ID = 2393

Create_gridctrl_box(Text name,Integer num_rows, Integer num_columns,Widget column_widgets[],Integer column_readonly[], Integer show_nav,Message_Box messages,Integer width,Integer height) For V10 only

Name

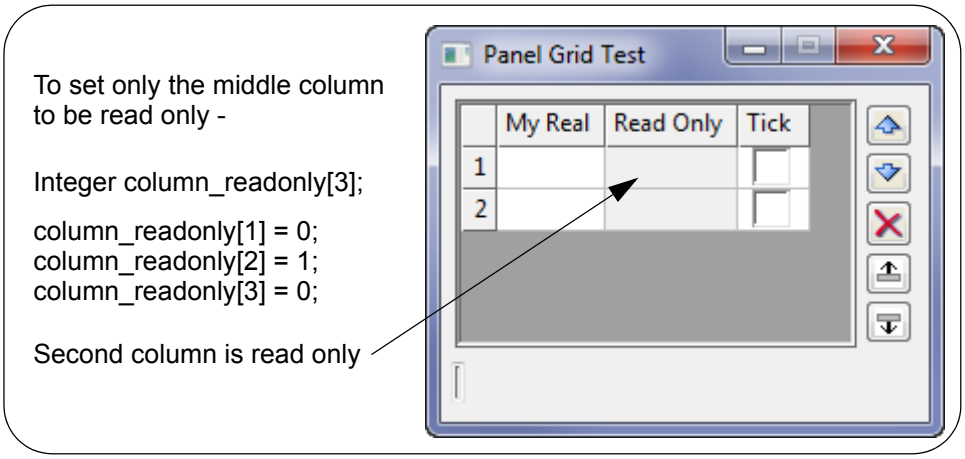
GridCtrl_Box Create_gridctrl_box(Text name,Integer num_rows,Integer num_columns,Widget column_widgets[],Integer column_readonly[],Integer show_nav,Message_Box messages,Integer width,Integer height)

Description

This call creates a new **GridCtrl_Box** object which can be added to Panels.

This is the same as the previous **GridCtrl_Box** function except that there is also the array **column_readonly[]** where

column_readonly[] is an Integer array of size **num_columns** where a value of 1 means that the cell is read only, and 0 means that the cell can be edited.



See Create_gridctrl_box(Text name,Integer num_rows,Integer num_columns,Widget column_widgets[],Integer show_nav,Message_Box messages,Integer width,Integer height) for more documentation for this function.

The created GridCtrl_Box is returned as the function return value.

ID = 2654

Load_widgets_from_row(GridCtrl_Box grid,Integer row_num)

Name

Integer Load_widgets_from_row(GridCtrl_Box grid,Integer row_num)

Description

Let **column_widgets[]** be the array that was used to define the GridCtrl_Box columns in the *Create_gridcltrl_box* call. See Create_gridctrl_box(Text name,Integer num_rows,Integer num_columns,Widget column_widgets[],Integer show_nav,Message_Box messages,Integer width,Integer height) .

Load_widgets_from_row loads the values in row **row_num** of the GridCtrl_Box **grid** into **column_widgets[]**.

Load_widgets_from_row allows you to validate grid values for a row, or to get the values to use for other purposes.

To change grid values, you first call *Load_widgets_from_row* to place the existing values for a row into **column_widgets[]**, change the values that you wish to change in **column_widgets[]**, and then call *Load_row_from_widgets* to load the new values from **column_widgets[]** back into the row. SeeLoad_row_from_widgets(GridCtrl_Box grid,Integer row_num) .

Note - this call can only be made after the *Show_widget* call is made to display the panel containing the GridCtrl_Box.

A function return value of zero indicates the load was successful.

ID = 2394

Load_row_from_widgets(GridCtrl_Box grid,Integer row_num)

Name

Integer Load_row_from_widgets(GridCtrl_Box grid,Integer row_num)

Description

Let **column_widgets[]** be the array that was used to define the GridCtrl_Box columns in the *Create_gridcltrl_box* call. See Create_gridctrl_box(Text name,Integer num_rows,Integer num_columns,Widget column_widgets[],Integer show_nav,Message_Box messages,Integer width,Integer height) .

Load_row_from_widgets loads the values of **column_widgets[]** into row **row_num** of the GridCtrl_Box **grid**.

Note - this call can only be made after the *Show_widget* call is made to display the panel containing the GridCtrl_Box.

A function return value of zero indicates the load was successful.

ID = 2395

Insert_row(GridCtrl_Box grid)

Name

Integer Insert_row(GridCtrl_Box grid)

Description

This call inserts a blank row at the bottom of the GridCtrl_Box **grid**.

Note - this call can only be made after the *Show_widget* call is made to display the panel containing the GridCtrl_Box.

A function return value of zero indicates the insertion was successful.

ID = 2396

Insert_row(GridCtrl_Box grid,Integer row_num,Integer is_before)

Name

Integer Insert_row(GridCtrl_Box grid,Integer row_num,Integer is_before)

Description

This call inserts a blank row into the GridCtrl_Box **grid**.

If **is_before** = 1, a blank row is inserted before **row_num**, so that the blank row becomes the new **row_num**'th row. The old rows from row **row_num** onwards are all pushed down one row.

If **is_before** = 0, a blank row is after row **row_num**, so that the blank row becomes a new **(num_row+1**)'th row. The old rows from row (**num_row+1**) onwards are pushed down one row.

t row number **row_num** of the GridCtrl_Box **grid**.

If you wish it to be inserted before the specified row, set **is_before** to 1, otherwise the row will be inserted after.

Note: a GridCtrl_Box(grid) call should be done after the *Insert_row(GridCtrl_Box grid,Integer row_num,Integer is_before)* call. See Format_grid(GridCtrl_Box grid) .

A function return value of zero indicates the insertion was successful.

ID = 2397

Delete_row(GridCtrl_Box grid,Integer row_num)

Name

Integer Delete_row(GridCtrl_Box grid,Integer row_num)

Description

Delete the row **row_num** from the GridCtrl_Box **grid**.

A function return value of zero indicates the row was successfully deleted.

ID = 2408

Delete_all_rows(GridCtrl_Box grid)

Name

Integer Delete_all_rows(GridCtrl_Box grid)

Description

Delete all the rows of the GridCtrl_Box **grid**.

A function return value of zero indicates the rows were successfully deleted.

ID = 2409

Get_row_count(GridCtrl_Box grid)

Name

Integer Get_row_count(GridCtrl_Box grid)

Description

This call returns the number of rows currently in a GridCtrl_Box **grid** as the function return value.

ID = 2398

Format_grid(GridCtrl_Box grid)

Name

Integer Format_grid(GridCtrl_Box grid)

Description

This call formats the GridCtrl_Box **grid**.

This means it makes sure all columns and rows are large enough to fit any entered data.

A function return value of zero indicates the format was successful.

ID = 2399

Set_cell(GridCtrl_Box grid,Integer row_num,Integer col_num,Text value)

Name

Integer Set_cell(GridCtrl_Box grid,Integer row_num,Integer col_num,Text value)

Description

For the cell with row number **row_num** and column number **col_num** of the GridCtrl_Box **grid**, set the *text* value of the cell to **text**.

It is recommended that you use the **Load_row_from_widgets** call, as this call will not provide any validation of data.

This call will return 0 if successful.

A function return value of zero indicates the set was successful.

ID = 2400

Get_cell(GridCtrl_Box grid,Integer row_num,Integer col_num,Text &value)

Name

Integer Get_cell(GridCtrl_Box grid,Integer row_num,Integer col_num,Text &value)

Description

Get the text value of the cell at row number **row_num** and column number **col_num** of the GridCtrl_Box **grid**, and returns the text in **value**.

It is recommended that you use the **Load_widgets_from_row** call instead, as this call will not provide any validation of data.

A function return value of zero indicates the get was successful.

ID = 2401

Set_column_width(GridCtrl_Box grid,Integer col,Integer width)

Name

Integer Set_column_width(GridCtrl_Box grid,Integer col,Integer width)

Description

For the GridCtrl_Box **grid**, set the width of column number **col** to **width**. The units of width are screen units (pixels).

The column can be made invisible by setting its width to 0.

A function return value of zero indicates the width was successfully set.

ID = 2402

Set_modified(GridCtrl_Box grid,Integer modified)

Name

Integer Set_modified(GridCtrl_Box grid,Integer modified)

Description

This call sets the *modified* state of the GridCtrl_Box **grid**.

If *modified* = 0 then the modified state is set to *off*.
If *modified* = 1 then the modified state is set to *on*.

A function return value of zero indicates the modified state was successfully set.

ID = 2403

Set_warn_on_modified(GridCtrl_Box grid,Integer warn_on_modified)

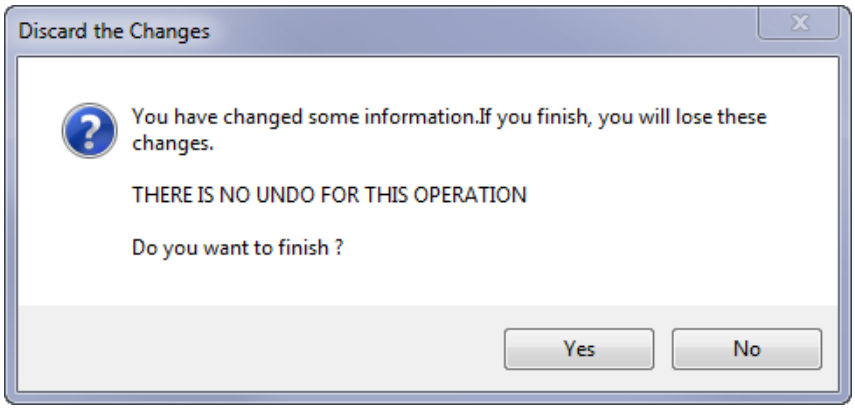
Name

Integer Set_warn_on_modified(GridCtrl_Box grid,Integer warn_on_modified)

Description

This call sets the *warn on modified* state of the GridCtrl_Box **grid.**

If *warn_on_modified* = 1 then if the panel containing **grid** is being closed and **grid** is in a modified state, then the user is prompted to confirm that **grid** is to be closed.



If *warn_on_modified* = 0 then there is no warning when the panel containing **grid** is being closed even if the panel has been modified.

Note: a GridCtrl_Box is in a in a modified state if data in the GridCtrl_Box has been changed and the modified state has not been set off by a ***Set_modified(grid,0)*** call. See Set_modified(GridCtrl_Box grid,Integer modified)

The *default* for a GridCtrl_Box is that a warning **is** given when attempting to close it.

A function return value of zero indicates the *warn on modified* state was successfully set.

ID = 2404

Get_selected_cells(GridCtrl_Box grid,Integer &start_row,Integer &start_col,Integer &end_row,Integer &end_col)

Name

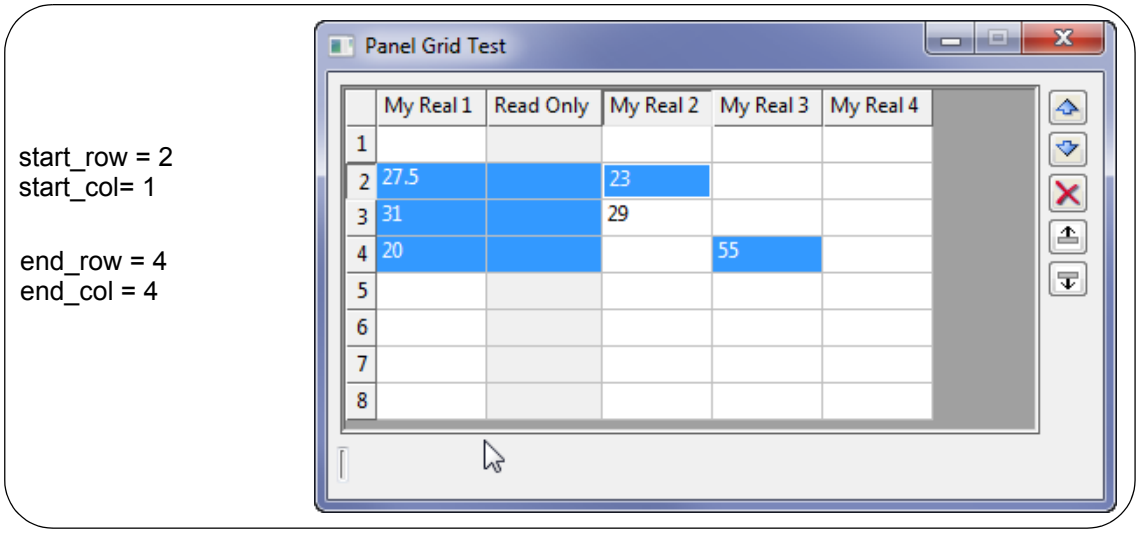
Integer Get_selected_cells(GridCtrl_Box grid,Integer &start_row,Integer &start_col,Integer &end_row,Integer &end_col)

Description

For the GridCtrl_Box **grid**, return the minimum and maximum row and column numbers for the current selected cells (the range of the selected cells).

The minimum and maximums are returned in **start_row**, **start_col** and **end_row** and **end_col**.

Note that not all the cells in the range need to be selected.



The function return value is zero if there are selected cells and the range is returned successfully.

The function return value is non-zero is there are no selected rows.

ID = 2410

Set_fixed_row_count(GridCtrl_Box grid,Integer num_fixed_rows)

Name

Integer Set_fixed_row_count(GridCtrl_Box grid,Integer num_fixed_rows)

Description

Sets the number of fixed rows in the GridCtrl_Box **grid**.

Fixed rows can not be deleted or moved and rows can not be inserted between two other fixed rows.

A function return value of zero indicates the set was successful.

ID = 2655

Get_fixed_row_count(GridCtrl_Box grid)

Name

Integer Get_fixed_row_count(GridCtrl_Box grid)

Description

Gets the number of fixed rows in the GridCtrl_Box **grid**.

Fixed rows can not be deleted or moved and rows can not be inserted between two other fixed rows.

The number of fixed rows is returned as the function return value.

ID = 2656

Set_cell_read_only(GridCtrl_Box grid,Integer row,Integer col,Integer read_only)

Name

Integer Set_cell_read_only(GridCtrl_Box grid,Integer row,Integer col,Integer read_only)

Description

For the GridCtrl_Box **grid,** set the cell specified by row **row** and column **col** as read only.

Note that colouring may be removed when **grid** is formatted and the *format_grid* message should be trapped to reapply these settings.

A function return value of zero indicates the set was successful.

ID = 2657

Get_cell_read_only(GridCtrl_Box grid,Integer row,Integer col)

Name

Integer Get_cell_read_only(GridCtrl_Box grid,Integer row,Integer col)

Description

For the GridCtrl_Box **grid**, check if the cell specified by row **row** and column **column** is read only.

The function return value is:

1 if the cell is read only

zero if the cell is not read only.

ID = 2658

Tree Box Calls

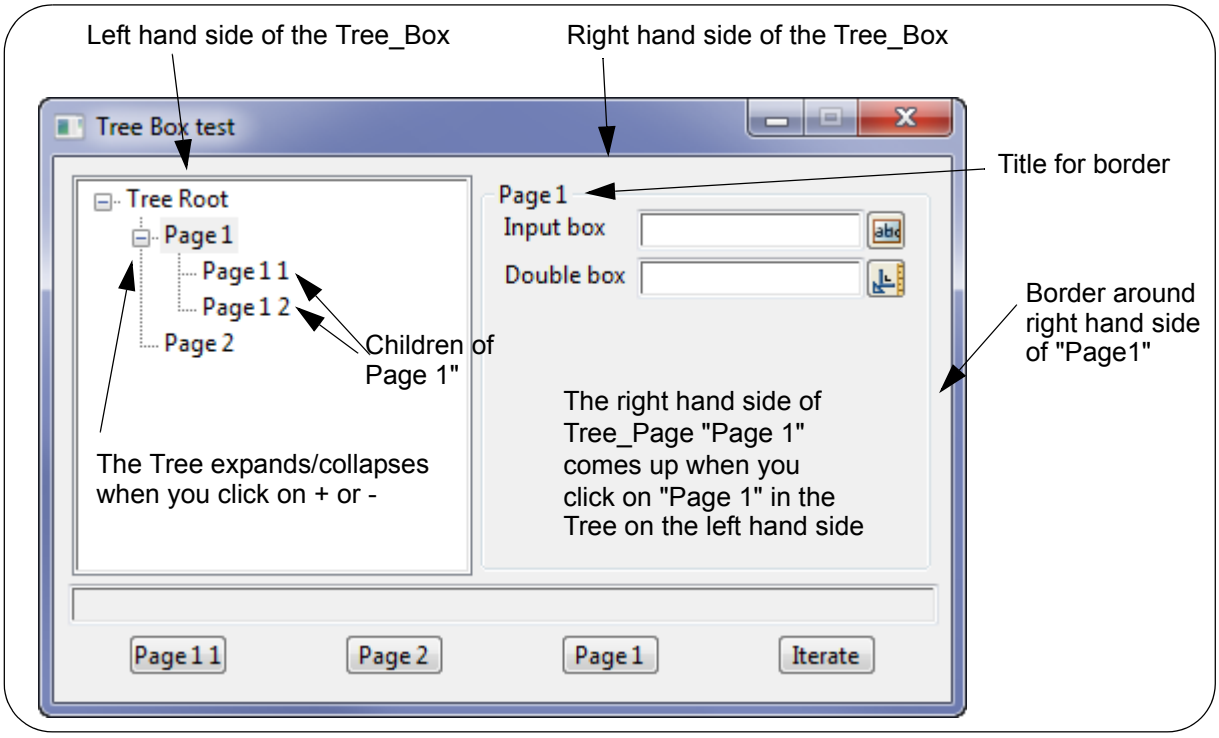
The tree box is a widget that consists of two parts - a left hand side (Tree) and a right hand side for displaying information for a particular part of the tree.

The tree on the left hand side is made up of **nodes** (or **pages**).

Each node (**page**) can have a set of Widgets that are displayed on the right hand side, when that node is selected on the left hand side.

Each node (**page**) can have zero or more of children pages.

The Tree_Box is similar in style to the *12d Model* panels for Super Alignment Parts Editor, the Chain editor and the Env.4d editor.



Create_tree_box(Text name,Text root_item_text,Integer tree_width,Integer tree_height)

Name

Tree_Box Create_tree_box(Text name,Text root_item_text,Integer tree_width,Integer tree_height)

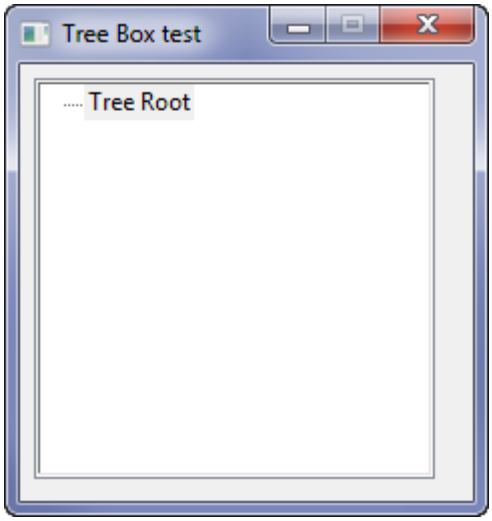
Description

This call creates a Tree_Box with the name **name** and with width **tree_width** and height **tree_height**. The units for width and height are screen units (pixels).

An empty node/page at the root of the tree is created with the title **root_item_text**. This is called the root page.

An example of a section of the code required to create a Tree_Box with its root page is:

```
Tree_Box tree_box = Create_tree_box("Tree", "Tree Root", 200, 200);
```



The created Tree_Box is returned as the function return value.

ID = 2571

Get_root_page(Tree_Box tree_box)

Name

Tree_Page Get_root_page(Tree_Box tree_box)

Description

Get the root page of the Tree_Box **tree_box** and return it as the function return value.

All Tree_Box's automatically have a root page.

ID = 2572

Create_tree_page(Tree_Page parent_page,Text name,Integer show_border, Integer use_name_for_border)

Name

Tree_Page Create_tree_page(Tree_Page parent_page,Text name,Integer show_border,Integer use_name_for_border)

Description

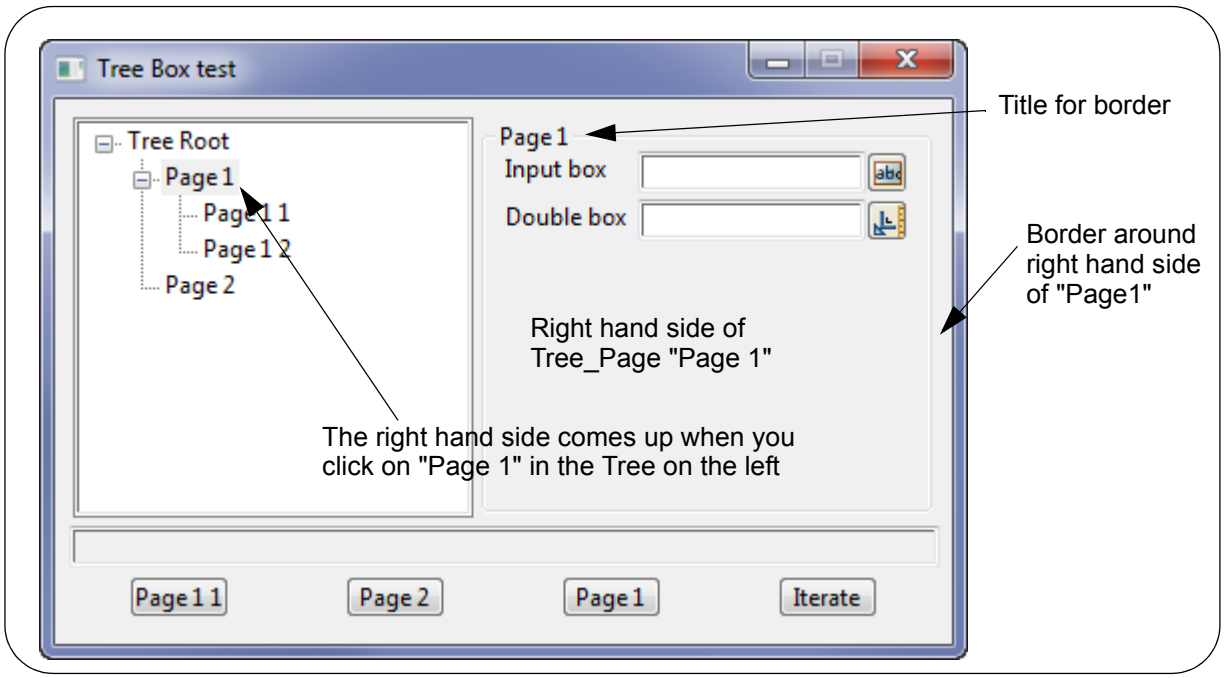
This call creates a new Tree_Page with the name **name**, as a child of the Tree_Page **parent_page**.

When the right hand side of the created page exists and there is none or more than one Group (either Horizontal_Group's and/or Vertical_Group's), then the right hand side can have an optional border and be given the name of the Tree_Page as a title for the border.

If *show_border* = *1*, a border is drawn around the right had side of the created Tree_Page.
If *show_border* = *0*, no border is drawn around the right had side of the created Tree_Page.

If *use_name_for_border* = *1*, **name** is used as the title when the border is drawn around the right had side of the created Tree_Page.

If *use_name_for_border* = *0*, there is no title when the border is drawn around the right had side of the created Tree_Page.

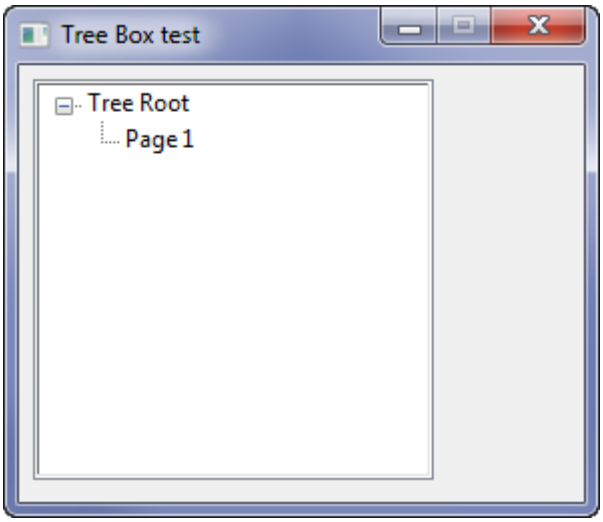


A parent page must exist before a child page can be created. The parent page may be the root page that is automatically created for a Tree_Box and the *Get_root_page* call is used to get the root page of a Tree_Box. *See* Get_root_page(Tree_Box tree_box)

A Tree_Page can contain any number of children pages.

An example of a section of the code required to create a Tree_Box with its root page, and then one child page of the root page is:

```
Tree_Box tree_box = Create_tree_box("Tree", "Tree Root", 200, 200);
// get the root page to add a child page called "Page 1" to
Tree_Page root_page = Get_root_page(tree_box);
Tree_Page page_1 = Create_tree_page(root_page, "Page 1", 1, 1);
```



The created Tree_Box is returned as the function return value.

ID = 2577

Append(Widget widget,Tree_Page page)

Name

Integer Append(Widget widget,Tree_Page page)

Description

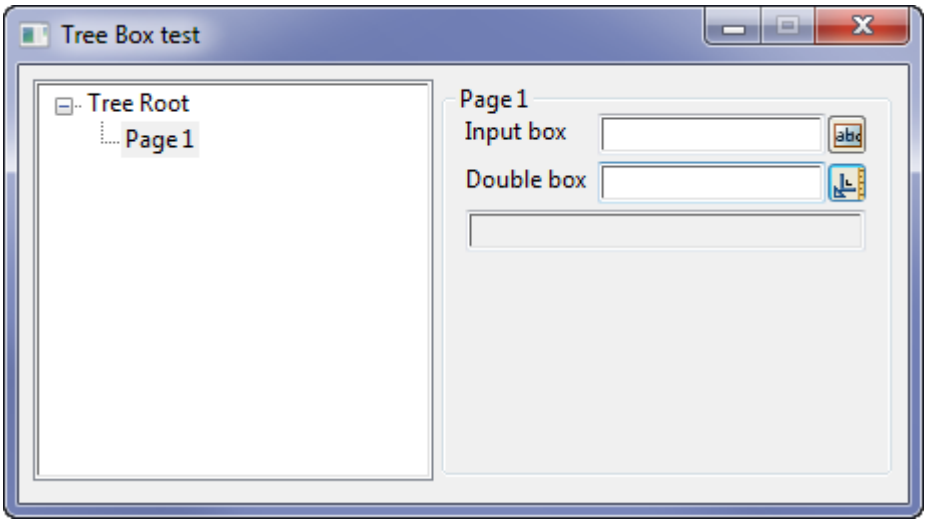
Append the Widget **widget** to the Tree_Page **page**.

All Widgets appended to a Tree_Page **page** are displayed on the right hand side of the Tree_Box when the user clicks on **page** on the left hand side of the Tree_Box.

A function return value of zero indicates the Widget was successfully appended.

An example of a section of the code required to create a Tree_Box with its root page, one child page of the root page, and some boxes to show on the right had side of the child page is:

```
 Panel panel   = Create_panel("Tree Box test");
Tree_Box tree_box = Create_tree_box("Tree", "Tree Root", 200, 200);
// get the root page to add a child page to
Tree_Page root_page = Get_root_page(tree_box);
Tree_Page page_1 = Create_tree_page(root_page, "Page 1", 1, 1);
Message_Box message_box = Create_message_box("");
Input_Box ib_1 = Create_input_box("Input box", message_box);
Real_Box db_1 = Create_real_box("Double box", message_box);
Append(ib_1,page_1);
Append(db_1,page_1);
Append(message_box,page_1);
Append(tree_box, panel);
Show_widget(panel);
```



ID = 2583

Get_number_of_pages(Tree_Page page)

Name

Integer Get_number_of_pages(Tree_Page page)

Description

For the Tree_Page **page**, return the number of child pages belonging to **page** as the function return value.

ID = 2578

Get_page(Tree_Page parent,Integer n,Tree_Page &child_page)

Name

Integer Get_page(Tree_Page parent,Integer page_index,Tree_Page &child_page)

Description

For the Tree_Page **parent**, find the **n**'th child page of **parent** and return the page as **child_page**.

A function return value of zero indicates a child page was successfully returned.

ID = 2579

Integer Has_child_page(Tree_Page parent,Tree_Page child)

Name

Has_child_page(Tree_Page parent,Tree_Page child)

Description

This call checks if the given child Tree_Page **child** belongs to the parent Tree_Page **parent**.

A non-zero function return value indicates that **child** is a child page of **parent**.

Warning this is the opposite of most 12dPL function return values

ID = 2580

Has_widget(Tree_Page page,Widget w)

Name

Integer Has_widget(Tree_Page page,Widget w)

Description

This call checks if the Tree_Page **page** contains the Widget **w**.

A non-zero function return value indicates that **w** is in **page**.

Warning this is the opposite of most 12dPL function return values

ID = 2581

Get_page_name(Tree_Page page)

Name

Text Get_page_name(Tree_Page page)

Description

For the Tree_Page **page**, return the Text name of page as the function return value.

ID = 2582

Set_page(Tree_Box tree_box,Widget w)

Name

Integer Set_page(Tree_Box tree_box,Widget w)

Description

Set the current displayed page of the Tree_Box **tree** to the Tree_Page that contains the Widget **w**.

This is particularly useful for validation, when validation fails.

A function return value of zero indicates the page was successfully displayed.

ID = 2573

Set_page(Tree_Box tree_box,Tree_Page page)

Name

Integer Set_page(Tree_Box tree_box,Tree_Page page)

Description

Set the current displayed page of the Tree_Box **tree** to the Tree_Page **page**.

A function return value of zero indicates the page was successfully displayed.

ID = 2574

Set_page(Tree_box tree_box,Text name)

Name

Integer Set_page(Tree_box tree_box,Text name)

Description

Set the current displayed page of the Tree_Box **tree** to the Tree_Page with name **name**.

A function return value of zero indicates the page was successfully displayed.

ID = 2575

Get_current_page(Tree_Box tree_box,Tree_Page &current_page)

Name

Integer Get_current_page(Tree_Box tree_box,Tree_Page &current_page)

Description

Get the Tree_Page that is currently selected and return it in **current_page**.

A function return value of zero indicates the page was successfully returned.

ID = 2576

General

See Quick Sort
See Name Matching
See Null Data
See Strings Edits
See Place Meshes

See Contour
See Drape

See Volumes

See Interface
See Templates
See Applying Templates

Quick Sort

The Quick Sort routines sort into increasing order, the n values held in either an Integer array, a Real array or a Text array, say val_array.

The data in the arrays is not actually moved but instead an Integer array *index[ ]* (called the Index array) is also passed into the Quick Sort routines and the Index array is returned holding the order of the sorted values.

That is, the i'th array value of Index is the array position of the i'th sorted value in val_array.

For example, if

```
ipos = Index[7],
```

and

```
val = val_array[ipos]
```

then *val* is the seventh sorted value from val_array.

So the loop below will go through the values in val_array in the sorted order from lowest value to the highest value:

```
for (Integer i=1;i<=n;i++) {
    val = val_array[index[i]];
```

Quick_sort(Integer count,Integer index[],Integer val_array[])

Name

Integer Quick_sort(Integer count,Integer index[],Integer val_array[])

Description

Sort the Integer array **val_array[count]** of size **count**, and return the sort order for **val_array[]** in the Index array **index[]**. For more information see Quick Sort .

The array **index[]** must be of at least size count.

A function return value of zero indicates that the sort was successful.

ID = 2745

Quick_sort(Integer count,Integer index[],Read val_array[])

Name

Integer Quick_sort(Integer count,Integer index[],Real val_array[])

Description

Sort the Real array **val_array[count]** of size **count**, and return the sort order for **val_array[]** in the Index array **index[]**. For more information see Quick Sort .

The array **index[]** must be of at least size count.

A function return value of zero indicates that the sort was successful.

ID = 2746

Quick_sort(Integer count,Integer index[],Text val_array[])

Name

Integer Quick_sort(Integer count,Integer index[],Text val_array[])

Description

Sort the Text array **val_array[count]** of size **count**, and return the sort order for **val_array[]** in the Index array **index[]**. For more information see Quick Sort .

The array **index[]** must be of at least size count.

A function return value of zero indicates that the sort was successful.

ID = 2747

Name Matching

Match_name(Text name,Text reg_exp)

Name

Integer Match_name(Text name,Text reg_exp)

Description

Checks to see if the Text **name** matches a regular expression given by Text **reg_exp**.

The regular expression uses

*	for a wild cards
?	for a wild character

A non-zero function return value indicates that there is a match.

A function return value of zero indicates there were no errors in the matching calculations.

Warning - this is the opposite of most 12dPL function return values

ID = 188

Match_name(Dynamic_Element de,Text reg_exp,Dynamic_Element &matched)

Name

Integer Match_name(Dynamic_Element de,Text reg_exp,Dynamic_Element &matched)

Description

Returns all the Elements from the Dynamic_Element **de** whose names match the regular expression Text **reg_exp**.

The matching elements are returned by appended them to the Dynamic_Element **matched**.

A function return value of zero indicates there were no errors in the matching calculations.

ID = 189

Null Data

It often happens in modelling that the plan position of a point is known (that is, the (x,y) co-ordinates are known) but the z-value is not defined.

For these situations, 12d Model has a special null z-value that is used to indicate that the z-value is to be ignored.

Is_null(Real value)

Name

Integer Is_null(Real value)

Description

Checks to see if the Real **value** is null or not.

A non-zero function return value indicates the value is null.

A zero function return value indicates the value is not null.

Warning - this is the opposite of most 12dPL function return values

ID = 469

Null(Real &value)

Name

void Null(Real &value)

Description

This function sets the Real **value** to the 12d Model null-value.

There is no function return value.

ID = 470

Null_ht(Dynamic_Element elements,Real height)

Name

Integer Null_ht(Dynamic_Element elements,Real height)

Description

This function examines the z-values of each point for all non-Alignment strings in the Dynamic_Element **elements**, and if the z-value of the point equals **height**, the z-value is reset to the null value.

A returned value of zero indicates there were no errors in the null operation.

ID = 407

Null_ht_range(Dynamic_Element elements,Real ht_min,Real ht_max)

Name

Integer Null_ht_range(Dynamic_Element elements,Real ht_min,Real ht_max)

Description

This function examines the z-values of each point for all non-Alignment strings in the Dynamic_Element **elements**, and if the z-value of the point is between ht_min and ht_max, the z-

value is reset to the null value.

A returned value of zero indicates there were no errors in the null operation.

ID = 408

Reset_null_ht(Dynamic_Element elements,Real height)

Name

Integer Reset_null_ht(Dynamic_Element elements,Real height)

Description

This function resets all the null z-values of all points of non-Alignment strings in the Dynamic_Element **elements**, to the value **height**.

A returned value of zero indicates there were no errors in the reset operation.

ID = 409

Contour

Contour(Tin tin,Real cmin,Real cmax,Real cinc,Real cont_ref,Integer cont_col,Dynamic_Element &cont_de,Real bold_inc,Integer bold_col,Dynamic_Element &bold_de)

Name

Integer Contour(Tin tin,Real cmin,Real cmax,Real cinc,Real cont_ref,Integer cont_col,Dynamic_Element &cont_de,Real bold_inc,Integer bold_col,Dynamic_Element &bold_de)

Description

Contour the triangulation **tin** between the minimum and maximum z values **cmin** and **cmax**.

The contour increment is **cinc**, and **cref** is a z value that the contours will pass through.

ccol is the colour of the normal contours and they are added to the Dynamic_Element cont_de.

bold_inc and **bold_col** are the increment and colour of the bold contours respectively. If **bold_inc** is zero then no bold contour are produced.

Any bold contours are added to the Dynamic_Element **bold_de**.

A function return value of zero indicates the contouring was successful.

ID = 143

Tin_tin_depth_contours(Tin original,Tin new,Integer cut_colour,Integer zero_colour,Integer fill_colour,Real interval,Real start_level,Real end_level,Integer mode,Dynamic_Element &de)

Name

Integer Tin_tin_depth_contours(Tin original,Tin new,Integer cut_colour,Integer zero_colour,Integer fill_colour,Real interval,Real start_level,Real end_level,Integer mode,Dynamic_Element &de)

Description

Calculate depth contours (isopachs) between the triangulations **original** and **new**.

The contour increment is **interval**, and the range is from **start_level** to **end_level**.

cut_colour, **zero_colour** and **fill_colour** are the colours of the cut, zero and fill contours respectively.

If the value of **mode** is

0	2d strings are produced with depth as the z-value
1	3d strings are produced with the depth contours projected onto the Tin **original**.
2	3d strings are produced with the depth contours projected onto the Tin **new**.

The new strings are added to the Dynamic_Element **de**.

A function return value of zero indicates the contouring was successful.

ID = 394

Tin_tin_intersect(Tin original,Tin new,Integer colour,Dynamic_Element &de)

Name

Integer Tin_tin_intersect(Tin original,Tin new,Integer colour,Dynamic_Element &de)

Description

Calculate the intersection (daylight lines) between the triangulations **original** and **new**.

The intersection lines have colour **colour** and are added to the Dynamic_Element **de**.

Note

This is the same as the zero depth contours projected onto either Tin **original** or **new** (mode 1 or 2) that are produced by the function Tin_tin_depth_contours.

A function return value of zero indicates the intersection was successful.

ID = 479

Tin_tin_intersect(Tin original,Tin new,Integer colour,Dynamic_Element &de,Integer mode)

Name

Integer Tin_tin_intersect(Tin original,Tin new,Integer colour,Dynamic_Element &de,Integer mode)

Description

Calculate the intersection (daylight lines) between the triangulations **original** and **new**.

The intersection lines have colour **colour** and are added to the Dynamic_Element **de**.

If **mode** is

0	the intersection line with z = 0 (2d string) is produced
1	the full 3d intersection is created.

A function return value of zero indicates the intersection was successful.

ID = 393

Drape

Drape(Tin tin,Model model,Dynamic_Element &draped_elts)

Name

Integer Drape(Tin tin,Model model,Dynamic_Element &draped_elts)

Description

Drape all the Elements in the Model **model** onto the Tin **tin**.

The draped Elements are returned in the Dynamic_Element **draped_elts**.

A function return value of zero indicates the drape was successful.

Drape(Tin tin,Dynamic_Element de, Dynamic_Element &draped_elts)

Name

Integer Drape(Tin tin,Dynamic_Element de, Dynamic_Element &draped_elts)

Description

Drape all the Elements in the Dynamic_Element **de** onto the Tin **tin**.

The draped Elements are returned in the Dynamic_Element **draped_elts**.

A function return value of zero indicates the drape was successful.

Face_drape(Tin tin,Model model, Dynamic_Element &face_draped_elts)

Name

Integer Face_drape(Tin tin,Model model, Dynamic_Element &face_draped_elts)

Description

Face drape all the Elements in the Model **model** onto the Tin **tin**.

The draped Elements are returned in the Dynamic_Element **face_draped_elts**.

A function return value of zero indicates the face drape was successful.

Face_drape(Tin tin,Dynamic_Element de,Dynamic_Element &face_draped_strings)

Name

Integer Face_drape(Tin tin,Dynamic_Element de,Dynamic_Element &face_draped_strings)

Description

Face drape all the Elements in the Dynamic_Element **de** onto the Tin **tin**.

The face draped Elements are returned in the Dynamic_Element **face_draped_elts**.

A function return value of zero indicates the face drape was successful.

ID = 145

Drainage

Get_drainage_intensity(Text rainfall_filename,Integer rainfall_method,Real frequency,Real duration,Real &intensity)

Name

Integer Get_drainage_intensity(Text rainfall_filename,Integer rainfall_method,Real frequency,Real duration,Real &intensity)

Description

The Rainfall Intensity information is part of a *12d Model* Rainfall File (that ends in ".12dhdyro").

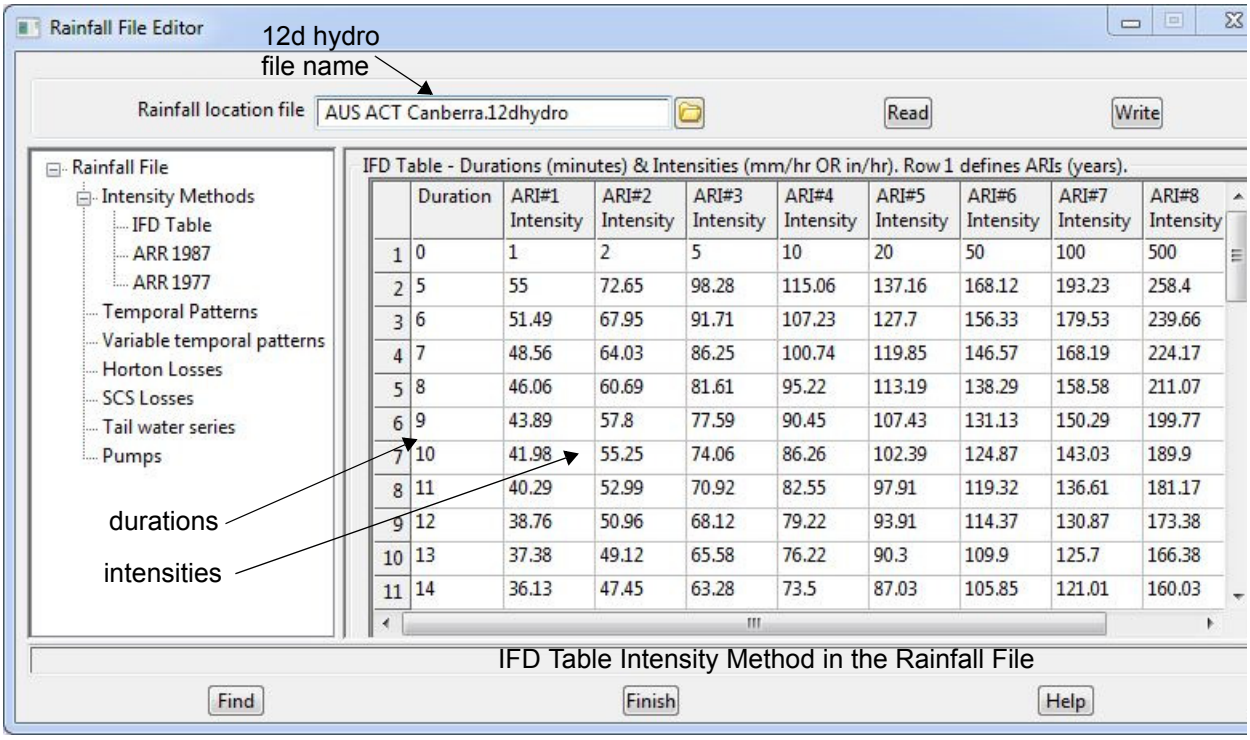
The Rainfall Files can be created and/or edited by the *12d Model* Rainfall File Editor:

Design =>Drainage-Sewer =>Rainfall editor.

12d Model comes with some Rainfall Files and others can be created by users.

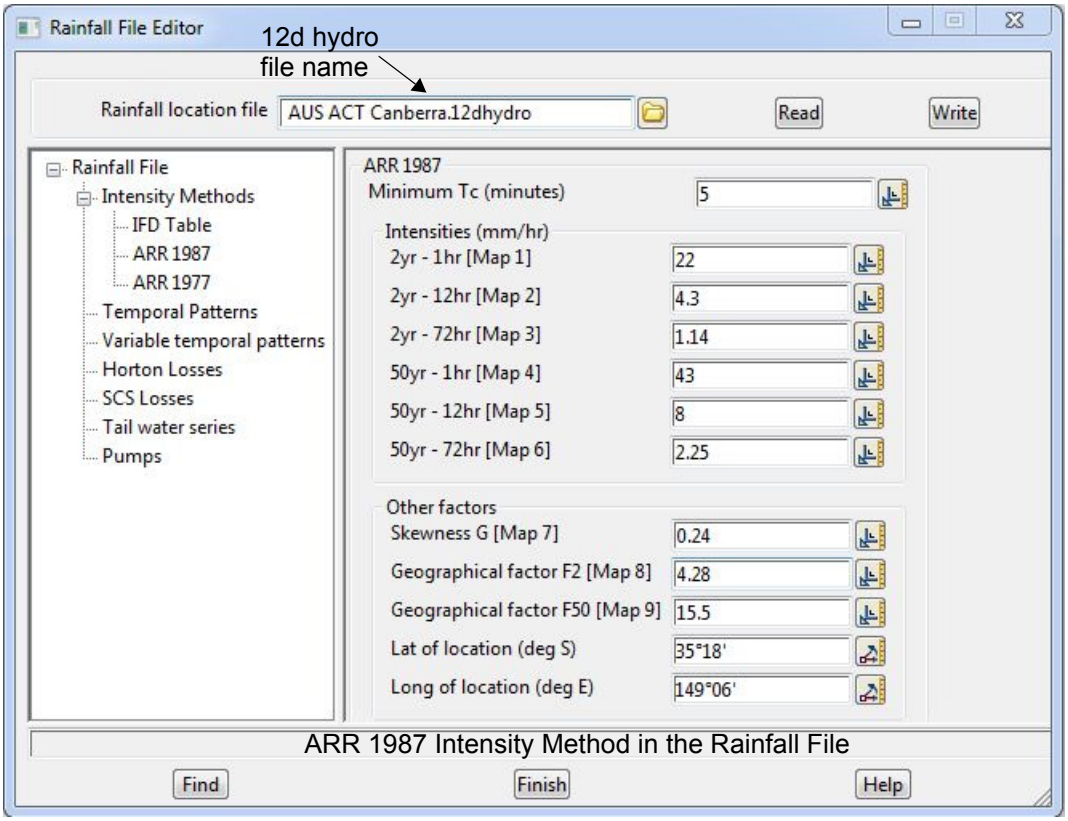
The *Get_drainage_intensity* call returns the intensity for a given rainfall method, frequency storm duration.

The image below are the rainfall Intensity Methods from the "AUS ACT Canberra.12dhydro" file loaded into the Rainfall File Editor.

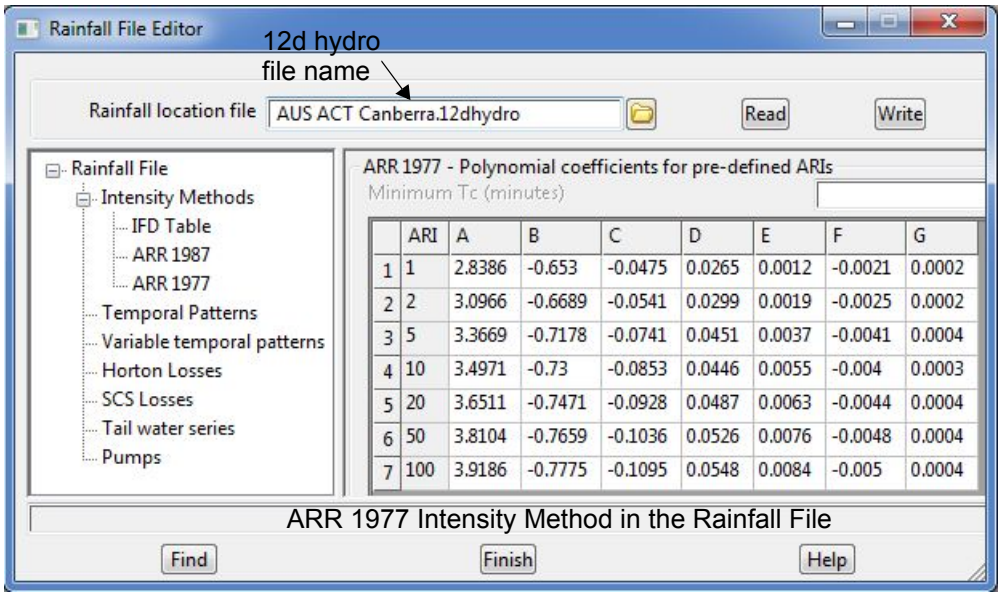


	Duration	ARI#1 Intensity	ARI#2 Intensity	ARI#3 Intensity	ARI#4 Intensity	ARI#5 Intensity	ARI#6 Intensity	ARI#7 Intensity	ARI#8 Intensity
1	0	1	2	5	10	20	50	100	500
2	5	55	72.65	98.28	115.06	137.16	168.12	193.23	258.4
3	6	51.49	67.95	91.71	107.23	127.7	156.33	179.53	239.66
4	7	48.56	64.03	86.25	100.74	119.85	146.57	168.19	224.17
5	8	46.06	60.69	81.61	95.22	113.19	138.29	158.58	211.07
6	9	43.89	57.8	77.59	90.45	107.43	131.13	150.29	199.77
7	10	41.98	55.25	74.06	86.26	102.39	124.87	143.03	189.9
8	11	40.29	52.99	70.92	82.55	97.91	119.32	136.61	181.17
9	12	38.76	50.96	68.12	79.22	93.91	114.37	130.87	173.38
10	13	37.38	49.12	65.58	76.22	90.3	109.9	125.7	166.38
11	14	36.13	47.45	63.28	73.5	87.03	105.85	121.01	160.03

IFD Table Intensity Method in the Rainfall File



ARR 1987 Intensity Method in the Rainfall File



	ARI	A	B	C	D	E	F	G
1	1	2.8386	-0.653	-0.0475	0.0265	0.0012	-0.0021	0.0002
2	2	3.0966	-0.6689	-0.0541	0.0299	0.0019	-0.0025	0.0002
3	5	3.3669	-0.7178	-0.0741	0.0451	0.0037	-0.0041	0.0004
4	10	3.4971	-0.73	-0.0853	0.0446	0.0055	-0.004	0.0003
5	20	3.6511	-0.7471	-0.0928	0.0487	0.0063	-0.0044	0.0004
6	50	3.8104	-0.7659	-0.1036	0.0526	0.0076	-0.0048	0.0004
7	100	3.9186	-0.7775	-0.1095	0.0548	0.0084	-0.005	0.0004

ARR 1977 Intensity Method in the Rainfall File

The function arguments are:

rainfall_filename is the local name of the ".12dhydro" file to get the Intensity from.

rainfall_method is one of:

"IFD Table"
"ARR 1987"
"ARR 1977"

frequency is the frequency (ARI) in years

duration is the duration in minutes

intensity is returned and is the intensity calculated from the table given by the rainfall_method, frequency and the duration.

A function return value of zero indicates that the intensity was successfully returned.

A non zero function return indicates that there was an error getting the intensity.

The value of the non-zero function value indicates the type of error:

Error Codes

-999 = no Drainage Analysis license
-99 = error reading file
-9 = no valid data found for specified method
-8 = frequency outside valid range
-4 = unsupported rainfall method
-3 = error building ARR1977 storm data
-2 = error building ARR1987 storm data
-1 = error building IFD storm data

ID = 2209

Get_rainfall_temporal_pattern(Text rainfall_filename,Integer storm_num,Integer &run,Text &zone_filter,Real &duration,Real &from_ari,Real &to_ari,Real &interval,Real pattern[],Integer max_num,Integer &ret_num)

Name

Integer Get_rainfall_temporal_pattern(Text rainfall_filename,Integer storm_num,Integer &run,Text &zone_filter,Real &duration,Real &from_ari,Real &to_ari,Real &interval,Real pattern[],Integer max_num,Integer &ret_num)

Description

The Rainfall Temporal Pattern information is part of a *12d Model* Rainfall File (that ends in ".12dhdyro").

The Rainfall Files can be created and/or edited by the *12d Model* Rainfall File Editor:

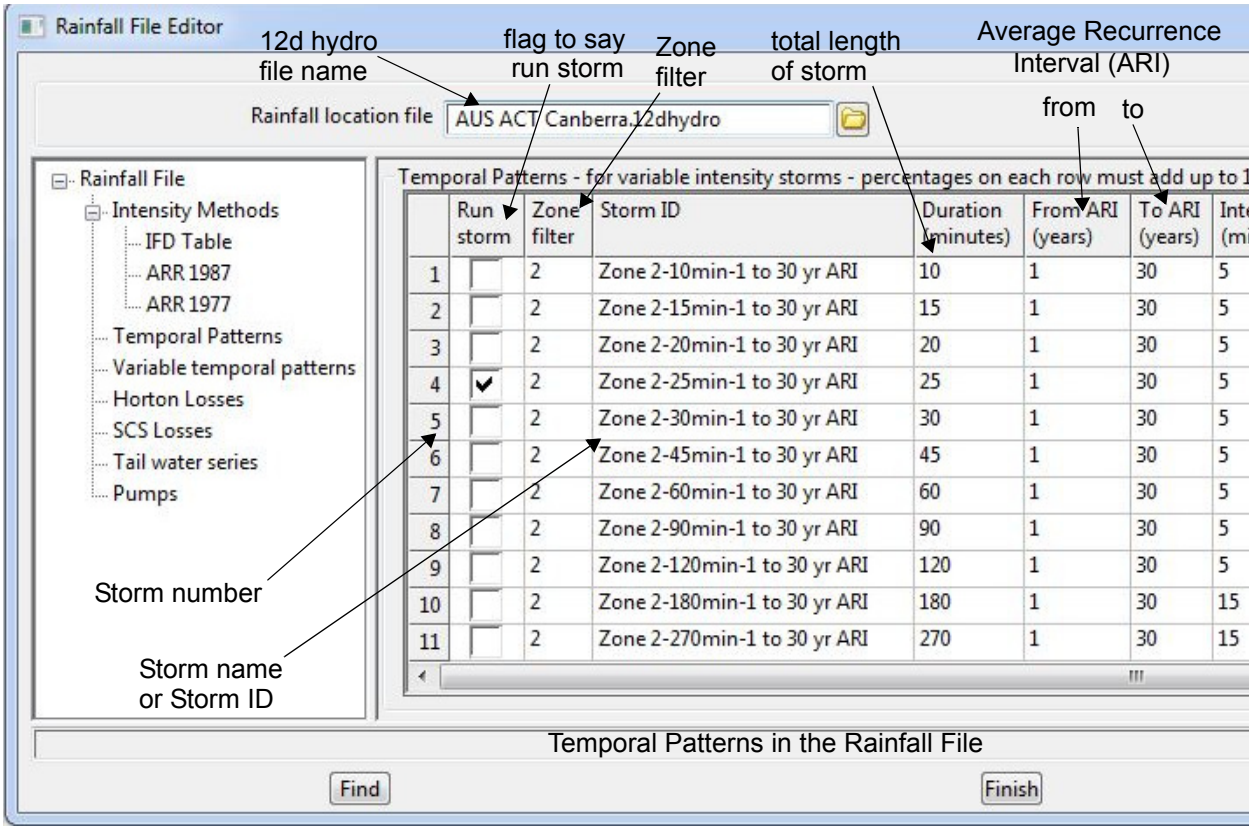
Design =>Drainage-Sewer =>Rainfall editor.

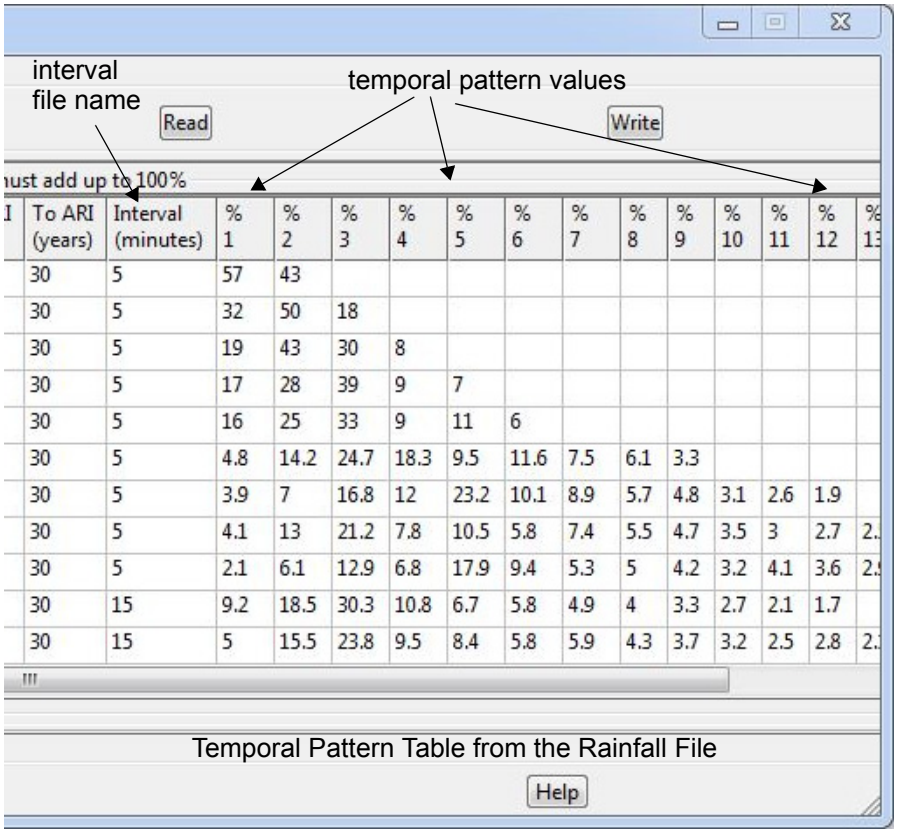
12d Model comes with some Rainfall Files and others can be created by users.

The rainfall Temporal Patterns give the mathematical description of one or more storms.

The *Get_rainfall_temporal_pattern* call returns the information for **one** storm from the rainfall Temporal Patterns in a Rainfall File.

The image below table is the is of the rainfall Temporal Patterns from the "AUS ACT Canberra.12dhydro" file loaded into the Rainfall File Editor.





The function arguments are:

rainfall_filename is the local name of the ".12dhydro" file to get the temporal pattern

values from.

storm_num is the number of the storm in the file

The rest of the arguments of the call return values from the storm_num'th line of the Temporal Pattern table.

run returns 1 if "Run Storm" is ticked
0 if "Run Storm" is not ticked

zone_filter returns the value from "Zone Filter"

duration returns the total length of the storm

from_ari returns the "from ARI" (Average Recurrence Interval, also known as the Frequency or Return Period)

to_ari returns the "to ARI" (Average Recurrence Interval, also known as the Frequency or Return Period)

interval returns the time interval for each of the values in the temporal patterns table (which give the percentage of the total storm that occurs in that period)

pattern[ ] is an array to return the values of the temporal pattern

max_num is the maximum size of the array pattern[]

ret_num returns the actual number of values returned in pattern

A function return value of zero indicates the data was successfully returned.

ID = 2405

Get_rainfall_temporal_pattern(Text rainfall_filename,Text storm_name,Integer &run,Text &zone_filter,Real &duration,Real &from_ari,Real &to_ari,Real &interval, Real pattern[],Integer max_num,Integer &ret_num)

Name

Integer Get_rainfall_temporal_pattern(Text rainfall_filename,Text storm_name,Integer &run,Text &zone_filter,Real &duration,Real &from_ari,Real &to_ari,Real &interval,Real pattern[],Integer max_num,Integer &ret_num)

Description

The Rainfall Temporal Pattern information is part of a *12d Model* Rainfall File (that ends in ".12dhdyro").

The Rainfall Files can be created and/or edited by the *12d Model* Rainfall File Editor:

Design =>Drainage-Sewer =>Rainfall editor.

12d Model comes with some Rainfall Files others can be created by users.

The rainfall Temporal Patterns give the mathematical description of one or more storms.

The *Get_rainfall_temporal_pattern* call returns the information for **one** storm from the rainfall Temporal Patterns in a Rainfall File.

The image of the rainfall Temporal Patterns from the "AUS ACT Canberra.12dhydro" file loaded into the Rainfall File Editor is given in Get_rainfall_temporal_pattern(Text rainfall_filename,Integer storm_num,Integer &run,Text &zone_filter,Real &duration,Real &from_ari,Real &to_ari,Real &interval,Real pattern[],Integer max_num,Integer &ret_num) .

The difference between the two calls is that in the other call, the required storm in the Temporal Patterns is given by a line number whereas in this function the storm is found by giving a storm ID (storm name).

storm_name is the name (Storm ID) of the required storm in the file. The Storm ID is will give the line in the Temporal Patterns to return the data from.

All the return values are the same as for the documentation in Get_rainfall_temporal_pattern(Text rainfall_filename,Integer storm_num,Integer &run,Text &zone_filter,Real &duration,Real &from_ari,Real &to_ari,Real &interval,Real pattern[],Integer max_num,Integer &ret_num) .

A function return value of zero indicates the data was successfully returned.

ID = 2406

Volumes

See End Area
See Exact Volumes

End Area

Volume(Tin tin_1,Real ht,Element poly,Real ang,Real sep,Text report_name,Integer report_mode,Real &cut,Real &fill,Real &balance)

Name

Integer Volume(Tin tin_1,Real ht,Element poly,Real ang,Real sep,Text report_name,Integer report_mode,Real &cut,Real &fill,Real &balance)

Description

Calculate the volume from a tin **tin_1** to a height **ht** inside the polygon **poly** using the end area method. The sections used for the end area calculations are taken at the angle **ang** with a separation of **sep**.

A report file is created called **report_name** which contains cut, fill and balance information.

If **report_mode** is equal to

0	only the total cut, fill and balance is given
1	the cut and fill value for every section is given.

If the file **report_name** is blank ("), no report is created.

The variables **cut**, **fill** and **balance** return the total cut, fill and balance.

A function return value of zero indicates the volume calculation was successful.

ID = 147

Volume(Tin tin_1,Tin tin_2,Element poly,Real ang,Real sep,Text report_name,Integer report_mode,Real &cut,Real &fill,Real &balance)

Name

Integer Volume(Tin tin_1,Tin tin_2,Element poly,Real ang,Real sep,Text report_name,Integer report_mode,Real &cut,Real &fill,Real &balance)

Description

Calculate the volume from tin **tin_1** to tin **tin_2** inside the polygon **poly** using the end area method. The sections used for the end area calculations are taken at the angle **ang** with a separation of **sep**.

A report file is created called **report_name** which contains cut, fill and balance information.

If **report_mode** is equal to

0	only the total cut, fill and balance is given
1	the cut and fill value for every section is given.

If the file **report_name** is blank ("), no report is created.

The variables **cut**, **fill** and **balance** return the total cut, fill and balance.

A function return value of zero indicates the volume calculation was successful.

ID = 148

Exact Volumes

Volume_exact(Tin tin_1,Real ht,Element poly,Real &cut,Real &fill,Real &balance)

Name

Integer Volume_exact(Tin tin_1,Real ht,Element poly,Real &cut,Real &fill,Real &balance)

Description

Calculate the volume from a tin **tin_1** to a height **ht** inside the polygon **poly** using the exact method.

The variables **cut**, **fill** and **balance** return the total cut, fill and balance.

A function return value of zero indicates the volume calculation was successful.

ID = 149

Volume_exact(Tin tin_1,Tin tin_2,Element poly,Real &cut,Real &fill,Real &balance)

Name

Integer Volume_exact(Tin tin_1,Tin tin_2,Element poly,Real &cut,Real &fill,Real &balance)

Description

Calculate the volume between tin **tin_1** and tin **tin_2** inside the polygon **poly** using the exact method.

The variables cut, fill and balance return the total **cut**, **fill** and **balance**.

A function return value of zero indicates the volume calculation was successful.

ID = 150

Interface

Interface(Tin tin,Element string,Real cut_slope,Real fill_slope,Real sep,Real search_dist,Integer side,Element &interface_string)

Name

Integer Interface(Tin tin,Element string,Real cut_slope,Real fill_slope,Real sep,Real search_dist,Integer side,Element &interface_string)

Description

Perform an interface to the tin **tin** along the Element **string**.

Use cut and fill slopes of value **cut_slope** and **fill_slope** and a distance between sections of **sep**. The units for slopes is 1:x.

Search to a maximum distance **search_dist** to find an intersection with the tin.

If **side** is negative, the interface is made to the left hand side of the string.

If **side** is positive, the interface is made to the right hand side of the string.

The resulting string is returned as the Element **interface_string**.

A function return value of zero indicates the interface was successful.

ID = 151

Interface(Tin tin,Element string,Real cut_slope,Real fill_slope,Real sep,Real search_dist,Integer side, Element &interface_string,Dynamic_Element &tadpoles)

Name

Integer Interface(Tin tin,Element string,Real cut_slope,Real fill_slope,Real sep,Real search_dist,Integer side,Element &interface_string,Dynamic_Element &tadpoles)

Description

Perform the interface as given in the previous function with the addition that slope lines are created and returned in the Dynamic_Element **tadpoles**.

A function return value of zero indicates the interface was successful.

ID = 152

Templates

Template_exists(Text template_name)

Name

Integer Template_exists(Text template_name)

Description

Checks to see if a template with the name **template_name** exists in the project.

A non-zero function return value indicates the template does exist.

A zero function return value indicates that no template of that name exists.

Warning - this is the opposite of most 12dPL function return values

ID = 201

Get_project_templates(Dynamic_Text &template_names)

Name

Integer Get_project_templates(Dynamic_Text &template_names)

Description

Get the names of all the templates in the project.

The dynamic array of template names is returned in the Dynamic_Text **template_names**.

A function return value of zero indicates success.

ID = 233

Template_rename(Text original_name,Text new_name)

Name

Integer Template_rename(Text original_name,Text new_name)

Description

Change the name of the Template **original_name** to the new name **new_name**.

A function return value of zero indicates the rename was successful.

ID = 424

Applying Templates

Apply(Real xpos,Real ypos,Real zpos,Real ang,Tin tin,Text template,Element &xsect)

Name

Integer Apply(Real xpos,Real ypos,Real zpos,Real ang,Tin tin,Text template,Element &xsect)

Description

Applies the templates template at the point (xpos,ypos,zpos) going out at the plan angle, ang.

The Tin **tin** is used as the surface for any interface calculations and the calculated section is returned as the Element **xsect**.

A function return value of zero indicates the apply was successful.

ID = 399

Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance)

Name

Integer Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance)

Description

Applies the templates **left_template** and **right_template** to the Element **string** going from start chainage **start_ch** to end chainage **end_ch** with distance **sep** between each section. The Tin **tin** is used as the surface for any interface calculations.

The variables **cut**, **fill** and **balance** return the total cut, fill and balance for the apply.

A function return value of zero indicates the apply was successful.

ID = 195

Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance,Text report)

Name

Integer Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance,Text report)

Description

Applies templates as for the previous function with the addition of a report being created with the name **report**.

A function return value of zero indicates the apply was successful.

ID = 196

Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance,Text report,Integer do_strings,Dynamic_Element &strings,Integer do_sections,Dynamic_Element &sections,Integer section_colour,Integer do_polygons,Dynamic_Element &polygons,Integer do_differences,Dynamic_Element &diffs,Integer difference_colour)

Name

Integer Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance,Text report,Integer do_strings,Dynamic_Element &strings,Integer do_sections,Dynamic_Element &sections,Integer section_colour,Integer do_polygons,Dynamic_Element &polygons,Integer do_differences,Dynamic_Element &diffs,Integer difference_colour)

Description

Applies templates as for the previous function with the additions:

If **do_strings** is non-zero, the strings are returned in **strings**.

If **do_sections** is non-zero, design sections of colour **section_colour** are returned in **sections**.

If **do_polygons** is non-zero, polygons are returned in **polygons**.

If **do_differences** is non-zero, difference sections of colour **difference_colour** are returned in **diffs**.

A function return value of zero indicates the apply was successful.

ID = 197

Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut,Real &fill,Real &balance)

Name

Integer Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut,Real &fill,Real &balance)

Description

Applies the templates as specified in the file **many_template_file** to the Element **string** with distance **sep** between each section. The Tin **tin** is used as the surface for any interface calculations.

The variables **cut**, **fill** and **balance** return the total cut, fill and balance for the apply.

A function return value of zero indicates success.

ID = 198

Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut_volume,Real &fill_volume,Real &balance_volume,Text report)

Name

Integer Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut_volume,Real &fill_volume,Real &balance_volume,Text report)

Description

Applies templates as for the previous function with the addition of a report being created with the name **report**.

A function return value of zero indicates success.

ID = 199

Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut,Real &fill,Real &balance,Text report,Integer do_strings,Dynamic_Element &strings,Integer do_sections,Dynamic_Element &sections,Integer section_colour,Integer do_polygons,Dynamic_Element &polygons,Integer

do_difference,Dynamic_Element &diffs,Integer difference_colour)

Name

Integer Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut,Real &fill,Real &balance,Text report,Integer do_strings,Dynamic_Element &strings,Integer do_sections,Dynamic_Element &sections,Integer section_colour,Integer do_polygons,Dynamic_Element &polygons,Integer do_difference,Dynamic_Element &diffs,Integer difference_colour)

Description

Applies templates as for the previous function with the additions:

If **do_strings** is non-zero, the strings are returned in strings.

If **do_sections** is non-zero, design sections of colour **section_colour** are returned in **sections**.

If **do_polygons** is non-zero, polygons are returned in **polygons**.

If **do_differences** is non-zero, difference sections of colour **difference_colour** are returned in **diffs**.

A function return value of zero indicates the apply was successful.

ID = 200

Strings Edits

String_reverse(Element in,Element &out)

Name

Integer String_reverse(Element in,Element &out)

Description

This functions creates a reversed copy of the string Element **in** and the reversed string is returned in **out**. That is, the chainage of string *out* starts at the end of the original string *in* and goes to the beginning of the original string *in*.

If successful, the new reversed string is returned in Element **out**.

A function return value of zero indicates the reverse was successful.

ID = 1134

Extend_string(Element elt,Real before,Real after,Element &newelt)

Name

Integer Extend_string(Element elt,Real before,Real after,Element &newelt)

Description

Extend the start and end of the string in Element **elt**.

The start of the string is extended by Real **before**.

The end of the string is extended by Real **after**.

If successful, the new element is returned in Element **newelt**.

A function return value of zero indicates the chainage was returned successfully.

ID = 664

Clip_string(Element string,Real chainage1,Real chainage2, Element &left_string,Element &mid_string,Element &right_string)

Name

Integer Clip_string(Element string,Real chainage1,Real chainage2, Element &left_string,Element &mid_string,Element &right_string)

Description

Clip a string about 2 chainages for the Element **string**. This will result in 3 new strings being created.

The part that exists before Real **chainage1** is returned in Element **left_string**.

The part that exists after Real **chainage2** is returned in Element **right_string**.

The part that exists between Real **chainage1** and Real **chainage2** is returned in Element **mid_string**.

A function return value of zero indicates the clip was successful.

Note

If the string is closed, right_string is not used.

If **chainage1** is on or before the start of the string, left_string is not used.

If **chainage2** is on or after the end of the string, right_string is not used.

If **chainage1** is greater than **chainage2**, they are first swapped.

ID = 542

Clip_string(Element string,Integer direction,Real chainage1,Real chainage2,Element &left_string,Element &mid_string,Element &right_string)

Name

Integer Clip_string(Element string,Integer direction,Real chainage1,Real chainage2,Element &left_string,Element &mid_string,Element &right_string)

Description

Clip a string about 2 chainages for the string Element **string**. This will result in 3 new strings being created. The clipped parts are returned relative to Integer **direction**. If direction is negative, **string** is first reversed before being clipped.

The part that exists before Real **chainage1** is returned in Element **left_string**.

The part that exists after Real **chainage2** is returned in Element **right_string**.

The part that exists between Real **chainage1** and Real **chainage2** is returned in Element **mid_string**.

A function return value of zero indicates the clip was successful.

Note

If the string is closed, **right_string** is not used.

If **chainage1** is on or before the start of the string, **left_string** is not used.

If **chainage2** is on or after the end of the string, **right_string** is not used.

If **chainage1** is greater than **chainage2**, they are first swapped.

ID = 549

Polygons_clip(Integer npts_clip,Real xclip[],Real yclip[],Integer npts_in,Real xarray_in[],Real yarray_in [],Real zarray_in [],Integer &npts_out,Real xarray_out[],Real yarray_out[],Real yarray_out[])

Name

Integer Polygons_clip(Integer npts_clip,Real xclip[],Real yclip[],Integer npts_in,Real xarray_in[],Real yarray_in [],Real zarray_in [],Integer &npts_out,Real xarray_out[],Real yarray_out[],Real yarray_out[])

Description

ID = 1440

Split_string(Element string,Real chainage,Element &string1,Element &string2)

Name

Integer Split_string(Element string,Real chainage,Element &string1,Element &string2)

Description

Split a string about a chainage for ELement string

This will result in 2 new strings being created.

The part that exists before Real **chainage** is returned in Element **string1**.

The part that exists after Real **chainage** is returned in Element **string2**.

A function return value of zero indicates the split was successful.

ID = 543

Join_strings(Element string1,Real x1,Real y1,Real z1,Element string2,Real x2,Real y2,Real z2,Element &joined_string)

Name

Integer Join_strings(Element string1,Real x1,Real y1,Real z1,Element string2,Real x2,Real y2,Real z2,Element &joined_string)

Description

Join the 2 strings Element **string1** and Element **string2** together to form 1 new string. The end of string1 closest to **x1,y1,z1** is joined to the end of string2 closest to **x2,y2,z2**.

The joined string is returned in Element **joined_string**.

A function return value of zero indicates the interface was successful.

Note

If the ends joined are no coincident, then a line between the ends is inserted.

The joined string is always of a type that preserves as much as possible about the original strings.

If you join 2 strings of the same type, the joined string is of the same type.

ID = 544

Rectangle_clip(Real x1,Real y1,Real x2,Real y2,Integer npts_in,Real xarray_in [],Real yarray_in [],Integer &npts_out,Real xarray_out[],Real yarray_out[])

Name

Integer Rectangle_clip(Real x1,Real y1,Real x2,Real y2,Integer npts_in,Real xarray_in [],Real yarray_in [],Integer &npts_out,Real xarray_out[],Real yarray_out[])

Description

<no description>

ID = 1438

Place Meshes

Place_mesh(Real x,Real y,Real z,Integer source_type,Text source_name,Vector3 offset,Vector3 rotate,Vector3 scale,Element &mesh_string)

Name

Integer Place_mesh(Real x,Real y,Real z,Integer source_type,Text source_name,Vector3 offset,Vector3 rotate,Vector3 scale,Element &mesh_string)

Description

This call places a mesh on the vertex of a new super string, at the co-ordinate specified by parameters **x, y, z**.

The **source_type** determines where the mesh will be loaded from:
source_type = 0 for the Mesh Library
, 1 for from a file

The **source_name** specifies the name of the mesh in the library or file, as defined by the **source_type** parameter.

You can also set any additional offset, rotation or scale parameters in the **offset**, **rotate** or **scale** vectors. If you are not intending to set additional parameters, you must set them to at least default values:

```
offset(0.0, 0.0, 0.0)
rotate(0.0, 0.0, 0.0)
scale(1.0, 1.0, 1.0);
```

The created super string will be stored in the element **mesh_string**.

This function returns 0 if it succeeds and non zero if it fails.

ID = 2803

Place_mesh(Real x,Real y,Real z,Text mesh_name,Vector3 offset,Vector3 rotate,Vector3 scale,Tin anchor_tin,Element &mesh_string)

Name

Integer Place_mesh(Real x,Real y,Real z,Text mesh_name,Vector3 offset,Vector3 rotate,Vector3 scale,Tin anchor_tin,Element &mesh_string)

Description

This call places a mesh from the mesh library on the vertex of a new super string, at the co-ordinate specified by parameters **x, y, z** and anchors it to the tin **anchor_tin**.

The Text **mesh_name** specifies the name of the mesh in the library.

You can also set any additional offset, rotation or scale parameters in the **offset**, **rotate** or **scale** vectors. If you are not intending to set additional parameters, you must set them to at least default values:

```
offset(0.0, 0.0, 0.0)
rotate(0.0, 0.0, 0.0)
scale(1.0, 1.0, 1.0);
```

The created super string will be stored in the Element **mesh_string**.

This function returns 0 if it succeeds and non zero if it fails.

ID = 2804

Utilities

See Affine Transformation
See Chains
See Convert
See Cuts Through Strings
See Factor
See Fence
See Filter
See Head to Tail
See Helmert Transformation
See Rotate
See Swap XY
See Translate

Affine Transformation

Affine(Dynamic_Element elements, Real rotate_x,Real rotate_y,Real scale_x,Real scale_y,Real dx,Real dy)

Name

Integer Affine(Dynamic_Element elements,Real rotate_x,Real rotate_y,Real scale_x,Real scale_y,Real dx,Real dy)

Description

Apply to all the elements in the Dynamic_Element **elements**, the Affine transformation with parameters:

X axis rotation	**rotate_x** (in radians)
Y axis rotation	**rotate_y** (in radians)
X scale factor	**scale_x**
Y scale factor	**scale_y**
Translation	(**dx,dy**)

A function return value of zero indicates the transformation was successful.

ID = 414

Chains

Run_chain(Text chain)

Name

Integer Run_chain(Text chain)

Description

Run the chain in the file named **chain**.

A function return value of zero indicates the chain was successfully run.

ID = 2096

Convert

Convert(Dynamic_Element in_de,Integer mode,Integer pass_others, Dynamic_Element &out_de)

Name

Integer Convert(Dynamic_Element in_de,Integer mode,Integer pass_others,Dynamic_Element &out_de)

Description

Convert the strings in Dynamic_Element in_de using Integer **mode** and when **mode** equals

1	convert 2d to 3d
2	convert 3d to 2d if the 3d string has constant z
3	convert 4d to 3d (the text is dropped at each point)

The converted strings are returned by appending them to the Dynamic_Element out_de.

If Integer **pass_others** is non zero, any strings in **in_de** that cannot be converted will be copied to **out_de**.

A function return value of zero indicates the conversion was successful.

ID = 139

Convert(Element elt,Text type,Element &newelt)

Name

Integer Convert(Element elt,Text type,Element &newelt)

Description

Tries to convert the Element **elt** to the Element type given by Text **type**.

If successful, the new element is returned in Element **newelt**.

A function return value of zero indicates the conversion was successful.

ID = 655

Cuts Through Strings

Cut_strings(Dynamic_Element seed,Dynamic_Element strings,Dynamic_Element &result)

Name

Integer Cut_strings(Dynamic_Element seed,Dynamic_Element strings,Dynamic_Element &result)

Description

Cut all the strings from the list Dynamic_Element **seed** with the strings from the list Dynamic_Element **strings** and add to Dynamic_Element **result**.

The strings created are 4d strings which have at each vertex the string cut.

Cuts are only considered valid if they have heights. Any cut at a point where the string height is null, will not be included.

A function return value of zero indicates the cut calculations was successful.

ID = 541

Cut_strings_with_nulls(Dynamic_Element seed,Dynamic_Element strings,Dynamic_Element &result)

Name

Integer Cut_strings_with_nulls(Dynamic_Element seed,Dynamic_Element strings,Dynamic_Element &result)

Description

Cut all the strings from the list Dynamic_Element **seed** with the strings from the list Dynamic_Element **strings** and add to Dynamic_Element **result**.

The strings created are 4d strings which have at each vertex the string cut.

A function return value of zero indicates the cut calculations was successful.

ID = 548

Factor

Factor(Dynamic_Element elements,Real xf,Real yf,Real zf)

Name

Integer Factor(Dynamic_Element elements,Real xf,Real yf,Real zf)

Description

Multiply all the co-ordinates of all the **elements** in the Dynamic_Element elements by the factors (**xf,yf,zf**).

A function return value of zero indicates the factor was successful.

ID = 411

Fence

Fence(Dynamic_Element data_to_fence,Integer mode,Element user_poly,Dynamic_Element &ret_inside,Dynamic_Element &ret_outside)

Name

Integer Fence(Dynamic_Element data_to_fence,Integer mode,Element user_poly,Dynamic_Element &ret_inside,Dynamic_Element &ret_outside)

Description

This function fences all the Elements in the Dynamic_Element **data_to_list** against the user supplied polygon Element **user_poly**.

The fence mode is given by Integer **mode** and when **mode** equals

0	get the inside of the polygon
1	get the outside of the polygon
2	get the inside and the outside of the polygon

If the inside is required, the data is returned by appending it to the Dynamic_Element **ret_inside**.

If the outside is required, the data is returned by appending it to the Dynamic_Element **ret_outside**

A returned value of zero indicates there were no errors in the fence operation.

Fence(Dynamic_Element data_to_fence,Integer mode,Dynamic_Element polygon_list,Dynamic_Element &ret_inside,Dynamic_Element &ret_outside)

Name

Integer Fence(Dynamic_Element data_to_fence,Integer mode,Dynamic_Element polygon_list,Dynamic_Element &ret_inside,Dynamic_Element &ret_outside)

Description

This function fences all the Elements in the Dynamic_Element **data_to_list** against one or more user supplied polygons given in the Dynamic_Element **polygon_list**.

The fence mode is given by Integer **mode** and when **mode** equals

0	get the inside of each of the polygons
1	get the outside of all the polygons
2	get the inside and the outside of the polygons

If the inside is required, the data is returned by appending it to the Dynamic_Element **ret_inside**.

If the outside is required, the data is returned by appending it to the Dynamic_Element **ret_outside**

A returned value of zero indicates there were no errors in the fence operation Head to Tail

ID = 137

Filter

Filter(Dynamic_Element in_de,Integer mode,Integer pass_others,Real tolerance,Dynamic_Element &out_de)

Name

Integer Filter(Dynamic_Element in_de,Integer mode,Integer pass_others,Real tolerance,Dynamic_Element &out_de)

Description

Filter removes points from 2d and/or 3d strings that do not deviate by more than the distance **tolerance** from the straight lines joining successive string points.

Hence the function Filter filters the data from **in_de** where **mode** means:

0	only 2d strings are filtered.
1	2d and 3d strings are filtered.

The filtered data is placed in the Dynamic_Element **out_de**.

If **pass_others** is non-zero, elements that can't be processed using the mode will be copied to **out_de**.

A function return value of zero indicates the filter was successful.

ID = 140

Head to Tail

Head_to_tail(Dynamic_Element in_list,Dynamic_Element &out_list)

Name

Integer Head_to_tail(Dynamic_Element in_list,Dynamic_Element &out_list)

Description

Perform head to tail processing on the data in Dynamic_Element **in_list**.

The resulting elements are returned by appending them to the Dynamic_Element **out_list**.

A function return value of zero indicates there were no errors in the head to tail process.

ID = 138

Helmert Transformation

Helmert(Dynamic_Element elements,Real rotate,Real scale,Real dx,Real dy)

Name

Integer Helmert(Dynamic_Element elements,Real rotate,Real scale,Real dx,Real dy)

Description

Apply to all the elements in the Dynamic_Element **elements**, the Helmert transformation with parameters:

Rotation	**rotate** (in radians)
Scale factor	**scale**
Translation	(**dx,dy**)

A function return value of zero indicates the transformation was successful.

ID = 413

Rotate

Rotate(Dynamic_Element elements,Real xorg,Real yorg,Real ang)

Name

Integer Rotate(Dynamic_Element elements,Real xorg,Real yorg,Real ang)

Description

Rotate all the elements in the Dynamic_Element **elements** about the centre point (**xorg,yorg**) through the angle ang.

A function return value of zero indicates the rotate was successful.

ID = 410

Swap XY

Swap_xy(Dynamic_Element elements)

Name

Integer Swap_xy(Dynamic_Element elements)

Description

Swap the x and y co-ordinates for all the elements in the Dynamic_Element **elements**.

A function return value of zero indicates the swap was successful.

ID = 412

Translate

Translate(Dynamic_Element elements,Real dx,Real dy,Real dz)

Name

Integer Translate(Dynamic_Element elements,Real dx,Real dy,Real dz)

Description

Translate translates all the elements in the Dynamic_Element **elements** by the amount (**dx,dy,dz**).

A function return value of zero indicates the translate was successful.

ID = 400

12d Model Macro_Functions

A *12d Model Function* is not a function call in the macro language, but a special type of object in *12d Model*. Typical 12d Model Functions are the Apply, Apply Many, Interface and Survey Data Reduction functions.

The macro language also allows the creation of Functions called *Macro_Functions*, or *Functions* for short that will appear in the standard 12d Model Function list and can be run from the standard 12d Model Recalc option.

The special things about *12d Model Functions* and *Macro_Functions* are that they:

(a) Have a **unique name** amongst all the *12d Model* Functions in a project.

(b) Have a **unique function type** so that pop-ups can be restricted to only Functions of that type.

(c) **Remember** t**he answers** for the fields in the panel that creates the Function (the Function input data) so that when Editing the Function, all the fields can be automatically filled in with the same answers as when the Function was last run.

(d) Can **record** which input Elements are such that if they are modified in *12d Model*, then the results of the Function will be incorrect and the Functions needs to be rerun (recalced) to update the results. These Elements are known as the Functions **dependency Elements**.

For a Macro_Function, the dependency Elements are set and retrieved using function dependency calls and the other answers for the panel fields are recorded as Function attributes.

(e) **Remember** the **data** that was **created** by the Function.

For a Macro_Function, these are normally elements and are recorded as function attributes as Uids and/or Uid ranges. This is the data that needs to be deleted when the Function is rerun.

(f) Can be Recalculated (or **Recalced** for short).

When a *12d Model Function* is recalced, the Function first deletes all the data that it created in the previously run, and then runs the Function again.

(g) Can on command, **replace** (delete or modify) all the **data** that the Function created on the pervious run with the data from this run.

The Macro_Function macro is just **one** macro and it is called with different *command line arguments* to let it know which mode it is in, and how it must behave.

The command line arguments that are used for a Macro_Function macro_function are:

(a) macro_function with no command line arguments

When there are no command line arguments, the function is being run for the first time and the macro panel is displayed.

(b) macro_function -function_recalc

The command line argument **-function_recalc** tells the macro that it is being recalced. So the macro needs to delete all the old data it created, and run the option again using the input information already stored in the Function. No panel is displayed when the macro_function is recalced.

12d Model calls the macro with the *-function_recalc* command line argument when the macro function is called from the 12d Model *Utilities =>Functions =>Recalc* option.

(c) macro_function -function_edit

The command line argument **-function_edit** tells the macro that it is being pulled up to be *edited*. That is, the macro_function needs to create the panel for the macro but the panel fields are filled with the input information that is stored with the function.

The panel fields can be modified and when the process button is pressed, the old data created by the function must be deleted and the option run again.

12d Model calls the macro with the *-function_edit* command line argument when the macro function is called from the 12d Model *Utilities =>Functions=> Recalc=>Editor* option.

(d) macro_function -function_delete

12d Model calls the macro with the *-function_delete* command line argument when the macro function is called from the 12d Model *Utilities =>Functions=> Recalc=>Delete* option.

So the macro must first check for a *command line argument.*

More detailed information to help understand how the Macro_Function calls are used in a macro is given in the following sections:

See Processing Command Line Arguments in a Macro_Function
See Creating and Populating the Macro_Function Panel
See Storing the Panel Information for Processing
See Recalcing
See Storing Calculated Information

All the *12d Model* Macro_Function calls are given in Macro_Function Functions .

Processing Command Line Arguments in a Macro_Function

The command line arguments -function_recalc, -function_edit, -function_delete and no arguments at all, need to be recognised and processed by the Macro_Function (for general information on command line arguments, see Command Line-Arguments).

The following is an example of some code from Example 15 (see Example 15) to trap and process the command line arguments for a Macro_Function:

```
void main()
// ------------------------------------------------------------------------------------------
// This is where the macro starts and checks for command line arguments
// ------------------------------------------------------------------------------------------
{
  Integer argc = Get_number_of_command_arguments();
  if(argc > 0) {
    Text arg;
    Get_command_argument(1,arg);          // check for the first command line argument
    if(arg == "-function_recalc") {       // check if it is -function_recalc
      Text function_name;
      Get_command_argument(2,function_name);    // the second command line argument
                                                // is the function name
      recalc_macro(function_name);
    } else if(arg == "-function_edit") {      // check if it is -function_edit
      Text function_name;
      Get_command_argument(2,function_name);    // the second command line argument
                                                // is the function name
      show_panel(function_name,1);          // tell show_panel the name of the function to
                                            // get the panel field answers from for recalc
```

```
                                        // See Creating and Populating the Macro_Function Panel
    } else if(arg == "-function_delete") {
// not implemented yet
      Text function_name;
      Get_command_argument(2,function_name);
      Error_prompt("function_delete not implemented");
    } else if(arg == "-function_popup") {
// not implemented yet
      Text function_name;
      Get_command_argument(2,function_name);
      Error_prompt("function_popup not implemented");
    } else {
// normal processing?
      Error_prompt("huh ? say what");                    // don't know what the command is
    }
  } else {                          // there are no command line arguments
                                    // show the panel with no information from a previous run
                                    // See Creating and Populating the Macro_Function Panel
    show_panel("",0);
  }
}
```

Continue to Creating and Populating the Macro_Function Panel

All the *12d Model* Macro_Function calls are given in Macro_Function Functions .

Creating and Populating the Macro_Function Panel

The main difference between a panel in a standard macro and a panel in a Macro_Function is that for the Macro_Function, the panel has an **Edit** mode.

When in Edit mode, the Macro_Function has already been run before and the panel information for the macro is loaded from the previous run of the macro.

The easiest way to set this up is to build the panel in a function in the same way as you would in a standard macro, but pass down to the panel function an **edit** flag where:

when **edit** is zero, the panel is being run for the first time and there is no data to load from a previous run. This is the case when there are no command line arguments. *See* Processing Command Line Arguments in a Macro_Function .

when **edit** is one, the panel is in Edit mode and the values from a previous run are loaded into the panel fields. This is the case when the command line argument is "-function_edit". *See* Processing Command Line Arguments in a Macro_Function .

The following is an example of some code from Example 15 (see Example 15) to build a panel for both the first time the Macro_Function is called, and when it is called in Edit mode:

```
Integer show_panel(Text function_name,Integer edit)
// ----------------------------------------------------------------------------------------------
```

```
// edit = 0 for the first time that the macro has been run
// edit = 1 when in edit mode. That is, the macro has been run before
// function_name is the function name. This is only known if the macro has been run before.
// That is, when edit = 1
// Note: in the section that processes the command line arguments,
// edit is set to 1 when the command line argument is "-function_edit"
// edit is set to 0 when their are no command line arguments
// See Processing Command Line Arguments in a Macro_Function
//-----------------------------------------------------------------------------------------------
// Macro_Function Dependencies
//     "string"          Element
//
// Macro_Function attributes
//     "offset"          Real
//     "start point"     Text
//     "end point"       Text
//     "new name"        Text
//     "new model"       Text
//     "new colour"      Text
//     "functype"        Text
//     "model"           Uid
//     "element"         Uid
//-----------------------------------------------------------------------------------------------
{
  Macro_Function macro_function;

  Get_macro_function(function_name,macro_function);

  Panel          panel      = Create_panel("Parallel String Section");
  Vertical_Group vgroup     = Create_vertical_group(0);
  Message_Box    message    = Create_message_box(" ");

// function box

  Function_Box function_box = Create_function_box("Function name", message,
                                        CHECK_FUNCTION_CREATE,RUN_MACRO_T);

  Set_type(function_box,"parallel_part");   // set the function type so that the pop-up for the
                                            // function_box only shows functions of this type
  Append(function_box,vgroup);

  if(edit) Set_data(function_box,function_name);   // when in edit mode, function name is known
                                                   // so load function_box with function_name

// string to parallel

  New_Select_Box select_box = Create_new_select_box("String to parallel","Select string",
                                                     SELECT_STRING, message);

  Append(select_box,vgroup);

  if(edit) {                // when in edit mode, load select_box with the string from the last run.

    Element string;

    Get_dependancy_element(macro_function,"string",string);
    Set_data(select_box,string);
  }

// offset distance

  Real_Box value_box = Create_real_box("Offset",message);
  Append(value_box,vgroup);

  if(edit) {    // when in edit mode, load value_box with the offset from the last run
                // offset was stored as a Real macro function attribute called "offset"
```

```
   Real offset;
   Get_function_attribute(macro_function,"offset",offset);
   Set_data(value_box,offset);
 }
```

▪ ▪ ▪

Continue to Storing the Panel Information for Processing

All the *12d Model* Macro_Function calls are given in Macro_Function Functions .

Storing the Panel Information for Processing

The panel information needs to be stored in the Macro_Function so that it is available at future times.

The following is an example of some code from Example 15 (see Example 15) that goes in the section after the Process button has been selected. The panel information has been validated and the next step is to store the information into the Macro_Function and call macro_recalc.

```
// Store the panel information in the Macro_Function

   Delete_all_dependancies(macro_function);    // clean out any data already there

   Set_function_attribute(macro_function,"functype","parallel_part");   // type of function

   Add_dependancy_element(macro_function,"string",string);   // string to be paralleled

   Set_function_attribute(macro_function,"offset", offset);         // offset value
   Set_function_attribute(macro_function,"start point",start);    // start chainage for parallel
   Set_function_attribute(macro_function,"end point",end);   // end chainage for parallel
   Set_function_attribute(macro_function,"new name",name);   // name of the created string
   Set_function_attribute(macro_function,"new model",name);   // model for the created string
   Set_function_attribute(macro_function,"new colour",colour_txt);  // colour of the created string

// Now do the processing
      Integer res = recalc_macro(function_name);
```

▪ ▪ ▪

Continue to Recalcing

All the *12d Model* Macro_Function calls are given in Macro_Function Functions .

Recalcing

For a Macro_Function, it is usually best to put all the processing into its own function, say recalc_macro.

That way the one calculation function can be used for each of the three processing cases:

1. when a Recalc is done.
2. when the Macro_Function is run for the first time and the process button is selected

3. when an Edit is done, the panels fields modified and the process button then selected

In the first case of a Recalc, all the information required for processing must already be contained in the Macro_Function itself and it is accessed via Get_dependency and Get_function_attribute calls.

For cases 2 and 3, a panel is actually displayed, information collected and then a process button selected. In both cases, the Macro_Function structure can be used to pass information through to the processing function by simply loading the information into Macro_Function via the function dependencies and function attributes **before** the processing function recalc_macro.

So in all cases, the information is accused by the processing function recalc_maco in exactly the same way (*See* Storing the Panel Information for Processing on how to store the information).

So with recalc_macro function should:

(a) load and validate the panel data stored in the Macro_Function

(b) check that the data created by the previous run can be replaced (deleted or modified), and clean it up as required.

For example, a string can not be deleted if it is locked by another option.

(c) if there are no problems, do the processing.

(d) save links to the new created data as attributes in the Macro_Function.

Continue to Storing the Panel Information for Processing

All the *12d Model* Macro_Function calls are given in Macro_Function Functions .

Storing Calculated Information

The data created by the Macro_Function are usually Elements such as Tins, Model and Strings.

Models and Tins could be stored by their names since their names are unique to a project. On the other hand, a Model or Tin name may be changed so maybe their Uid's should be saved. Or both the name and the Uid could be saved.

Strings do not have unique names and usually it is best to save them by their Uids. If the processing produces strings with sequential Uids, then just the first and the last Uids need to be stored.

There is no definite answer to how the information should be stored because it varies with every macro.

In the code extract below from Example 15 (see Example 15) the paralleled string is stored as the Uid of the model containing the string, and the Uid of the string.

```
// store details of the created string in function attributes

   Uid mid, eid;

   Get_id(model,mid);                  // get the Uid of the model containing elt
   Get_id(elt,eid);                    // get the Uid of elt

   Set_function_attribute(macro_function,"model",mid);
   Set_function_attribute(macro_function,"element",eid);
```

All the *12d Model* Macro_Function calls are given in Macro_Function Functions .

Macro_Function Functions

Create_macro_function(Text function_name,Macro_Function &func)

Name

Integer Create_macro_function(Text function_name,Macro_Function &func)

Description

Create a user defined *12d Model* Function with the name **function_name** and return the created Function as **func**.

If a Function with the name function_name already exists, the function fails and a non-zero function return value is returned.

A function return value of zero indicates the Function was successfully created.

ID = 1135

Function_recalc(Function func)

Name

Integer Function_recalc(Function func)

Description

Recalc (i.e. re-run) the Function **func**.

A function return value of zero indicates the recalc was successful.

ID = 1138

Function_exists(Text function_name)

Name

Integer Function_exists(Text function_name)

Description

Checks to see if a 12d or user 12d Function with the name **function_name** exists.

A non-zero function return value indicates a Function does exist.

A zero function return value indicates that no Function of name **function_name** exists.

Warning - this is the opposite of most 12dPL function return values.

ID = 1141

Function_rename(Text original_name,Text new_name)

Name

Integer Function_rename(Text original_name,Text new_name)

Description

Change the name of the Function **original_name** to the new name **new_name**.

A function return value of zero indicates the rename was successful.

ID = 425

Get_name(Function func,Text &name)

Name

Integer Get_name(Function func,Text &name)

Description

Get the name of the Function **func** and return it in **name**.

A function return value of zero indicates the Function name was successfully returned.

ID = 1455

Get_function(Text function_name)

Name

Function Get_function(Text function_name)

Description

Get the Function with the name **function_nam**e and return it as the function return value.

LJG? what if the function does not exist?

The existence of a function with the name function_name can first be checked by the call Function_exists(**function_nam**e).

ID = 1140

Get_macro_function(Text function_name,Macro_Function &func)

Name

Integer Get_macro_function(Text function_name,Macro_Function &func)

Description

Get the Macro Function with the name **function_nam**e and return it as **func**.

If the Function named **function_name** does not exist, or it does exist but is not a Macro Function, then the function fails and a non-zero function return value is returned.

A function return value of zero indicates the Macro Function was successfully returned.

ID = 1142

Get_all_functions(Dynamic_Text &functions)

Name

Integer Get_all_functions(Dynamic_Text &functions)

Description

Get all names of the 12d and user defined Function currently in the project. The Function names are returned in the Dynamic_Text **functions**.

A function return value of zero indicates the Function names are returned successfully.

ID = 1139

Function_delete(Text function_name)

Name

Integer Function_delete(Text function_name)

Description

Delete the Function with the name **function_name**.

Note that the data in the function is not deleted.

If a Function with the name **function_name** does not exist, the function fails and a non-zero function return value is returned.

A function return value of zero indicates the Function was successfully deleted.

ID = 1137

Get_time_created(Function func,Integer &time)

Name

Integer Get_time_created(Function func,Integer &time)

Description

Get the time that the Function **func** was created and return the time in **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully returned.

ID = 2117

Get_time_updated(Function func,Integer &time)

Name

Integer Get_time_updated(Function func,Integer &time)

Description

Get the time that the Function **func** was last updated and return the time in **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully returned.

ID = 2118

Set_time_updated(Function func,Integer time)

Name

Integer Set_time_updated(Function func,Integer time)

Description

Set the update time for the Function **func** to **time**.

The time **time** is given as seconds since January 1 1970.

A function return value of zero indicates the time was successfully set.

ID = 2119

Add_dependancy_file(Macro_Function func,Text name,Text file)

Name

Integer Add_dependancy_file(Macro_Function func,Text name,Text file)

Description

Record in the Macro Function **func**, that the disk file named **file** is dependant on **func** and on a recalc of **func**, needs to be checked for changes from the last time that **func** was recalced.

The dependency is added with the unique name **name.**

If a dependency called **name** already exists, a non-zero function return value is returned and no

dependency is added.

A function return value of zero indicates the dependency was successfully set.

ID = 1143

Add_dependancy_model(Macro_Function func,Text name,Model model)

Name

Integer Add_dependancy_model(Macro_Function func,Text name,Model model)

Description

Record in the Macro Function **func**, that the Model **model** is dependant on **func** and on a recalc of **func**, needs to be checked for changes from the last time that **func** was recalced.

If a dependency called **name** already exists, a non-zero function return value is returned and no dependency is added.

A function return value of zero indicates the dependency was successfully set.

ID = 1144

Add_dependancy_tin(Macro_Function func,Text name,Tin tin)

Name

Integer Add_dependancy_tin(Macro_Function func,Text name,Tin tin)

Description

Record in the Macro Function **func**, that the Tin **tin** is dependant on **func** and on a recalc of **func**, needs to be checked for changes from the last time that **func** was recalced.

If a dependency called **name** already exists, a non-zero function return value is returned and no dependency is added.

A function return value of zero indicates the dependency was successfully set.

ID = 1145

Integer Add_dependancy_template(Macro_Function func,Text name,Text template)

Name

Integer Add_dependancy_template(Macro_Function func,Text name,Text template)

Description

Record in the Macro Function **func**, that the template name **template** is dependant on **func** and on a recalc of **func**, needs to be checked for changes from the last time that **func** was recalced.

If a dependency called **name** already exists, a non-zero function return value is returned and no dependency is added.

A function return value of zero indicates the dependency was successfully set.

ID = 1146

Add_dependancy_element(Macro_Function func,Text name,Element elt)

Name

Integer Add_dependancy_element(Macro_Function func,Text name,Element elt)

Description

Record in the Macro Function **func**, that the Element **elt** is dependant on **func** and on a recalc of **func**, needs to be checked for changes from the last time that **func** was recalced.

If a dependency called **name** already exists, a non-zero function return value is returned and no dependency is added.

A function return value of zero indicates the dependency was successfully set.

ID = 1147

Get_number_of_dependancies(Macro_Function func,Integer &count)

Name

Integer Get_number_of_dependancies(Macro_Function func,Integer &count)

Description

For the Macro_Function **func**, return the number of dependencies that exist for func and return the number in **count**.

A function return value of zero indicates the count was successfully returned.

ID = 1148

Get_dependancy_name(Macro_Function func,Integer i,Text &name)

Name

Integer Get_dependancy_name(Macro_Function func,Integer i,Text &name)

Description

For the Macro_Function **func**, return the name of the **i**'th dependencies in **name**.

A function return value of zero indicates the name was successfully returned.

ID = 1149

Get_dependancy_type(Macro_Function func,Integer i,Text &type)

Name

Integer Get_dependancy_type(Macro_Function func,Integer i,Text &type)

Description

For the Macro_Function **func**, return the *type* of the **i**'th dependencies as the Text **type**.

The valid types are:

unknown
File
Element
Model
Template
Tin
Integer
Real
Text

A function return value of zero indicates the type was successfully returned.

ID = 1150

Get_dependancy_file(Macro_Function func,Integer i,Text &file)

Name

Integer Get_dependancy_file(Macro_Function func,Integer i,Text &file)

Description

For the Macro_Function **func**, if the **i**'th dependency is a file then return the name of the file in **name**.

If the **i'**th dependency is not a file then a non-zero function return value is returned.

A function return value of zero indicates the file name was successfully returned.

ID = 1151

Get_dependancy_model(Macro_Function func,Integer i,Model &model)

Name

Integer Get_dependancy_model(Macro_Function func,Integer i,Model &model)

Description

For the Macro_Function **func**, if the **i**'th dependency is a Model then return the Model in **model**.

If the **i'**th dependency is not a Model then a non-zero function return value is returned.

A function return value of zero indicates the Model was successfully returned.

ID = 1152

Get_dependancy_tin(Macro_Function func,Integer i,Tin &tin)

Name

Integer Get_dependancy_tin(Macro_Function func,Integer i,Tin &tin)

Description

For the Macro_Function **func**, if the **i**'th dependency is a Tin then return the Tin in **tin**.

If the **i'**th dependency is not a Tin then a non-zero function return value is returned.

A function return value of zero indicates the Tin was successfully returned.

ID = 1153

Get_dependancy_template(Macro_Function func,Integer i,Text &template)

Name

Integer Get_dependancy_template(Macro_Function func,Integer i,Text &template)

Description

For the Macro_Function **func**, if the **i**'th dependency is a Template then return the template name in **template**.

If the **i'**th dependency is not a Template then a non-zero function return value is returned.

A function return value of zero indicates the Tin was successfully returned.

ID = 1154

Get_dependancy_element(Macro_Function func,Integer i,Element &element)

Name

Integer Get_dependancy_element(Macro_Function func,Integer i,Element &element)

Description

For the Macro_Function **func**, if the **i**'th dependency is an Element then return the Element in **elt**.

If the **i'**th dependency is not an Element then a non-zero function return value is returned.

A function return value of zero indicates the Element was successfully returned.

ID = 1155

Get_dependancy_data(Macro_Function func,Integer i,Text &text)

Name

Integer Get_dependancy_data(Macro_Function func,Integer i,Text &text)

Description

For the Macro_Function **func**, a text description of the **i**'th dependency is returned in **text**.

For an Element, the text description is: model_name->element_name is return in text.

For a File/Model/Template/Tin, the text description is the name of the File/Model/Template/Tin.

For an Integer, the text description is the Integer converted to Text.

For a Real, the text description is the Real converted to Text. LJG? how many decimals

For a Text, the text description is just the text.

A function return value of zero indicates the Macro_Function description was successfully returned.

ID = 1156

Get_dependancy_type(Macro_Function func,Text name,Text &type)

Name

Integer Get_dependancy_type(Macro_Function func,Text name,Text &type)

Description

For the Macro_Function **func**, return the *type* of the dependency with the name name as the Text **type**.

The valid types are:

```
unknown
File
Element
Model
Template
Tin
Integer        // not implemented or accessible from macros
Real           // not implemented or accessible from macros
Text           // not implemented or accessible from macros
```

If a dependency called **name** does not exist then a non-zero function return value is returned.

A function return value of zero indicates the type was successfully returned.

ID = 1157

Get_dependancy_file(Macro_Function func,Text name,Text &file)

Name

Integer Get_dependancy_file(Macro_Function func,Text name,Text &file)

Description

For the Macro_Function **func**, get the dependency called **name** and if it is a File, return the file name as **file**.

If no dependency called name exists, or it does exist and it is not a file, then a non-zero function return value is returned.

LJG? if error, is text returned as blank?

A function return value of zero indicates the file name was successfully returned.

ID = 1158

Get_dependancy_model(Macro_Function func,Text name,Model &model)

Name

Integer Get_dependancy_model(Macro_Function func,Text name,Model &model)

Description

For the Macro_Function **func**, get the dependency called **name** and if it is a Model, return the Model as **model**.

If no dependency called **name** exists, or it does exist and it is not a Model, then a non-zero function return value is returned.

LJG? if error, is model returned as null?

A function return value of zero indicates the Model was successfully returned.

ID = 1159

Get_dependancy_tin(Macro_Function func,Text name,Tin &tin)

Name

Integer Get_dependancy_tin(Macro_Function func,Text name,Tin &tin)

Description

For the Macro_Function **func**, get the dependency called **name** and if it is a Tin, return the Tin as **tin**.

If no dependency called **name** exists, or it does exist and it is not a Tin, then a non-zero function return value is returned.

LJG? if error, is tin returned as null?

A function return value of zero indicates the Tin was successfully returned.

ID = 1160

Get_dependancy_template(Macro_Function func,Text name,Text &template)

Name

Integer Get_dependancy_template(Macro_Function func,Text name,Text &template)

Description

For the Macro_Function **func**, get the dependency called **name** and if it is a Template, return the Template name as **template**.

If no dependency called **name** exists, or it does exist and it is not a Template, then a non-zero

function return value is returned.

LJG? if error, is template returned as blank?

A function return value of zero indicates the template name was successfully returned.

ID = 1161

Get_dependancy_element(Macro_Function func,Text name,Element &elt)

Name

Integer Get_dependancy_element(Macro_Function func,Text name,Element &element)

Description

For the Macro_Function **func**, get the dependency called **name** and if it is an Element, return the Element as **elt**.

If no dependency called **name** exists, or it does exist and it is not an Element, then a non-zero function return value is returned.

LJG? if error, is elt returned as null?

A function return value of zero indicates the Element was successfully returned.

ID = 1162

Get_dependancy_data(Macro_Function func,Text name,Text &text)

Name

Integer Get_dependancy_data(Macro_Function func,Text name,Text &text)

Description

For the Macro_Function **func**, get the dependency called **name** and if it is a Text, return the Text as **text**.

If no dependency called **name** exists, or it does exist and it is not a Text, then a non-zero function return value is returned.

LJG? if error, is text returned as blank?

A function return value of zero indicates the Text was successfully returned.

ID = 1163

Delete_dependancy(Macro_Function func,Text name)

Name

Integer Delete_dependancy(Macro_Function func,Text name)

Description

For the Macro_Function **func**, if the dependency called **name** exist then it is deleted from the list of dependencies for **func**.

Warning: if a dependency is deleted then the dependency number of all dependencies after the deleted one will be reduced by one.

If no dependency called **name** exists then a non-zero function return value is returned.

A function return value of zero indicates the dependency was successfully deleted.

ID = 1164

Delete_all_dependancies(Macro_Function func)

Name

Integer Delete_all_dependancies(Macro_Function func)

Description

For the Macro_Function **func**, delete all the dependencies.

A function return value of zero indicates all the dependency were successfully deleted.

ID = 1165

Get_id(Function func,Uid &id)

Name

Integer Get_id(Function func,Uid &id)

Description

For the Function/Macro_Function **func**, get its unique Uid in the Project and return it in **id**.

A function return value of zero indicates the Uid was successfully returned.

ID = 1909

Get_id(Function func,Integer &id)

Name

Integer Get_id(Function func,Integer &id)

Description

For the Function/Macro_Function **func**, get its unique id in the Project and return it in **id**.

A function return value of zero indicates the id was successfully returned.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Get_id(Function func,Uid &id)* instead.

ID = 1177

Get_function_id(Element elt,Uid &id)

Name

Integer Get_function_id(Element elt,Uid &id)

Description

For an Element **elt**, check if it has a function Uid and if it has, return it in **id**.

LJG? What if it doesn't have a function Uid. Is that a error return code or is something like 0 returned?

A function return value of zero indicates the Uid was successfully returned.

ID = 1910

Get_function_id(Element elt,Integer &id)

Name

Integer Get_function_id(Element elt,Integer &id)

Description

For an Element **elt**, check if it has a function id and if it has, return it in **id**.

LJG? What if it doesn't have a function id. Is that a error return code or is something like 0 returned?

A function return value of zero indicates the id was successfully returned.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Get_function_id(Element elt,Uid &id)* instead.

ID = 1178

Set_function_id(Element elt,Uid id)

Name

Integer Set_function_id(Element elt,Uid id)

Description

For an Element **elt**, set its function Uid to **id**.

A function return value of zero indicates the function Uid was successfully set.

ID = 1911

Set_function_id(Element elt,Integer id)

Name

Integer Set_function_id(Element elt,Integer id)

Description

For an Element **elt**, set its function id to **id**.

A function return value of zero indicates the function id was successfully set.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Set_function_id(Element elt,Uid id)* instead.

ID = 1179

Get_function(Uid function_id)

Name

Function Get_function(Uid function_id)

Description

Find the Function/Macro_Function with the Uid **function_id.**

The Function is returned as the function return value.

If there is no Function/Macro_Function with the Uid **function_id,** then a null Function/Macro_Function is returned as the function return value.

ID = 1916

Get_function(Integer function_id)

Name

Function Get_function(Integer function_id)

Description

Find the Function/Macro_Function with the Id **function_id.**

The Function is returned as the function return value.

If there is no Function/Macro_Function with the Id **function_id,** then a null Function/Macro_Function is returned as the function return value.

Deprecation Warning - this function has now been deprecated and will no longer exist unless special compile flags are used. Use *Get_function(Uid function_id)* instead.

ID = 1188

Function_exists(Uid function_id)

Name

Integer Function_exists(Uid function_id)

Description

Checks to see if a Function/Macro_Function with Uid function_id exists.

A non-zero function return value indicates that a Function does exist.

A zero function return value indicates that no Function exists.

Warning this is the opposite of most 12dPL function return values

ID = 1915

Function_attribute_exists(Macro_Function fcn,Text att_name)

Function_attribute_exists(Function fcn,Text att_name)

Name

Integer Function_attribute_exists(Macro_Function fcn,Text att_name)

Integer Function_attribute_exists(Function fcn,Text att_name)

Description

Checks to see if an attribute with the name **att_name** exists for the Macro_Function/Function **fcn**.

A non-zero function return value indicates that the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 1109

Function_attribute_exists(Function fcn,Text name,Integer &no)

Function_attribute_exists(Macro_Function fcn,Text name,Integer &no)

Name

Integer Function_attribute_exists(Function fcn,Text name,Integer &no)

Integer Function_attribute_exists(Macro_Function fcn,Text name,Integer &no)

Description

Checks to see if an attribute with the name **att_name** exists for the Macro_Function/Function **fcn**.

If the attribute exists, its position is returned in Integer **no**.

This position can be used in other Attribute functions described below.

A non-zero function return value indicates the attribute does exist.

A zero function return value indicates that no attribute of that name exists.

Warning this is the opposite of most 12dPL function return values

ID = 1110

Function_attribute_delete(Macro_Function fcn,Text att_name)

Function_attribute_delete(Function fcn,Text att_name)

Name

Integer Function_attribute_delete(Macro_Function fcn,Text att_name)

Integer Function_attribute_delete(Function fcn,Text att_name)

Description

Delete the attribute with the name **att_name** from the Macro_Function/Function **fcn**.

A function return value of zero indicates the attribute was deleted.

ID = 1111

Function_attribute_delete(Macro_Function fcn,Integer att_no)

Function_attribute_delete(Function fcn,Integer att_no)

Name

Integer Function_attribute_delete(Macro_Function fcn,Integer att_no)

Integer Function_attribute_delete(Function fcn,Integer att_no)

Description

Delete the attribute with the number **att_no** from the Macro_Function/Function **fcn**.

A function return value of zero indicates the attribute was deleted.

ID = 1112

Function_attribute_delete_all(Function fcn)

Function_attribute_delete_all(Macro_Function fcn)

Name

Integer Function_attribute_delete_all(Function fcn)

Integer Function_attribute_delete_all(Macro_Function fcn)

Description

Delete all the attributes from the Macro_Function/Function **fcn**.

A function return value of zero indicates all the attribute were deleted.

ID = 1113

Function_attribute_dump(Function fcn)

Function_attribute_dump(Macro_Function fcn)

Name

Integer Function_attribute_dump(Function fcn)

Integer Function_attribute_dump(Macro_Function fcn)

Description

Write out information about the Macro_Function/Function attributes to the Output Window.

A function return value of zero indicates the function was successful.

ID = 1114

Function_attribute_debug(Macro_Function fcn)

Function_attribute_debug(Function fcn)

Name

Integer Function_attribute_debug(Macro_Function fcn)

Integer Function_attribute_debug(Function fcn)

Description

Write out even more information about the Macro_Function/Function attributes to the Output Window.

A function return value of zero indicates the function was successful.

ID = 1115

Get_function_number_of_attributes(Function fcn,Integer &no_atts)

Get_function_number_of_attributes(Macro_Function fcn,Integer &no_atts)

Name

Integer Get_function_number_of_attributes(Function fcn,Integer &no_atts)

Integer Get_function_number_of_attributes(Macro_Function fcn,Integer &no_atts)

Description

Get the number of top level attributes in the Macro_Function/Function **fcn** and return it in **no_atts**.

A function return value of zero indicates the number is successfully returned

ID = 1116

Get_function_attribute(Macro_Function fcn,Text att_name,Text &txt)

Get_function_attribute(Function fcn,Text att_name,Text &txt)

Name

Integer Get_function_attribute(Macro_Function fcn,Text att_name,Text &att)

Integer Get_function_attribute(Function fcn,Text att_name,Text &txt)

Description

For the Macro_Function/Function **fcn**, get the attribute called **att_name** and return the attribute value in **txt**. The attribute must be of type Text.

If the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called

att_name.

ID = 1117

Get_function_attribute(Macro_Function fcn,Text att_name,Integer &int)

Get_function_attribute(Function fcn,Text att_name,Integer &int)

Name

Integer Get_function_attribute(Macro_Function fcn,Text att_name,Integer &int)

Integer Get_function_attribute(Function fcn,Text att_name,Integer &int)

Description

For the Macro_Function/Function **fcn**, get the attribute called **att_name** and return the attribute value in **int**. The attribute must be of type Integer.

If the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1118

Get_function_attribute(Function fcn,Text att_name,Real &real)

Get_function_attribute(Macro_Function fcn,Text att_name,Real &real)

Name

Integer Get_function_attribute(Function fcn,Text att_name,Real &real)

Integer Get_function_attribute(Macro_Function fcn,Text att_name,Real &real)

Description

For the Macro_Function/Function **fcn**, get the attribute called **att_name** and return the attribute value in **real**. The attribute must be of type Real.

If the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1119

Get_function_attribute(Function fcn,Integer att_no,Text &txt)

Get_function_attribute(Macro_Function fcn,Integer att_no,Text &txt)

Name

Integer Get_function_attribute(Function fcn,Integer att_no,Text &txt)

Integer Get_function_attribute(Macro_Function fcn,Integer att_no,Text &txt)

Description

For the Macro_Function/Function **fcn**, get the attribute with attribute number **att_no** and return the attribute value in **txt**. The attribute must be of type Text.

If the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1120

Get_function_attribute(Function fcn,Integer att_no,Integer &int)

Get_function_attribute(Macro_Function fcn,Integer att_no,Integer &int)

Name

Integer Get_function_attribute(Function fcn,Integer att_no,Integer &int)

Integer Get_function_attribute(Macro_Function fcn,Integer att_no,Integer &int)

Description

For the Macro_Function/Function **fcn**, get the attribute with attribute number **att_no** and return the attribute value in **int**. The attribute must be of type Integer.

If the attribute is not of type Integer then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1121

Get_function_attribute(Function fcn,Integer att_no,Real real)

Get_function_attribute(Macro_Function fcn,Integer att_no,Real real)

Name

Integer Get_function_attribute(Function fcn,Integer att_no,Real real)

Integer Get_function_attribute(Macro_Function fcn,Integer att_no,Real real)

Description

For the Macro_Function/Function **fcn**, get the attribute with attribute number **att_no** and return the attribute value in **real**. The attribute must be of type Real.

If the attribute is not of type Real then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1122

Get_function_attribute_name(Macro_Function fcn,Integer att_no,Text &txt)

Get_function_attribute_name(Function fcn,Integer att_no,Text &txt)

Name

Integer Get_function_attribute_name(Macro_Function fcn,Integer att_no,Text &txt)

Integer Get_function_attribute_name(Function fcn,Integer att_no,Text &txt)

Description

For the Macro_Function/Function **fcn**, get the attribute with attribute number **att_no** and return

the attribute value in **txt**. The attribute must be of type Text.

If the attribute is not of type Text then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1123

Get_function_attribute_type(Macro_Function fcn,Text att_name,Integer &att_type)

Get_function_attribute_type(Function fcn,Text att_name,Integer &att_type)

Name

Integer Get_function_attribute_type(Macro_Function fcn,Text att_name,Integer &att_type)

Integer Get_function_attribute_type(Function fcn,Text att_name,Integer &att_type)

Description

For the Macro_Function/Function **fcn**, get the type of the attribute called **att_name** and return the attribute type in **att_type**.

A function return value of zero indicates the attribute type is successfully returned.

ID = 1124

Get_function_attribute_type(Function fcn,Integer att_no,Integer &att_type)

Get_function_attribute_type(Macro_Function fcn,Integer att_no,Integer &att_type)

Name

Integer Get_function_attribute_type(Function fcn,Integer att_no,Integer &att_type)

Integer Get_function_attribute_type(Macro_Function fcn,Integer att_no,Integer &att_type)

Description

For the Macro_Function/Function **fcn**, get the type of the attribute with attribute number **att_no** and return the attribute type in **att_type**.

A function return value of zero indicates the attribute type is successfully returned.

ID = 1125

Get_function_attribute_length(Function fcn,Text att_name,Integer &att_len)

Get_function_attribute_length(Macro_Function fcn,Text att_name,Integer &att_len)

Name

Integer Get_function_attribute_length(Function fcn,Text att_name,Integer &att_len)

Integer Get_function_attribute_length(Macro_Function fcn,Text att_name,Integer &att_len)

Description

For the Macro_Function/Function **fcn**, get the length in bytes of the attribute of name **att_name**. The number of bytes is returned in **att_len**.

This is mainly for use with attributes of types Text and Binary (blobs)

A function return value of zero indicates the attribute length is successfully returned.

ID = 1126

Get_function_attribute_length(Function fcn,Integer att_no,Integer &att_len)

Get_function_attribute_length(Macro_Function fcn,Integer att_no,Integer &att_len)

Name

Integer Get_function_attribute_length(Function fcn,Integer att_no,Integer &att_len)

Integer Get_function_attribute_length(Macro_Function fcn,Integer att_no,Integer &att_len)

Description

For the Macro_Function/Function **fcn**, get the length in bytes of the attribute with attribute number **att_no**. The number of bytes is returned in **att_len**.

This is mainly for use with attributes of types Text and Binary (blobs)

A function return value of zero indicates the attribute length is successfully returned.

ID = 1127

Set_function_attribute(Function fcn,Text att_name,Text txt)

Set_function_attribute(Macro_Function fcn,Text att_name,Text txt)

Name

Integer Set_function_attribute(Function fcn,Text att_name,Text txt)

Integer Set_function_attribute(Macro_Function fcn,Text att_name,Text txt)

Description

For the Macro_Function/Function **fcn**,
if the attribute called **att_name** does not exist then create it as type Text and give it the value **txt**.
if the attribute called **att_name** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text, or the attribute does not exist, then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1128

Set_function_attribute(Function fcn,Text att_name,Integer int)

Set_function_attribute(Macro_Function fcn,Text att_name,Integer int)

Name

Integer Set_function_attribute(Function fcn,Text att_name,Integer int)

Integer Set_function_attribute(Macro_Function fcn,Text att_name,Integer int)

Description

For the Macro_Function/Function **fcn**,
if the attribute called **att_name** does not exist then create it as type Integer and give it the value **int**.
if the attribute called **att_name** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer, or the attribute does not exist, then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1129

Set_function_attribute(Macro_Function fcn,Text att_name,Real real)

Set_function_attribute(Function fcn,Text att_name,Real real)

Name

Integer Set_function_attribute(Macro_Function fcn,Text att_name,Real real)

Integer Set_function_attribute(Function fcn,Text att_name,Real real)

Description

For the Macro_Function/Function **fcn**,
if the attribute called **att_name** does not exist then create it as type Real and give it the value **real**.
if the attribute called **att_name** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real, or the attribute does not exist, then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_function_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1130

Set_function_attribute(Macro_Function fcn,Integer att_no,Text txt)

Set_function_attribute(Function fcn,Integer att_no,Text txt)

Name

Integer Set_function_attribute(Macro_Function fcn,Integer att_no,Text txt)

Integer Set_function_attribute(Function fcn,Integer att_no,Text txt)

Description

For the Macro_Function/Function **fcn**,
if the attribute with attribute number **att_no** does not exist then create it as type Text and give it the value **txt**.
if the attribute with attribute number **att_no** does exist and it is type Text, then set its value to **txt**.

If the attribute exists and is not of type Text, or the attribute does not exist, then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_function_attribute_type call can be used to get the type of the attribute with attribute number **att_no.**

ID = 1131

Set_function_attribute(Function fcn,Integer att_no,Integer int)

Set_function_attribute(Macro_Function fcn,Integer att_no,Integer int)

Name

Integer Set_function_attribute(Function fcn,Integer att_no,Integer int)

Integer Set_function_attribute(Macro_Function fcn,Integer att_no,Integer int)

Description

For the Macro_Function/Function **fcn**,
if the attribute with attribute number **att_no** does not exist then create it as type Integer and give it the value **int**.
if the attribute with attribute number **att_no** does exist and it is type Integer, then set its value to **int**.

If the attribute exists and is not of type Integer, or the attribute does not exist, then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_function_attribute_type call can be used to get the type of the attribute with attribute number **att_no.**

ID = 1132

Set_function_attribute(Macro_Function fcn,Integer att_no,Real real)

Set_function_attribute(Function fcn,Integer att_no,Real real)

Name

Integer Set_function_attribute(Macro_Function fcn,Integer att_no,Real real)

Integer Set_function_attribute(Function fcn,Integer att_no,Real real)

Description

For the Macro_Function/Function **fcn**,
if the attribute with attribute number **att_no** does not exist then create it as type Real and give it the value **real**.
if the attribute with attribute number **att_no** does exist and it is type Real, then set its value to **real**.

If the attribute exists and is not of type Real, or the attribute does not exist, then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_function_attribute_type call can be used to get the type of the attribute with attribute number **att_no.**

ID = 1133

Get_function_attributes(Function fcn,Attributes &att)

Get_function_attributes(Macro_Function fcn,Attributes &att)

Name

Integer Get_function_attributes(Function fcn,Attributes &att)

Integer Get_function_attributes(Macro_Function fcn,Attributes &att)

Description

For the Function/Macro_Function **fcn**, return the Attributes for the Function/Macro_Function as **att**.

If **fcn** has no Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute is successfully returned.

ID = 1992

Set_function_attributes(Function fcn,Attributes att)

Set_function_attributes(Macro_Function fcn,Attributes att)

Name

Integer Set_function_attributes(Function fcn,Attributes att)

Integer Set_function_attributes(Macro_Function fcn,Attributes att)

Description

For the Function/Macro_Function **fcn**, set the Attributes for the Function/Macro_Function **fcn** to **att**.

A function return value of zero indicates the attribute is successfully set.

ID = 1993

Get_function_attribute(Function fcn,Text att_name,Uid &uid)

Get_function_attribute(Macro_Function fcn,Text att_name,Uid &uid)

Name

Integer Get_function_attribute(Function fcn,Text att_name,Uid &uid)

Integer Get_function_attribute(Macro_Function fcn,Text att_name,Uid &uid)

Description

From the Function/Macro_Function **fcn**, get the attribute called **att_name** and return the attribute value in **uid**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1994

Get_function_attribute(Macro_Function fcn,Text att_name,Attributes &att)

Get_function_attribute(Function fcn,Text att_name,Attributes &att)

Name

Integer Get_function_attribute(Macro_Function fcn,Text att_name,Attributes &att)

Integer Get_function_attribute(Function fcn,Text att_name,Attributes &att)

Description

From the Function/Macro_Function **fcn**, get the attribute called **att_name** and return the attribute value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1995

Get_function_attribute(Macro_Function fcn,Integer att_no,Uid &uid)

Get_function_attribute(Function fcn,Integer att_no,Uid &uid)

Name

Integer Get_function_attribute(Macro_Function fcn,Integer att_no,Uid &uid)

Integer Get_function_attribute(Function fcn,Integer att_no,Uid &uid)

Description

From the Function/Macro_Function **fcn**, get the attribute with number **att_no** and return the attribute value in **uid**. The attribute must be of type Uid.

If the attribute is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1996

Get_function_attribute(Function fcn,Integer att_no,Attributes &att)

Get_function_attribute(Macro_Function fcn,Integer att_no,Attributes &att)

Name

Integer Get_function_attribute(Function fcn,Integer att_no,Attributes &att)

Integer Get_function_attribute(Macro_Function fcn,Integer att_no,Attributes &att)

Description

From the Function/Macro_Function **fcn**, get the attribute with number **att_no** and return the attribute value in **att**. The attribute must be of type Attributes.

If the attribute is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully returned.

Note - the Get_attribute_type call can be used to get the type of the attribute with attribute number **att_no**.

ID = 1997

Set_function_attribute(Function fcn,Text att_name,Uid uid)

Set_function_attribute(Macro_Function fcn,Text att_name,Uid uid)

Name

Integer Set_function_attribute(Function fcn,Text att_name,Uid uid)

Integer Set_function_attribute(Macro_Function fcn,Text att_name,Uid uid)

Description

For the Function/Macro_Function **fcn**,

if the attribute called **att_name** does not exist then create it as type Uid and give it the value **uid**.
if the attribute called **att_name** does exist and it is type Uid, then set its value to **att**.

If the attribute exists and is not of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1998

Set_function_attribute(Macro_Function fcn,Text att_name,Attributes att)

Set_function_attribute(Function fcn,Text att_name,Attributes att)

Name

Integer Set_function_attribute(Macro_Function fcn,Text att_name,Attributes att)

Integer Set_function_attribute(Function fcn,Text att_name,Attributes att)

Description

For the Function/Macro_Function **fcn**,
if the attribute called **att_name** does not exist then create it as type Attributes and give it the value **att**.
if the attribute called **att_name** does exist and it is type Attributes, then set its value to **att**.

If the attribute exists and is not of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_name**.

ID = 1999

Set_function_attribute(Macro_Function fcn,Integer att_no,Uid uid)

Set_function_attribute(Function fcn,Integer att_no,Uid uid)

Name

Integer Set_function_attribute(Macro_Function fcn,Integer att_no,Uid uid)

Integer Set_function_attribute(Function fcn,Integer att_no,Uid uid)

Description

For the Function/Macro_Function **fcn**, if the attribute number **att_no** exists and it is of type Uid, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Uid then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 2000

Set_function_attribute(Function fcn,Integer att_no,Attributes att)

Set_function_attribute(Macro_Function fcn,Integer att_no,Attributes att)

Name

Integer Set_function_attribute(Function fcn,Integer att_no,Attributes att)

Integer Set_function_attribute(Macro_Function fcn,Integer att_no,Attributes att)

Description

For the Function/Macro_Function **fcn**, if the attribute number **att_no** exists and it is of type Attributes, then its value is set to **att**.

If there is no attribute with number **att_no** then nothing can be done and a non-zero return code is returned.

If the attribute of number **att_no** exists and is **not** of type Attributes then a non-zero return value is returned.

A function return value of zero indicates the attribute value is successfully set.

Note - the Get_attribute_type call can be used to get the type of the attribute called **att_no**.

ID = 2001

Function Property Collections

Create_function_property_collection()

Name

Function_Property_Collection Create_function_property_collection()

Description

Create a **Function_Property_Collection**.

Function_Property_Collection's are used to transfer information about a function such as the Apply Many function instead of needing a large number of function calls which would need to be updated every time a new parameter was added to the Apply Many,

The function return value is the created Function_Property_Collection.

ID = 2726

Set_property(Function_Property_Collection collection,Text name,Integer int_val)

Name

Integer Set_property(Function_Property_Collection collection,Text name,Integer int_val)

Description

In the Function Property Collection **collection**, set the value of the Integer property called **name** to **int_val**.

For more information on which properties are available for the function in question, please see the section Function Properties .

LJG? is it non-zero return if name doesn't exist or it is not Integer property?

A function return value of zero indicates the value is successfully set.

ID = 2727

Set_property(Function_Property_Collection collection,Text name,Real real_val)

Name

Integer Set_property(Function_Property_Collection collection,Text name,Real real_val)

Description

In the Function Property Collection **collection**, set the value of the Real property called **name** to **real_val**.

For more information on which properties are available for the function in question, please see the section Function Properties .

LJG? is it non-zero return if name doesn't exist or it is not Integer property?

A function return value of zero indicates the value is successfully set.

ID = 2728

Set_property(Function_Property_Collection collection,Text name,Text txt_val)

Name

Integer Set_property(Function_Property_Collection collection,Text name,Text txt_val)

Description

In the Function Property Collection **collection**, set the value of the Text property called **name** to

txt_val.

For more information on which properties are available for the function in question, please see the section Function Properties .

LJG? is it non-zero return if name doesn't exist or it is not Integer property?

A function return value of zero indicates the value is successfully set.

ID = 2729

Set_property_colour(Function_Property_Collection collection,Text name,Text colour_name)

Name

Integer Set_property_colour(Function_Property_Collection collection,Text name,Text colour_name)

Description

In the Function Property Collection **collection**, set the value of the Colour property called **name** to the colour given by **colour_name**.

For more information on which properties are available for the function in question, please see the section Function Properties .

LJG? is it non-zero return if name doesn't exist or it is not Integer property?

A function return value of zero indicates the value is successfully set.

ID = 2730

Set_property(Function_Property_Collection collection,Text name,Element element)

Name

Integer Set_property(Function_Property_Collection collection,Text name,Element element)

Description

In the Function Property Collection **collection**, set the value of the Element property called **name** to **element**.

For more information on which properties are available for the function in question, please see the section Function Properties .

LJG? is it non-zero return if name doesn't exist or it is not Integer property?

A function return value of zero indicates the value is successfully set.

ID = 2731

Set_property(Function_Property_Collection collection,Text name,Tin tin)

Name

Integer Set_property(Function_Property_Collection collection,Text name,Tin tin)

Description

In the Function Property Collection **collection**, set the tin of the Tin property called **name** to **tin**.

For more information on which properties are available for the function in question, please see the section Function Properties .

LJG? is it non-zero return if name doesn't exist or it is not Integer property?

A function return value of zero indicates the value is successfully set.

ID = 2732

Set_property(Function_Property_Collection collection,Text name,Model model)

Name

Integer Set_property(Function_Property_Collection collection,Text name,Model model)

Description

In the Function Property Collection **collection**, set the model of the Model property called **name** to **model**.

For more information on which properties are available for the function in question, please see the section Function Properties .

LJG? is it non-zero return if name doesn't exist or it is not Integer property?

A function return value of zero indicates the value is successfully set.

ID = 2733

Get_property(Function_Property_Collection collection,Text name,Integer &int_val)

Name

Integer Get_property(Function_Property_Collection collection,Text name,Integer &int_val)

Description

From the Function Property Collection **collection**, get the value of the Integer property called **name** and return it in **int_val**.

For more information on which properties are available for the function in question, please see the section Function Properties .

The function return value is non zero if there is no property called **name**, or if it does exist, it is not of type Integer.

A function return value of zero indicates the value was successfully returned.

ID = 2737

Get_property(Function_Property_Collection collection,Text name,Real &real_val)

Name

Integer Get_property(Function_Property_Collection collection,Text name,Real &real_val)

Description

From the Function Property Collection **collection**, get the value of the Real property called **name** and return it in **real_val**.

For more information on which properties are available for the function in question, please see the section Function Properties .

The function return value is non zero if there is no property called **name**, or if it does exist, it is not of type Real.

A function return value of zero indicates the value was successfully returned.

ID = 2738

Get_property(Function_Property_Collection collection,Text name,Text &txt_val)

Name

Integer Get_property(Function_Property_Collection collection,Text name,Text &txt_val)

Description

From the Function Property Collection **collection**, get the value of the Text property called **name** and return it in **txt_val**.

For more information on which properties are available for the function in question, please see the section Function Properties .

The function return value is non zero if there is no property called **name**, or if it does exist, it is not of type Text.

A function return value of zero indicates the value was successfully returned.

ID = 2739

Get_property(Function_Property_Collection collection,Text name,Tin &tin)

Name

Integer Get_property(Function_Property_Collection collection,Text name,Tin &tin)

Description

From the Function Property Collection **collection**, get the Tin from the Tin property called **name** and return it in **tin**.

For more information on which properties are available for the function in question, please see the section Function Properties .

The function return value is non zero if there is no property called **name**, or if it does exist, it is not of type Tin.

A function return value of zero indicates the value was successfully returned.

ID = 2740

Get_property(Function_Property_Collection collection,Text name,Element &element)

Name

Integer Get_property(Function_Property_Collection collection,Text name,Element &element)

Description

From the Function Property Collection **collection**, get the Element from the Element property called **name** and return it in **element**.

For more information on which properties are available for the function in question, please see the section Function Properties .

The function return value is non zero if there is no property called **name**, or if it does exist, it is not of type Element.

A function return value of zero indicates the value was successfully returned.

ID = 2741

Get_property(Function_Property_Collection collection,Text name,Model &model)

Name

Integer Get_property(Function_Property_Collection collection,Text name,Model &model)

Description

From the Function Property Collection **collection**, get the Model from the Tin property called **name** and return it in **model**.

For more information on which properties are available for the function in question, please see the section Function Properties .

The function return value is non zero if there is no property called **name**, or if it does exist, it is not of type Model.

A function return value of zero indicates the value was successfully returned.

ID = 2742

Get_property_colour(Function_Property_Collection collection,Text name,Text &colour_name)

Name

Integer Get_property_colour(Function_Property_Collection collection,Text name,Text &colour_name)

Description

From the Function Property Collection **collection**, get the Colour from the Colour property called **name** and return the name of the colour in **colour_name**.

For more information on which properties are available for the function in question, please see the section Function Properties .

The function return value is non zero if there is no property called **name**, or if it does exist, it is not of type Colour.

A function return value of zero indicates the value was successfully returned.

ID = 2743

Create_apply_many_function(Text function_name,Function_Property_Collection properties,Apply_Many_Function &function,Text &msg)

Name

Integer Create_apply_many_function(Text function_name,Function_Property_Collection properties,Apply_Many_Function &function,Text &msg)

Description

Create an Apply Many function with the function name **function_name** using the values supplied in the Function_Property_Collection **properties**.

For more information on which properties are available, please see Apply Many Function Properties .

Any errors such as missing properties, or properties of an incorrect type, will be reported in the Text **msg**.

A non zero function return value indicates that there was a problem creating the Apply Many function.

A function return value of zero indicates the Apply Many was successfully created.

ID = 2734

Set_apply_many_function_properties(Apply_Many_Function function,Function_Property_Collection properties,Text &msg)

Name

Integer Set_apply_many_function_properties(Apply_Many_Function function,

Function_Property_Collection properties,Text &msg)

Description

For the Apply_Many_Function **function**, set the values of **function** to be those in the Function_Property_Collection **properties**.

For more information on which properties are available, please see Apply Many Function Properties .

Any errors such as missing properties, or properties of an incorrect type, will be reported in the Text **msg**.

A non zero function return value indicates that there was a problem creating the Apply Many function.

A function return value of zero indicates the Apply Many was successfully created.

ID = 2735

Get_apply_many_function_properties(Apply_Many_Function function,Function_Property_Collection &properties)

Name

Integer Get_apply_many_function_properties(Apply_Many_Function function,Function_Property_Collection &properties)

Description

Load the values of the Function_Property_Collection **properties** from the Apply Many Function **function**.

For more information on which properties are available, please see Apply Many Function Properties .

A function return value of zero indicates the get was successful.

ID = 2736

Get_apply_many_function(Text name, Apply_Many_Function &function)

Name

Integer Get_apply_many_function(Text name, Apply_Many_Function &function)

Description

Get and existing 12d Model Apply Many Function with the name **name** and create an Apply_Many_Function with the values from the existing 12d MOdel Apply Many Function.

A non zero function return value indicates that there was no 12d Model Apply Many Function with the name **name**, or thee was a problem creating the Apply_Many_Function.

A function return value of zero indicates the creation of the Apply_Many_Function was successful.

ID = 2748

Function Properties

Apply Many Function Properties

Name	Type	Description
tin	Tin / Text	The tin to be used by the apply many
Mtf	Text	The mtf used by the apply many
Separation	Real	The separation between sections
start_chainage	Real	The optional start chainage for the apply many
end_chainage	Real	The optional end chainage for the apply many
left_prefix	Text	The optional left prefix for template names
right_prefix	Text	The optional right prefix for template names
Reference	Element	The centreline / reference string to run the apply many down
Hinge	Element	The optional hinge string
report_file	Text	The optional report file
road_surface_strings	Model/Text	The road strings model to be created by the apply many
road_surface_sections	Model/Text	The road sections model to be created by the apply many
road_surface_colour	Text	The name of the colour for the road surface strings and sections
boxing_strings_N	Model/Text	The optional model or name of a model for boxing strings for layer N (1 to 8)
boxing_sections_N	Model/ Text	The optional model or name of a model for boxing sections for layer N (1 to 8)
boxing_colour_N	Text	The optional name of the colour for the strings created for boxing layer N (1 to 8)
difference_sections	Model/Text	The optional model or name of a model for difference sections
difference_colour	Text	The name of the colour for difference sections
polygons_model	Model/Text	The optional model or name of a model for apply many polygons
road_boundary_model	Model/Text	The optional model or name of a model for the road boundary
create_arcs	Integer	What type of arcs to create 0 - no arcs 1 - alignments 2 - polylines 3 - super strings
chord_arc_tolerance	Real	The chord arc tolerance value
volume_correction	Integer	Whether or not to perform volume correction (0 or 1)
partial_interfaces	Integer	Whether or not to create partial interfaces (0 or 1)
sections_as_4d	Integer	Whether or not to create sections as 4d strings (0 or 1)
copy_hinge	Integer	Whether or not to copy the hinge string (0 or 1)

use_stripping	Integer	Whether or not to use stripping (0 or 1)
show_stripping_volumes	Integer	Whether or not to show detailed stripping volumes (0 or 1)
calculate_natural_surface_to_design_volumes	Integer	Whether or not to calculate natural surface to design volumes (0 or 1)
calculate_road_to_subgrade_volume	Integer	Whether or not to calculate road to subgrade volumes (0 or 1)
calculate_inter_boxing_layer_volumes	Integer	Whether or not to calculate inter boxing layer volumes (0 or 1)
map_file	Text	The optional name of a map file to create
create_road_tin	Integer	Whether or not to create a tin (0 or 1)
road_tin	Tin/Text	The tin or the name of the tin to create
road_tin_colour	Text	The name of the colour for the created tin
road_tin_model	Model/Text	The model or the name of the model to create the tin in
create_depth_range_polygons	Integer	Whether or not to create depth range polygons (0 or 1)
depth_range_file	Text	The name of the depth range file to use when creating depth range polygons
depth_range_polygons_model	Model/Text	The model or name of the model to create depth range polygons in
road_tin_number_extra_models	Integer	The optional number of extra models for the road tin
road_tin_extra_model_N	Model/Text	The model or name of the Nth model to be used as an extra model for the road tin
calculate_sight_distance	Integer	Whether or not to calculate sight distances (0 or 1)
sight_distance_min	Real	The minimum sight distance
sight_distance_max	Real	The maximum sight distance
sight_distance_eye_height	Real	The eye height for the sight distance calcs
sight_distance_eye_offset	Real	The eye offset for the sight distance calcs
sight_distance_target_height	Real	The target height for the sight distance calcs
sight_distance_target_offset	Real	The target offset for the sight distance calcs
sight_distance_calc_interval	Real	The calc interval for the sight distance calcs
sight_distance_trial_interval	Real	The trial interval for the sight distance calcs
sight_distance_report	Text	The optional report for the sight distance calc
create_separation_barrier_lines	Integer	Whether or not to create separation and barrier lines (0 or 1)
barrier_distance	Real	The barrier distance
min_barrier_road_length	Real	The min barrier road length
min_barrier_line_length	Real	The min barrier line length
min_barrier_between	Real	The min distance between barriers
filter_cross_sections	Integer	Whether or not to filter cross sections (0 or 1)
filter_sections_model	Model/Text	The model or name of model for filtered cross sections
filter_sections_colour	Text	The name of the colour for filtered cross sections

filter_sections_interval	Real	The interval at which to filter cross sections
filter_sections_tolerance	Real	The culling tolerance for filtering cross sections
filter_sections_include_start	Integer	Whether or not to include the start section (0 or 1)
filter_sections_include_end	Integer	Whether or not to include the end section (0 or 1)
filter_sections_include_equalities	Integer	Whether or not to include equalities (0 or 1)
filter_sections_include_h_tangent	Integer	Whether or not to include horizontal tangent sections (0 or 1)
filter_sections_include_v_tangent	Integer	Whether or not to include vertical tangent sections (0 or 1)
filter_sections_include_crest_sag	Integer	Whether or not to include crest/sag sections (0 or 1)
filter_sections_spc_file	Text	The optional special chainages file for filtering cross sections
generate_long_section_plot	Integer	Whether or not to generate a long section plot (0 or 1)
long_section_ppf	Text	The name of the ppf for the long section plot
long_section_plotter_type	Text	The name of the plotter to plot a long section with
long_section_plot_stem	Text	The stem for the long section plot
long_section_plot_clean	Integer	Whether or not to clean the long section plot model first (0 or 1)
generate_cross_section_plot	Integer	Whether or not to generate a cross section plot (0 or 1)
cross_section_ppf	Text	The name of the ppf for the cross section plot
cross_section_plotter_type	Text	The name of the plotter to plot a cross section with
cross_section_plot_stem	Text	The stem for the cross section plot
cross_section_plot_clean	Integer	Whether or not to clean the cross section plot model first (0 or 1)
create_tadpoles	Integer	Whether or not to create tadpoles (0 or 1)
tadpole_model	Model/Text	The model or name of model for tadpoles
tadpole_interval	Real	The interval at which to create tadpoles
tadpole_search_width	Real	The search width for creating tadpoles
tadpole_search_side	Integer	The side on which to create tadpoles 0 - Left and Right 1 - Left 2 - Right
tadpole_count	Integer	The number of tadpole types to be created
tadpole_N_string_1_name	Text	The name of string 1 for the Nth tadpole entry
tadpole_N_string_2_name	Text	The name of string 2 for the Nth tadpole entry
tadpole_N_start_ch	Real	The start chainage for the Nth tadpole entry (optional)

tadpole_N_end_ch	Real	The end chainage for the Nth tadpole entry (optional)
tadpole_N_symbol_1_name	Text	The name of the first tadpole symbol for the Nth tadpole entry
tadpole_N_symbol_1_colour	Text	The name of the colour of the first tadpole symbol for the Nth tadpole entry
tadpole_N_symbol_1_size	Real	The size of the first tadpole symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_1_rotation	Real	The rotation of the first tadpole symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_1_offset_x	Real	The x offset of the first tadpole symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_1_offset_y	Real	The y ofset of the first tadpole symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_1_percent	Real	The percentage modifier for the first symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_2_name	Text	The name of the second tadpole symbol for the Nth tadpole entry
tadpole_N_symbol_2_colour	Text	The name of the colour of the second tadpole symbol for the Nth tadpole entry
tadpole_N_symbol_2_size	Real	The size of the second tadpole symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_2_rotation	Real	The rotation of the second tadpole symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_2_offset_x	Real	The x offset of the second tadpole symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_2_offset_y	Real	The y offset of the second tadpole symbol for the Nth tadpole entry (optional)
tadpole_N_symbol_2_percent	Real	The percentage modifier for the second symbol for the Nth tadpole entry (optional)

Plot Parameters

12d Model plot parameters control the look of the different plots that *12d Model* can generate.

The Plot_Parameter_File is a *12d Model* Variable that can contain plot parameters and the plot parameter values for a given plot type.

Plot_Parameter_File Types

The valid Plot_Parameter_File types are:

section_x_plot
section_long_plot
melb_water_sewer_long_plot
pipeline_long_plot
drainage_long_plot
drainage_plan_plot
plot_frame_plot
rainfall_methods
design_parameters

Each type of plot has its own set of valid plot parameters.

When a Plot_Parameter_File, say *ppf,* is first defined, it starts as an empty structure until it has its type defined using the *Create_XX_parameter* calls. The *ppf* then knows what plot parameters are valid for that type of plot.

The Plot_Parameter_File *ppf* is then loaded with particular plot parameters and their values by making *Set_Parameter* calls and/or reading in data from a plot parameter file stored already disk (*Read_Parameter_File*).

When all the required plot parameters have been set, the Plot_Parameter_File *ppf* can be used to create a plot (*Plot_parameter_file*).

The Plot_Parameter_File *ppf* can also be written out as a disk file so that it can be used in the future (*Write_parameter_file*).

Note: note all the available parameters for a particular plot type need to be set for a Plot_Parameter_File. For most plot parameters, there is a default value used for plotting and that is used if the parameter is not given a value by a *Set_Parameter* call.

Create_parameter_file(Plot_Parameter_File ppf,Text ppf_type)

Name

Integer Create_parameter_file(Plot_Parameter_File ppf,Text ppf_type)

Description

Set the Plot_Parameter_File *ppf* to be of type *ppf_type* and clear out any information already contained in *ppf*. For the valid types, see Plot_Parameter_File Types .

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2447

Create_section_long_plot_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_section_long_plot_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type section_long_plot, and clear out any information already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2448

Create_section_x_plot_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_section_x_plot_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type section_x_plot, and clear out any information already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2449

Create_melb_water_sewer_long_plot_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_melb_water_sewer_long_plot_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type melb_water_sewer_long_plot, and clear out any information already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2450

Create_pipeline_long_plot_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_pipeline_long_plot_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type pipeline_long_plot, and clear out any information already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2451

Create_drainage_long_plot_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_drainage_long_plot_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type drainage_long_plot, and clear out any information already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2452

Create_drainage_plan_plot_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_drainage_plan_plot_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type drainage_plan_plot, and clear out any information already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2453

Create_plot_frame_plot_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_plot_frame_plot_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type plot_frame_plot, and clear out any information already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2454

Create_rainfall_methods_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_rainfall_methods_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type rainfall_methods, and clear out any information already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2455

Create_design_parameters_parameter_file(Plot_Parameter_File ppf)

Name

Integer Create_design_parameters_parameter_file(Plot_Parameter_File ppf)

Description

Set the Plot_Parameter_File *ppf* to be of type design_parameters, and clear out any information

already contained in *ppf*.

Hence if *ppf* already contained plot information, then all that information will be lost.

A function return value of zero indicates the type is successfully set.

ID = 2456

Read_parameter_file(Plot_Parameter_File ppf,Text filename,Integer expand_includes)

Name

Integer Read_parameter_file(Plot_Parameter_File ppf,Text filename,Integer expand_includes)

Description

Reads from disk a binary plot parameter file of file name *filename* and load the data into the Plot_Parameter_File *ppf*. The type of the Plot_Parameter_File is determined by the file extension of filename.

If *expand_includes* is no-zero then any Includes listed in filename will be read in.

Any information that is already in *ppf* is cleared before loading the data from *filename*.

A function return value of zero indicates the file was successfully read and loaded into *ppf*.

ID = 2457

Write_parameter_file(Plot_Parameter_File ppf,Text filename)

Name

Integer Write_parameter_file(Plot_Parameter_File ppf,Text filename)

Description

Write out to a file on disk, the information in the Plot_Parameter_File *ppf*.

The name of the disk file is *filename*, plus the appropriate extension given by the type of *ppf* (*see* Plot_Parameter_File Types)

A function return value of zero indicates the file was successfully written.

ID = 2458

Set_parameter(Plot_Parameter_File ppf,Text parameter_name, Element parameter_value)

Name

Integer Set_parameter(Plot_Parameter_File ppf,Text parameter_name,Element parameter_value)

Description

Sets the value of the plot parameter *parameter_name* in the Plot_Parameter_File *ppf* to be the Element *parameter_value*.

For example, setting the plot parameter *string_to_plot* to be a selected string. *Aside* - in the plot parameter file written to the disk, an element is stored with three things - the string name, the string id and the model id of the model containing the element.

If the plot parameter does not require an Element, then a non-zero return function return value is returned.

A function return value of zero indicates the parameter value is successfully set.

ID = 2641

Get_parameter(Plot_Parameter_File ppf,Text parameter_name,Element &parameter_value)

Name

Integer Get_parameter(Plot_Parameter_File ppf,Text parameter_name,Element &parameter_value)

Description

Get the value for the plot parameter *parameter_name* in the Plot_Parameter_File *ppf* and return it as the Element *parameter_value*.

If the value for the plot parameter is not of type Element, then a non-zero return function return value is returned.

A function return value of zero indicates the parameter value is successfully found.

ID = 2642

Set_parameter(Plot_Parameter_File ppf,Text parameter_name,Text parameter_value)

Name

Integer Set_parameter(Plot_Parameter_File ppf,Text parameter_name,Text parameter_value)

Description

Sets the value of the plot parameter *parameter_name* in the Plot_Parameter_File *ppf* to be the Text *parameter_value*.

For example, setting the plot parameter *box_titles_x* to have the value 24.5

Note - even though a plot parameter file may be used as a real number or an integer, it is stored in the Plot_Parameter_File as a Text.

A function return value of zero indicates the parameter value is successfully set.

ID = 2459

Get_parameter(Plot_Parameter_File ppf,Text parameter_name,Text &parameter_value)

Name

Integer Get_parameter(Plot_Parameter_File ppf,Text parameter_name,Text &parameter_value)

Description

so get back as text and you need to decode it.

Get the value for the plot parameter *parameter_name* in the Plot_Parameter_File *ppf* and return it as the Text *parameter_value*.

Note - if the parameter value is to be used as say an Integer, then the returned Text parameter_value will need to be decoded.

If the value for the plot parameter is not of type Text, then a non-zero return function return value is returned.

A function return value of zero indicates the parameter value is successfully found.

ID = 2460

Parameter_exists(Plot_Parameter_File ppf,Text parameter_name)

Name

Integer Parameter_exists(Plot_Parameter_File ppf,Text parameter_name)

Description

Check to see if a plot parameter of name *parameter_name* exists in the Plot_Parameter_File *ppf*.

returns no-zero if exists

A non-zero function return value indicates that an plot parameter exists.

Warning this is the opposite of most 12dPL function return values.

ID = 2461

Remove_parameter(Plot_Parameter_File ppf,Text parameter_name)

Name

Integer Remove_parameter(Plot_Parameter_File ppf,Text parameter_name)

Description

Remove the plot parameter of name *parameter_name* and its value from the Plot_Parameter_File *ppf*.

Note - the Plot_Parameter_File *ppf* does not necessarily contain values for all the possible plot parameters that are available for a given Plot_Parameter_File. Many parameters can have default values which are used if the plot parameter is not set.

A function return value of zero indicates the parameter was successfully removed.

ID = 2462

Plot_parameter_file(Plot_Parameter_File ppf)

Name

Integer Plot_parameter_file(Plot_Parameter_File ppf)

Description

Plot the Plot_Parameter_File *ppf*.

Note - *ppf* needs to contain all the appropriate information on where the plot is plotted to.

A function return value of zero indicates the plot was successfully created

ID = 2463

Plot_parameter_file(Text file)

Name

Integer Plot_parameter_file(Text file)

Description

Plot the plot parameter file in the binary plot parameter disk file **name**.

Note - the file needs to contain all the appropriate information on where the plot is plotted to.

A function return value of zero indicates the plot was successfully created.

ID = 2464

Plot_ppf_file(Text name)

Name

Integer Plot_ppf_file(Text name)

Description

Plot the plot parameter file in the ascii plot parameter disk file **name**.

Note - the file needs to contain all the appropriate information on where the plot is plotted to.

A function return value of zero indicates the plot was successfully created.

ID = 652

Undos

12d Model has an Undo system which allows operations to be undone (option *Edit =>Undo* or using <Ctrl>-Z) and the Undo macro calls gives access to the 12d Model Undo system.

For an operation to be undone, enough information must be stored to allow for the operation to be reversed.

For example, if an Element **elt** is created, then the undo of this operation it to delete **elt**.

Or if an Element **original** is modified to create a new Element **changed**, then the original element and the new element both need to be recorded so that the undo operation can replace the original Element.

To correctly create items for undos, 12dPL has an **Undo** structure and calls to create the Undo structure with the appropriate information for an undo. Creating the Undo also automatically adds it to the 12d Model Undo system.

Creating an undo for even a simple operation, may need a number of pieces of information stored.

For example, if you were splitting a string into two pieces and only leaving the two pieces, for an undo to work, you would need to have a copy of the original string that is being split (since the macro would delete it after is did the split), plus information about the two strings that are created by the split. This is because the undo must find and delete the two strings created by the split, and then bring the original string back.

So the calls needed would be

```
Undo  a = Add_undo_delete("deleted string",original_string,1);
Undo  b = Add_undo_add("split 1",split_1);
Undo  c = Add_undo_add("split 2",split_2);
```

where original_string is the string what is split and split_1 and split_2 are the two pieces that are created by the split (*See* Functions to Create Undos *for the documentation on each call).*

However, each call automatically adds the operation to the 12d Model Undo system so making the three calls actually places three items on the 12d Model Undo system with the text "Deleted string", "split 1" and "split 2".

So as it stands, to make the undo happen would need three *Edit =>Undo's, or three <ctrl>-z's.*

To wrap the three items into one item on the 12d Model Undo system, you need to use a 12dPL Undo_List.

Basically you just add the three items that are to be done as one 12d Model Undo onto a Undo_List, add the three Undos to the Undo_list, and then add the Undo_List to the 12d Model Undo system:

```
Undo_List  ul;
Append (a,ul);
Append (b,ul);
Append (c,ul);
Add_undo_list ("split",ul);
```

Note: Add_undo_list adds the Undo_List with three items to the 12d Model Undo system and gives it the name "split". At the same time, it removes the three separate Undos a, b, c from the 12d Model Undo system so only the item called "split" is left on the 12d Model Undo system.

Important Note: Leaving the three Undo's a, b, c without combining them is a great way of

debugging your creation of an 12d Model Undo. You will see them as three separate items and they can be undone one at a time to see what is going on.

For information on the Undo function calls:

See Functions to Create Undos
See Functions for a 12dPL Undo_List

Functions to Create Undos

Add_undo_add(Text name,Element elt)

Name

Undo Add_undo_add(Text name,Element elt)

Description

Create an Undo from the Element **elt** and give it the name **name.**

The Undo is automatically added to the 12d Model Undo system.

Return the created Undo as the function return value.

This is telling the 12d Model Undo system that a new element has been created in *12d Model.*

Note: **name** is the text that appears when the Undo is displayed in the *12d Model Undo List*.

ID = 1563

Add_undo_add(Text name,Dynamic_Element de)

Name

Undo Add_undo_add(Text name,Dynamic_Element de)

Description

Create an Undo from the Dynamic_Element **de** and give it the name **name.**

The Undo is automatically added to the 12d Model Undo system.

Return the created Undo as the function return value.

This is telling the Undo system that a list of new element (stored in the Dynamic_Element **de**) has been created in *12d Model.*

Note: **name** is the text that appears when the Undo is displayed in the *12d Model Undo List*.

ID = 1564

Add_undo_change(Text name,Element original,Element changed)

Name

Undo Add_undo_change(Text name,Element original,Element changed)

Description

Create an Undo from a *copy* of the original Element **original** and the modified Element **changed,** and give it the name **name.**

The Undo is automatically added to the 12d Model Undo system.

Return the created Undo called name as the function return value.

The Element **original** should not exist in a Model. The Element **changed** does exist in a Model.

This is telling the Undo system that an Element **original** has been modified to create the Element **changed**. If the Model for **original** is ever needed then the parent structure of **original** can be used to get it.

Note: **name** is the text that appears when the Undo is displayed in the *12d Model Undo List*.

ID = 1565

Add_undo_delete(Text name,Element original,Integer make_copy)

Name

Undo Add_undo_delete(Text name,Element original,Integer make_copy)

Description

If **make_copy** is non zero, create a copy of the Element **original** and transfer the Uid from **original** to the copy.

If **make_copy** is zero, then a reference to **original is use.** Warning - **make_copy** = 0 should never be used because if **original** is then deleted in 12d Model, the Undo list could be corrupted.

The Undo is given the name **name.**

The Undo is automatically added to the 12d Model Undo system.

Return the created Undo called name as the function return value.

This is telling the Undo system that an Element **original** has been deleted.

Note: **name** is the text that appears when the Undo is displayed in the *12d Model Uno List*.

ID = 1566

Add_undo_range(Text name,Integer id1,Integer id2)

Name

Undo Add_undo_range(Text name,Integer id1,Integer id2)

Description

Create an Undo that consists of the id range form 1d1 to id2.

The Undo is given the name **name.**

The Undo is automatically added to the 12d Model Undo system.

Return the created Undo called name as the function return value.

This is telling the Undo system that all the Elements in the id range from Id1 to Id2 have been created.

Note: **name** is the text that appears when the Undo is displayed in the *12d Model Undo List*.

Important note - Id's are no longer used in 12d Model and have been replaced by Uids. This macro has been deprecated (i.e. won't exist) unless the macro is compiled with a special flag. This function has been replaced by *Undo Add_undo_range(Text name,Uid id1,Uid id2).*

ID = 1567

Add_undo_range(Text name,Uid id1,Uid id2)

Name

Undo Add_undo_range(Text name,Uid id1,Uid id2)

Description

Create an Undo that consists of the Uid range form id1 to id2.

The Undo is given the name **name.**

The Undo is automatically added to the 12d Model Undo system.

Return the created Undo called name as the function return value.

This is telling the Undo system that all the Elements in the Uid id range from Id1 to Id2 have been created.

Note: **name** is the text that appears when the Undo is displayed in the *12d Model Undo List*.

ID = 1919

For information on adding/removing Undo's to an internal 12dPL list and how it interacts with the 12d Model Undo system, go to the next section Functions for a 12dPL Undo_List

Functions for a 12dPL Undo_List

Get_number_of_items(Undo_List &undo_list,Integer &count)

Name

Integer Get_number_of_items(Undo_List &undo_list,Integer &count)

Description

Get the number of items in the Undo_List **undo_list** and return the number in **count**.

A function return value of zero indicates the number was successfully returned.

ID = 1557

Get_item(Undo_List &undo_list,Integer n,Undo &undo)

Name

Integer Get_item(Undo_List &undo_list,Integer n,Undo &undo)

Description

Get the **n**'th item from the Undo_List **undo_list** and return the item (which is an Undo) as **undo**.

A function return value of zero indicates the Undo was successfully returned.

ID = 1558

Set_item(Undo_List &undo_list,Integer n,Undo undo)

Name

Integer Set_item(Undo_List &undo_list,Integer n,Undo undo)

Description

Set the **n**'th item in the Undo_List **undo_list** to be the Undo **undo**.

A function return value of zero indicates the Undo was successfully set.

ID = 1559

Append(Undo undo,Undo_List &undo_list)

Name

Integer Append(Undo undo,Undo_List &undo_list)

Description

Append the Undo **undo** to the Undo_List **undo_list**.

That is, the Undo **undo** is added to the end of the Undo_List and so the number of items in the Undo_List is increased by one.

A function return value of zero indicates the Undo was successfully appended.

ID = 1560

Append(Undo_List list,Undo_List &to_list)

Name

Integer Append(Undo_List from_list,Undo_List &to_list)

Description

Append the Undo_list **list** to the Undo_List **to_list.**

A function return value of zero indicates the Undo_List was successfully appended.

ID = 1561

Null(Undo_List &undo_list)

Name

Integer Null(Undo_List &undo_list)

Description

Removes and nulls all the Undo's from the Undo_list **undo_list** and sets the number of items in **undo_list** to zero.

That is, all the items on the Undo_List are nulled and the number of items in the Undo_List is set back to zero.

A function return value of zero indicates the Undo_List was successfully nulled.

ID = 1562

Add_undo_list(Text name,Undo_List list)

Name

Undo Add_undo_list(Text name,Undo_List list)

Description

Adds the Undo_List **list** to the 12d Model Undo system and gives it the name **name.**

At the same time, it automatically removes each of the Undo's in **list** from the 12d Model Undo system. So all the items in **list** are removed from the 12d Model Undo system and replaced by the one item called name.

ID = 1568

ODBC Macro Calls

The ODBC (Open Database Connectivity) macro calls allow a macro to interface with external data sources via ODBC. These data sources include any ODBC enabled database or spreadsheets such as Excel. This is particularly useful for custom querying of GIS databases.

Terminology

- A Connection refers to a connection to a known data source.
- A Query refers to an operation against the database (See Query Types for more information)
- A Query Condition is a set of conditions applied against a query to constrain the information being returned.
- A Transaction refers to an atomic, discrete operation that has a known start and end. Any changes to your data source will not apply until the transaction is committed.
- A Parameter refers to a known keyword pair for supplied values, which is important for security purposes

See Connecting to an external data source
See Querying against a data source
See Navigating results with Database_Result
See Insert Query
See Update Query
See Delete Query
See Manual Query
See Query Conditions
See Transactions
See Parameters

Connecting to an external data source

Before running queries, a connection must be made to the database. It is also good practise to close the connection when you are finally finished with it.

Create_ODBC_connection()

Name

Connection Create_ODBC_connection()

Description

Creates an ODBC connection object, which may then by used to connect to a database.

ID = 2501

Connect(Connection connection,Text connection_string,Text user,Text password)

Name

Integer Connect(Connection connection,Text connection_string,Text user,Text password)

Description

This call attempts to connect to an external data source, with a username and password. A connection string must also be supplied. This is data source specific and ODBC driver specific. For more information on connection strings, see the vendor of the data source or data source driver.

This call returns 0 if successful.

ID = 2502

Connect(Connection connection,Text connection_string)

Name

Integer Connect(Connection connection, Text connection_string)

Description

This call attempts to connect to an external data source. A connection string must also be supplied. This is data source specific and ODBC driver specific. For more information on connection strings, see the vendor of the data source or data source driver.

This call returns 0 if successful.

ID = 2503

Close(Connection connection)

Name

Integer Close(Connection connection)

Description

This call determines if there was an error performing an operation against the connection. This call will return 1 if there was an error.

ID = 2504

Has_error(Connection connection)

Name

Integer Has_error(Connection connection)

Description

This call will check if an error has occurred as the result of an operation. Has_error should always be called after any operation. If there is an error, Get_last_error can be used to retrieve the result.

This call will return 0 if there is no error, and 1 if there is.

ID = 2512

Get_last_error(Connection connection,Text &status,Text &message)

Name

Integer Get_last_error(Connection connection,Text &status,Text &message)

Description

This call will get the last error, if there is one, and retrieve the status and message of the error. This call will return 0 if successful.

ID = 2513

Return to ODBC Macro Calls

Querying against a data source

Once connected, you may query the data source in a number of ways. Queries are typically implemented in SQL (the Structured Query Language). To make it easier to use, the macro language provides an interface to building up queries without having to use SQL. There are several types of query building objects.

The query is not run until the appropriate Execute function is called.

- **Select_Query** - Used to retrieve information from the data source
- **Insert_Query** - Used to insert new information into the data source
- **Update_Query** - Used to update existing information in the data source
- **Delete_Query** - Used to delete information from a data source

A **Manual_Query** also exists, if you wish to define the SQL yourself.

Note that a query execution may return as successful even if no data was changed.

Select Query

Select queries are used to retrieve information, with or without constraints, from the data source. Select queries are defined by tables and columns, from which to retrieve results, and optional query conditions to constrain them.

Create_select_query()

Name

Select_Query Create_select_query()

Description

Creates and returns a select query object.

ID = 2528

Add_table(Select_Query query,Text table_name)

Name

Integer Add_table(Select_Query query,Text table_name)

Description

This call adds a table of a given name to the supplied query. The query will look at this table when retrieving data.

This call returns 0 if successful.

ID = 2529

Add_result_column(Select_Query query,Text table,Text column_name)

Name

Integer Add_result_column(Select_Query query,Text table,Text column_name)

Description

This call adds a result column that belongs to a given table to the query. Note that the table must already be added for this to work. The query will retrieve that column from the supplied table when it runs.

The call returns 0 if successful.

ID = 2531

Add_result_column(Select_Query query,Text table,Text column_name,Text return_as)

Name

Integer Add_result_column(Select_Query query,Text table,Text column_name,Text return_as)

Description

This call adds a result column that belongs to a given table to the query. Note that the table must already be added for this to work. The query will retrieve that column from the supplied table when it runs, but in the results it will be called by the name you supply.

The call returns 0 if successful.

ID = 2530

Add_order_by(Select_Query query,Text table_name,Text column_name,Integer sort_ascending)

Name

Integer Add_order_by(Select_Query query,Text table_name,Text column_name,Integer sort_ascending)

Description

This call will instruct the query to order the results for a column in a table. Set sort_ascending to 1 if you wish the results to be sorted in ascending order.

This call returns 0 if successful.

ID = 2533

Set_limit(Select_Query query,Integer start,Integer number_to_retrieve)

Name

Integer Set_limit(Select_Query query,Integer start,Integer number_to_retrieve)

Description

This call will set an upper limit on the number of results to read, as well as defining the start index of the returned results. This is useful when you have many results that you wish to return in discrete sets or pages.

This call returns 0 if successful.

ID = 2534

Add_group_by(Select_Query query,Text table_name,Text column_name)

Name

Integer Add_group_by(Select_Query query,Text table_name,Text column_name)

Description

This call will group results by a given table and column name. This is useful if your data provider allows aggregate functions for your queries.

This call returns 0 if successful.

ID = 2532

Add_condition(Select_Query query,Query_Condition condition)

Name

Integer Add_condition(Select_Query query,Query_Condition condition)

Description

This call will add a query condition to a select query. A query condition will allow you to constrain your results to defined values. See the section Query Conditions on how to create and defined Query Conditions.

This call returns 0 if successful.

ID = 2535

Execute(Connection connection,Select_Query query)

Name

Integer Execute(Connection connection,Select_Query query)

Description

This call will execute a created select query for a scalar value. The return value of the call will be the result of the query.

ID = 2505

Execute(Connection connection,Select_Query query,Database_Result &result)

Name

Integer Execute(Connection connection,Select_Query query,Database_Result &result)

Description

This call will execute a created select query and return a set of results in the result argument. See the section on Navigating results with Database_Result for more information on the **Database_Result** object.

This call will return 0 if successful.

ID = 2506

Return to ODBC Macro Calls

Navigating results with Database_Result

If a select or manual query returns results, they will be stored in a **Database_Result** object. A **Database_Result** may be visualised as a table of rows and columns. The **Database_Result** can be used to access these results in a sequential fashion, in a forward only direction.

Move_next(Database_Result result)

Name

Integer Move_next(Database_Result result)

Description

This call moves a database result to the next row. Depending on your provider, you may need to call this before reading the first row.

This call will return 0 if the **Database_Result** was able to move to the next row.

ID = 2514

Close(Database_Result result)

Name

Integer Close(Database_Result result)

Description

This call will close a database result. This is generally good practise as your data provider may not allow more than one **Database_Result** to exist at one time.

This call will return 0 if successful.

ID = 2515

Get_result_column(Database_Result result,Integer column,Text &res)

Name

Integer Get_result_column(Database_Result result,Integer column,Text &res)

Description

This call will retrieve a text value from a **Database_Result**, at the current index as given by column. The value will be stored in *res*.

This call will return 0 if successful.

ID = 2516

Get_result_column(Database_Result result,Integer column,Integer &res)

Name

Integer Get_result_column(Database_Result result,Integer column,Integer &res)

Description

This call will retrieve an Integer value from a **Database_Result**, at the current index as given by column. The value will be stored in *res*.

This call will return 0 if successful.

ID = 2517

Get_result_column(Database_Result result,Integer column,Real &res)

Name

Integer Get_result_column(Database_Result result,Integer column,Real &res)

Description

This call will retrieve a Real value from a **Database_Result,** at the current index as given by column. The value will be stored in *res*.

This call will return 0 if successful.

ID = 2518

Get_time_result_column(Database_Result result,Integer column,Integer &time)

Name

Integer Get_time_result_column(Database_Result result,Integer column,Integer &time)

Description

This call will retrieve a timestamp, as an Integer value, from a **Database_Result**, at the current index as given by column. The value will be stored in *res*.

This call will return 0 if successful.

ID = 2519

Get_result_column(Database_Result result,Text column,Text &res)

Name

Integer Get_result_column(Database_Result result,Text column,Text &res)

Description

This call will retrieve a text value from a **Database_Result,** from the column named by the argument column. The value will be stored in *res*.

This call will return 0 if successful.

ID = 2520

Get_result_column(Database_Result result,Database_Result result,Text column, Integer &res)

Name

Integer Get_result_column(Database_Result result,Database_Result result,Text column,Integer &res)

Description

This call will retrieve an Integer value from a **Database_Result,** from the column named by the argument column. The value will be stored in *res.*

This call will return 0 if successful.

ID = 2521

Get_result_column(Database_Result result,Text column,Real &res)

Name

Integer Get_result_column(Database_Result result,Text column,Real &res)

Description

This call will retrieve a Real value from a **Database_Result**, from the column named by the argument column. The value will be stored in *res.*

This call will return 0 if successful.

ID = 2522

Get_time_result_column(Database_Result result,Text column,Integer &time)

Name

Integer Get_time_result_column(Database_Result result,Text column,Integer &time)

Description

This call will retrieve a timestamp value, as an Integer, from a **Database_Result,** from the column named by the argument column. The value will be stored in *res*.

This call will return 0 if successful.

ID = 2523

Return to ODBC Macro Calls

Insert Query

An insert query is used to insert new data into a data provider. Typically, this will insert one row of data into one table at a time.

Create_insert_query(Text table)

Name

Insert_Query Create_insert_query(Text table)

Description

This call creates and returns an insert query object. The insert will be applied against the value supplied in table.

ID = 2536

Add_data(Insert_Query query,Text column_name,Integer value)

Name

Integer Add_data(Insert_Query query,Text column_name,Integer value)

Description

This call will add Integer data to be inserted to a created **Insert_Query** when it is executed. The data will be inserted into the column named by the **column_name** argument.

This call returns 0 if successful.

ID = 2537

Add_data(Insert_Query query,Text column_name,Text value)

Name

Integer Add_data(Insert_Query query,Text column_name,Text value)

Description

This call will add Text data to be inserted to a created **Insert_Query** when it is executed. The data will be inserted into the column named by the **column_name** argument.

This call returns 0 if successful.

ID = 2538

Add_data(Insert_Query query,Text column_name,Real value)

Name

Integer Add_data(Insert_Query query,Text column_name,Real value)

Description

This call will add Real data to be inserted to a created **Insert_Query** when it is executed. The data will be inserted into the column named by the **column_name** argument.

This call returns 0 if successful.

ID = 2539

Add_time_data(Insert_Query query,Text column_name,Integer time)

Name

Integer Add_time_data(Insert_Query query,Text column_name,Integer time)

Description

This call will add timestamp data, stored as an Integer value, to be inserted to a created **Insert_Query** when it is executed. The data will be inserted into the column named by the **column_name** argument.

This call returns 0 if successful.

ID = 2540

Execute(Connection connection,Insert_Query query)

Name

Integer Execute(Connection connection,Insert_Query query)

Description

This call will execute a created **Insert_Query** against the data provider to insert some new data.

This call will return 0 if successful.

ID = 2507

Return to ODBC Macro Calls

Update Query

An update query is used to update existing data in a table in a data provider. One or more rows may be updated by using query conditions to constrain which rows the update should be applied against.

Create_update_query(Text table)

Name

Update_Query Create_update_query(Text table)

Description

This call creates and returns an **Update_Query**. The update query will be applied against the

table given by the table argument.

ID = 2541

Add_data(Update_Query query,Text column_name,Integer value)

Name

Integer Add_data(Update_Query query,Text column_name,Integer value)

Description

This call will add Integer data for a column update, when the **Update_Query** is executed. The data will be updated for the column named by the **column_name** argument.

This call returns 0 if successful.

ID = 2542

Add_data(Update_Query query,Text column_name,Text value)

Name

Integer Add_data(Update_Query query,Text column_name,Text value)

Description

This call will add Text data for a column update, when the **Update_Query** is executed. The data will be updated for the column named by the **column_name** argument.

This call returns 0 if successful.

ID = 2543

Add_data(Update_Query query,Text column_name,Real value)

Name

Integer Add_data(Update_Query query,Text column_name,Real value)

Description

This call will add Real data for a column update, when the **Update_Query** is executed. The data will be updated for the column named by the **column_name** argument.

This call returns 0 if successful.

ID = 2544

Add_time_data(Update_Query query,Text column_name,Integer time)

Name

Integer Add_time_data(Update_Query query,Text column_name,Integer time)

Description

This call will add timestamp data, stored as an Integer value, for a column update, when the **Update_Query** is executed. The data will be updated for the column named by the **column_name** argument.

This call returns 0 if successful.

ID = 2545

Add_condition(Update_Query query,Query_Condition condition)

Name

Integer Add_condition(Update_Query query,Query_Condition condition)

Description

This call will add a created **Query_Condition** to an update query. Using a **Query_Condition** enables the operation to be constrained to a number of rows, rather than applying to an entire table.

This call will return 0 if successful.

ID = 2546

Execute(Connection connection,Update_Query query)

Name

Integer Execute(Connection connection,Update_Query query)

Description

This call will execute a created **Update_Query** against the data provider to update existing data.

This call will return 0 if successful.

ID = 2508

Return to ODBC Macro Calls

Delete Query

A delete query will delete data from a table in a data provider. It should always be constrained using a **Query Condition**, or you may delete all data from a table.

Create_delete_query(Text table)

Name

Delete_Query Create_delete_query(Text table)

Description

This call will create and return a **Delete_Query.** When it is executed, it will delete data from the table named by the table argument.

ID = 2547

Add_condition(Delete_Query query,Query_Condition condition)

Name

Integer Add_condition(Delete_Query query,Query_Condition condition)

Description

This call will add a **Query_Condition** to a **Delete_Query**. Adding a **Query_Condition** will allow you to constrain which rows of data are deleted from the table.

This call will return 0 if successful.

ID = 2548

Execute(Connection connection,Delete_Query query)

Name

Integer Execute(Connection connection,Delete_Query query)

Description

This call will execute a created **Delete_Query** against the data provider to delete existing data.

This call will return 0 if successful.

ID = 2509

Return to ODBC Macro Calls

Manual Query

Using a manual query gives you direct access to the underlying SQL used by most data providers. If you are familiar with SQL, it may be faster for you to use this method. This also gives you access to Parameters, for secure and sanitized inputs. See the section on **Parameters** for more information.

Create_manual_query(Text query_text)

Name

Manual_Query Create_manual_query(Text query_text)

Description

This call will create a new **Manual_Query**. The SQL for the query must be supplied in the **query_text** argument.

ID = 2549

Get_parameters(Manual_Query query,Parameter_Collection parameters)

Name

Integer Get_parameters(Manual_Query query,Parameter_Collection parameters)

Description

This call will retrieve the set of Parameters that a Manual Query uses. Parameters are not required but provide greater security when using user input. See the section on **Parameters** for more details.

This call will return 0 if successful.

ID = 2550

Execute(Connection connection,Manual_Query query)

Name

Integer Execute(Connection connection,Manual_Query query)

Description

This call will execute a created **Manual_Query** against the data provider to perform a custom operation.

This call will return 0 if successful.

ID = 2510

Execute(Connection connection,Manual_Query query,Database_Result &result)

Name

Integer Execute(Connection connection,Manual_Query query,Database_Result &result)

Description

This call will execute a created **Manual_Query** against the data provider to perform a custom operation. If the Manual Query returns results, they will be stored in the result argument.

This call will return 0 if successful.

ID = 2511

Return to ODBC Macro Calls

Query Conditions

A query condition constrains the results of a select, update or delete query. They are generally optimised and much more efficient that attempting to cull down a large result set on your own, as the operation is performed by the data provider. For those familiar with SQL, a Query Condition helps build up the 'WHERE' clause in an SQL statement.

One or more query conditions can be used to constrain a query.

The following Query Conditions are available:

- **A value condition** - Constrains by checking if a column value matches a constant, user defined value
- **Column match condition** - Performs an 'explicit join'. If you are retrieving results from more than one table, this can be used to determine which rows from each table are related to one another. Typically you would match id columns from each table.
- **Value in list condition** - Checks if a column value is inside a list of values
- **Value in sub query** - Checks if a column value is inside the result of another select query
- **Manual condition** - A manual condition, defined by SQL. This gives greater flexibility and provides access to the Parameter functions, for security and sanitization of inputs.

Value and Column match conditions allow various operators to be used.

These operators are defined below:

```
Match_Equal = 0
Match_Greater_Than = 1
Match_Less_Than = 2
Match_Greater_Than_Equal = 3
Match_Less_Than_Equal = 4
Match_Not_Equal = 5
Match_Like = 6
Match_Not_Like = 7
```

Create_value_condition(Text table_name,Text column_name,Integer operator,Text value)

Name

Query_Condition Create_value_condition(Text table_name,Text column_name,Integer operator,Text value)

Description

This call creates and returns a Value Condition Query Condition for a given table, column, operation and Text value. See the list of operators for available values of operator.

When executed, the data provider will check that the value in column **colum_name** inside table **table_name** matches (as appropriate for the given operator) against the supplied value.

ID = 2555

Create_value_condition(Text table_name,Text column_name,Integer operator, Integer value)

Name

Query_Condition Create_value_condition(Text table_name,Text column_name,Integer operator,Integer value)

Description

This call creates and returns a Value Condition Query Condition for a given table, column, operation and Integer value. See the list of operators for available values of operator.

When executed, the data provider will check that the value in column **colum_name** inside table **table_name** matches (as appropriate for the given operator) against the supplied value.

ID = 2556

Create_value_condition(Text table_name,Text column_name,Integer operator, Real value)

Name

Query_Condition Create_value_condition(Text table_name,Text column_name,Integer operator,Real value)

Description

This call creates and returns a Value Condition Query Condition for a given table, column, operation and Real value. See the list of operators for available values of operator.

When executed, the data provider will check that the value in column **colum_name** inside table **table_name** matches (as appropriate for the given operator) against the supplied value.

ID = 2557

Create_time_value_condition(Text table_name,Text column_name,Integer operator,Integer value)

Name

Query_Condition Create_time_value_condition(Text table_name,Text column_name,Integer operator,Integer value)

Description

This call creates and returns a Value Condition Query Condition for a given table, column, operation and timestamp value, as defined by an Integer. See the list of operators for available values of operator.

When executed, the data provider will check that the value in column **colum_name** inside table **table_name** matches (as appropriate for the given operator) against the supplied value.

ID = 2558

Create_column_match_condition(Text left_table,Text left_column,Integer operator,Text right_table,Text right_column)

Name

Query_Condition Create_column_match_condition(Text left_table,Text left_column,Integer operator,Text right_table,Text right_column)

Description

This call will create and return a Column Match Query Condition to match two columns in two tables against each other, using a supplied operator.

When executed, the data provider will check that the left_column in table **left_table** matches (as appropriate for the given operator) against the **right_column** in table **right_table.**

ID = 2559

Create_value_in_sub_query_condition(Text table_name,Text column_name, Integer not_in,Select_Query sub_query)

Name

Query_Condition Create_value_in_sub_query_condition(Text table_name,Text column_name,Integer not_in,Select_Query sub_query)

Description

This call will create and return a Value In Sub Query **Query_Condition**, to match the value of a column against the results of another query.

When executed, the data provider will check that the value in column c**olumn_name** in table **table_name** is or is not inside (as defined by **not_in**) the results of the Select Query, **sub_query**.

ID = 2560

Create_value_in_list_condition(Text table_name,Text column_name,Integer not_in,Dynamic_Integer values)

Name

Query_Condition Create_value_in_list_condition(Text table_name,Text column_name,Integer not_in,Dynamic_Integer values)

Description

This call will create and return a Value In List **Query_Condition**, to see if the value of a column is in a list of integers.

When executed, the data provider will check that the value in column **column_name** in table **table_name** is or is not inside (as defined by **not_in**) the list defined by values.

ID = 2561

Create_value_in_list_condition(Text table_name,Text column_name,Integer not_in,Dynamic_Text values)

Name

Query_Condition Create_value_in_list_condition(Text table_name,Text column_name,Integer not_in,Dynamic_Text values)

Description

This call will create and return a Value In List **Query_Condition**, to see if the value of a column is in a list of Text values.

When executed, the data provider will check that the value in column **column_name** in table **table_name** is or is not inside (as defined by **not_in**) the list defined by values.

ID = 2562

Create_value_in_list_condition(Text table_name,Text column_name,Integer not_in,Dynamic_Real values)

Name

Query_Condition Create_value_in_list_condition(Text table_name,Text column_name,Integer not_in,Dynamic_Real values)

Description

This call will create and return a Value In List **Query_Condition**, to see if the value of a column is in a list of Real values.

When executed, the data provider will check that the value in column **column_name** in table **table_name** is or is not inside (as defined by **not_in**) the list defined by values.

ID = 2563

Create_manual_condition(Text sql)

Name

Manual_Condition Create_manual_condition(Text sql)

Description

This call will create a Manual **Query_Condition**. The operation of the manual condition is totally defined by the SQL fragment defined in argument sql.

ID = 2564

Add_table(Manual_Condition manual,Text table)

Name

Integer Add_table(Manual_Condition manual,Text table)

Description

This call will add a table to be used by a Manual Condition. This is required when using Parameters.

This call will return 0 if successful.

ID = 2565

Get_parameters(Manual_Condition manual,Parameter_Collection &param)

Name

Integer Get_parameters(Manual_Condition manual,Parameter_Collection &param)

Description

This call will add a table to be used by a Manual Condition. This is required when using Parameters. See the section on Parameters for more information.

This call will return 0 if successful.

ID = 2566

Return to ODBC Macro Calls

Transactions

A transaction is an atomic operation. While a transaction is running against a connection, a series of queries can be made and executed. Using a transaction, the final result (updates, deletes, inserts) will not actually be applied until the transaction is committed. This gives the user the opportunity to rollback the changes a transaction has made if they are no longer required.

To use a transaction, create it using **Create_Transaction.**

You must then call **Begin_Transaction**.

Create and execute all your queries.

Finally, choose to either commit it (**Commit_transaction**) or roll it back (**Rollback_transaction**)

Create_transaction(Connection connection)

Name

Transaction Create_transaction(Connection connection)

Description

This call creates and returns a transaction object for a given Connection.

ID = 2524

Begin_transaction(Transaction transaction)

Name

Integer Begin_transaction(Transaction transaction)

Description

This call begins a new transaction. It will return 0 if successful.

ID = 2525

Commit_transaction(Transaction transaction)

Name

Integer Commit_transaction(Transaction transaction)

Description

This call will commit the operations performed inside a transaction to the data provider. The call will return 0 if successful.

ID = 2526

Rollback_transaction(Transaction transaction)

Name

Integer Rollback_transaction(Transaction transaction)

Description

This call will cancel or rollback the operations performed inside a transaction from the data provider. The call will return 0 if successful.

ID = 2527

Return to ODBC Macro Calls

Parameters

Parameters can be used for extra security. When you are working with user input to your queries, you may wish to consider using parameters to 'sanitize' them. If you are working with untrusted users, users may be able to use the SQL to perform malicious queries against your data provider.

To prevent this from happening, it is generally recommended that you use Parameters.

When you are using parameters, instead of specifying column names in your Manual Query or Manual Query Condition, simply use a **?** instead.

You should then add your parameters for those columns in the same order.

To start, you must retrieve the **Parameter_Collection** using the appropriate **Get_Parameters** function for either a **Manual_Query** or **Manual_Condition**.

Add_parameter(Parameter_Collection parameters,Integer value)

Name

Integer Add_parameter(Parameter_Collection parameters,Integer value)

Description

This call will add a new Integer parameter to a **Parameter_Collection**.

This will return 0 if successful.

ID = 2551

Add_parameter(Parameter_Collection parameters,Text value)

Name

Integer Add_parameter(Parameter_Collection parameters,Text value)

Description

This call will add a new Text parameter to a **Parameter_Collection**.

This will return 0 if successful.

ID = 2552

Add_parameter(Parameter_Collection parameters,Real value)

Name

Integer Add_parameter(Parameter_Collection parameters,Real value)

Description

This call will add a new Real parameter to a **Parameter_Collection**.

This will return 0 if successful.

ID = 2553

Add_time_parameter(Parameter_Collection parameters,Integer value)

Name

Integer Add_time_parameter(Parameter_Collection parameters,Integer value)

Description

This call will add a new timestamp parameter, from an Integer value, to a **Parameter_Collection**.

This will return 0 if successful.

ID = 2554

6 Examples

When using these code examples check the ends of lines for word wrapping.

The file *set_ups.h* contains constants and values that are used in, or returned by, 12dPL supplied functions. Before any of the constants or values in *set_ups.h* can be used, *set_ups.h* needs to be included in a 12dPL program by using the command *#include "set_ups.h*" at the top of the 12dPL program. For an example see Example 11 .

For more information on set_ups.h, see Appendix - Set_ups.h File

Example 1

A macro to select a string and write out how many vertices there are in the string.

See Example 1 example using macro console, and goto's

See Example 1a example using macro console, without goto's

See Example 1b example using a panel

Example 2

Macro to select a string and ask if its ok to delete it.

This macro uses the Macro Console.

See Example 2 example with goto's

See Example 2a example without goto's

Example 3

Write four lines of data out to a file.

This macro uses the Macro Console.

See Example 3

Example 4

Read a file in and calculate the number of lines and words.

This macro uses the Macro Console.

See Example 4

Example 5

Write four lines of data out to a file and then read it back in again.

This macro uses the Macro Console.

See Example 5 close and reopen the file

Example 5a

Create a Unicode and an ANSI (Ascii) file.

This macro uses the Macro Console.

See Example 5a ANSI and Unicode files

Example 5b

Create all the Unicode/ANSI file types.

This example has a User Defined Function.

This macro uses the Macro Console.

See Example 5b all the ANSI/Unicode file types

Example 6

1. select a pad
2. ask for cut and fill interface slopes
3. ask for a separation between the interface calcs
4. ask if interface is to left or right of pad
5. ask for a tin to interface against

Then

- calculate the interface string
- display the interface on all the views the pad is on
- ·check if the interface is ok to continue processing
- check for intersections in the interface and if so, ask for a good point so loop removal can be done.
- display the cleaned interface
- calculate the tin for the pad and the cleaned interface
- calculate and display the volumes between the original tin and the new tin

The macro includes a called function as well as main.

This macro uses the Macro Console.

See Example 6

Example 7

Macro to label each point of a user selected string with the string id and the string point number.

The labels are created as a 4d string.

This macro uses the Macro Console.

See Example 7

Example 8

A macro that exercises many of the Text functions

This macro uses the Macro Console.

See Example 8 .

Example 9

A macro to label the spiral and curve lengths of an Alignment string

This macro uses the Macro Console.

See Example 9

Example 10

Macro to take the (x,y) position for each point on a string and then produce a text string of the z-values at each point on the tin

This macro uses the Macro Console.

See Example 10

Example 11

Macro to delete a selected empty model or all empty models in a project.

This macro uses a 12d Model Panel.

See Example 11

Example 12

Macro to change names of selected strings

See Example 12

Example 13

Macro to use the x, y, z of a text string and create a new 3d point string at the same point.

This macro uses a 12d Model Panel.

See Example 13

Example 14

This is an example of the 12dPL functions for a dialogue that contains most of the common widget controls. The text for the widgets and the on/off switch are contained in the function call go_panel.

This macro uses a 12d Model Panel.

See Example 14

Example 15

This is an example of how to create a Macro_Function.

This macro uses a 12d Model Panel.

See Example 15

Example 1

```
//-------------------------------------------------------------
// Programmer   Lee Gregory
// Date         26/5/94
// Description of Macro
// Macro to select a string and write outs out to the console
// how many vertices there are in the string. This is then repeated.
// The macro terminates if cancel is selected from pick ops menu
// Note - This macro uses a Console.
// There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in
// any order in the panel.
//-------------------------------------------------------------
void main (){
  Element string;
  Integer ret,no_verts;
  Text text;

 Prompt("Select a string");   // write message to prompt message area of console

ask:
  ret = Select_string("Select a string",string); // message goes to 12d Model Output Window
  if(ret == -1) {
    Prompt("Macro finished - cancel selected");
    return;
  } else if (ret == 1) {
    if(Get_points(string,no_verts) !=0) goto ask;
    text = To_text(no_verts);
    text = "There are " + text + " vertices in the string. Select another string";
    Prompt(text);
    goto ask;
  } else {
    Prompt("Invalid pick. Select again");
    goto ask;
  }
}
```

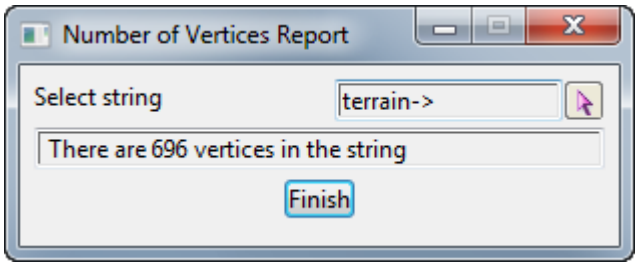
Example 1a

```
//-----------------------------------------------------------------------------------------------------------
// Programmer   Lee Gregory
// Date         24/8/13
// Description of Macro
// Macro to select a string and write outs out to the console
// how many vertices there are in the string. This is then repeated.
// The macro terminates if cancel is selected from pick ops menu.
// This macro is the same as Example 1 but does not use goto.
// Note - This macro uses a Console.
// There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in any order in the panel.
//-----------------------------------------------------------------------------------------------------------
void main(){
  Element string;
  Integer ret=0,no_verts;
  Text text;

 Prompt("Select a string");// write message to console

  while (ret != -1) {
    ret = Select_string("Select a string",string); //message to Output Window
    if(ret == -1) Prompt("Macro finished - cancel selected");
    else if (ret == 1) {
      if(Get_points(string,no_verts) !=0) continue;
      text = To_text(no_verts);
      text = "There are " + text + " vertices in the string. Select another string";
      Prompt(text);
    } else Prompt("Invalid pick. Select again");
  }
  return;
}
```

Example 1b

```
//-----------------------------------------------------------------------------------------------------
// Programmer   Lee Gregory
// Date         22/9/13
// Description of Macro
// Macro using a panel to select a string and when a string is
// selected, the number of vertices in the string is written to the message box.
// The macro terminates when the Finish button, or X is selected
//-----------------------------------------------------------------------------------------------------
#include "set_ups.h"

void main() {
  Panel   panel = Create_panel("Number of Vertices Report");
  Message_Box    new_msg_box = Create_message_box("");
  New_Select_Box new_select_box = Create_new_select_box("Select string",
                                  "Select a string",SELECT_STRING,new_msg_box);
  Button  finish_button = Create_finish_button("Finish","finish_reply");
  Vertical_Group vgroup = Create_vertical_group(BALANCE_WIDGETS_OVER_HEIGHT);

  Append(new_select_box,vgroup);
  Append(new_msg_box,vgroup);

  Horizontal_Group hgroup = Create_button_group();
   Append(finish_button,hgroup);
  Append(hgroup,vgroup);
  Append(vgroup,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1, id; Text cmd,msg;

  while(doit) {
    Integer ret = Wait_on_widgets(id,cmd,msg);
    switch(id) {
      case Get_id(panel) : {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button) : {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(new_select_box) : {
        Set_data(new_msg_box,"");
```

```
        if(cmd == "accept select") {
          Element string; Integer ierr,no_verts;
          ierr = Validate(new_select_box,string);
          if(ierr != TRUE) {
            Set_data(new_msg_box,"Invalid pick.");
          } else {
            if(Get_points(string,no_verts)==0) {
              Set_data(new_msg_box,"There are " + To_text(no_verts) + " vertices in the string");
            } else {
              Set_data(new_msg_box,"error in string");
            }
          }
        }
        break;
      }
    }
  }
}
```



Panel for Exercise 1b

Example 2

```
// ---------------------------------------------------------
// Programmer   Lee Gregory
// Date         26/5/94
// Description of Macro
// Macro to select a string and ask if it is ok to delete it.
// The macro loops round until cancel is selected from the pick ops menu.
// Note - This macro uses a Console.
//  There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
// ---------------------------------------------------------
void main (){
  Element string;
  Integer ret;
  Text text;

  Prompt("Select a string to delete");   // write message to prompt message area of console
  ask:
    ret = Select_string("Select a string to delete",string);
    if(ret == -1) {
      Prompt("Macro finished - cancel selected");
      return;
    } else if (ret == 1) {
      Prompt("ok to delete the string y or n",text);
        if(text == "y") {
          Element_delete(string);
          Prompt("Sting deleted. Pick another string");
        } else {
          Prompt("No string deleted. Pick another string");
        }
    } else {
      Prompt("Invalid pick. Select again");
    }
  goto ask;
}
```

Example 2a

```
//-----------------------------------------------------------------------------------------------------
// Programmer   Lee Gregory
// Date         24/8/13
// Description of Macro
// Macro to select a string and ask if it is ok to delete it.
// The macro loops round until cancel is selected from the pick ops menu.
// This macro is the same as Example 2 but does not use goto.
// Note - This macro uses a Console.
// There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in any order in the panel.
//-----------------------------------------------------------------------------------------------------
void main(){
  Element string;
  Integer ret=0;
  Text text;

 Prompt("Select a string to delete");// write message to console

  while (ret != -1) {
    ret = Select_string("Select a string to delete",string); //message to Output Window
    if(ret == -1) Prompt("Macro finished - cancel selected");
    else if (ret == 1) {
     Prompt("ok to delete the string y or n",text);
        if(text == "y") {
          Element_delete(string);
          Prompt("Sting deleted. Pick another string");
        } else {
          Prompt("No string deleted. Pick another string");
        }
    } else Prompt("Invalid pick. Select again");
  }
  return;
}
```

Example 3

```
//-----------------------------------------------------------
// Programmer   Alan Gray
// Date         27/5/94
// Description of Macro
// Write four lines of data out to a file
// Note - This macro uses a Console.
// There are very few Console macros since most people
// prefer to use full Panels as in12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
//-----------------------------------------------------------
void main()
{
  File file;
  File_open("report.rpt","w+"," ",file);              // ANSI file - also do UNICODE
  File_write_line(file,"1st line of file");
  File_write_line(file,"2nd line of file");
  File_write_line(file,"3rd line of file");
  File_write_line(file,"4th line of file");
  File_flush(file);
  File_close(file);
}
```

Example 4

```
//-------------------------------------------------------------
// Programmer   Alan Gray and Lee Gregory
// Date         3/9/13
// Description of Macro
// Read a file in and calculate the number of lines and words.
// Write to the console the number of lines and words,
// and also the individual words.
// Note - This macro uses a Console.
//-------------------------------------------------------------
void main()
{
  Text file_name; File file;

  while (1) {
    File_prompt("Enter the file name","*.rpt",file_name);
    if(!File_exists(file_name)) continue;
    File_open(file_name,"r","ccs=UNICODE",file);
    break;
  }
  Integer eof,count = 0 wordc = 0;
  Text line;

  while(1) {
    if(File_read_line(file,line) != 0) break;
    ++count;

// break line into words
    Dynamic_Text words;
    Integer      no_words = From_text(line,words);
    wordc = wordc + no_words;     // this could also be writen as wordc += no_words;
    Get_number_of_items(words,no_words);
    for(Integer i=1;i<=no_words;i++) {
      Text t;
      Get_item(words,i,t);
      Print(t); Print();
    }
  }
  File_close(file);

// display the number of lines and words read
  Text out;
  out = To_text(count) + " lines & " + To_text(wordc) + "words read";
  Prompt(out);  Print(out);
  Print("\nMacro finished\n"); // write to the Output Window
}
```

Example 5

```
//-------------------------------------------------------------
// Programmer   Alan Gray and Lee Gregory
// Date         2/9/13
// Description of Macro
// Write four lines of data out to a file and close the file.
// Then open the file and then read it back in again.
// Report the number of lines read in.
// Note - This macro uses a Console.
//  There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
//-------------------------------------------------------------
void main()
{
  File file;
  File_open("report.rpt","w+","",file);
  File_write_line(file,"1st line of file");
  File_write_line(file,"2nd line of file");
  File_write_line(file,"3rd line of file");
  File_write_line(file,"4th line of file");
  File_flush(file);

// Because files may be Unicode with a BOM then
// it is best to close the file and reopen it again for reading.
// File_rewind, w+, r+  can destroy the BOM.

  File_close(file);
 File_open("report.rpt","r","",file);
  Integer count = 0;
  while(1) {
    Text line;
    if(File_read_line(file,line) != 0) break;
    ++count;
  }
  File_close(file);
// display # lines read
  Prompt(To_text(count) + " lines read");
}
```

Example 5a

```
//-------------------------------------------------------
// Programmer   Lee Gregory
// Date         2/9/13
// Description of Macro
// Delete and open a new file as a UNICODE file
// Get the Start position and write it out to the output .
// Write "one line" into the file.
// Repeat this for a ANSI file.
// Note - This macro uses a Console.
// There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
//-------------------------------------------------------
void main()
{
  File file;
  Text file_name, file_type;
  Integer file_start;
  Clear_console();

  file_name = "test_unicode.rpt";
  file_type = "ccs=UNICODE";
  if(File_exists(file_name)) File_delete(file_name);
  File_open(file_name,"w",file_type,file);
  File_tell(file,file_start);     // record the beginning of the file
  File_write_line(file,"one line");
  Print("File <" + file_name + ">  Start pos = " + To_text(file_start) + "\n");
  File_close(file);

  file_name = "test_ansi.rpt";
  file_type = "";
  if(File_exists(file_name)) File_delete(file_name);
  File_open(file_name,"w",file_type,file);
  File_tell(file,file_start);     // record the beginning of the file
  File_write_line(file,"one line");
  Print("File <" + file_name + ">  Start pos = " + To_text(file_start) + "\n");
  File_close(file);

  Print("\nMacro finished\n"); // write to the Output Window
}
```

Example 5b

```
//-----------------------------------------------------------
// Programmer   Lee Gregory
// Date         2/9/13
// Description of Macro
// This is an example of using a User Defined Function.
// The function create_new_file hsa the Text arugments file_name
// and file type nad creates a new file called file_name
// and with type file_type. It also writes information
// to the Output Window.
//
// The main function calls this function numerous times
// to create filees of type Unicodde, UTF-8, UTF-16LE and ANSI.

// Note - This macro uses a Console.
// There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
//-----------------------------------------------------------
Integer create_new_file(Text file_name,Text file_type)
{
  File file;
  Integer file_start,file_end;

  if(File_exists(file_name)) File_delete(file_name);
  File_open(file_name,"w",file_type,file);
  File_tell(file,file_start);     // record the beginning of the file
  File_write_line(file,"one line");
  File_tell(file,file_end);     // record after writing a line
  Print("File <" + file_name + ">  Start pos = " + To_text(file_start) + "  End pos = " +
          To_text(file_end) + "\n");
  File_close(file);
  return(0);
}
void main()
{
  Clear_console();
  create_new_file("test_unicode.4dm","ccs=UNICODE");
  create_new_file("test_utf_8.4dm","ccs=UTF-8");
  create_new_file("test_utf_16.4dm","ccs=UTF-16LE");
  create_new_file("test_ansi.4dm","");
  Print("\nMacro finished\n"); // write to the Output Window
}
```

Example 6

```
//-------------------------------------------------------------
// Programmer     Lee Gregory
// Date           26/5/94
//
// Description of Macro
// (a) select a pad
// (b) ask for cut and fill interface slopes
// (c) ask for a separation between the interface calcs
// (d) ask if interface is to left or right of pad
// (d) ask for a tin to interface against
//  Then
// (a) calculate the interface string
// (b) display the interface on all the views the pad is on
// (c) check if the interface is ok to continue processing
// (d) check for intersections in the interface and if so, ask
//     for a good point so loop removal can be done.
// (e) display the cleaned interface
// (f) calculate the tin for the pad and the cleaned interface
// (g) calculate and display the volumes between the original tin
//     and the new tin
// The macro includes some user defined function as well as main.
//
// Note - This macro uses a Console.
// There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
//
// Modifications
// Programmer   Lee Gregory
// Date         15/2/2013
//
// Description of Modifications
// Added more error checks, and routines to
// (a) get all the views that a model is on, then delete the model
//     and refresh that list of views
// (b) Example of two overloaded function called redraw_views
//     redraw_views(Model model) - redraw all views the model is on
//     redraw_views(Dynamic_Text dtviews) - redraw all view in in list
//-------------------------------------------------------------

// Function to add new_model to all the non-section views that
// old_model is on

void add_to_non_section_views(Model new_model,Model old_model)
{
  Dynamic_Text  dtviews;
  Integer no_views;

// get all the views that old_model is on
```

```
  Model_get_views(old_model,dtviews);

// add new_model to all the views

  Get_number_of_items(dtviews,no_views);
  View view;
  Text view_name,type;
  if(no_views <= 0) return;
  for (Integer i=1;i <= no_views;i++) {
    Get_item(dtviews,i,view_name);
    view = Get_view(view_name);
    Get_type(view,type);
    if(type == "Section") continue;
    View_add_model(view,new_model);
  }
  return;
}

// Function to redraw all the non section views that
// old_model is on

void redraw_views(Model old_model)
{
  Dynamic_Text  dtviews;
  Integer no_views;

// get all the views that old_model is on
  Model_get_views(old_model,dtviews);

//  redraw all the plan views

  Get_number_of_items(dtviews,no_views);
  View view;
  Text view_name,type;
  if(no_views <= 0) return;

  for (Integer i=1;i<=no_views;i++) {
    Get_item(dtviews,i,view_name);
    view = Get_view(view_name);
 //   Get_type(view,type);
 //   if(type == "Section") continue;
    View_redraw(view);
  }
  return;
}

// Function to redraw all the non section views
// named in the Dynamic Text dviews

void redraw_views(Dynamic_Text dtviews)
{
  Integer no_views;

//  redraw all the non section views
```

```
  Get_number_of_items(dtviews,no_views);
  View view;
  Text view_name,type;
  if(no_views <= 0) return;

  for (Integer i=1;i <= no_views;i++) {
    Get_item(dtviews,i,view_name);
    view = Get_view(view_name);
//    Get_type(view,type);
//    if(type == "Section") continue;
    View_redraw(view);
  }
  return;
}

// Function that if model model_name exists, get all the views that
// model_name is on, delete model_name and then redraw all the
// views model_name was on.

void delete_model_redraw_views(Text model_name)
{
  Dynamic_Text  dtviews;
  Model         model;

// if model model_name exists, get all the views that model_name is on
// then delete model_name and redraw all the views that model_name was on.

  if(Model_exists(model_name)) {
    model = Get_model(model_name);
    Model_get_views(model,dtviews);
    Model_delete(model);
    redraw_views(dtviews);  // redraw all the views that model_name was on
  }

  return;
}

// Main program to calculate the interface for a pad
// and then do volumes on it

void main ()
{
  Element pad,int_string,clean_string,sgood;
  Point   pt;
  Integer ret,side,error,closed;
  Text    text,tside,ok;
  Real    cut,fill,sep;

  Text    combined_model_name     = "pad combined";
  Text    combined_tin_name       = "pad combined";
  Text    combined_tin_model_name = "tin pad combined";
  Model   combined_model,combined_tin__model,pad_model;
  Tin     ground_tin,combined_tin;
```

```
  Dynamic_Text  dtviews;


clean_up:
// Delete the tin combined_tin_name

  Tin_delete(Get_tin(combined_tin_name));

// delete models called combined_model_name and combined_tin_model_name
// and redraw all non-section views they were on.

  delete_model_redraw_views(combined_model_name);
  delete_model_redraw_views(combined_tin_model_name);

// start the option proper

  Prompt("Select a pad");    // write message to prompt message area of console

ask:
  ret = Select_string("Select a pad",pad);
  if(ret == -1) {
    Prompt("Macro finished - cancel selected");
    return;
  } else if (ret != 1) {
    Prompt("bad pick, try selecting a string again");
    goto ask;
  } else {        // case of valid pick
// check if pad is closed
    error = String_closed(pad,closed);
    if(closed !=1) {
      Prompt("Pad not a closed string. Select another string");
      goto ask;
    }
  }

// getting here means we have selected a pad

// get cut and fill slopes, side to interface
// and separation between sections

  Integer ierr;

cut:
  ierr = Prompt("Cut slope 1:",cut);
  if(ierr != 0) goto cut;

fill:
  ierr = Prompt("Fill slope 1:",fill);
  if(ierr != 0) goto fill;

sep:
  ierr = Prompt("Separation",sep);
  if(ierr != 0) goto sep;
```

```
lr:
  ierr = Prompt("Left or Right (l or r)",tside);
  if(ierr != 0) goto lr;

  if((tside == "l")||(tside == "L")){
    side = -1;
  } else if((tside == "r")|| (tside == "R")) {
    side = 1;
  } else {
    Prompt("incorrect answer. Try again");
    goto lr;
  }

tin:
  Tin_prompt("Tin name",1,text);
  if(text == "") return;

  if(!Tin_exists(text)) goto tin;
  ground_tin = Get_tin(text);

// calculate the interface

  Interface(ground_tin,pad,cut,fill,sep,1000.0,side,int_string);

// Draw the interface to see if l or r was ok
// Get the model for the selected pad string,
// add the interface string onto the same views
// and check that its ok to continue

  combined_model = Get_model_create(combined_model_name); // create the model called
                                          // combined_model_name and add int_string

  Set_model(int_string,combined_model);
  Get_model(pad,pad_model);
  add_to_non_section_views(combined_model,pad_model);
  redraw_views(pad_model);                    // redraw the non section views pad_model is on

  Prompt("OK to continue (y or n)",ok);
  if(ok == "n") {
    Element_delete(int_string);
    goto clean_up;          // need to start again
  }

// check if the interface needs cleaning

  Integer no_self;
  String_self_intersects(int_string,no_self);
  if(no_self < 1) {
    clean_string = int_string;
    goto cleaned;
  }

//  clean the interface string
```

```
  Real x,y,z,ch,ht;

good:
  Prompt("Pick a good point");    // write message to prompt message area of console
  ret = Select_string("Pick a good point",sgood,x,y,z,ch,ht);
  Set_x(pt,x);
  Set_y(pt,y);
  Set_z(pt,z);
  Loop_clean(int_string,pt,clean_string);
  String_self_intersects(clean_string,no_self);

  if(no_self < 1) goto cleaned;

// still not a clean interface

  Element_delete(clean_string);
  goto good;

//  add the interface string to a new model which is added to the
//  same non-section views that the model containing the string was on

cleaned:
  Element duplicate_pad;
  Element_duplicate(pad,duplicate_pad);

  Set_model(duplicate_pad,combined_model);   // add duplicate of pad string to combined_model
  Set_model(clean_string,combined_model);     // add cleaned interface string to combined model
  Calc_extent(combined_model);

  add_to_non_section_views(combined_model,pad_model); // add combined model to all
                                          // non sections views that pad_model is on

// triangulate the combined model - pad and interface strings

  Triangulate(combined_model,combined_tin_name,1,1,1,combined_tin);

  Model combined_tin_model = Get_model_create(combined_tin_model_name); // create model
                                                                  // called combined_tin_model
  Set_model(combined_tin,combined_tin_model);        // add combined_tin to model
                                                     // combined_tin_model
  Calc_extent(combined_tin_model);

// add combined_tin_model to all non section views that pad_model is on

  add_to_non_section_views(combined_tin_model,pad_model);
                                                     //
// do volumes between the ground tin and the combined_tin with interface string as polygon

  Real  cut_vol,fill_vol,bal_vol;
  Volume_exact(ground_tin,combined_tin,clean_string,cut_vol,fill_vol,bal_vol);

// display the volumes
```

```
  Text ret_text;
  Text out_text,cut_text,fill_text,bal_text;
  cut_text  = To_text(cut_vol,3);
  fill_text = To_text(fill_vol,3);
  bal_text  = To_text(bal_vol,3);
  out_text = "cut " + cut_text + " fill " + fill_text + " bal " + bal_text + " <enter> to exit";
  Prompt(out_text,ret_text);

  return;
}
```

Example 7

```
//-------------------------------------------------------
// Programmer    Andre Mazzone
// Date          3rd June 1994
// Description of Macro
// Macro to label each point of a user selected string with
// the string id and the string point number.
// The labels are created as a 4d string.
// Note - This macro uses a Console.
//  There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
//-------------------------------------------------------
void Gen_get(Element string,Real& x,Real& y,Real& z,Integer i)
// a function that extracts the x, y, and z for the ith point in
// any string (this routine reused from drape line
// point sexample)
// in:  string,i
// out: x,y,z
{
  Text    type;
  Element result;
  // get the type
  Get_type(string, type);
  if(type == "2d") {
    // 2d strings have only one z value
    //   (this is not needed for this example
    //    and is only here for completeness)
    Get_2d_data(string, i, x, y);
    Get_2d_data(string, z);
  } else if(type == "3d") {
    // 3d strings have all the information
    Get_3d_data(string, i, x, y, z);
  } else if(type == "4d") {
    // 4d strings have too much information
    // so any text is thrown away
    Text tmp;
    Get_4d_data(string, i, x, y, z, tmp);
```

```
  } else if(type == "Interface") {
    // interface strings have too much information
    // so the flags are thrown away
    Integer tmp;
    Get_interface_data(string, i, x, y, z, tmp);
  }
}
Element create_label_string(Element string)
// create a 4d string with labels for string id and point num
// in:  string
// out: return value
{
  Integer npts, i, id;
  Real    x[200], y[200], z[200];
  Text    t[200], buf;
  Element str4d;
  // get number of points
  Get_points(string, npts);
  // get the id
  Get_id(string, id);
  // convert id to text
  buf = To_text (id) + "-";
  // loop through all points
  for (i = 1; i <= npts; i++) {
    // get x, y, z data
    Gen_get(string, x[i], y[i], z[i], i);
    // create text message with id-pt no
    t [i] = buf + To_text (i);
  }
  // create the string and return it
  return Create_4d(x, y, z, t, npts);
}
void main ()
// Asks for a model to use plus a string to be picked.
// The program then creates a label string and adds
// it to the model.
{
  Integer ret;
  Element poly;
  // get the model to use
  Text model_name;
```

```
  ret = Prompt ("model to store labels", model_name);
  while (ret != 0) {
    // loop until there are no errors in input
    Text x;
    Prompt ("error in input, press return", x);
    ret = Prompt ("model to store labels", model_name);
  }
  // get a handle to a new or existing model
  Model model = Get_model_create (model_name);
  // get the polyline from user
  Text select_msg = "Id_string: string to label";
  Prompt ("Select a polygon from a view");
  ret = Select_string (select_msg, poly);
  // loop until success or cancel
  Integer done = 0;
  while ((ret != -1) && (ret !=1) && (!done)) {
    if (ret == 0) {
      // this means the select failed, so try again
      Prompt ("select failed, please try again");
      Prompt ("Select a polygon from a view");
      ret = Select_string (select_msg, poly);
    } else if (!Element_exists (poly)) {
      // this means that there were no selections, so try again
      Prompt ("no polygon selected, please try again");
      ret = Select_string (select_msg, poly);
    }
  }
  // if user chooses cancel from the select box then end
  if (ret == -1) {
    Prompt ("action cancelled");
    return;
  }
  // create string
  Element labels = create_label_string(poly);
  // add to model
  Set_model (labels, model);
  // finished processing
  Prompt("Finished labelling");
}
```

Example 8

```
//-----------------------------------------------------------
// Programmer   Alan Gray
// Date         14/7/94
// Description of Macro
// A macro which exercises many of the Text functions
//-----------------------------------------------------------
void main()
{
  Text t1 = " A very very long string with lots of simple words";
  Integer l1 = Text_length(t1);
  Print("<");  Print(t1);  Print(">\n");
  Text    t2 = Get_subtext(t1,1,10);
  Integer l2 = Text_length(t2);
  Print("<");  Print(t2);  Print(">\n");
  Text    t3 = Text_justify(t1);
  Integer l3 = Text_length(t3);
  Print("<");  Print(t3);  Print(">\n");
  Text    t4 = Text_upper(t1);
  Integer l4 = Text_length(t4);
  Print("<");  Print(t4);  Print(">\n");
  Text    t5 = Text_lower(t1);
  Integer l5 = Text_length(t5);
  Print("<");  Print(t5);  Print(">\n");
  Integer p  = Find_text(t1,"words");
  Print("p=<");  Print(p);  Print(">\n");
  Text    t6 = t1;  Set_subtext(t6,p,"mindless words");
  Integer l6 = Text_length(t6);
  Print("<");  Print(t6);  Print(">\n");
  Text    t7 = t1;  Set_subtext(t7,10,"[mindless words]");
  Integer l7 = Text_length(t7);
  Print("<");  Print(t7);  Print(">\n");
  Text    t8 = t1;  Insert_text(t8,p,"mindless ");
  Integer l8 = Text_length(t8);
  Print("<");  Print(t8);  Print(">\n");
// formatting
  Integer I = 1234567;
  Real    r = 987654.321;
  Text    b = To_text(I,"I = %8Id") + " "+ To_text(r,"r = %12.4lf") + " :";
  Print("<");  Print(b);  Print(">\n");
```

```
// decoding
  Integer ll;
  From_text(Get_subtext(b,Find_text(b,"l = "),9999),ll,"l = %ld");
  Print("ll = ");  Print(ll);  Print("\n");
  Real rr;
  From_text((Get_subtext(b,Find_text(b,"r = "),9999),rr,"r = %lf");
  Print("rr = ");  Print(rr);  Print("\n");
}
```

Example 9

```
//-----------------------------------------------------------
// Programmer   Lee Gregory
// Date         30/9/94
// Description of Macro
// A macro to label the spiral and curve lengths of
// an Alignment string (not for a Super Alignment)
// Note - This macro uses a Console.
//  There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
//-----------------------------------------------------------
void get_hip_info(Element align,Integer hip,Integer &type,
                  Real xval[],Real yval[],Real lengths[])
// -----------------------------------------------------------

//  Get the horizontal info for an horizontal ip
//      - the co-ordinates of the special points
//      - the curve radius and curve length
//      - the left and right spiral lengths
//
//  Type of HIP is returned as type where
//     type = 0  HIP only
//          1  Curve only
//          2  LH Spiral only
//          3  LH spiral and curve
//          4  RH spiral only
//          5  curve, RH spiral
//          6  LH spiral, RH spiral

//          7  LH spiral, curve, RH spiral
//   Co-ordinates of special points returned in
//       xval[1...6],yval[1...6]
//   where the array position gives
//       position  1  LH tangent, TS or TC
//                 2  RH tangent, ST or CT
//                 3  curve centre
//                 4  SC
```

```
//              5  CS
//              6  HIP
//   NOTE -
//  If the IP is an HIP only, 1-5 are all given the HIP co-ords.

//  If the IP has a curve and no spirals, 1 is set equal
//     to  4 (TC=SC), and 2 is set equal to 5 (CT=CS).
//   The curve radius, curve and spiral lengths are returned in
//    the array lengths[1...4]
//      position  1  circle radius
//              2  circle length
//              3  left spiral length
//              4  right spiral length
//
// ------------------------------------------------------------
{
  Text hip_type;
  Integer ret;
  ret = Get_hip_type(align,hip,hip_type);
// Get the co-ordinates of the special points for the HIP
  if(hip_type == "IP") {
//  case of HIP only with no curve or spiral
    Real xip,yip;  ret = Get_hip_geom(align,hip,0,xip,yip);
    xval[6] = xip; yval[6] = yip;
    type = 0;
// fill in other array positions - set them all to the HIP
//   position
    xval[1] = xip; yval[1] = yip;
    xval[2] = xip; yval[2] = yip;
    xval[3] = xip; yval[3] = yip;
    xval[4] = xip; yval[4] = yip;
    xval[5] = xip; yval[5] = yip;
  } else if(hip_type == "Curve") {
//  case of HIP with and curve and no spirals
    Real xip,yip;  ret = Get_hip_geom(align,hip,0,xip,yip);
    Real xtc,ytc;  ret = Get_hip_geom(align,hip,1,xtc,ytc);
    Real xct,yct;  ret = Get_hip_geom(align,hip,2,xct,yct);
    Real xcc,ycc;  ret = Get_hip_geom(align,hip,3,xcc,ycc);
    xval[1] = xtc; yval[1] = ytc;
    xval[2] = xct; yval[2] = yct;
    xval[3] = xcc; yval[3] = ycc;
```

```
    xval[6] = xip; yval[6] = yip;
    type = 2;
// fill in the other array positions
    xval[4] = xtc; yval[4] = ytc;
    xval[5] = xct; yval[5] = yct;
  } else if(hip_type == "Spiral") {
    Real xip,yip;  ret = Get_hip_geom(align,hip,0,xip,yip);
    Real xts,yts;  ret = Get_hip_geom(align,hip,1,xts,yts);
    Real xsc,ysc;  ret = Get_hip_geom(align,hip,4,xsc,ysc);
    Real xcs,ycs;  ret = Get_hip_geom(align,hip,5,xcs,ycs);
    Real xst,yst;  ret = Get_hip_geom(align,hip,2,xst,yst);
    Real xcc,ycc;  ret = Get_hip_geom(align,hip,3,xcc,ycc);
    Integer left_spiral = ((xts != xsc) || (yts != ysc)) ? 1 : 0;
    Integer right_spiral= ((xst != xcs) || (yst != ycs)) ? 1 : 0;
    Integer curve       = ((xsc != xcs) || (ysc != ycs)) ? 1 : 0;
    xval[1] = xts; yval[1] = yts;
    xval[2] = xst; yval[2] = yst;
    xval[3] = xcc; yval[3] = ycc;
    xval[4] = xsc; yval[4] = ysc;
    xval[5] = xcs; yval[5] = ycs;
    xval[6] = xip; yval[6] = yip;
    type = 2*curve + 2*left_spiral + 2*right_spiral;
  }
//  Get the curve radius, curve and spiral lengths
  Real x,y,radius,left_spiral,right_spiral;
  Get_hip_data(align,hip,x,y,radius,left_spiral,right_spiral);
  Real ch1,ch2,xf,yf,zf,dir,off;  // to get curve length
  if(radius != 0) {
    Drop_point(align,xval[4],yval[4],0.0,xf,yf,zf,ch1,dir,off);
    Drop_point(align,xval[5],yval[5],0.0,xf,yf,zf,ch2,dir,off);
    lengths[2] = ch2 - ch1;
  } else {
    lengths[2] = 0.0;
  }
  lengths[1] = radius;
  lengths[3] = left_spiral;
  lengths[4] = right_spiral;
  return;
}
Element position_text(Text text,Real size,Integer colour,Real x1,Real y1,Real x2,Real y2)
// ------------------------------------------------------------
```

```
// Routine to position text
// At the moment it centres it between (x1,y1) and (x2,y2)
// with (bottom,centre) justification
// ------------------------------------------------------------
{
  Real xpos,ypos,angle;
  xpos = 0.5 * (x1 + x2);
  ypos = 0.5 * (y1 + y2);
  angle = Atan2(y2 - y1,x2 - x1);
  Element elt = Create_text(text,xpos,ypos,size,colour,angle,4,1);
  return (elt);
}
void main()
// ------------------------------------------------------------
//  Select an alignment string and then label it in plan with
//  spiral lengths, curve radii and tangent length.
//
//  The positions of the labels is midway between the
//  two critical points.
//  This should be changed to whatever is required

// ------------------------------------------------------------
{
  Integer ret;
  Element cl;
  Real    text_size;
  Integer colour;
  Text    colour_name,model_name;
  Model   model;
  Real    x_prev_tangent,y_prev_tangent;
// Get model for text
model :
   Model_prompt("Model name for text ? ",model_name);
   if(!Model_exists(model_name)) goto model;
    model = Get_model(model_name);
// Get text size
text_size :
   if(Prompt("Text size ? ",text_size) != 0) goto text_size;

// Get text colour
text_colour:
```

```
   Colour_prompt("Colour for text ? ",colour_name);
   if(!Colour_exists(colour_name)) goto text_colour;
    if(Convert_colour(colour_name,colour) != 0) goto text_colour;
// Get alignment string
  Prompt("Select alignment string");
align:
  ret = Select_string("Select alignment string",cl);
  if(ret == -1) {
    Prompt("Finished");
    return;
  } else if(ret != 1) {
    Prompt("Try again ");
    goto align;
  }
  Text type_name;  Get_type(cl,type_name);
  if(type_name != "Alignment") {
    Prompt("not an alignment string - try again");
    goto align;
  }
// query all alignment info
  Integer no_hip;
  Get_hip_points(cl,no_hip);
  if(no_hip <= 1) {
    Prompt("<= 1 HIP point");
    return;
  }
// label the alignment
  for(Integer i=1;i<= no_hip;i++) {
    Integer  type;
    Real     xval[6],yval[6],lengths[4];
    get_hip_info(cl,i,type,xval,yval,lengths);

// label the spiral lengths and curve radius
    Real     xpos,ypos,angle;
    Text     text;
    Element  elt;
    Integer curve        = (lengths[1] == 0) ? 0 : 1;
    Integer left_spiral  = (lengths[3] == 0) ? 0 : 1;
    Integer right_spiral = (lengths[4] == 0) ? 0 : 1;
// label the left spiral length
    if(left_spiral) {
```

```
      text = "spiral length = " + To_text(lengths[3],1) + "m";
      elt = position_text(text,text_size,colour,xval[1],yval[1],xval[4],yval[4]);
      Set_model(elt,model);
    }
// label the curve radius
    if(curve) {
      text = "Radius = " + To_text(lengths[1],1) + "m";
      elt = position_text(text,text_size,colour,xval[4],yval[4],xval[5],yval[5]);
      Set_model(elt,model);
    }
// label the right spiral length
    if(right_spiral) {
      text = "spiral length = " + To_text(lengths[4],1) + "m";
      elt = position_text(text,text_size,colour,xval[5],yval[5],xval[2],yval[2]);
      Set_model(elt,model);
    }
// label the tangent
    if(i==1) {
      x_prev_tangent = xval[6];
      y_prev_tangent = yval[6];
    } else {
      Real xx,yy,tangent;
      xx = xval[1] - x_prev_tangent;
      yy = yval[1] - y_prev_tangent;
      tangent = Sqrt(xx*xx+ yy*yy);
      text = "tangent length = " + To_text(tangent,1) + "m";
      elt = position_text(text,text_size,colour,x_prev_tangent,y_prev_tangent,xval[1],yval[1]);
      Set_model(elt,model);
      x_prev_tangent = xval[2];
      y_prev_tangent = yval[2];
     }
  }
  Prompt ("Finished");
}
```

Example 10

```
//-------------------------------------------------------
// Programmer    Andre Mazzone
// Date          3rd September 1994
// Description of Macro
// Macro to take the (x,y) position for each point on a
// string and then produce a text string of the z-values
// at each point on the tin
// Note - This macro uses a Console.
//  There are very few Console macros since most people
// prefer to use full Panels as in 12d Model itself.
// However Panel macros are more difficult to write since
// they are not sequential, but things can be filled in in
// any order in the panel.
//-------------------------------------------------------
void process_elt(Tin tin,Element elt,Model model,Real size,Integer colour,Real angle,Real
offset,Integer decimals)

// ---------------------------------------------------
// Find the z-value on the tin for each point in elt.
// Only process 2d, 3d strings.
// ---------------------------------------------------
{
  Text     type,number;
  Integer  i,no_pts,justif;
  Real     x,y,z,height,rise;
  Element  text_elt;
  Get_type(elt,type);
  Get_points(elt,no_pts);
  justif = 1;
  rise   = 0.0;
  if(!(type =="2d" || type == "3d")) return;
  for (i=1;i<=no_pts;i++) {

    if(type == "2d") {
      Get_2d_data(elt,i,x,y);
    } else if (type == "3d") {
      Get_3d_data(elt,i,x,y,z);
    }
// get value on the tin at (x,y)
```

```
    if(Tin_height(tin,x,y,height) != 0) continue;
    number   = To_text(height,decimals);
    text_elt = Create_text(number,x,y,size,colour,angle,justif,1,offset,rise);
    Set_model(text_elt,model);
  }
  return;
}
void main ()
// -------------------------------------------------------

// Macro to take the (x,y) position for each point on a
// string and then produce a text string of the z-values
// at each point on the tin
// -------------------------------------------------------
{
  Text    tin_name,model_name,text_model_name,colour_name;
  Tin     tin;
  Model   model,text_model;
  Real    text_size,offset,angle,radians;
  Integer colour,decimals;
// Get the name of the tin
get_tin:
  Tin_prompt("Give the name of the tin :",tin_name);

  if(!Tin_exists(tin_name)) goto get_tin;
  tin = Get_tin(tin_name);
// Get model for text
model1 :
  Model_prompt("Model to drape :",model_name);
  if(!Model_exists(model_name)) goto model1;
  model = Get_model(model_name);
// Get model for text
model2 :
  Model_prompt("Model for text :",text_model_name);
  text_model = Get_model_create(text_model_name);
  if(!Model_exists(text_model)) goto model2;
// Get text size

text_size :
  if(Prompt("Text size :",text_size) != 0) goto text_size;
// Get text colour
```

```
text_colour:
  Colour_prompt("Colour for text :",colour_name);
  if(!Colour_exists(colour_name)) goto text_colour;
  if(Convert_colour(colour_name,colour) != 0)
                                                goto text_colour;
angle:
  if(Prompt("Angle for text(degrees) :",angle) != 0)
                                               goto angle;
  Degrees_to_radians(angle,radians);
offset:
  if(Prompt("Offset for text :",offset) != 0) goto offset;

decimals:
  if(Prompt("No. decimal places for text :",decimals) != 0)
                                                            goto decimals;
  decimals = Absolute(decimals);
// Get all the strings in the model and drop their nodes
// onto the tin
  Dynamic_Element  strings;
  Integer          no_strings,i;
  Element          elt;
  Prompt("Processing");
  Get_elements(model,strings,no_strings);
  for (i=1;i<=no_strings;i++) {
    Get_item(strings,i,elt);
    process_elt(tin,elt,text_model,text_size,colour,radians,offset,decimals);

  }
  Prompt("Finished");
}
```

Example 11

```
//-----------------------------------------------------------------------
// Programmer          Van Hanh Cao
// Date                14/07/99
// 12d Model           V4.0
// Version             1.0
// Macro Name          Del_empty_model_panel
// Description
// Delete a selected empty model or all empty models in a project.
//
// Note - this example uses a full 12d Model Panel rather than
// a simple console that the examples 1 to 10 used
//-----------------------------------------------------------------------
// Update/Modification
// (C) Copyright 1990-2003 by 12D Solutions Pty Ltd. All Rights Reserved
// This macro, or parts thereof, may not be reproduced in any form
// without permission of 12D Solutions Pty Ltd
//-----------------------------------------------------------------------
#include "set_ups.H"
// function to delete the model called model_name if it is empty
Integer delete_model(Text model_name,Integer &no_deleted)
{
  Model model = Get_model(model_name);
  Integer no_elts;
  Get_number_of_items(model,no_elts);
  if(!Model_exists(model)) return(-1);
// if model empty then delete it

  if(no_elts == 0) {
    Model_delete(model);
    no_deleted++;
  }
  return(0);
}

// function to delete all the emaply models in a project
Integer delete_all_model(Integer &no_deleted)
{
  Integer      no_models;
  Dynamic_Text project_models;
  Get_project_models (project_models);
  Get_number_of_items(project_models,no_models);
```

```
  no_deleted = 0;
  for(Integer i;i<=no_models;i++) {
    Text    model_name;
    Model   model;
    Integer no_elts;
    Get_item(project_models,i,model_name);
    delete_model(model_name,no_deleted);
  }
  return(0);
}
// function to make a list for a CHoice_Box of all empty models
Integer update_list(Choice_Box &model_list)
{
  Integer      no_models;
  Dynamic_Text project_models;
  Get_project_models (project_models);
  Get_number_of_items(project_models,no_models);
  if(no_models == 0) return(-1);
  Dynamic_Text empty_models;   // a list to contain the names of all empty models
  for(Integer i=1;i<=no_models;i++) {
// validate model
    Text model_name;
    Get_item(project_models,i,model_name);
    Model model = Get_model(model_name);

    if(!Model_exists(model)) continue;
    Integer no_elts;
    Get_number_of_items(model,no_elts);
    if(no_elts == 0) Append(model_name,empty_models);
  }
  Integer no_empty = 0;
  Get_number_of_items(empty_models,no_empty);
// add to choice box
  Text list[no_empty];
  for(Integer j=1;j<=no_empty;j++) Get_item(empty_models,j,list[j]);
  Set_data(model_list,no_empty,list);
  return(0);
}
void manage_a_panel()
{
// create the panel
```

```
  Panel         panel   = Create_panel("Delete Empty Models");
  Message_Box    message = Create_message_box("  ");
  Choice_Box model_list = Create_choice_box("Empty models",message);
  update_list(model_list);
// have buttons Delete, Delete All and Finish in a Horiziontal_Group
  Horizontal_Group bgroup = Create_button_group();
  Button delete     = Create_button("&Delete","delete");
  Button delete_all = Create_button("Delete &All","delete all");
  Button finish     = Create_button("&Finish" ,"finish");
  Append(delete,bgroup);
  Append(delete_all,bgroup);
  Append(finish,bgroup);

// add Widgets to the Panel
  Append(model_list,panel);        // add the Choice_Box with list of empty models
  Append(message,panel);           // add the Message_Box
  Append(bgroup,panel);            // add the Horizontal_Groups of buttons

// Display the panel on the screen
  Show_widget(panel);
  Integer doit = 1;
  Integer no_deleted = 0;
  while(doit) {
    Integer id;
    Text    cmd;
    Text    msg;

// Process events from any of the Widgets on the panel
    Integer ret = Wait_on_widgets(id,cmd,msg);
    if(cmd == "keystroke") continue;
    switch(id) {
      case Get_id(panel) : {
        if(cmd == "Panel Quit") doit = 0;
        break;
      }
      case Get_id(finish) : {
        if(cmd == "finish") doit = 0;
        break;
      }
      case Get_id(model_list) : {
        update_list(model_list);
```

```
        Set_data(message,"Update");
        break;
      }
// delete the selected model
      case Get_id(delete) : {
        Integer ierr;
        Text model_name;
        ierr = Validate(model_list,model_name);
        if(ierr != TRUE) break;
        delete_model(model_name,no_deleted);
        Set_data(message,"empty model \"" + model_name + "\" deleted");
        update_list(model_list);
        Set_data(model_list,"");
        break;
      }
// delete all empty models
      case Get_id(delete_all): {
        delete_all_model(no_deleted);
        Set_data(message,To_text(no_deleted) + " empty model(s) deleted");
        update_list(model_list);
        Set_data(model_list,"");
        break;
      }
    }
  }
}
void main()
{
  manage_a_panel();
}
```

Example 12

```
//-----------------------------------------------------------------------
// Programmer          Van Hanh Cao
// Date                14 Jul 2003
// 12d Model           V4.0
// Version             1.0
// Macro Name          Newname_panel
// Description
// routine to change names of selected strings
// Note - this example uses a full 12d Model Panel rather than
// a simple console that the examples 1 to 10 used
//-----------------------------------------------------------------------
#include "set_ups.H"
void set_names(Element string,Text stem,Integer &number)
{
  Text new_name = stem + To_text(number);
  Set_name(string,new_name);
  number++;
}
void set_names(Model model,Text stem,Integer &number)
{
  Integer         no_items;
  Dynamic_Element  items;

  Get_elements(model,items,no_items);
  for(Integer i=1;i<=no_items;i++) {
    Element elt;
    Get_item(items,i,elt);
    set_names(elt,stem,number);
  }
}
void set_names(View view,Text stem,Integer &number)
{
  Integer      no_items;
  Dynamic_Text items;
  View_get_models     (view,items);
  Get_number_of_items (items,no_items);
  for(Integer i=1;i<=no_items;i++) {
    Text    model_name;
    Get_item(items,i,model_name);
```

```
    Model model = Get_model(model_name);

    if(!Model_exists(model)) continue;
    set_names(model,stem,number);
  }
}
void manage_a_panel()
// -------------------------------------------------------
{
// create the panel
  Panel          panel   = Create_panel("Set new string name(s)");
  Vertical_Group vgroup  = Create_vertical_group(0);
  Message_Box    message = Create_message_box("  ");
  Integer no_choices = 3;
  Text    choices[5];
  choices[1] = "String";
  choices[2] = "Model";

  choices[3] = "View";
  Choice_Box pages_box = Create_choice_box("Data source",message);
  Set_data(pages_box,no_choices,choices);
  Set_data(pages_box,choices[2]);
  Append(pages_box,vgroup);
// create 3 vertical groups for each page of widgets
  Horizontal_Group g1 = Create_button_group();     Set_border(g1,0,0);
  Vertical_Group   g2 = Create_vertical_group(-1); Set_border(g2,0,0);
  Vertical_Group   g3 = Create_vertical_group(-1); Set_border(g3,0,0);
// add these groups to the pages widget

  Widget_Pages pages = Create_widget_pages();
  Append(g1,pages);
  Append(g2,pages);
  Append(g3,pages);
// page 1
  Select_Box select_box = Create_select_box("&Pick a string","Pick a string", SELECT_STRING,
                                                                                  message);

  Append(select_box,g1);

// page 2
```

```
  Model_Box model_box =
Create_model_box("Model",message,CHECK_MODEL_MUST_EXIST);

  Append(model_box,g2);

// page 3
  View_Box view_box = Create_view_box ("View",message,CHECK_VIEW_MUST_EXIST);
  Append(view_box,g3);

// top of panel
  Append(pages_box,vgroup);
  Append(pages    ,vgroup);

// setting
  Vertical_Group ogroup      = Create_vertical_group(0);
  Name_Box       name_box    = Create_name_box("Name stem"  ,message);
  Integer_Box    integer_box = Create_integer_box("Next number",message);

// Default values
  Set_data(name_box,"new name");
  Set_data(integer_box ,1);
  Append(name_box   ,ogroup);
  Append(integer_box,ogroup);

// buttons along the bottom
  Horizontal_Group bgroup = Create_button_group();
  Button process = Create_button("&Process","count");
  Button finish  = Create_button("&Finish" ,"finish");
  Append(process,bgroup);
  Append(finish ,bgroup);
  Append(vgroup ,panel);
  Append(ogroup ,panel);
  Append(message,panel);
  Append(bgroup ,panel);

// set page 2 active
  Integer page = 2;
  Set_page(pages,page);
  Show_widget(panel);
  Integer doit = 1;
  while(doit) {
```

```
    Integer id;
    Text    cmd;
    Text    msg;
    Integer ret = Wait_on_widgets(id,cmd,msg);
    if(cmd == "keystroke") continue;
    switch(id) {
      case Get_id(panel) : {
        if(cmd == "Panel Quit") doit = 0;
        break;
      }
      case Get_id(finish) : {
        if(cmd == "finish") doit = 0;
        break;
      }
      case Get_id(pages_box) : {
        Text page_text;
        Integer ierr = Validate(pages_box,page_text);
        if(ierr != TRUE) break;
        if(page_text == choices[1]) {
          page = 1;
        } else if(page_text == choices[2]) {
          page = 2;
        } else if(page_text == choices[3]) {
          page = 3;
        } else {
          page = 0;
        }
        Set_page(pages,page);
        break;
      }
      case Get_id(select_box) : {
        Integer ierr;
        if(cmd == "accept select") {

// validate name and text size
          Integer next;
          ierr = Validate(integer_box,next);
          if(ierr != TRUE) break;
          Text name;
          ierr = Validate(name_box,name);
          if(ierr != TRUE) break;
```

```
          Element string;
          ierr = Validate(select_box,string);
          if(ierr != TRUE) break;

// set the new name
          set_names(string,name,next);

// restart select
          Select_start(select_box);
          Set_data(integer_box,next);
          Set_data(message,"new name \"" + name + To_text(next-1) + "\" ok");
        }
        break;
      }
      case Get_id(process) : {
        Integer ierr;

// validate name and text size
        Integer next;
        ierr = Validate(integer_box,next);
        if(ierr != TRUE) break;
        Text name;
        ierr = Validate(name_box,name);
        if(ierr != TRUE) break;

// validate model
        if(page == 1) {
          Element string;
          ierr = Validate(select_box,string);
          if(ierr != TRUE) break;
          set_names(string,name,next);
          Set_data(message,"new name \"" + name + To_text(next-1) + "\" ok");
        } else if(page == 2) {
          Model model;
          ierr = Validate(model_box,GET_MODEL_ERROR,model);
          if(ierr != MODEL_EXISTS) break;
          Integer no_strings = next;

          set_names(model,name,next);
          no_strings = next - no_strings;
          Set_data(message, To_text(no_strings) + " new name(s) were set");
```

```
        } else if(page == 3) {
          View view;
          ierr = Validate(view_box,GET_VIEW_ERROR,view);
          if(ierr != VIEW_EXISTS) break;
          Integer no_strings = next;
          set_names(view,name,next);
          no_strings = next - no_strings;
          Set_data(message, To_text(no_strings) + " new name(s) were set");
        }
        Set_data(integer_box,next);

// display data
        break;
      }
    }
  }
}
void main()

//-------------------------------------------------------
{
  manage_a_panel();
}
```

Example 13

```
//-----------------------------------------------------------------------
// Programmer          Van Hanh Cao
// Date                16/07/99
// 12d Model           V4.0
// Version             1.0
// Macro Name          Textto3d_panel
// Description
// User is asked to select view, model or a text string that contains
// the text strings. The macro will create a 3d point string at those text
// positions, and then put this string in a user selected model.If there
// is no user specified model, the default model "0", will be created
// and used.
// Note - this example uses a full 12d Model Panel rather than
// a simple console that the examples 1 to 10 used
//-----------------------------------------------------------------------
// Update/Modification
// (C) Copyright 1990-2011 by 12d Solutions Pty Ltd. All Rights Reserved
// This macro, or parts thereof, may not be reproduced in any form without
// permission of 12d Solutions Pty Ltd
//-----------------------------------------------------------------------
#include "set_ups.H"

#define MAX_NO_POINTS 1000
Integer get_text_points(Model model,Dynamic_Element &strings)
{
  Dynamic_Element elts;
  Integer         no_elts;
  Get_elements(model,elts,no_elts);
  for(Integer i=1;i<=no_elts;i++) {
    Element string;
    Get_item(elts,i,string);
    Text string_type;
    Get_type(string,string_type);
    if(string_type == "Text")  Append(string,strings);
  }
  return(0);
}
Integer get_text_points(View view,Dynamic_Element &strings)

{
  Dynamic_Text models;
  Integer      no_models;
  View_get_models(view,models);
  Get_number_of_items(models,no_models);
  for(Integer i=1;i<=no_models;i++) {
```

```
    Text model_name;
    Get_item(models,i,model_name);
    Model model;
    Get_model(model_name);
    if(!Model_exists(model)) continue;
    get_text_points(model,strings);
  }
  return(0);
}
Integer make_string(Model &tmodel,Dynamic_Element &strings,Real dx,
                    Real dy,Real maxz,Real minz)

//-----------------------------------------------------------------
// Create a 4d string with point numbers for each point in the strings
// from setout_model.
// Begin the point numbers at start_no and leave start_no as the next
// point number.
//-----------------------------------------------------------------
{
  Integer no_strings;
  Get_number_of_items(strings,no_strings);
  if(no_strings == 0) return(-1);
  Integer count = 1;
  Real                    x[MAX_NO_POINTS],y[MAX_NO_POINTS],z[MAX_NO_POINTS];

  for (Integer i=1;i<=no_strings;i++) {
    Text    string_type;
    Element string;
    Get_item(strings,i,string);
    Get_type(string,string_type);
    if(string_type == "Text") {
      Text t_z;
      Get_text_value(string, t_z);
      Dynamic_Text dtext;

      From_text(t_z,dtext);
      Integer no_text;
      Get_number_of_items(dtext,no_text);
      if(no_text != 1) continue;
      Real temp;
      if (From_text(t_z,temp) == 0) {
```

```
        z[count] = temp;
        if(z[count]<maxz && z[count]>minz) {
          Get_text_xy(string,x[count],y[count]);
          x[count] += dx;
          y[count] += dy;
          count++;
        }
      }
    }
  }
  count--;

  Element new_string;
  new_string = Create_3d(x,y,z,count);
  Set_model(new_string, tmodel);
  Set_breakline(new_string, 0);
  Calc_extent(tmodel);
  return(0);
}
void manage_a_panel()
// ---------------------------------------------------------
{
  Panel          panel   = Create_panel("Convert text strings to 3d string");
  Vertical_Group vgroup  = Create_vertical_group(0);
  Message_Box    message = Create_message_box(" ");
  Integer no_choices = 2;
  Text    choices[5];
  choices[1] = "Model";
  choices[2] = "View";
  Choice_Box pages_box = Create_choice_box("Data source",message);
  Set_data(pages_box,no_choices,choices);
  Set_data(pages_box,choices[1]);
  Append(pages_box,vgroup);

// create 3 vertical groups for each page of widgets
  Vertical_Group    g1  = Create_vertical_group(-1);      Set_border(g1,0,0);
  Vertical_Group    g2  = Create_vertical_group(-1);   Set_border(g2,0,0);
// add these groups to the pages widget
  Widget_Pages pages = Create_widget_pages();
  Append(g1,pages);
  Append(g2,pages);
```

```
// page 1
  Model_Box model_box = Create_model_box("Model containing text", message,
CHECK_MODEL_MUST_EXIST);
  Append(model_box,g1);

// page 2
  View_Box view_box = Create_view_box("View name", message,    CHECK_VIEW_MUST_EXIST);
  Append(view_box,g2);
  Model_Box model_box2 = Create_model_box("Model for 3d points"   , message,
        CHECK_MODEL_CREATE);
  Real_Box  dx_box     = Create_real_box ("Horizontal offset (dx)" ,message);
  Real_Box  dy_box     = Create_real_box("Vertical offset (dy)"   ,message);
  Real_Box  maxz_box   = Create_real_box("Max z value"           ,message);
  Real_Box  minz_box   = Create_real_box("Min z value"           ,message);
  Set_optional(maxz_box,1);
  Set_optional(minz_box,1);

// default data
  Set_data(dx_box    ,0.0);
  Set_data(dy_box    ,0.0);
  Append(pages_box ,vgroup);
  Append(pages     ,vgroup);
  Append(model_box2,vgroup);
  Append(dx_box    ,vgroup);
  Append(dy_box    ,vgroup);
  Append(maxz_box  ,vgroup);
  Append(minz_box  ,vgroup);
  Append(message   ,vgroup);

// buttons along the bottom
  Horizontal_Group bgroup = Create_button_group();
  Button process = Create_button("&Process" ,"count");
  Button finish  = Create_button("&Finish"  ,"finish");
  Append(process ,bgroup);
  Append(finish  ,bgroup);
  Append(vgroup  ,panel);
  Append(bgroup  ,panel);

// set page 1 active
```

```
Integer page = 1;
Set_page(pages,page);
Show_widget(panel);
Integer doit = 1;
while(doit) {
  Integer id;
  Text    cmd;
  Text    msg;
  Integer ret = Wait_on_widgets(id,cmd,msg);
  if(cmd == "keystroke") continue;
  Dynamic_Element strings;
  switch(id) {
    case Get_id(panel) : {
      if(cmd == "Panel Quit") doit = 0;
      break;
    }
    case Get_id(finish) : {
      if(cmd == "finish") doit = 0;
      break;
    }
    case Get_id(pages_box) : {
      Text page_text;
      Integer ierr = Validate(pages_box,page_text);
      if(ierr != TRUE) {
        Set_data(message,"bad page");
        break;
      }
      if(page_text == choices[1]) {
                              page = 1;
      } else if(page_text == choices[2]) {
        page = 2;
      } else {
        page = 0;
      }
      Set_page(pages,page);
      break;
    }
    case Get_id(process) : {
      Integer ierr;
// validate model box
      Model tmodel;
```

```
ierr = Validate(model_box2,GET_MODEL_CREATE,tmodel);
if(ierr != MODEL_EXISTS) break;
Real dx,dy;
ierr = Validate(dx_box,dx);
if(ierr != TRUE) break;
ierr = Validate(dy_box,dy);
if(ierr != TRUE) break;
Real maxz =  9999.9, minz = -9999.9;
Text temp_max,temp_min;
Get_data(maxz_box,temp_max);
if(temp_max != "") {
  Real temp;
  ierr = Validate(maxz_box,temp);
  if(ierr != TRUE) break;
  maxz = temp;
}
Get_data(minz_box,temp_min);
if(temp_min != "") {
  Real temp;
  ierr = Validate(minz_box,temp);
  if(ierr != TRUE) break;
  minz = temp;
}
if(minz >= maxz) {
  Set_data(message,"max z must be greater than min z");
  break;
}
if(page == 1) {
    Model   model;
    ierr = Validate(model_box,GET_MODEL_ERROR,model);
    if(ierr != MODEL_EXISTS) break;
    get_text_points(model,strings);
  } else if(page == 2) {
    View    view;
    ierr = Validate(view_box,GET_VIEW_ERROR,view);
    if(ierr != VIEW_EXISTS) break;
    get_text_points(view,strings);
  } else {
  Set_data(message,"bad choice");
  break;
}
```

```
        make_string(tmodel,strings,dx,dy,maxz,minz);
        Text tmodel_name;
        Get_name(tmodel,tmodel_name);
        Set_data(message,"model " + tmodel_name + " created");
        Null(strings);
        break;
      }
    }
  }
}

void main()
//-------------------------------------------------------
{
  manage_a_panel();
}
```

Example 14

```
#include "set_ups.H"

Integer my_function(Model model1_model, File file1_file ,Tin tin1_tin,Real real1_value,
     View view1_view ,Text input1_text,Integer colour1_value,Integer tick1_value,
     Text select1_text,Real select1_x,Real select1_y ,Real select1_z ,
     Real  select1_prof_chainage ,Real  select1_prof_z ,Element select1_string,
     Integer xyz1_value)
{
  return 0;
}
Integer go_panel(
       Text panel_title   ,                    Text panel_help  , Text file_default          ,
       Integer draw1_on    ,Text draw1_name    , Integer draw1_box_width, Integer draw1_box_height,
       Integer choice1_on ,Text choice1_title , Text    choice1_name   , Text choice1_help, Text
choice1_title_default , Text choice1[]       , Integer no_choice1,
       Integer model1_on  ,Text model1_title  , Text    model1_name    , Text model1_help , Text
model1_title_default  , Text model1_ceme      ,
      Integer file1_on   ,Text file1_title   , Text    file1_name     , Text file1_help  , Text file1_title_default   ,
Text file1_rw         , Text file1_ext ,
      Integer tin1_on    ,Text tin1_title    , Text    tin1_name      , Text tin1_help   , Text tin1_title_default    ,
Integer tin1_supertin ,
       Integer real1_on   ,Text real1_title   , Real    real1_value    , Text real1_help  , Text
real1_title_default   , Text real1_check      , Real real1_low   , Real real1_high ,
       Integer view1_on   ,Text view1_title   , Text    view1_name     , Text view1_help  , Text
view1_title_default   ,
       Integer input1_on  ,Text input1_title  , Text    input1_text    , Text input1_help , Text
input1_title_default  , Text input1_not_blank ,
       Integer colour1_on ,Text colour1_title , Text    colour1_text   , Text colour1_help, Text
colour1_title_default ,
       Integer select1_on ,Text select1_title , Text    select1_text   , Text select1_help, Text
select1_title_default , Text select1_type,Text select1_go,
      Integer tick1_on   ,Text tick1_title   , Integer tick1_value    , Text tick1_help  , Text tick1_title_default
,
       Integer xyz1_on    ,Text xyz1_title    , Integer xyz1_value     , Text xyz1_help   , Text
xyz1_title_default    ,
       Integer process_on, Text process_title ,                        Text process_finish_help)
{
  // ==============================================
  // get defaults at the start of a routine and set up the panel

  Integer ok=0;

  //-----------------------------------------------------------------------------
  //                    CREATE THE PANEL
  //-----------------------------------------------------------------------------
  Panel panel = Create_panel(panel_title);
  Vertical_Group  vgroup = Create_vertical_group(0);
  Message_Box message_box = Create_message_box("");
  //-------------------------------------------------------------------------------------
  //         draw1_box
  //-------------------------------------------------------------------------------------


  Horizontal_Group hgroup_box = Create_horizontal_group(0);
  Draw_Box draw1_box = Create_draw_box(draw1_box_width,draw1_box_height,0);
```

```
if (draw1_on) Append(draw1_box,hgroup_box);
// ---------------- choice1_ name ------------------------------------------------------------

Choice_Box choice1_box = Create_choice_box(choice1_title,message_box);
Set_data(choice1_box,no_choice1,choice1);
ok += Set_help(choice1_box,choice1_help);
if (choice1_on) Append(choice1_box,vgroup);

// ---------------- model1_ name -------------------------------------------------------------
// model1_name
Model_Box model1_box;
switch (model1_ceme) {
  case "c" : {
    model1_box = Create_model_box(model1_title,message_box,CHECK_MODEL_CREATE);
    break;
  }
  case "e" : {
    model1_box = Create_model_box(model1_title,message_box,CHECK_MODEL_EXISTS);
    break;
  }
  case "me" : {
    model1_box = Create_model_box(model1_title,message_box,CHECK_MODEL_MUST_EXIST);
    break;
  }
}
ok += Set_help(model1_box,model1_help);
if (model1_on) Append(model1_box,vgroup);

// ---------------- file1_ name ---------------------------------------------------------------
File_Box file1_box;
switch (file1_rw) {
  case "c" : {
    file1_box = Create_file_box(file1_title,message_box,CHECK_FILE_CREATE,file1_ext);
    break;
  }
  case "w" : {
    file1_box = Create_file_box(file1_title,message_box,CHECK_FILE_WRITE,file1_ext);
    break;
  }
  case "n" : {
    file1_box = Create_file_box(file1_title,message_box,CHECK_FILE_NEW,file1_ext);
    break;
  }
  case "r" : {
    file1_box = Create_file_box(file1_title,message_box,CHECK_FILE_MUST_EXIST,file1_ext);
    break;
  }
  case "a" : {
    file1_box = Create_file_box(file1_title,message_box,CHECK_FILE_APPEND,file1_ext);
    break;
  }
}
ok += Set_help(file1_box,file1_help);
if (file1_on) Append(file1_box,vgroup);

// ---------------- tin1_ ---------------------------------------------------------------
Tin_Box tin1_box = Create_tin_box(tin1_title,message_box,CHECK_TIN_MUST_EXIST);
ok += Set_supertin(tin1_box,tin1_supertin);
```

```
  ok += Set_help(tin1_box,tin1_help);
  if (tin1_on)   Append(tin1_box,vgroup);

  // ----------------- real1_ data ------------------------------------------------------------
  Real_Box real1_box = Create_real_box(real1_title,message_box);
  ok += Set_help(real1_box,real1_help);
  if (real1_on) Append(real1_box,vgroup);

  // ----------------- view1_ data ------------------------------------------------------------
  View_Box view1_box = Create_view_box(view1_title,message_box,CHECK_VIEW_MUST_EXIST);
  ok += Set_help(view1_box,view1_help);
  if (view1_on) Append(view1_box,vgroup);

  // ----------------- input1_ ----------------------------------------------------------------
  Input_Box input1_box =       Create_input_box(input1_title,message_box);
  ok += Set_help(input1_box,input1_help);
  ok += Set_optional(input1_box,(input1_not_blank != "not blank"));
  if (input1_on) Append(input1_box,vgroup);

  // ----------------- colour1_ ---------------------------------------------------------------
  Colour_Box colour1_box =       Create_colour_box(colour1_title,message_box);
  ok += Set_help(colour1_box,colour1_help);
  if (colour1_on) Append(colour1_box,vgroup);

  // ----------------- select1_ ---------------------------------------------------------------
  Element select1_string;
  Real select1_x,select1_y,select1_z,select1_prof_chainage,select1_prof_z;
  Select_Button select1_button =
Create_select_button(select1_title,SELECT_STRING,message_box);
  ok += Set_help(select1_button,select1_help);
  if(select1_type != "") ok += Set_select_type(select1_button,select1_type);
  if (select1_on) Append(select1_button,vgroup);

  // ----------------- tick1_ -----------------------------------------------------------------
  Named_Tick_Box tick1_box =     Create_named_tick_box(tick1_title,tick1_value,"");
  ok += Set_help(tick1_box,tick1_help);
  if (tick1_on) Append(tick1_box,vgroup);

  // ----------------- xyz1_ ------------------------------------------------------------------
  Real xyz1_xvalue,xyz1_yvalue,xyz1_zvalue;
  XYZ_Box xyz1_box =     Create_xyz_box(xyz1_title,message_box);
  ok += Set_help(xyz1_box,xyz1_help);
  if (xyz1_on) Append(xyz1_box,vgroup);

// ----------------- message area -----------------------------------------------------------
  Append(message_box,vgroup);

  // ----------------- bottom of panel buttons ------------------------------------------------------------
  Horizontal_Group button_group = Create_button_group();
  Button process_button = Create_button(process_title,"process");
  ok += Set_help(process_button,process_finish_help);
  if(process_on) Append(process_button,button_group);
  Button finish_button = Create_button("Finish","finish");
  ok += Set_help(finish_button,process_finish_help);
  Append(finish_button,button_group);
  Append(button_group,vgroup);
  Append(vgroup,hgroup_box);
  Append(hgroup_box,panel);
```

```
  // ---------------- display the panel ------------------------------------------------------------
  Integer wx = 100,wy = 100;
  Show_widget(panel,wx,wy);

  //-----------------------------------------------------------------------------------------------
  //         draw bit map
  //-----------------------------------------------------------------------------------------------
  if (draw1_on) {
    Get_size(draw1_box,draw1_box_width,draw1_box_height);
    Start_batch_draw(draw1_box);
    //// the following RGB values match my screen setup
    //// set it to Clear(draw_box,-1,0,0) to see if you can get the window default
    //// or if that doesn't work set it to your RGB values
    Clear(draw1_box,192,192,192);
    Draw_transparent_BMP(draw1_box,draw1_name,0,draw1_box_height);
    End_batch_draw(draw1_box);
   }
  // ---------------------------------------------------------------------------------------------------------------
  //                    GET AND VALIDATE DATA
  // ---------------------------------------------------------------------------------------------------------------
  Integer done = 0;
  while (1) {
    Integer id,ierr;
    Text cmd,msg;
    Wait_on_widgets(id,cmd,msg);
    #if DEBUG
      Print(" id  <"+To_text(id));
      Print("> cmd <"+cmd);
      Print("> msg <"+msg+">\n");
    #endif

//-----------------------------------------------------------------------------------------------
//  first process the command that are common to all wigits or are rarely processed by the wigit ID
//-----------------------------------------------------------------------------------------------
    switch(cmd) {
      case "keystroke" : {
        continue;
        break;
      }
      case "set_focus"  :
      case "kill_focus" : {
        continue;
        break;
      }
      case "Help" : {
        Winhelp(panel,"12d.hlp",'a',msg);
        continue;
        break;
      }
    }

//-----------------------------------------------------------------------------------------------
//  process each event by the wigit id
//  most wigits do not need to be processed until the PROCESS button is pressed
//  only the ones that change the appearance of the panel need to be processed in this loop
//-----------------------------------------------------------------------------------------------
    switch(id) {
```

```
      case Get_id(panel) :{
        if(cmd == "Panel Quit") return 1;
        if(cmd == "Panel About") continue;
        break;
      }
      case Get_id(finish_button) : {
        Print("Normal Exit\n");
        return(0);
        break;
      }
      case Get_id(select1_button) : {
        switch (cmd) {
          case "accept select" : {
            if(Get_subtext(select1_go,1,2) != "go") continue;
            break;
          }
/*
// other select cmds
          case "cancel select" : {
            continue;
            break;
          }
*/
        }
        continue;
        break;
       }
      case Get_id(process_button) : {
//-----------------------------------------------------------------------------
//  verify / retrieve all the data in the panel
//-----------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------
//                    select box
//-------------------------------------------------------------------------------------------
        Validate(select1_button,select1_string);
        Get_select_coordinate(select1_button,select1_x,select1_y,select1_z,select1_prof_chainage,
   select1_prof_z);
 // create the file handle
//-----------------------------------------------------------------------------
//                    MODEL CHECK
//-----------------------------------------------------------------------------
        Model model1_model;
        if(model1_on) {
          switch (model1_ceme) {
            case "c" : {
              if(Validate(model1_box,GET_MODEL_CREATE,model1_model) != MODEL_EXISTS)
                          continue;
              break;
            }
            case "e" : {
              if(Validate(model1_box,GET_MODEL,model1_model) != MODEL_EXISTS) continue;
              break;
            }
            case "me" : {
             if(Validate(model1_box,GET_MODEL_ERROR,model1_model) != MODEL_EXISTS) continue;
              break;
            }
          }
```

```
        }
        Tin tin1_tin;
        if(tin1_on) {
          if(Validate(tin1_box,CHECK_TIN_MUST_EXIST,tin1_tin) != TIN_EXISTS) continue;
          ok += Get_data(tin1_box,tin1_name);
        }
        View view1_view;
        if(view1_on) {
          if(Validate(view1_box,CHECK_VIEW_MUST_EXIST,view1_view) != VIEW_EXISTS) continue;
          ok += Get_data(view1_box,view1_name);
        }
        if(real1_on) {
          if(Validate(real1_box,real1_value) == !OK) continue;
        }
        if(input1_on) {
          input1_text = "*******";
          if(!Validate(input1_box,input1_text)) continue;
          if ((input1_text == "") && (input1_not_blank == "not blank")) {
            Set_data(message_box,"Text must be entered");
            continue;
          }
        }
        Integer colour1_value;
        if(colour1_on) {
          if(!Validate(colour1_box,colour1_value)) continue;
          Get_data(colour1_box,colour1_text);
        }
// save the file checks for last
//-----------------------------------------------------------------------------------------
//         FILE CHECK BEFORE PROCESSING
//-----------------------------------------------------------------------------------------
//  if the file already exists
//  Error_prompt(To_text(Validate(file1_box,GET_FILE_CREATE,file1_name)));
//   replace y/n n=NO_FILE_ACCESS y = NO_FILE
//  Error_prompt(To_text(Validate(file1_box,GET_FILE_WRITE,file1_name)));
//  append  y/n n= NO_FILE y = FILE_EXISTS
//  Error_prompt(To_text(Validate(file1_box,GET_FILE_NEW,file1_name)));
//  new     error_message =  FILE_EXISTS
// Error_prompt(To_text(Validate(file1_box,GET_FILE_MUST_EXIST,file1_name)));
        // must exist ok message =  FILE_EXISTS
        //Error_prompt(To_text(Validate(file1_box,GET_FILE_APPEND,file1_name)));
        // append  y/n n =  NO_FILE y = FILE_EXISTS

        //  if the file does not exist
        //Error_prompt(To_text(Validate(file1_box,GET_FILE_CREATE,file1_name)));
        // message will be created = NO_FILE
        //Error_prompt(To_text(Validate(file1_box,GET_FILE_WRITE,file1_name)));
        // message will be created = NO_FILE
        //Error_prompt(To_text(Validate(file1_box,GET_FILE_NEW,file1_name)));
        // message will be created = NO_FILE
        //Error_prompt(To_text(Validate(file1_box,GET_FILE_MUST_EXIST,file1_name)));
        // error message =  NO_FILE
        //Error_prompt(To_text(Validate(file1_box,GET_FILE_APPEND,file1_name)));
        // message will be created = NO_FILE

        File file1_file;
        if(file1_on) {
          switch (file1_rw) {
```

```
          case "c" : {
            if(Validate(file1_box,GET_FILE_CREATE,file1_name) == NO_FILE_ACCESS) continue;
            break;
          }
          case "w" : {
            if(Validate(file1_box,GET_FILE_WRITE,file1_name) == NO_FILE_ACCESS) continue;
            break;
          }
          case "n" : {
            if(Validate(file1_box,GET_FILE_NEW,file1_name) != NO_FILE) continue;
            break;
          }
          case "r" : {
            if(Validate(file1_box,GET_FILE_MUST_EXIST,file1_name) != FILE_EXISTS) continue;
            break;
          }
          case "a" : {
            if(Validate(file1_box,GET_FILE_APPEND,file1_name) == NO_FILE_ACCESS) continue;
            break;
          }
        }
        ok += File_open(file1_name,file1_rw,file1_file);
      } // if file1_on
//-----------------------------------------------------------------------------------------
//                this is the function call to your program
//-----------------------------------------------------------------------------------------
      my_function(model1_model          ,file1_file         ,tin1_tin           ,real1_value,
                  view1_view            ,input1_text        ,colour1_value      ,tick1_value,
                  select1_text          ,select1_x          ,select1_y          ,select1_z,
                  select1_prof_chainage ,select1_prof_z     ,select1_string,
                  xyz1_value);

      if(select1_on && (select1_go == "go again")) {
        Set_data(message_box,"select another "+select1_type+" string: <RB> to cancel");
        Select_start(select1_button);
        continue;
      } else  Set_data(message_box,"Processing complete");
    } break; // process
    default : {
      continue;
    }
  }  // switch id
 }  // while !done
 return ok;
}

void main() {
 Clear_console();
 Text macro_help = "help";
 //-----------------------------------------------------------------------------
 //                    Example call
 //-----------------------------------------------------------------------------
 Integer no_choice1 = 3;
 Text choice1[no_choice1];
 choice1[1] = "choice 1";
 choice1[2] = "choice 2";
 choice1[3] = "choice 3";
```

```
//      wigit label         , default data  , help assoc key   , default data name , check data
  go_panel(
        "Sample Panel"          ,                macro_help , "sample.mdf"      ,
        1,"12dlogo2.bmp"        , 180, 180,
        1,"Choice1_title"     , choice1[1]    , macro_help , "choice1"         , choice1, no_choice1,
        1,"Model_title"       , ""            , macro_help , "model1"          , "c"        ,
        1,"Input file"        , ""            , macro_help , "file1"           , "r"        , "*.txt" ,
        1,"tin1_title"        , "tin name xx" , macro_help , "tin1"            ,  1,
        1,"real1_title"       , 99.9          , macro_help , "real1"           , "check data", 0.0 , 100.0 ,
        1,"view1_title"       , "1"           , macro_help , "view1"           ,
        1,"input1_title"      , "input text"  , macro_help , "input1"          , "not blank" ,
        1,"Section colour"    , "red"         , macro_help , "colour1"         ,
        1,"select1_title"     , ""            , macro_help , "select1"         ,"" ,"no go again",
        1,"tick title"        , 0             , macro_help , "tick1"           ,
        1,"xyz1_title"        , 0             , macro_help , "xyz1"            ,
        1,"Process",                            macro_help );
// Select codes
   // go          executes the process command automatically after an accept
   // go again    start another select immediately after the last accept
// Model codes
   // c   message it exists or a create message if it does not exist
   // e   message it exists or a message that it does not exist
   // me  message it exists or a error message if the model does not exist
//File codes
   // n   create a new file and will not overwrite an existing file
   // c   asks if you want to overwrite
   // w   asks if you want to append (overwrites if you say no)
   // a   asks if you want to append
   // r   the file must exist
}
```

Example 15

```
// ---------------------------------------------------------------------------------------------
// Macro:        macro_function.4dm
// Author:       alg
// Organization: 12d Solutions Pty Ltd
// Date:         Tue Sep 15 19:02:19 1998
// Modified      ljg
// Date          11 August 2011
// ---------------------------------------------------------------------------------------------
// Brief description
// Macro_Function to parallel a string between two chainages.
// ---------------------------------------------------------------------------------------------
// Description
// Macro_Function to parallel a string between two chainages.
// A strinng is selected and then two chainages to offset between.
// An offset value is given and optionally a new name, colour and model
// for the created string. If name, colour or model is blank,
// then the property is taken from the selected string.
//
// Note - this example uses a full 12d Model Panel rather than
// a simple console that the examples 1 to 10 used
// ---------------------------------------------------------------------------------------------
// Update/Modification
//
//
// (C) Copyright 1990-2011 by 12d Solutions Pty Ltd. All Rights Reserved
//
// This macro, or parts thereof, may not be reproduced in any form without
// permission of 12d Solutions Pty Ltd
// ---------------------------------------------------------------------------------------------
//
// Macro_Function Dependencies
//
//      "string"   Element
//
// Macro_Function attributes
//
//      "offset"            Real
//      "start point"       Text
//      "end point"         Text
//      "new name"          Text
//      "new model"         Text
//      "new colour"        Text
//      "functype"          Text
//
//      "model"             Uid
//      "element"           Uid
// ---------------------------------------------------------------------------------------------

#include "Set_ups.H"

Integer get_chainage_value(Element string,Text mode,Text ch_text,Real &chainage)
// ---------------------------------------------------------------------------------------------
// Convert the text to chainage and check that it is on the string.
// Blank text means use string start/end chainage.
```

```
// ------------------------------------------------------------------------------------------
{
  Integer ierr;
  Real    start,end;

  ierr = Get_chainage(string,start);
  if (ierr != 0) return(1);

  ierr = Get_end_chainage(string,end);
  if (ierr != 0) return(1);

  if(mode == "start") {  // if text is blank then use string start chainage
    if(ch_text == "") {
      chainage = start;
      return(0);
    } else {
      ierr = From_text(ch_text,chainage);
      if (ierr != 0) return(1);
    }

  } else if(mode == "end") {
    if(ch_text == "") {
      chainage = end;
      return(0);
    } else {
      ierr = From_text(ch_text,chainage);
      if(ierr != 0) return(1);
    }

  } else {
    return (1);   // invalid mode
  }

// check if chainage is on the string

  if(chainage > end) return(1);
  if(chainage < start) return(1);
  return(0);
}
void set_error(Macro_Function macro_function,Text error)
// ---------------------------------------------------------------------
// If there is a non blank error message than store it as the function attribute
// if the error message is blank, remove the error message attribute
// ---------------------------------------------------------------------
{
  if(error != "") {
    Set_function_attribute(macro_function,"error message",error);
  } else {
    Function_attribute_delete(macro_function,"error message");
  }
}
Integer recalc_macro(Text function_name)
// ---------------------------------------------------------------------
// Do the processing for the macro.
//
// recalc_macro is used to do the recalcs where all the panel answers are recorded
// as function depecencies and attributes.
//
```

```
// recalc_macro is also used to do the processing for the first run of the panel,
// and for the Edit case where the panel and answers are displayed and can be modified.
//
// In the first run and Edit case , the panel information has been loaded into
// function dependecies and function attributes so the information
// is all there in the function just like it is for a Recalc.
//
// The only major difference is that for the first run, there are no strings etc
// created from a previous run that need to be deleted.
//
// In all cases, all panel answers must be checked before continuing to calculations
// since there is no guarentee that something hasn't been deleted since the
// last Recalc.
//
// For example, in this macro, the string to be paralleled may have been deleted.
//
// NOTE: Before any processing takes place, any strings that were created in
// in a previous run and are to be deleted, must first be checked that they
// can be deleted. For example, that they are not locked.
// If they can't be deleted then the macro terminates with an error message.
// ---------------------------------------------------------------------------
{
  Integer ierr;

  Macro_Function macro_function;
  Get_macro_function(function_name,macro_function);

  Element string;
  Get_dependancy_element(macro_function,"string",string);

  Real offset;
  Get_function_attribute(macro_function,"offset",offset);

  Text start_pt;
  Get_function_attribute(macro_function,"start point",start_pt);

  Text end_pt;
  Get_function_attribute(macro_function,"end point",end_pt);

  Text name_txt,name;
  Get_function_attribute(macro_function,"new name",name_txt);
  if(name == "") {
    Get_name(string,name);  // name is existing string name
  } else {
    name = name_txt;
  }

  Text model_txt;
  Model model;
  Uid   mid;
  Integer model_exists = 0;

  Get_function_attribute(macro_function,"new model",model_txt);
  if(model_txt == "") {
    ierr = Get_model(string,model);// model name is blank so use strings model
    model_exists = 1;
  } else if(Model_exists(model_txt)) {
    model = Get_model(model_txt);
```

```
    ierr = Get_id(model,mid);
    model_exists = 1;
  }

  if(model_exists) {
    ierr = Get_id(model,mid);
    if(Is_global(mid)) {    // check if model is shared from another project
      set_error(macro_function,"new model is write protected");
      return(-1);
    }
  }

// haven't created a new model if needed as yet. Wait to all validation is complete

  Text colour_txt;
  Integer colour;
  Get_function_attribute(macro_function,"new colour",colour_txt);
  if(colour_txt == "") {
    Get_colour(string,colour);  // colour is existing string colour
  } else {
    Convert_colour(colour_txt,colour);
  }

// are start and end chainages valid

  Real start_ch;
  if(get_chainage_value(string,"start",start_pt,start_ch) != 0) {
    set_error(macro_function,"start chainage is bad");
    return(-1);
  }

  Real end_ch;
  if(get_chainage_value(string,"end",end_pt,end_ch) != 0) {
    set_error(macro_function,"end chainage is bad");
    return(-1);
  }

// get the parallel elt from a previous run

  Integer first_time = 0;

  Uid eid;
  if(Get_function_attribute(macro_function,"model"  ,mid) != 0) first_time = 1;
  if(Get_function_attribute(macro_function,"element",eid) != 0) first_time = 1;

  Element elt;
  if(Get_element(mid,eid,elt) != 0) first_time = 1;  // can't find elt by mid and eid

  if(first_time == 0) {  // not the first time and previous created elt has been found by mid and eid
                         // check elt is not locked since it is going to be modified
    Integer locks;
    Get_write_locks(elt,locks);

    if(locks > 0) {
      set_error(macro_function,"paralled string is locked");
      return(-1);
    }
  }
```

```
// compute new string

  Element left_str,mid_str,right_str;

// get partial string

  if(Clip_string(string,start_ch,end_ch,left_str,mid_str,right_str) != 0) {

    set_error(macro_function,"cannot get string between clip points");
    return(-1);
  }

// parallel the string between the two chainages

  Element elt_new;
  ierr = Parallel(mid_str,offset,elt_new);

// clean up clipping bits

  Element_delete(left_str);
  Element_delete(mid_str);
  Element_delete(right_str);

// did parallel work ?

  if(ierr != 0) {

    set_error(macro_function,"parallel failed");
    return(-1);
  }

// we can replace string

  Element_draw(elt,0);     // draw elt as blank

  if(!model_exists) model = Create_model(model_txt);  // model doesn't exist so create it

  if(first_time) {

    Set_model(elt_new,model);  // put string in model
    elt = elt_new;

// store details of the created string in function attributes

    Get_id(model,mid);
    Get_id(elt  ,eid);

    Set_function_attribute(macro_function,"model"  ,mid);
    Set_function_attribute(macro_function,"element",eid);

  } else {

// replace contents of string - so eid will stay the same
// copy switch attributes !

    Text sw1;  Integer a1 = Get_attribute(elt,"start switch",sw1);
    Text sw2;  Integer a2 = Get_attribute(elt,"end switch"  ,sw2);
```

```
    String_replace(elt_new,elt);

    if(a1 == 0) Set_attribute(elt,"start switch",sw1);
    if(a2 == 0) Set_attribute(elt,"end switch"  ,sw2);

// store details of the created string in function attributes
// the string has same Uid. The model Uid may have cdhanged

    Get_id(model,mid);
    Set_function_attribute(macro_function,"model" ,mid);

// clean up

    Element_delete(elt_new);
  }

// set name, model  and colour details

  Set_name  (elt,name);
  Set_model (elt,model);
  Set_colour(elt,colour);

// parallel finished

  Element_draw(elt);

// tell element what function it belongs to

  Uid fid;  Get_id(macro_function,fid);

  Set_function_id(elt,fid);

// finished

  return(0);
}


Integer show_panel(Text function_name,Integer edit)
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
{
  Macro_Function macro_function;
  Get_macro_function(function_name,macro_function);

  Panel           panel   = Create_panel("Parallel String Section");
  Vertical_Group   vgroup  = Create_vertical_group(0);
  Message_Box      message = Create_message_box(" ");

// function

  Function_Box function_box = Create_function_box("Function name", message,
                                        CHECK_FUNCTION_CREATE,RUN_MACRO_T);

  Set_type(function_box,"parallel_part");     // set the unique type for the Macro_Function

  Append(function_box,vgroup);
```

```
  if(edit) Set_data(function_box,function_name);

// string

  New_Select_Box select_box = Create_new_select_box("String to parallel","Select
string",SELECT_STRING,message);

  Append(select_box,vgroup);

  if(edit) {     // this is when -function_edit is found
                 // get the panel data from the last run

    Element string;
    Get_dependancy_element(macro_function,"string",string);

// check the model is not shared from another project.
// If it is then the model can't be used for the new string.

    Set_data(select_box,string);
  }

// offset distance

  Real_Box value_box = Create_real_box("Offset",message);
  Append(value_box,vgroup);

  if(edit) {     // this is when -function_edit is found
                 // get the panel data from the last run
    Real offset;
    Get_function_attribute(macro_function,"offset",offset);
    Set_data(value_box,offset);
  }

// chainage of start point - optional. If not filled in then use string start

  Chainage_Box start_box = Create_chainage_box("Start chainage",message);
  Set_optional(start_box,1);
  Append(start_box,vgroup);

  if(edit) {     // this is when -function_edit is found
                 // get the panel data from the last run
    Text start_value;
    Get_function_attribute(macro_function,"start point",start_value);
    Set_data(start_box,start_value);
  }

// chainage of end point - optional. If not filled in then use string end

  Chainage_Box end_box = Create_chainage_box("End chainage",message);
  Set_optional(end_box,1);
  Append(end_box,vgroup);

  if(edit) {     // this is when -function_edit is found
                 // get the panel data from the last run
    Text end_value;
    Get_function_attribute(macro_function,"end point",end_value);
    Set_data(end_box,end_value);
```

```
  }

// details about new string

  Name_Box name_box = Create_name_box("New name",message);
  Set_optional(name_box,1);
  Append(name_box,vgroup);

  if(edit) {     // this is when -function_edit is found
                 // get the panel data from the last run

    Text name;
    Get_function_attribute(macro_function,"New name",name);
    Set_data(name_box,name);
  }

  Model_Box model_box = Create_model_box("New model",message,CHECK_MODEL_CREATE);
  Set_optional(model_box,1);
  Append(model_box,vgroup);

  if(edit) {     // this is when -function_edit is found
                 // get the panel data from the last run

    Text model_txt;
    Get_function_attribute(macro_function,"new model",model_txt);
    Set_data(model_box,model_txt);
  }

  Colour_Box colour_box = Create_colour_box("New colour",message);
  Set_optional(colour_box,1);
  Append(colour_box,vgroup);

  if(edit) {     // this is when -function_edit is found
                 // get the panel data from the last run
    Integer colour;
    Text colour_txt;
    Get_function_attribute(macro_function,"new colour",colour_txt);
    Set_data(colour_box,colour_txt);
  }

// message box

  Append(message,vgroup);

  Horizontal_Group bgroup = Create_button_group();

  Button       compute = Create_button       ("Parallel","compute");
  Button       finish  = Create_finish_button("Finish"  ,"Finish" );

  Append(compute,bgroup);
  Append(finish ,bgroup);

  Append(bgroup,vgroup);
  Append(vgroup,panel);

  Show_widget(panel);

// reset edit
```

```
  edit = 0;

// was there an error message !

  if(Function_attribute_exists(macro_function,"error message")) {

    Text error;
    Get_function_attribute(macro_function,"error message",error);

    Set_data(message,"last error was: " + error);
  }

// now wait on events

  Integer doit = 1;

  while(doit) {

    Integer id;
    Text    cmd;
    Text    msg;
    Integer ret = Wait_on_widgets(id,cmd,msg);  // this processes standard messages first ?

    if(cmd == "keystroke") continue;

    switch(id) {

      case Get_id(panel) : {

        if(cmd == "Panel Quit") {  // X on panel top right hand corner clicked
          doit = 0;
        }
        break;
      }

      case Get_id(finish) : {  //  finish button clicked

        doit = 0;

        break;
      }

      case Get_id(function_box) : {  // a function of this type has been selected. So the
                                     // information from that function needs to be put in the panel

        Function func;
        if(Validate(function_box,CHECK_FUNCTION_EXISTS,func) != FUNCTION_EXISTS) break;

        Get_data(function_box,function_name);
        if(Get_macro_function(function_name,macro_function) == 0) {

// load string

          Element string;
          Get_dependancy_element(macro_function,"string",string);

          Set_data(select_box,string);
```

```
// load offset

        Real offset;
        Get_function_attribute(macro_function,"offset",offset);

        Set_data(value_box,offset);

// start chainage

        Text start_val;
        Get_function_attribute(macro_function,"start point",start_val);

        Set_data(start_box,start_val);

// end chainage

        Text end_val;
        Get_function_attribute(macro_function,"end point",end_val);

        Set_data(end_box,end_val);

// new string details

        Text name;
        Get_function_attribute(macro_function,"new name",name);
        Set_data(name_box,name);

        Text model_txt;
        Get_function_attribute(macro_function,"new model",model_txt);
        Set_data(model_box,model_txt);

        Text colour_txt;
        Get_function_attribute(macro_function,"new colour",colour_txt);
        Set_data(colour_box,colour_txt);

// data retrieved

        if(Function_attribute_exists(macro_function,"error message")) {

          Text error;
          Get_function_attribute(macro_function,"error message",error);
          Set_data(message,"function retrieved - last error was: " + error);
        } else {
          Set_data(message,"function retrieved");
        }
      }
      break;
    }

    case Get_id(compute) : {

// for now - the only safe way to create a macro function is by
//          using Create_macro_function , NOT by Validate(Function,....)

      Get_data(function_box,function_name);
      if(Get_macro_function(function_name,macro_function) != 0) {
```

```
// create the function

        if(Create_macro_function(function_name,macro_function) != 0) {

          Error_prompt("failed to create function");
          break;
        }
      } else {

// stop other function type now!!!

        Function func;
        if(Validate(function_box,CHECK_FUNCTION_EXISTS,func) != FUNCTION_EXISTS) break;
      }
      Text type;

// validate string

      Element string;
      if(Validate(select_box,string) != TRUE) {

        Set_data(message,"string not valid");
        break;
      }

// validate offset

      Real offset;
      if(Validate(value_box,offset) != TRUE) break;

// start point

      Text start;
      Get_data(start_box,start);

      Real start_ch;
      if(get_chainage_value(string,"start",start,start_ch) != 0) {
        Set_error_message(start_box,"start chainage not valid");
        break;
      }

// end point

      Text end;
      Get_data(end_box,end);

      Real end_ch;
      if(get_chainage_value(string,"end", end,end_ch) != 0) {
        Set_error_message(end_box,"end chainage not valid");
        break;
      }

// new string details

      Text name;
      Integer val = Validate(name_box,name);
      if(val == 0) break;   // validation error in mame box
```

```
        Model model;
                                Text  model_txt;
                                Uid mid;
                                Integer ierr;

                                Get_data(model_box,model_txt);

                                if(model_txt == "") {   // model name is blank so use selected strings model.
                                                        // Need to check model is not shared from another project
         ierr = Get_model(string,model);
                                  ierr = Get_id(model,mid);
                                  if(Is_global(mid)) break;    // validation error in model box
                                } else if(Model_exists(model_txt)) {
                                   model = Get_model(model_txt);
                                  ierr = Get_id(model,mid);
                                  if(Is_global(mid)) break;    // can't add data to shared model
                                                        // validation error in model box
        }

        Integer colour;
        Text    colour_txt;
        val = Validate(colour_box,colour);
        if(val == 0) break;   // validation error in colour box

        if(val == NO_NAME) {
          colour_txt = "";
        } else {
          Convert_colour(colour,colour_txt);
        }

    // Store the panel information in the Macro_Function

        Delete_all_dependancies(macro_function);

        Set_function_attribute(macro_function,"functype"    ,"parallel_part");

        Add_dependancy_element(macro_function,"string"      ,string);

        Set_function_attribute(macro_function,"offset"      ,offset);
        Set_function_attribute(macro_function,"start point" ,start);
        Set_function_attribute(macro_function,"end point"   ,end);
        Set_function_attribute(macro_function,"new name"    ,name);
        Set_function_attribute(macro_function,"new model"   ,model_txt);
        Set_function_attribute(macro_function,"new colour"  ,colour_txt);

    // Now do the processing

        Integer res = recalc_macro(function_name);

        Text error;
        if(Get_function_attribute(macro_function,"error message",error) != 0) error = "ok";

        Set_data(message,error);

        if(res == 0) Set_finish_button(panel,1);
        break;
      }
    }
```

```
  }
  return(-1);
}
void main()
// -------------------------------------------------------
// this is where the macro starts
// -------------------------------------------------------
{
  Integer argc = Get_number_of_command_arguments();
  if(argc > 0) {

    Text arg;
    Get_command_argument(1,arg);

    if(arg == "-function_recalc") {

      Text function_name;
      Get_command_argument(2,function_name);

      recalc_macro(function_name);

    } else if(arg == "-function_edit") {

      Text function_name;
      Get_command_argument(2,function_name);

      show_panel(function_name,1);

    } else if(arg == "-function_delete") {

// not implimented yet

      Text function_name;
      Get_command_argument(2,function_name);

      Error_prompt("function_delete not implimented");

    } else if(arg == "-function_popup") {

// not implimented yet

      Text function_name;
      Get_command_argument(2,function_name);

      Error_prompt("function_popup not implimented");

    } else {

// normal processing ?

      Error_prompt("huh ? say what");
    }
  } else {

    show_panel("",0);
  }
}
```

A Appendix - Set_ups.h File

The file *set_ups.h* contains constants and values that are used in, or returned by, 12dPL supplied functions.

Before any of the constants or values in *set_ups.h* can be used, *set_ups. h* needs to be included in a 12dPL program by using the command *#include "set_ups.h*" at the top of the 12dPL program. For an example see Example 11 .

The following sections describe in detail what some of the values in the set_ups.h file are used for. For a full listing of *set_ups.h*, see Set Ups.h at the end of this Appendix.

See General Constants
See Model Mode
See File Mode
See View Mode
See Tin Mode
See Template Mode
See Project Mode
See Directory Mode
See Function Mode
See Linestyle Mode
See Symbol Mode
See Snap Mode
See Super String Use Modes
See Select Mode
See Widgets Mode
See Text Alignment Modes for Draw_Box
See Set Ups.h

General Constants

TRUE = 1

OK = 1

FALSE = 0

Model Mode

The Model **modes** are used in two ways.

(a) When a Model_Box is created with *Create_model_box(Text title_text,Message_Box message,Integer mode)*, **mode** determines the behaviour when information is entered into the Model_Box.

If information is typed and then an <enter> pressed in the Model_Box, or if a model is selected from the model pop-up list, automatic validation is performed by the Model_Box according to **mode.** What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

(b) A **mode** is also used with the *Validate(Model_Box box,Integer mode,Model &model)* call. Again **mode** will determine what validation occurs, what messages are written to the Message_Box, what actions are taken and what the function return value is.

There are CHECK modes which never create models and GET modes which may create models.

CHECK_MODEL_EXISTS = 3

If information is typed and then an <enter> pressed in the Model_Box, or if a model is selected from the model pop-up list:

(a) If the model exists, the message says "exists".

(b) If the model doesn't exist and the field is not blank, the messages says "does not exist"

(c) If field is blank and not optional, message says "no model specified"

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(model_box,mode,model)*:

(a) If the model exists, for Validate the message says "exists" and the return code is MODEL_EXISTS. The model is returned as the argument **model**.

(b) If the model doesn't exist and the field is not blank, for Validate the message says "does not exist" and the return code is NO_MODEL and no model is returned as the argument **model**.

(c) If field is blank and not optional, for Validate the message says "no model specified" and the return code of NO_NAME and no model is returned as the argument **model.**

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME and no model is returned as the argument **model.**

CHECK_MODEL_MUST_EXIST = 7

If information is typed and then an <enter> pressed in the Model_Box, or if a model is selected from the model pop-up list:

(a) If the model exists, the message says "exists".

(b) If the model doesn't exist and the field is not blank, the messages says "ERROR does not exist"

(c) If field is blank and not optional, message says "ERROR no model specified"

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(model_box,mode,model)*:

(a) If the model exists, for Validate the message says "exists" and the return code is MODEL_EXISTS. The model is returned as the argument **model**.

(b) If the model doesn't exist and the field is not blank, for Validate the messages says "ERROR does not exist" and the return code is NO_MODEL and no model is returned as the

argument **model**.

(c) If field is blank and not optional, for Validate the message says "ERROR no model specified" and the return code of NO_NAME and no model is returned as the argument **model.**

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME and no model is returned as the argument **model.**

CHECK_MODEL_CREATE = 4

If information is typed and then an <enter> pressed in the Model_Box, or if a model is selected from the model pop-up list:

(a) If the model exists, the message says "exists".

(b) If the model doesn't exist and the field is not blank, the messages says "will be created"

(c) If field is blank and not optional, message says "no model specified"

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(model_box,mode,model)*:

(a) If the model exists, for Validate the message says "exists" and the return code is MODEL_EXISTS. The model is returned as the argument **model**.

(b) If the model doesn't exist and the field is not blank, for Validate the messages says "will be created" and the return code is NO_MODEL and no model is returned as the argument **model**. Yes it is a confusing message but this mode should not be used with Validate.

(c) If field is blank and not optional, for Validate the message says "no model specified" and the return code of NO_NAME and no model is returned as the argument **model.**

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME and no model is returned as the argument **model.**

CHECK_MODEL_MUST_NOT_EXIST = 60

If information is typed and then an <enter> pressed in the Model_Box, or if a model is selected from the model pop-up list:

(a) If the model exists, the message says "ERROR exists".

(b) If the model doesn't exist and the field is not blank, the messages says "does not exist".

(c) If field is blank and not optional, message says "no model specified"

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(model_box,mode,model)*:

(a) If the model exists, for Validate the message says "ERROR exists" and the return code is MODEL_EXISTS. The model is returned as the argument **model**.

(b) If the model doesn't exist and the field is not blank, for Validate the messages says "does not exist" and the return code is NO_MODEL and no model is returned as the argument **model**.

(c) If field is blank and not optional, for Validate the message says "no model specified" and the return code of NO_NAME and no model is returned as the argument **model.**

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME and no model is returned as the argument **model.**

CHECK_DISK_MODEL_MUST_EXIST = 33

CHECK_EITHER_MODEL_EXISTS = 38

GET_MODE = 10

If information is typed and then an <enter> pressed in the Model_Box, or if a model is selected from the model pop-up list:

(a) If the model exists, the message says "exists".

(b) If the model doesn't exist and the field is not blank, the messages says "ERROR does not exist"

(c) If field is blank and not optional, there is no message

(d) If field is blank and optional, there is no message.

For *Validate(model_box,mode,model)*:

(a) If the model exists, for Validate the message says "exists" and the return code is MODEL_EXISTS. The model is returned as the argument **model**.

(b) If the model doesn't exist and the field is not blank, for Validate the message says "ERROR does not exist" and the return code is NO_MODEL and no model is returned as the argument **model**.

(c) If field is blank and not optional, for Validate there is no message and the return code is NO_NAME and no model is returned as the argument **model.**

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME and no model is returned as the argument **model.**

GET_MODEL_CREATE = 5

If information is typed and then an <enter> pressed in the Model_Box, or if a model is selected from the model pop-up list:

(a) If the model exists, the message says "exists".

(b) If the model doesn't exist and the field is not blank, the messages says "created" and the **model is created**.

(c) If field is blank and not optional, the message says "ERROR no model specified"

(d) If field is blank and optional, there is no message.

For *Validate(model_box,mode,model)*:

(a) If the model exists, for Validate the message says "exists" and the return code is MODEL_EXISTS. The model is returned as the argument **model**.

(b) If the model doesn't exist and the field is not blank, for Validate the message says "created" and the model is created. The return code is MODEL_EXISTS and the model is returned as the argument **model**.

(c) If field is blank and not optional, for Validate the message says "ERROR no model specified" and the return code is NO_MODEL and no model is returned as the argument **model.**

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME and no model is returned as the argument **model.**

GET_MODEL_ERROR = 13

If information is typed and then an <enter> pressed in the Model_Box, or if a model is selected from the model pop-up list:

(a) If the model exists, the message says "exists".

(b) If the model doesn't exist and the field is not blank, the messages says "ERROR does not exist".

(c) If field is blank and not optional, the message says "ERROR no model specified"

(d) If field is blank and optional, there is no message.

For *Validate(model_box,mode,model)*:

(a) If the model exists, for Validate the message says "exists" and the return code is MODEL_EXISTS. The model is returned as the argument **model**.

(b) If the model doesn't exist and the field is not blank, for Validate the message says "ERROR does not exist" and the return code is NO_MODEL and no model is returned as the argument **model**.

(c) If field is blank and not optional, for Validate the message says "ERROR no model specified" and the return code is NO_MODEL and no model is returned as the argument **model.**

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME and no model is returned as the argument **model.**

GET_DISK_MODEL_ERROR = 34

MODEL FUNCTION RETURN CODES

NO_MODEL = 1

MODEL_EXISTS = 2

DISK_MODEL_EXISTS = 19

NEW_MODEL = 3

NO_NAME = 10 // when no name is entered (i.e. blank)

NO_CASE = 8

File Mode

The File **modes** are used in two ways.

(a) When a File_Box is created with *Create_file_box(Text title_text,Message_Box message,Integer mode)*, **mode** determines the behaviour when information is entered into the File_Box.

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list, automatic validation is performed by the File_Box according to **mode.** What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

(b) A **mode** is also used with the *Validate(File_Box box,Integer mode,Text &result)* call. Again **mode** will determine what validation occurs, what messages are written to the Message_Box, what actions are taken and what the function return value is.

Because of many different ways files can be opened, files are never created by the *Create_file_box(Text title_text,Message_Box message,Integer mode)* or *Validate(File_Box box,Integer mode,Text &result) calls.*

Regardless of the modes, the text typed into the File_Box is returned as **result** in the Validate *Validate(File_Box box,Integer mode,Text &result) call.*

CHECK_FILE_MUST_EXIST = 1

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists".

(b) If the file doesn't exist and the field is not blank, the messages says "ERROR ... does not exist"

(c) If field is blank and not optional, message says "ERROR File must specify a file name"

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the model exists, for Validate the message says "exists" and the return code is FILE_EXISTS. The text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the message says "ERROR ... does not exist" and the return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate the message says "ERROR File must specify a file name" and the return code of NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME. **result** is returned as ""

CHECK_FILE_CREATE = 14

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists".

(b) If the file doesn't exist and the field is not blank, the messages says "will be created"

(c) If field is blank and not optional, message says "ERROR must specify a file name"

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and the return code is FILE_EXISTS.

The text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the messages says "will be created" and the return code is NO_FILE. The text in the File_Box is returned in the argument **result**. Yes it is a confusing message but this mode should not be used with Validate.

(c) If field is blank and not optional, for Validate the message says "ERROR must specify a file name" and the return code of NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME. **result** is returned as "".

CHECK_FILE = 22

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists".

(b) If the file doesn't exist and the field is not blank, the messages says "ERROR File must specify an existing file"

(c) If field is blank and not optional, message says "ERROR File must specify an existing file"

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and the return code is FILE_EXISTS. The text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the messages says "ERROR File must specify an existing file" and the return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate the message says "ERROR File must specify an existing file" and the return code of NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME. **result** is returned as "".

CHECK_FILE_NEW = 20

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "ERROR ... exists".

(b) If the model doesn't exist and the field is not blank, the messages says "File ... will be created".

(c) If field is blank and not optional, message says "ERROR File must specify a file name"

(d) If field is blank and optional, message says "ok - field is optional".

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "ERROR ... exists" and the return code is FILE_EXISTS. The text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the messages says "File ... will be created" and the return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate the message says "ERROR File must specify a file name" and the return code of NO_FILE. **result** is returned as "".

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_FILE. **result** is returned as ""

CHECK_FILE_APPEND = 21

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists".

(b) If the file doesn't exist and the field is not blank, the messages says "will be created"

(c) If field is blank and not optional, message says "ERROR must specify a file"

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and the return code is FILE_EXISTS. The text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the messages says "will be created" and the return code is NO_FILE. The text in the File_Box is returned in the argument **result**. Yes it is a confusing message but this mode should not be used with Validate.

(c) If field is blank and not optional, for Validate the message says "ERROR must specify a file" and the return code of NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME. **result** is returned as "".

CHECK_FILE_WRITE = 23

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists".

(b) If the file doesn't exist and the field is not blank, the messages says "will be created"

(c) If field is blank and not optional, message says

(d) If field is blank and optional, message says "ok - field is optional"

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and the return code is FILE_EXISTS. The text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the messages says "will be created" and the return code is NO_FILE. The text in the File_Box is returned in the argument **result**. Yes it is a confusing message but this mode should not be used with Validate.

(c) If field is blank and not optional, for Validate the message says and the return code of NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate the message says "ok - field is optional" and the return code is NO_NAME. **result** is returned as "".

GET_FILE = 16

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists".

(b) If the file doesn't exist and the field is not blank, the messages says "ERROR File must specify an existing file"

(c) If field is blank and not optional, there is no message

(d) If field is blank and optional, there is no message.

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and the return code is FILE_EXISTS. The text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the message says "ERROR File must specify an existing file" and the return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

GET_FILE_MUST_EXIST = 7

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists".

(b) If the file doesn't exist and the field is not blank, the messages says "ERROR File file ... does not exist".

(c) If field is blank and not optional, the message says "ERROR File must specify a file name"

(d) If field is blank and optional, there is no message.

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and the return code is FILE_EXISTS. The text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the message says "ERROR File file ... does not exist" and the return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate the message says "ERROR File must specify a file name" and the return code is NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

GET_FILE_CREATE = 15

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists", and a "File_Box Not Optional" panel comes up and asks if you would like to *Replace* or *Cancel*. If Replace if selected, the file is deleted. If Cancel is Selected, the file is not deleted and "overwrite aborted by user".

(b) If the file doesn't exist and the field is not blank, the messages says "File ... will be created" but **no file** is created.

(c) If field is blank and not optional, there is no message.

(d) If field is blank and optional, there is no message.

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and a "File_Box Not Optional" panel comes up and asks if you would like to *Replace* or *Cancel*. If Replace if selected, the file is deleted and the return code is NO_FILE. If Cancel is Selected, the file is not deleted and "overwrite aborted by user" and the return code is NO_FILE_ACCESS. In both bases, the text in the File_Box is returned in the argument **result**.

Hence when the file already exist, the user is asked to *Replace* or *Cancel and the* return code differentiates between the two possibilities:

NO_FILE indicates that *Replace* was chosen (and the file is automatically deleted).

NO_FILE_ACCESS indicates that *Cancel* was chosen and so the file is not to be used.

(b) If the file doesn't exist and the field is not blank, for Validate the message says "will be created" but no file is created. The return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

GET_FILE_NEW = 18

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "ERROR File ... exists". The file is not deleted.

(b) If the file doesn't exist and the field is not blank, the messages says "File ... will be created" but **no file** is created.

(c) If field is blank and not optional, the message says "ERROR File must specify a file name".

(d) If field is blank and optional, there is no message.

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "ERROR File ... exists" and the return code is FILE_EXISTS. The file is not deleted.T he text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the message says "will be created" but no file is created. The return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate the message says "ERROR File must specify a file name" and the return code is NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

GET_FILE_APPEND = 19

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists", and a "File_Box Not Optional" panel comes up and asks if you would like to *Append*, *Replace* or *Cancel*. If *Append* is selected nothing is done, if *Replace* if selected, the file is deleted. If *Cancel* is Selected, the file is not deleted and "overwrite aborted by user".

(b) If the file doesn't exist and the field is not blank, the messages says "File ... will be created" but **no file** is created.

(c) If field is blank and not optional, there is no message.

(d) If field is blank and optional, there is no message.

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and a "File_Box Not Optional" panel comes up and asks if you would like to *Append*, *Replace* or *Cancel*. If *Append* is selected nothing is done to the file and the return code is FILE_EXISTS, If *Replace* if selected, the file is deleted and the return code is NO_FILE. If *Cancel* is Selected, the file is not deleted

and "overwrite aborted by user" and the return code is NO_FILE_ACCESS. In both bases, the text in the File_Box is returned in the argument **result**.

Hence when the file already exist, the user is asked to *Append*, *Replace* or *Cancel and the* return code differentiates between the three possibilities:

FILE_EXISTS indicates that *Append* was chosen.

NO_FILE indicates that *Replace* was chosen (and the file is automatically deleted).

NO_FILE_ACCESS indicates that *Cancel* was chosen and so the file is not to be used.

(b) If the file doesn't exist and the field is not blank, for Validate the message says "will be created" but **no file** is created. The return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

GET_FILE_WRITE = 24

If information is typed and then an <enter> pressed in the File_Box, or if a file is selected from the file pop-up list:

(a) If the file exists, the message says "exists", and a "File_Box Not Optional" panel comes up and asks if you would like to *Append*, *Replace* or *Cancel*. If *Append* is selected ?, if *Replace* if selected, the file is deleted. If *Cancel* is Selected, the file is not deleted and "overwrite aborted by user".

(b) If the file doesn't exist and the field is not blank, the messages says "File ... will be created" but **no file** is created.

(c) If field is blank and not optional, there is no message.

(d) If field is blank and optional, there is no message.

For *Validate(File_Box box,Integer mode,Text &result)*:

(a) If the file exists, for Validate the message says "exists" and a "File_Box Not Optional" panel comes up and asks if you would like to *Append*, *Replace* or *Cancel*. If *Append* is selected ? and the return code is FILE_EXISTS, If *Replace* if selected, the file is deleted and the return code is NO_FILE. If *Cancel* is Selected, the file is not deleted and "overwrite aborted by user" and the return code is NO_FILE_ACCESS. In both bases, the text in the File_Box is returned in the argument **result**.

(b) If the file doesn't exist and the field is not blank, for Validate the message says "will be created" but **no file** is created. The return code is NO_FILE. The text in the File_Box is returned in the argument **result**.

(c) If field is blank and not optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME. **result** is returned as "".

FILE RETURN CODES

NO_FILE = 4

FILE_EXISTS = 5

NO_FILE_ACCESS = 6

```
NO_NAME = 10          // when no name is entered (i.e. blank)
NO_CASE = 8
```

View Mode

The View **modes** are used in two ways.

(a) When a View_Box is created with *Create_view_box(Text title_text,Message_Box message,Integer mode)*, **mode** determines the behaviour when information is entered into the View_Box.

If information is typed and then an <enter> pressed in the View_Box, or if a view is selected from the view pop-up list, automatic validation is performed by the View_Box according to **mode.** What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

(b) A **mode** is also used with the *Validate(View_Box box,Integer mode,View &view)* call. Again **mode** will determine what validation occurs, what messages are written to the Message_Box, what actions are taken and what the function return value is.

CHECK_VIEW_MUST_EXIST = 2

If information is typed and then an <enter> pressed in the View_Box, or if a view is selected from the view pop-up list:

(a) If the view exists, the message says "exists".

(b) If the model doesn't exist and the field is not blank, the messages says "ERROR does not exist"

(c) If field is blank and not optional, message says "ERROR no view specified"

(d) If field is blank and optional, message says "ok"

For *Validate(view_box,mode,view)*:

(a) If the model exists, for Validate the message says "exists" and the return code is VIEW_EXISTS. The view is returned as the argument **view**.

(b) If the view doesn't exist and the field is not blank, for Validate the messages says "ERROR does not exist" and the return code is NO_VIEW and no view is returned as the argument **view**.

(c) If field is blank and not optional, for Validate the message says "ERROR no view specified" and the return code of NO_NAME and no view is returned as the argument **view**.

(d) If field is blank and optional, for Validate the message says "ok" and the return code is NO_NAME and no view is returned as the argument **view**.

CHECK_VIEW_MUST_NOT_EXIST = 25

If information is typed and then an <enter> pressed in the View_Box, or if a view is selected from the view pop-up list:

(a) If the view exists, the message says "ERROR exists".

(b) If the model doesn't exist and the field is not blank, the messages says "will be created".

(c) If field is blank and not optional, message says "ERROR no view specified"

(d) If field is blank and optional, message says "ok"

For *Validate(view_box,mode,view)*:

(a) If the view exists, for Validate the message says "ERROR exists" and the return code is VIEW_EXISTS. The view is returned as the argument **view**.

(b) If the view doesn't exist and the field is not blank, for Validate the messages says "will be created" and the return code is NO_VIEW and no view is returned as the argument **model**.

(c) If field is blank and not optional, for Validate the message says "no view specified" and the return code of NO_NAME and no view is returned as the argument **view.**

(d) If field is blank and optional, for Validate the message says "ok" and the return code is NO_NAME and no view is returned as the argument **view**.

GET_VIEW = 11

If information is typed and then an <enter> pressed in the View_Box, or if a view is selected from the view pop-up list:

(a) If the view exists, the message says "exists".

(b) If the view doesn't exist and the field is not blank, the messages says "ERROR does not exist"

(c) If field is blank and not optional, there is no message

(d) If field is blank and optional, there is no message.

For *Validate(view_box,mode,view)*:

(a) If the view exists, for Validate the message says "exists" and the return code is VIEW_EXISTS. The view is returned as the argument **view**.

(b) If the view doesn't exist and the field is not blank, for Validate the message says "ERROR does not exist" and the return code is NO_VIEW and no view is returned as the argument **view**.

(c) If field is blank and not optional, for Validate there is no message and the return code is NO_NAME and no view is returned as the argument **view.**

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME and no view is returned as the argument **view.**

GET_VIEW_ERROR = 6

If information is typed and then an <enter> pressed in the View_Box, or if a view is selected from the view pop-up list:

(a) If the view exists, the message says "exists".

(b) If the view doesn't exist and the field is not blank, the messages says "ERROR does not exist".

(c) If field is blank and not optional, the message says "ERROR no view specified"

(d) If field is blank and optional, there is no message.

For *Validate(view_box,mode,view)*:

(a) If the view exists, for Validate the message says "exists" and the return code is VIEW_EXISTS. The model is returned as the argument **view**.

(b) If the view doesn't exist and the field is not blank, for Validate the message says "ERROR does not exist" and the return code is NO_VIEW and no view is returned as the argument **view**.

(c) If field is blank and not optional, for Validate the message says "ERROR no view specified" and the return code is NO_NAME and no view is returned as the argument **view.**

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME and no view is returned as the argument **view.**

VIEW RETURN CODES

NO_VIEW = 6

VIEW_EXISTS = 7

NO_NAME = 10

NO_CASE = 8

Tin Mode

The Tin **modes** are used in two ways.

(a) When a Tin_Box is created with *Create_tin_box(Text title_text,Message_Box message,Integer mode)*, **mode** determines the behaviour when information is entered into the Tin_Box.

If information is typed and then an <enter> pressed in the Tin_Box, or if a tin is selected from the tin pop-up list, automatic validation is performed by the Tin_Box according to **mode.** What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

(b) A **mode** is also used with the *Validate(Tin_Box box,Integer mode,Tin &tin)* call. Again **mode** will determine what validation occurs, what messages are written to the Message_Box, what actions are taken and what the function return value is.

There are CHECK modes which never create tins and GET modes which may create tins.

CHECK_TIN_MUST_EXIST = 8

If information is typed and then an <enter> pressed in the Tin_Box, or if a tin is selected from the tin pop-up list:

(a) If the tin exists, the message says "exists".

(b) If the tin doesn't exist and the field is not blank, the messages says "ERROR does not exist"

(c) If field is blank and not optional, message says "ERROR no tin specified"

(d) If field is blank and optional, message says "ok"

For *Validate(tin_box,mode,tin)*:

(a) If the tin exists, for Validate the message says "exists" and the return code is TIN_EXISTS. The tin is returned as the argument **tin**.

(b) If the tin doesn't exist and the field is not blank, for Validate the messages says "ERROR does not exist" and the return code is NO_TIN and no tin is returned as the argument **tin**.

(c) If field is blank and not optional, for Validate the message says "ERROR no tin specified" and the return code of NO_NAME and no tin is returned as the argument **tin.**

(d) If field is blank and optional, for Validate the message says "ok" and the return code is NO_NAME and no tin is returned as the argument **tin.**

CHECK_TIN_EXISTS = 61

If information is typed and then an <enter> pressed in the Tin_Box, or if a tin is selected from the tin pop-up list:

(a) If the tin exists, the message says "exists".

(b) If the tin doesn't exist and the field is not blank, the messages says "does not exist"

(c) If field is blank and not optional, message says "no tin specified"

(d) If field is blank and optional, message says "ok"

For *Validate(tin_box,mode,tin)*:

(a) If the tin exists, for Validate the message says "exists" and the return code is TIN_EXISTS. The tin is returned as the argument **tin**.

(b) If the tin doesn't exist and the field is not blank, for Validate the message says "does not exist" and the return code is NO_TIN and no tin is returned as the argument **tin**.

(c) If field is blank and not optional, for Validate the message says "no tin specified" and the return code of NO_NAME and no tin is returned as the argument **tin.**

(d) If field is blank and optional, for Validate the message says "ok" and the return code is NO_NAME and no tin is returned as the argument **tin.**

CHECK_EITHER_TIN_EXISTS = 39

CHECK_TIN_NEW = 12

If information is typed and then an <enter> pressed in the Tin_Box, or if a tin is selected from the tin pop-up list:

(a) If the tin exists, the message says "ERROR must not exist".

(b) If the tin doesn't exist and the field is not blank, the messages says "ok - no Tin exists"

(c) If field is blank and not optional, message says "ERROR no tin specified"

(d) If field is blank and optional, message says "ok"

For *Validate(tin_box,mode,tin)*:

(a) If the tin exists, for Validate the message says "ERROR must not exist" and the return code is TIN_EXISTS. The tin is returned as the argument **tin**.

(b) If the tin doesn't exist and the field is not blank, for Validate the messages says "ok - no Tin exists" and the return code is NO_TIN and no tin is returned as the argument **tin**.

(c) If field is blank and not optional, for Validate the message says "ERROR no tin specified" and the return code of NO_NAME and no tin is returned as the argument **tin.**

(d) If field is blank and optional, for Validate the message says "ok" and the return code is NO_NAME and no tin is returned as the argument **tin.**

CHECK_TIN_MUST_NOT_EXIST = 91

If information is typed and then an <enter> pressed in the Tin_Box, or if a tin is selected from the tin pop-up list:

(a) If the tin exists, the message says "ERROR exists".

(b) If the tin doesn't exist and the field is not blank, the messages says "does not exist".

(c) If field is blank and not optional, message says "ERROR tin not specified"

(d) If field is blank and optional, message says "ok"

For *Validate(tin_box,mode,tin)*:

(a) If the tin exists, for Validate the message says "ERROR exists" and the return code is TIN_EXISTS. The tin is returned as the argument **tin**.

(b) If the tin doesn't exist and the field is not blank, for Validate the messages says "does not exist" and the return code is NO_TIN and no tin is returned as the argument **tin**.

(c) If field is blank and not optional, for Validate the message says "ERROR no tin specified" and the return code of NO_NAME and no tin is returned as the argument **tin.**

(d) If field is blank and optional, for Validate the message says "ok" and the return code is NO_NAME and no tin is returned as the argument **tin.**

CHECK_DISK_TIN_MUST_EXIST = 16

GET_TIN = 10

If information is typed and then an <enter> pressed in the Tin_Box, or if a tin is selected from the tin pop-up list:

(a) If the tin exists, the message says "exists".

(b) If the tin doesn't exist and the field is not blank, the messages says "ERROR does not exist"

(c) If field is blank and not optional, there is no message

(d) If field is blank and optional, there is no message.

For *Validate(tin_box,mode,tin)*:

(a) If the tin exists, for Validate the message says "exists" and the return code is TIN_EXISTS. The tin is returned as the argument **tin**.

(b) If the tin doesn't exist and the field is not blank, for Validate the message says "ERROR does not exist" and the return code is NO_TIN and no tin is returned as the argument **tin**.

(c) If field is blank and not optional, for Validate there is no message and the return code is NO_NAME and no tin is returned as the argument **model.**

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME and no tin is returned as the argument **model.**

GET_TIN_ERROR = 9

If information is typed and then an <enter> pressed in the Tin_Box, or if a tin is selected from the tin pop-up list:

(a) If the tin exists, the message says "exists".

(b) If the tin doesn't exist and the field is not blank, the messages says "ERROR does not exist".

(c) If field is blank and not optional, the message says "ERROR no tin specified"

(d) If field is blank and optional, there is no message.

For *Validate(tin_box,mode,tin)*:

(a) If the tin exists, for Validate the message says "exists" and the return code is TIN_EXISTS. The tin is returned as the argument **tin**.

(b) If the tin doesn't exist and the field is not blank, for Validate the message says "ERROR does not exist" and the return code is NO_TIN and no tin is returned as the argument **tin**.

(c) If field is blank and not optional, for Validate the message says "ERROR no tin specified" and the return code is NO_NAME and no tin is returned as the argument **tin.**

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME and no tin is returned as the argument **tin.**

GET_TIN_CREATE = 24

If information is typed and then an <enter> pressed in the Tin_Box, or if a tin is selected from the tin pop-up list:

(a) If the tin exists, the message says "exists".

(b) If the tin doesn't exist and the field is not blank, the messages says "created" and the **tin is created**.

(c) If field is blank and not optional, the message says "ERROR no tin specified"

(d) If field is blank and optional, there is no message.

For *Validate(tin_box,mode,tin)*:

(a) If the tin exists, for Validate the message says "exists" and the return code is TIN_EXISTS. The tin is returned as the argument **tin**.

(b) If the tin doesn't exist and the field is not blank, for Validate the message says "created" and the tin is created. The return code is TIN_EXISTS and the tin is returned as the argument **tin**.

(c) If field is blank and not optional, for Validate the message says "ERROR no tin specified" and the return code is NO_NAME and no tin is returned as the argument **tin.**

(d) If field is blank and optional, for Validate there is no message and the return code is NO_NAME and no tin is returned as the argument **tin.**

GET_DISK_TIN_ERROR = 35

TIN RETURN CODES

NO_TIN = 9

TIN_EXISTS = 11

DISK_TIN_EXISTS = 12

NO_NAME = 10 // when no name is entered (i.e. blank)

NO_CASE = 8

Template Mode

MODE	MODE NUMBER
CHECK_TEMPLATE_EXISTS1	7
CHECK_TEMPLATE_CREATE	18
CHECK_TEMPLATE_NEW	19
CHECK_TEMPLATE_MUST_EXIST	20
CHECK_TEMPLATE_MUST_NOT_EXIST	59
CHECK_DISK_TEMPLATE_MUST_EXIST	48
CHECK_EITHER_TEMPLATE_EXISTS	49
GET_TEMPLATE	21
GET_TEMPLATE_CREATE	22
GET_TEMPLATE_ERROR	23
GET_DISK_TEMPLATE_ERROR	40

TEMPLATE RETURN CODES	VALUE
NO_TEMPLATE	13
TEMPLATE_EXISTS	14
DISK_TEMPLATE_EXISTS	20
NEW_TEMPLATE	15
NO_NAME	10
NO_CASE	8

Project Mode

MODE	MODE NUMBER
CHECK_PROJECT_EXISTS	26
CHECK_PROJECT_CREATE	27
CHECK_PROJECT_NEW	28
CHECK_PROJECT_MUST_EXIST	29
CHECK_DISK_PROJECT_MUST_EXIST	36
GET_PROJECT	30
GET_PROJECT_CREATE	31
GET_PROJECT_ERROR	32
GET_DISK_PROJECT_ERROR	37

PROJECT RETURN CODES	VALUE
NO_PROJECT	16
PROJECT_EXISTS	17
NEW_PROJECT	18
NO_NAME	10
NO_CASE	8

Directory Mode

MODE	MODE NUMBER
CHECK_DIRECTORY_EXISTS	41
CHECK_DIRECTORY_CREATE	42
CHECK_DIRECTORY_NEW	43
CHECK_DIRECTORY_MUST_EXIST	44
GET_DIRECTORY	45
GET_DIRECTORY_CREATE	46
GET_DIRECTORY_ERROR	47

DIRECTORY RETURN CODES	VALUE
NO_DIRECTORY	21
DIRECTORY_EXISTS	22
NEW_DIRECTORY	23
NO_NAME	10
NO_CASE	8

Function Mode

MODE	MODE NUMBER
CHECK_FUNCTION_MUST_EXIST	50
CHECK_FUNCTION_EXISTS	51
CHECK_FUNCTION_CREATE	52
CHECK_DISK_FUNCTION_MUST_EXIST	53
CHECK_EITHER_FUNCTION_EXISTS	54
CHECK_FUNCTION_MUST_NOT_EXIST	90
GET_FUNCTION	55
GET_FUNCTION_CREATE	56
GET_FUNCTION_ERROR	57
GET_DISK_FUNCTION_ERROR	58

FUNCTION RETURN CODES	VALUE
NO_FUNCTION	24
FUNCTION_EXISTS	25
DISK_FUNCTION_EXISTS	26
NEW_FUNCTION	27
NO_NAME	10
NO_CASE	8

Linestyle Mode

MODE	MODE NUMBER
CHECK_LINESTYLE_MUST_EXIST	82
CHECK_LINESTYLE_MUST_NOT_EXIST	83
GET_LINESTYLE	84
GET_LINESTYLE_ERROR	85

LINESTYLE RETURN CODES	VALUE
LINESTYLE_EXISTS	80
NO_LINESTYLE	81
NO_NAME	10
NO_CASE	8

Symbol Mode

MODE	MODE NUMBER

Snap Mode

MODE	MODE NUMBER
Ignore_Snap	0
User_Snap	1
Program_Snap	2
Failed_Snap	-1
No_Snap	0
Point_Snap	1
Line_Snap	2
Grid_Snap	3
Intersection_Snap	4
Cursor_Snap	5
Name_Snap	6
Tin_Snap	7
Model_Snap	8
Height_Snap	9

Super String Use Modes

MODE	MODE NUMBER
Att_ZCoord_Value	1
Att_ZCoord_Array	2
Att_Radius_Array	3
Att_Major_Array	4
Att_Diameter_Value	5
Att_Diameter_Array	6
Att_Text_Array	7
Att_Colour_Value	8
Att_Colour_Array	9
Att_Point_Array	11
Att_Visible_Array	12
Att_Contour_Array	13
Att_Annotate_Value	14
Att_Annotate_Array	15
Att_Attribute_Array	16
Att_Symbol_Value	17
Att_Symbol_Array	18
Att_Segment_Attribute_Array	19
Att_Segment_Annotate_Value	20
Att_Segment_Annotate_Array	21
Att_Segment_Text_Value	22
Att_Pipe_Justify	23
Att_Culvert_Value	24
Att_Culvert_Array	25
Att_Hole_Value	26
Att_Hatch_Value	27
Att_Solid_Value	28
Att_Bitmap_Value	29
Att_World_Annotate	30
Att_Annotate_Type	31
Att_XCoord_Array	32
Att_YCoord_Array	33
Att_Pattern_Value	33 ?
Att_Vertex_UID_Array	35
Att_Segment_UID_Array	36
Att_Vertex_Tinable_Value	37
Att_Vertex_Tinable_Array	38
Att_Segment_Tinable_Value	39

Att_Segment_Tinable_Array 40
Att_Vertex_Visible_Value 41
Att_Vertex_Visible_Array 42
Att_Segment_Visible_Value 43
Att_Segment_Visible_Array 44
Att_Vertex_Paper_Annotate 45
Att_Segment_Paper_Annotate 46
Att_Database_Point_Array 47
Att_Extrude_Value 48

Att_Interval_Value 50
Att_Vertex_Image_Value 51
Att_Vertex_Image_Array 52
Att_Matrix_Value 53
Att_Autocad_Pattern_Value 54
Att_Null_Levels_Value 55

Select Mode

MODE	MODE NUMBER
SELECT_STRING	5509
SELECT_STRINGS	5510
NO_NAME	10
NO_CASE	8
TRUE	1
OK	1
FALSE	0

Widgets Mode

HORIZONTAL GROUP	MODE NUMBER
BALANCE_WIDGETS_OVER_WIDTH	1
ALL_WIDGETS_OWN_WIDTH	2
COMPRESS_WIDGETS_OVER_WIDTH	4

-1 is also allowed

VERTICAL GROUP	MODE NUMBER
BALANCE_WIDGETS_OVER_HEIGHT	1
ALL_WIDGETS_OWN_HEIGHT	2
COMPRESS_WIDGETS_OVER_HEIGHT	4

-1 is also allowed

Text Alignment Modes for Draw_Box

The text drawn in the Draw_Box uses the Text Alignments as given by the Microsoft SetTextAlign Function.

The text is drawn on a baseline and has a bounding box that surrounds the text.

The default values are TA_LEFT, TA_TOP and TA_NOUPDATECP.

MODE	MODE NUMBER
TA_NOUPDATECP	0

The current position is not updated after each text output call. The reference point is passed to the next text output function.

TA_UPDATECP	1

The current position is updated after each text output call. The current position is used as the reference point.

TA_LEFT	0

The reference point will be on the left edge of the bounding rectangle.

TA_RIGHT	2

The reference point will be on the right edge of the bounding rectangle.

TA_CENTER	6

The reference point will be aligned horizontally with the centre of the bounding rectangle.

TA_TOP	0

The reference point will be on the top edge of the bounding rectangle.

TA_BOTTOM	8

The reference point will be on the bottom edge of the bounding rectangle.

TA_BASELINE	24

The reference point will be on the base line of the text.

TA_RTLREADING	256

Middle East language edition of Windows: The text is laid out in right to left reading order, as opposed to the default left to right order. This applies only when the font selected into the device context is either Hebrew or Arabic. reference point will be on the base line of the text.

TA_MASK	(TA_BASELINE+TA_CENTER+TA_UPDATECP+TA_RTLREADING)

VTA_BASELINE	TA_BASELINE
VTA_LEFT	TA_BOTTOM
VTA_RIGHT	TA_TOP
VTA_CENTER	TA_CENTER
VTA_BOTTOM	TA_RIGHT
VTA_TOP	TA_LEFT

Set Ups.h

```
#ifndef set_ups_included
#define set_ups_included

// ---------------------------------------------------------------------------
// colour conversion stuff
// ---------------------------------------------------------------------------

Integer create_rgb(Integer r,Integer g,Integer b)
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
{
  return((1 << 31) | (r << 16) | (g << 8) | b);
}
Integer is_rgb(Integer colour)
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
{
  return((colour & (1 << 31)) ? 1 : 0);
}
Integer get_rgb(Integer colour,Integer &r,Integer &g,Integer &b)
// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
{
  if(colour & (1 << 31)) {

// a direct colour defined !

    r = (colour & 16711680) >> 16;
    g = (colour &    65280) >> 8;
    b = (colour &      255);

    return(1);
  }
  return(0);
}

#define VIEW_COLOUR 0x7fffffff
#define NO_COLOUR   -1

//-----------------------------------------------------------------------------
//                  SETUPS
//-----------------------------------------------------------------------------

#define CHECK_MODEL_MUST_EXIST         7
#define CHECK_MODEL_EXISTS             3
#define CHECK_MODEL_CREATE             4
#define CHECK_DISK_MODEL_MUST_EXIST    33
#define CHECK_EITHER_MODEL_EXISTS      38
#define GET_MODEL                      10
#define GET_MODEL_CREATE               5
#define GET_MODEL_ERROR                13
#define GET_DISK_MODEL_ERROR           34
#define CHECK_MODEL_MUST_NOT_EXIST     60

#define CHECK_FILE_MUST_EXIST          1
```

```
#define CHECK_FILE_CREATE          14
#define CHECK_FILE                 22
#define CHECK_FILE_CREATE          14
#define CHECK_FILE_NEW             20
#define CHECK_FILE_APPEND          21
#define CHECK_FILE_WRITE           23
#define GET_FILE                   16
#define GET_FILE_MUST_EXIST        17
#define GET_FILE_CREATE            15
#define GET_FILE_NEW               18
#define GET_FILE_APPEND            19
#define GET_FILE_WRITE             24

#define GET_TIN                    10


#define CHECK_VIEW_MUST_EXIST      2
#define CHECK_VIEW_MUST_NOT_EXIST  25
#define GET_VIEW                   11
#define GET_VIEW_ERROR             6

#define CHECK_TIN_MUST_EXIST       8
#define CHECK_TIN_EXISTS           61
#define CHECK_EITHER_TIN_EXISTS    39
#define CHECK_TIN_NEW              12
#define GET_TIN_ERROR              9
#define CHECK_DISK_TIN_MUST_EXIST  16
#define GET_TIN_CREATE             24
#define GET_DISK_TIN_ERROR         35
#define CHECK_TIN_MUST_NOT_EXIST   91

#define CHECK_TEMPLATE_EXISTS      17
#define CHECK_TEMPLATE_CREATE      18
#define CHECK_TEMPLATE_NEW         19
#define CHECK_TEMPLATE_MUST_EXIST  20
#define CHECK_TEMPLATE_MUST_NOT_EXIST  59
#define GET_TEMPLATE               21
#define GET_TEMPLATE_CREATE        22
#define GET_TEMPLATE_ERROR         23
#define GET_DISK_TEMPLATE_ERROR    40
#define CHECK_DISK_TEMPLATE_MUST_EXIST  48
#define CHECK_EITHER_TEMPLATE_EXISTS  49

#define CHECK_PROJECT_EXISTS       26
#define CHECK_PROJECT_CREATE       27
#define CHECK_PROJECT_NEW          28
#define CHECK_PROJECT_MUST_EXIST   29
#define CHECK_DISK_PROJECT_MUST_EXIST  36
#define GET_PROJECT                30
#define GET_PROJECT_CREATE         31
#define GET_PROJECT_ERROR          32
#define GET_DISK_PROJECT_ERROR     37

#define CHECK_DIRECTORY_EXISTS     41
#define CHECK_DIRECTORY_CREATE     42
#define CHECK_DIRECTORY_NEW        43
#define CHECK_DIRECTORY_MUST_EXIST 44
#define GET_DIRECTORY              45
```

```
#define GET_DIRECTORY_CREATE          46
#define GET_DIRECTORY_ERROR           47

#define CHECK_FUNCTION_MUST_EXIST        50
#define CHECK_FUNCTION_EXISTS            51
#define CHECK_FUNCTION_CREATE            52
#define CHECK_DISK_FUNCTION_MUST_EXIST   53
#define CHECK_EITHER_FUNCTION_EXISTS     54
#define GET_FUNCTION                     55
#define GET_FUNCTION_CREATE              56
#define GET_FUNCTION_ERROR               57
#define GET_DISK_FUNCTION_ERROR          58
#define CHECK_FUNCTION_MUST_NOT_EXIST    90

#define CHECK_LINESTYLE_MUST_EXIST       82
#define CHECK_LINESTYLE_MUST_NOT_EXIST   83
#define GET_LINESTYLE                    84
#define GET_LINESTYLE_ERROR              85

//    return codes

#define NO_NAME            10

#define NO_MODEL           1
#define MODEL_EXISTS       2
#define DISK_MODEL_EXISTS  19
#define NEW_MODEL          3

#define NO_FILE            4
#define FILE_EXISTS        5
#define NO_FILE_ACCESS     6

#define NO_VIEW            6
#define VIEW_EXISTS        7

#define NO_CASE            8

#define NO_TIN             9
#define TIN_EXISTS         11
#define DISK_TIN_EXISTS    12

#define NO_TEMPLATE        13
#define TEMPLATE_EXISTS    14
#define DISK_TEMPLATE_EXISTS  20
#define NEW_TEMPLATE       15

#define NO_PROJECT         16
#define PROJECT_EXISTS     17
#define NEW_PROJECT        18

#define NO_DIRECTORY       21
#define DIRECTORY_EXISTS   22
#define NEW_DIRECTORY      23

#define NO_FUNCTION        24
#define FUNCTION_EXISTS    25
#define DISK_FUNCTION_EXISTS  26
#define NEW_FUNCTION       27
```

```
#define LINESTYLE_EXISTS      80
#define NO_LINESTYLE          81

#define SELECT_STRING   5509
#define SELECT_STRINGS  5510

// teststyle data constants

#define Textstyle_Data_Textstyle 0x001
#define Textstyle_Data_Colour    0x002
#define Textstyle_Data_Type      0x004
#define Textstyle_Data_Size      0x008
#define Textstyle_Data_Offset    0x010
#define Textstyle_Data_Raise     0x020
#define Textstyle_Data_Justify_X 0x040
#define Textstyle_Data_Justify_Y 0x080
#define Textstyle_Data_Angle     0x100
#define Textstyle_Data_Slant     0x200
#define Textstyle_Data_X_Factor  0x400
#define Textstyle_Data_Name      0x800
#define Textstyle_Data_All       0xfff

// textstyle data box constants - V9 compatible - for V10 and beyond see below

#define Show_favorites_box     0x00000001
#define Show_textstyle_box     0x00000002
#define Show_colour_box        0x00000004
#define Show_type_box          0x00000008
#define Show_size_box          0x00000010
#define Show_offset_box        0x00000020
#define Show_raise_box         0x00000040
#define Show_justify_box       0x00000080
#define Show_angle_box         0x00000100
#define Show_slant_box         0x00000200
#define Show_x_factor_box      0x00000400
#define Show_name_box          0x00000800
#define Show_draw_box          0x00001000
#define Show_underline_box     0x00002000
#define Show_strikeout_box     0x00004000
#define Show_italic_box        0x00008000
#define Show_weight_box        0x00010000
#define Show_all_boxes         0x0001ffff
#define Show_std_boxes         0x0001f7ff

#define Optional_textstyle_box 0x00020000
#define Optional_colour_box    0x00040000
#define Optional_type_box      0x00080000
#define Optional_size_box      0x00100000
#define Optional_offset_box    0x00200000
#define Optional_raise_box     0x00400000
#define Optional_justify_box   0x00800000
#define Optional_angle_box     0x01000000
#define Optional_slant_box     0x02000000
#define Optional_x_factor_box  0x04000000
#define Optional_name_box      0x08000000
#define Optional_underline_box 0x10000000
#define Optional_strikeout_box 0x20000000
```

```
#define Optional_italic_box    0x40000000
#define Optional_weight_box    0x80000000
#define Optional_all_boxes     0xfffe0000
#define Optional_std_boxes     0xf7fe0000

// V10 textstyle data box constants - only to be used with
// Textstyle_Data_Box Create_textstyle_data_box(Text text,Message_Box box,Integer flags,
//                    Integer optionals)
// this is the only way to correctly access the additional fields introduced in V10 (whiteout, border,outline)

#define V10_Show_favorites_box     0x00000001
#define V10_Show_textstyle_box     0x00000002
#define V10_Show_colour_box        0x00000004
#define V10_Show_type_box          0x00000008
#define V10_Show_size_box          0x00000010
#define V10_Show_offset_box        0x00000020
#define V10_Show_raise_box         0x00000040
#define V10_Show_justify_box       0x00000080
#define V10_Show_angle_box         0x00000100
#define V10_Show_slant_box         0x00000200
#define V10_Show_x_factor_box      0x00000400
#define V10_Show_name_box          0x00000800
#define V10_Show_draw_box          0x00001000
#define V10_Show_underline_box     0x00002000
#define V10_Show_strikeout_box     0x00004000
#define V10_Show_italic_box        0x00008000
#define V10_Show_weight_box        0x00010000
#define V10_Show_whiteout_box      0x00020000
#define V10_Show_border_box        0x00040000
#define V10_Show_outline_box       0x00080000
#define V10_Show_all_boxes         0x000fffff

#define V10_Optional_textstyle_box 0x00000002
#define V10_Optional_colour_box    0x00000004
#define V10_Optional_type_box      0x00000008
#define V10_Optional_size_box      0x00000010
#define V10_Optional_offset_box    0x00000020
#define V10_Optional_raise_box     0x00000040
#define V10_Optional_justify_box   0x00000080
#define V10_Optional_angle_box     0x00000100
#define V10_Optional_slant_box     0x00000200
#define V10_Optional_x_factor_box  0x00000400
#define V10_Optional_name_box      0x00000800
#define V10_Optional_underline_box 0x00001000
#define V10_Optional_strikeout_box 0x00002000
#define V10_Optional_italic_box    0x00004000
#define V10_Optional_weight_box    0x00008000
#define V10_Optional_whiteout_box  0x00010000
#define V10_Optional_border_box    0x00020000
#define V10_Optional_outline_box   0x00040000
#define V10_Optional_all_boxes     0x0007fffe

#define V10_Show_std_boxes   0x0001f7ff ,
 V10_Optional_whiteout_box | V10_Optional_border_box | V10_Optional_outline_box
#define V10_Optional_std_boxes     0xf7fe0000

// note the critical placement of the , in V10_Show_std_boxes
// since the flags and optionals are now split into 2 separate words, the call to
```

```
// Textstyle_Data_Box Create_textstyle_data_box(Text text,Message_Box box,
//                Integer flags,Integer optionals)
// requires two arguments, so if
//
// Textstyle_Data_Box my_box = Create_textstyle_data_box("Contour label",messages,
//                V10_Show_std_boxes)
//
// is going the same as
//
// Textstyle_Data_Box my_box = Create_textstyle_data_box("Contour label",messages,
//          V10_Show_all_boxes & ~V10_Show_name_box,
//          V10_Optional_whiteout_box |  V10_Optional_border_box | V10_Optional_outline_box)
//

// source box constants

#define Source_Box_Model         0x001
#define Source_Box_View          0x002
#define Source_Box_String        0x004
#define Source_Box_Rectangle     0x008
#define Source_Box_Trapezoid     0x010
#define Source_Box_Polygon       0x020
#define Source_Box_Lasso         0x040
#define Source_Box_Filter        0x080
#define Source_Box_Models        0x100
#define Source_Box_Favorites     0x200
#define Source_Box_All           0xfff
#define Source_Box_Fence_Inside   0x01000
#define Source_Box_Fence_Cross    0x02000
#define Source_Box_Fence_Outside  0x04000
#define Source_Box_Fence_String   0x08000
#define Source_Box_Fence_Points   0x10000
#define Source_Box_Fence_All      0xff000
#define Source_Box_Standard       Source_Box_All | Source_Box_Fence_Inside |
Source_Box_Fence_Outside | Source_Box_Fence_Cross | Source_Box_Fence_String

// target box constants

#define Target_Box_Move_To_Original_Model 0x0001   /* change/replace data */
#define Target_Box_Move_To_One_Model      0x0002   /* move/delete original data */
#define Target_Box_Move_To_Many_Models    0x0004   /* move/delete original data */
#define Target_Box_Copy_To_Original_Model 0x0008   /* copy data */
#define Target_Box_Copy_To_One_Model      0x0010   /* copy data */
#define Target_Box_Copy_To_Many_Models    0x0020   /* copy data */
#define Target_Box_Move_Copy_All          0x00ff
#define Target_Box_Delete                 0x1000   /* delete data (exclusive of all others ?) */

// more constants

#define TRUE  1
#define FALSE 0

#define OK    1

// modes for Horizontal_Group (note -1 is also allowed)

#define BALANCE_WIDGETS_OVER_WIDTH  1
#define ALL_WIDGETS_OWN_WIDTH       2
```

```
#define COMPRESS_WIDGETS_OVER_WIDTH 4

// modes for Vertical_Group (note -1 is also allowed)

#define BALANCE_WIDGETS_OVER_HEIGHT  1
#define ALL_WIDGETS_OWN_HEIGHT       2
#define ALL_WIDGETS_OWN_LENGTH       4

// snap controls

#define Ignore_Snap  0
#define User_Snap    1
#define Program_Snap 2

// snap modes

#define Failed_Snap       -1
#define No_Snap           0
#define Point_Snap        1
#define Line_Snap         2
#define Grid_Snap         3
#define Intersection_Snap  4
#define Cursor_Snap       5
#define Name_Snap         6
#define Tin_Snap          7
#define Model_Snap        8
#define Height_Snap       9
#define Segment_Snap      11
#define Text_Snap         12
#define Fast_Snap         13
#define Fast_Accept       14

// super string dimensions

#define Att_ZCoord_Value            1
#define Att_ZCoord_Array            2
#define Att_Radius_Array            3
#define Att_Major_Array             4
#define Att_Diameter_Value          5
#define Att_Diameter_Array          6
#define Att_Vertex_Text_Array       7
#define Att_Segment_Text_Array      8
#define Att_Colour_Array            9
#define Att_Vertex_Text_Value       10
#define Att_Point_Array             11
#define Att_Visible_Array           12
#define Att_Contour_Array           13
#define Att_Vertex_Annotate_Value   14
#define Att_Vertex_Annotate_Array   15
#define Att_Vertex_Attribute_Array  16
#define Att_Symbol_Value            17
#define Att_Symbol_Array            18
#define Att_Segment_Attribute_Array  19
#define Att_Segment_Annotate_Value   20
#define Att_Segment_Annotate_Array   21
#define Att_Segment_Text_Value       22
#define Att_Pipe_Justify            23
#define Att_Culvert_Value           24
```

```
#define Att_Culvert_Array          25
#define Att_Hole_Value             26
#define Att_Hatch_Value            27
#define Att_Solid_Value            28
#define Att_Bitmap_Value           29
#define Att_Vertex_World_Annotate    30
#define Att_Segment_World_Annotate   31

#define Att_Geom_Array             32
#define Att_Pattern_Value          33

#define Att_Vertex_UID_Array        35
#define Att_Segment_UID_Array       36
#define Att_Vertex_Tinable_Value    37
#define Att_Vertex_Tinable_Array    38
#define Att_Segment_Tinable_Value   39
#define Att_Segment_Tinable_Array   40
#define Att_Vertex_Visible_Value    41
#define Att_Vertex_Visible_Array    42
#define Att_Segment_Visible_Value   43
#define Att_Segment_Visible_Array   44
#define Att_Vertex_Paper_Annotate   45
#define Att_Segment_Paper_Annotate  46
#define Att_Database_Point_Array    47
#define Att_Extrude_Value           48
#define Att_Interval_Value          50

#define concat(a,b) a##b
#define String_Super_Bit(n) (1 << concat(Att_,n))

#define All_String_Super_Bits 65535

// function identifiers

#define APPLY_TEMPLATE_MACRO_T       4100
#define APPLY_TEMPLATES_MACRO_T      4102
#define INTERFACE_MACRO_T            4103
#define TURKEY_NEST_MACRO_T          4104
#define KERB_RETURN_MACRO_T          4105
#define RETRIANGULATE_MACRO_T        4106
#define RUN_MACRO_T                  4107
#define STRING_MODIFIERS_MACRO_T     4108
#define SURVEY_DATA_REDUCTION_MACRO_T 4109
#define SIMPLE_MACRO_T               4110
#define CREATE_ROADS_MACRO_T         4111
#define SLF_MACRO_T                  4112

// constants for Create_select_box mode

#define SELECT_STRING 5509
#define SELECT_STRINGS 5510

#define SELECT_SUB_STRING 5515
#define SELECT_SUB_STRINGS 5516

// values for special characters

#define Degrees_character       176
```

```
#define Squared_character        178
#define Cubed_character          179
#define Middle_dot_character     183
#define Diameter_large_character 216
#define Diameter_small_character 248

#define Degrees_text             "°"
#define Squared_text             "²"
#define Cubed_text               "³"
#define Middle_dot_text          "·"
#define Diameter_small_text      "ø"
#define Diameter_large_text      "Ø"

// definitions for last parameter of Shell_execute

#define SW_HIDE            0
#define SW_SHOWNORMAL      1
#define SW_NORMAL          1
#define SW_SHOWMINIMIZED   2
#define SW_SHOWMAXIMIZED   3
#define SW_MAXIMIZE        3
#define SW_SHOWNOACTIVATE  4
#define SW_SHOW            5
#define SW_MINIMIZE        6
#define SW_SHOWMINNOACTIVE 7
#define SW_SHOWNA          8
#define SW_RESTORE         9
#define SW_SHOWDEFAULT     10
#define SW_FORCEMINIMIZE   11
#define SW_MAX             11

// ********************************************************************
// transparency
// ********************************************************************

#define TRANSPARENT        1
#define OPAQUE             2

// ********************************************************************
// Text Alignment Options
// ********************************************************************

#define TA_NOUPDATECP              0
#define TA_UPDATECP                1

#define TA_LEFT                    0
#define TA_RIGHT                   2
#define TA_CENTER                  6

#define TA_TOP                     0
#define TA_BOTTOM                  8
#define TA_BASELINE                24

#define TA_RTLREADING              256

#define TA_MASK      (TA_BASELINE+TA_CENTER+TA_UPDATECP+TA_RTLREADING)

#define VTA_BASELINE TA_BASELINE
```

```
#define VTA_LEFT     TA_BOTTOM
#define VTA_RIGHT    TA_TOP
#define VTA_CENTER   TA_CENTER
#define VTA_BOTTOM   TA_RIGHT
#define VTA_TOP      TA_LEFT

// ************************************************************************
// font types
// ************************************************************************

#define FW_DONTCARE        0
#define FW_THIN            100
#define FW_EXTRALIGHT      200
#define FW_LIGHT           300
#define FW_NORMAL          400
#define FW_MEDIUM          500
#define FW_SEMIBOLD        600
#define FW_BOLD            700
#define FW_EXTRABOLD       800
#define FW_HEAVY           900

#define FW_ULTRALIGHT      FW_EXTRALIGHT
#define FW_REGULAR         FW_NORMAL
#define FW_DEMIBOLD        FW_SEMIBOLD
#define FW_ULTRABOLD       FW_EXTRABOLD
#define FW_BLACK           FW_HEAVY

// ************************************************************************
// raster op codes
// ************************************************************************

#define R2_BLACK           1   /*  0      */
#define R2_NOTMERGEPEN     2   /* DPon    */
#define R2_MASKNOTPEN      3   /* DPna    */
#define R2_NOTCOPYPEN      4   /* PN      */
#define R2_MASKPENNOT      5   /* PDna    */
#define R2_NOT             6   /* Dn      */
#define R2_XORPEN          7   /* DPx     */
#define R2_NOTMASKPEN      8   /* DPan    */
#define R2_MASKPEN         9   /* DPa     */
#define R2_NOTXORPEN       10  /* DPxn    */
#define R2_NOP             11  /* D       */
#define R2_MERGENOTPEN     12  /* DPno    */
#define R2_COPYPEN         13  /* P       */
#define R2_MERGEPENNOT     14  /* PDno    */
#define R2_MERGEPEN        15  /* DPo     */
#define R2_WHITE           16  /*  1      */
#define R2_LAST            16

// ************************************************************************
// Ternary raster operations
// ************************************************************************

#define SRCCOPY            0x00CC0020 /* dest = source                  */
#define SRCPAINT           0x00EE0086 /* dest = source OR dest          */
#define SRCAND             0x008800C6 /* dest = source AND dest         */
#define SRCINVERT          0x00660046 /* dest = source XOR dest         */
#define SRCERASE           0x00440328 /* dest = source AND (NOT dest )  */
```

```
#define NOTSRCCOPY          0x00330008 /* dest = (NOT source)             */
#define NOTSRCERASE         0x001100A6 /* dest = (NOT src) AND (NOT dest) */
#define MERGECOPY           0x00C000CA /* dest = (source AND pattern)     */
#define MERGEPAINT          0x00BB0226 /* dest = (NOT source) OR dest     */
#define PATCOPY             0x00F00021 /* dest = pattern                  */
#define PATPAINT            0x00FB0A09 /* dest = DPSnoo                   */
#define PATINVERT           0x005A0049 /* dest = pattern XOR dest         */
#define DSTINVERT           0x00550009 /* dest = (NOT dest)               */
#define BLACKNESS           0x00000042 /* dest = BLACK                    */
#define WHITENESS           0x00FF0062 /* dest = WHITE                    */

// Quaternary raster codes

#define MAKEROP4(fore,back) (DWORD)((((back) << 8) & 0xFF000000) | (fore))

// Colour Message Box

#define MESSAGE_LEVEL_GENERAL 1
#define MESSAGE_LEVEL_WARNING 2
#define MESSAGE_LEVEL_ERROR   3
#define MESSAGE_LEVEL_GOOD    4


#endif
```

B Appendix - Ascii, Ansi and Unicode

From ***12d Model 10*** onwards, text is stored in the ***12d Model*** database as Unicode (UTF-16 Unicode) and the default format for all output files produced by ***12d Model*** is for them to be Unicode files.

But what does that mean?

Computers can only understands numbers (only zeros and ones actually) so a common code is needed for the numerical representation of characters such as 'a' or '1' or some action such as TAB and a number of common codes have evolved over time.

The common code is not only needed for text in a file or text on a Web page, but also for the names of the files and folders on a computer disc or an internet site.

See ASCII Character Set
See ANSI Character Set
See Unicode Character Set
See Unicode Encoding: UTF-8
See Unicode Encoding: UTF-16
See Endian and BOM

ASCII Character Set

The ASCII (American Standard Code for Information Exchange) was first published in 1963 and was adopted by the American National Standards Institute (ANSI) during the 1960's and has been in common use since then.

The ASCII definition used 7 bits to define characters and some non character codes such as tab, back space and line feed (new line). The seven bits means that only a maximum of 127 codes are allowed.

An examples of the ASCII codes are:

2 is the ASCII code for start of text (STX)

8 is the ASCII code for back space (BS)

9 is the ASCII code for horizontal tab (TAB)

10 is the ASCII code for line feed, new line (NL)

27 is the ASCII code for escape (ESC)

32 is the ASCII code for a space (" ")

36 is the ASCII code for a dollar sign **$**

40 is the ASCII code for a left parenthesis **(**

41 is the ASCII code for a right parenthesis **)**

48 is the ASCII code for the digit zero **0**

49 is the ASCII code for the digit zero **1**

65 is the ASCII code for the Latin capital letter A **A**

97 is the ASCII code for the Latin small letter a **a**

126 is the ASCII code for a tilde **~**

127 is not used

Even with the newer standards, the 7-bit ASCII table continues to be the backbone of modern computing and data storage. Is is so ubiquitous that the terms "text file" and "ascii file" have come to mean the same thing for most computer users.

The ASCII standard was good, as long as you were only working in US English.

ANSI Character Set

The ANSI standard extended the ASCII character set. In the ANSI standard, the first 128 characters were the same as for ASCII but from character 128 onwards, there were different ways depending on where you lived. These different ways were called ***code pages***.

For example, in Israel DOS used a code page called 862 while Greek users used code page 737.

The ANSI set of 218 characters (also know as Windows-1252) was the standard for core fonts supplied with US versions of Microsoft Windows up to and including Windows 95 and Windows NT 4 (character 218 was the euro currency symbol was added during this time).

ANSI characters 32 to 127 correspond to those in the 7-bit ASCII character set.

Some of the extra ANSI codes are:

163 is the ANSI code for a currency Pound sign

165 is the ANSI code for a currency Yen sign

If you use a version of Windows that is designed for a non-Latin alphabet such Arabic, Cyrillic, Greek or Thai to view a document that has been typed using the ANSI character set, then in the code page for the characters from these languages may replace some of those in the 128-255 range and so the document will look different.

There are similar problems when transferring ANSI documents to DOS or Macintosh computers, because DOS and MacRoman arrange characters differently in the 128-255 range.

Unicode Character Set

Today people want to transfer information around the world in emails and on Web sites but the ASCII and ANSII character sets can not work with a variety of Latin and non-Latin alphabets in the one document.

The solution is to move to a system that assigns a unique number to each character in each of the major languages of the world. Such as system has been developed and is known as **Unicode** and it is intended to be used on all computer systems, not just Windows.

The Unicode Standard covers more than 110,000 characters covering 100 scripts, a set of code charts for visual reference, an encoding methodology and set of standard character encodings, an enumeration of character properties such as upper and lower case, a set of reference data computer files, and a number of related items such as character properties, rules for normalisation, decomposition, collation rendering and bidirectional display order (for the correct display of text containing both right-to-left scripts such as Arabic and Hebrew and left-to-right scripts such as English). As of 2012, the most recent version is ***Unicode 6.1***

Unicode's success at unifying character sets has led to its widespread use in computer software and the standard has been implemented in XML, Java, Microsoft .NET Framework and modern operating systems.

To make it Unicode compatible with ASCII, the first 128 characters where the same as for ASCII but from character 128 onwards they are totally different.

All the Unicode characters can be covered with 32 bits but to use a 32-bit representation in a file means that a standard ASCII file would be four times as large when written out in Unicode.

So to save on disk space, and the size of files for emailing etc, there are a number of different mapping methods, or character encodings, for writing Unicode characters to a file.

The Unicode standard defines two mapping methods: the Unicode Transformation Format (UTF) encodings, and the Universal Characters Set (UCS) encodings. An encoding maps the range of Unicode characters (or possibly a subset) to sequences of values in some fixed-size range.

Note: Even though software stores Unicode characters, the computer system still needs the graphics for the character sets to be able to correctly display the Unicode characters.

Unicode Encoding: UTF-8

One of the most common character encodings is UTF-8.

In UTF-8 encoding, only 8-bits are used for any ASCII characters from 0 to 127. For the characters 128 and above, it uses between 16, 24 and up to 48 bits.

And because the representation of the first 128 characters are the same in Unicode and ASCII, US English text looks exactly the same in UTF-8 as it did in ASCII.

So why can't a standard ASCII text editor, or a program requiring plain ASCII text have problems with a Unicode file just containing ASCII characters?

The main reason is that in many Unicode files, a special character called a BOM (see Endian and BOM) is often placed at the beginning of the file, and the BOM would not be recognised by a program only expecting ASCII and would generate an error or show up as blank spaces or strange-looking characters.

Unicode Encoding: UTF-16

In UTF-16 encoding, 16-bits are the basic unit and depending on the Unicode character, UTF-16 encoding may require one or two 16-bit code units. Using the two 16-bit code units, UTF-16 is capable of encoding up to 1,112,064 numbers.

The basic unit of computers is a byte which consists of 8-bits. Because the UFT-16 encoding uses 16-bit and so is made up of two bytes, the order of the bytes may depend on the endianness (byte order) of the computer architecture.

To assist in recognizing the byte order of code units, UTF-16 allows a Byte Order Mark (BOM - see Endian and BOM), a code with a special value to precede the first actual coded value.

Because the fundamental unit in UFT-16 is 16 bits, storing a text file only containing ASCII text will take twice as much disk space as the ASCII version.

Microsoft has used UTF-16 for internal storage for Windows NT and its descendents including Windows 2000, WIndows XP, Windows Vista and Windows 7.

Endian and BOM

From early computing, the fundamental unit of storage was a byte consisting of 8-bits (a bit is a one or a zero). When computers started using 16-bits, this could be stored as two bytes but there was a choice of the order of storing the two bytes. Two different approaches arose and are referred to the endian or endianness.

Big endian stores the most significant byte first and the least significant byte second. Similar to a number written on paper. ***Little endian*** stores the least significant byte first and the most significant byte second.

The ***byte order mark*** (BOM) is a Unicode character used to signal endianness (byte order) of a text file or character stream.

A BOM is essential when the basic unit of an encoding consists of two bytes such as in UTF-16.

Beyond its specific use as a byte-order indicator, the BOM character may also indicate which of the Unicode encoding has been used because the values of the bits in the BOM will be different for the different Unicode encodings.

So although a BOM is not strictly necessary for UTF-8 when it only contains ASCII data, it still alerts the software that it is UTF-8.

Some common programs from Microsoft, such as Notepad and Visual C++, add BOMs to UTF-8 files by Default. Google Docs adds a BOM when a Microsoft Word document is downloaded as a .txt file.

When a BOM is used, it should appear at the **start** of the text.

Index

Symbols

, Integer num_pts) 302
,Integer max_pts,Integer &num_pts,Integer start_pt) 490, 565, 591, 596, 609, 614
,Integer max_pts,Integer &num_pts) 302, 489, 565, 591, 596, 608, 614
,Integer num_pts, Integer num_pits) 488
,Integer num_pts,Integer offset) 303
,Integer num_pts,Integer start_pt) 489, 566, 588, 592, 595, 610, 616
,Integer num_pts) 298, 488, 564, 566, 588, 590, 592, 594, 595, 608, 609, 613, 615
,Message_Box message) 783
,Real &zvalue,Integer max_pt,Integer &num_pts,Integer start_pt) 587
,Real &zvalue,Integer max_pts,Integer &num_pts) 586
,Real zvalue,Integer num_pts) 586
,Text &ret) 653
) 282, 660, 705, 720, 890, 891

A

Affine(Dynamic_Element elements, Real rotate_x,Real rotate_y,Real scale_x,Real scale_y,Real dx,Real dy) 894
Angle_intersect(Point pt_1,Real ang_1,Point pt_2, Real ang_2,Point &p) 195
Angle_prompt(Text msg,Text &ret) 658
Append (Widget widget,Widget_Pages pages) 689
Append_hip(Element elt,Real x,Real y,Real radius,Real left_spiral,Real right_spiral) 619
Append_hip(Element elt,Real x,Real y,Real radius) 619
Append_hip(Element elt,Real x,Real y) 618
Append_vip(Element elt,Real ch,Real ht,Real length,Integer mode) 624
Append_vip(Element elt,Real ch,Real ht,Real parabolic) 623
Append_vip(Element elt,Real ch,Real ht) 623
Append(Dynamic_Element from_de,Dynamic_Element &to_de) 156
Append(Dynamic_Text from_dt,Dynamic_Text &to_dt) 158
Append(Text text,Dynamic_Text &dt) 158
Append(Widget widget,Horizontal_Group group) 670
Append(Widget widget,Vertical_Group group) 673
Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut_volume,Real &fill_volume,Real &balance_volume,Text report) 887
Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut,Real &fill,Real &balance,Text report,Integer do_strings,Dynamic_Element &strings,Integer do_sections,Dynamic_Element &sections,Integer section_colour,Integer do_polygons,D 887
Apply_many(Element string,Real separation,Tin tin,Text many_template_file,Real &cut,Real &fill,Real &balance) 887
Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance,Text report,Integer do_strings,Dynamic_Element &strings,Integer do_sections,Dynamic_Element &sections,Integer section 886
Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance,Text report) 886
Apply(Element string,Real start_ch,Real end_ch,Real sep,Tin tin,Text left_template,Text right_template,Real &cut,Real &fill,Real &balance) 886
Apply(Real xpos,Real ypos,Real zpos,Real angle,Tin tin,Text template, Element &xsect) 886
ASCII 1096
Attribute_debug(Element elt) 271
Attribute_delete_all(Element elt) 266
Attribute_delete(Element elt,Integer att_no) 266
Attribute_delete(Element elt,Text att_name) 266

Attribute_dump(Element elt) 271
Attribute_exists(Element elt,Text att_name,Integer &att_no) 266
Attribute_exists(Element elt,Text att_name) 265

B

big endian 1098
Breakline (Tin tin,Integer p1,Integer p2) 283
buttons 11
byte order 1098

C

Calc_alignment(Element elt) 627
Calc_extent(Element elt) 261
Calc_extent(Model model) 234
Calc_extent(View view) 248
Change_of_angle(Line l1,Line l2,Real &angle) 200
Change_of_angle(Real x1,Real y1,Real x2,Real y2,Real x3,Real y3, Real &angle) 200
Clear (Draw_Box box,Integer r,Integer g,Integer b) 719
Clip_string(Element string,Integer direction,Real chainage1,Real chainage2,Element &left_string,Element &mid_string,Element &right_string) 890
Clip_string(Element string,Real chainage1,Real chainage2, Element &left_string,Element &mid_string,Element &right_string) 889
Colour_exists(Integer col_number) 201
Colour_exists(Text col_name) 201
Colour_prompt(Text msg,Text &ret) 651
Colour_triangles(Tin tin,Integer colour, Element poly,Integer mode) 286
Contour(Tin tin,Real cmin,Real cmax,Real cinc,Real cont_ref,Integer cont_col,Dynamic_Element &cont_de,Real bold_inc,Integer bold_col,Dynamic_Element &bold_de) 873
Convert_colour(Integer col_number, Text &col_name) 202
Convert_colour(Text col_name,Integer &col_number) 201
Convert_time(Integer t1,Text &t2) 125
Convert_time(Integer t1,Text format,Text &t2) 125
Convert_time(Text &t1,Integer t2) 125
Convert(Dynamic_Element in_de,Integer mode, Integer pass_others, Dynamic_Element &out_de) 896
Convert(Element elt,Text type,Element &newelt) 896
Create_2d(Integer num_pts,Element seed) 586
Create_2d(Integer num_pts) 586
Create_3d(Integer num_pts,Element seed) 590
Create_3d(Integer num_pts) 590
Create_3d(Line line) 590
Create_4d(Integer num_pts,Element seed) 595
Create_4d(Integer num_pts) 594
Create_align() 618
Create_align(Element seed) 618
Create_angle_box(Text title,Message_Box message) 693
Create_arc_2(Real xs,Real ys,Real zs,Real radius,Real arc_length,Real start_angle) 463
Create_arc_3(Real xs,Real ys,Real zs,Real radius,Real arc_length,Real chord_angle) 463
Create_arc(Arc arc) 461
Create_arc(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3) 461
Create_arc(Real xc,Real yc,Real zc,Real rad,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze) 461
Create_arc(Real xc,Real yc,Real zc,Real radius,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze) 462
Create_arc(Real xc,Real yc,Real zc,Real xs,Real ys,Real zs,Real sweep) 462
Create_arc(Real xc,Real yc,Real zc,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze,Integer dir) 462
Create_button(Menu menu,Text button_text,Text button_reply) 152
Create_button(Text title,Text reply) 844
Create_child_button(Text title) 844

Create_choice_box(Text title,Message_Box message) 704
Create_circle(Real x1,Real y1,Real z1,Real x2,Real y2,Real z2,Real x3,Real y3,Real z3) 467
Create_circle(Real xc,Real yc,Real zc, Real xp,Real yp,Real zp) 467
Create_circle(Real xc,Real yc,Real zc,Real radius) 467
Create_colour_box(Text title,Message_Box message) 708
Create_directory_box(Text title,Message_Box message,Integer mode) 714
Create_drainage(Integer num_pts,Integer num_pits) 488
Create_draw_box (Integer width,Integer height,Integer border) 717
Create_feature() 562
Create_feature(Element seed) 562
Create_feature(Text name,Integer colour,Real xc,Real yc,Real zc,Real radius) 562
Create_file_box(Text title,Message_Box message,Integer mode,Text wild) 724
Create_finish_button (Text title,Text reply) 845
Create_input_box(Text title,Message_Box message) 733
Create_integer_box(Text title,Message_Box message) 735
Create_interface(Integer num_pts,Element seed) 564
Create_interface(Integer num_pts) 564
Create_justify_box(Text title,Message_Box message) 739
Create_linestyle_box(Text title,Message_Box message,Integer mode) 742
Create_list_box (Text title,Message_Box message,Integer nlines) 745
Create_map_file_box(Text title,Message_Box message,Integer mode) 747
Create_menu(Text menu_title) 152
Create_message_box(Text title) 833
Create_model_box(Text title,Message_Box message,Integer mode) 750
Create_model(Text model_name) 229
Create_name_box(Text title,Message_Box message) 753
Create_named_tick_box(Text title,Integer state,Text response) 755
Create_pipe(Integer num_pts,Element seed) 608
Create_pipe(Integer num_pts) 608
Create_pipeline() 483
Create_pipeline(Element seed) 483
Create_plot_frame(Text name) 575
Create_plotter_box(Text title,Message_Box message) 766
Create_polyline(Integer num_pts,Element seed) 613
Create_polyline(Integer num_pts) 613
Create_polyline(Segment segment) 614
Create_real_box(Text title,Message_Box message) 771
Create_report_box(Text title,Message_Box message,Integer mode) 774
Create_screen_text (Text text) 776
Create_select_box(Text title,Text select_title,Integer mode,Message_Box message) 778
Create_select_button(Text title,Integer mode,Message_Box box) 846
Create_sheet_size_box(Text title,Message_Box message) 789
Create_super(Integer flag,Integer npts) 297
Create_super(Integer flag,Segment seg) 298
Create_super(Integer npts,Element seed) 297
Create_template_box(Text title,Message_Box message,Integer mode) 804
Create_text_edit_box(Text name,Message_Box box,Integer no_lines) 814
Create_text_style_box(Text title,Message_Box message) 807
Create_text_units_box(Text title,Message_Box message) 809
Create_text(Text text,Real x,Real y,Real size, Integer colour,Real angle) 469
Create_text(Text text,Real x,Real y,Real size, Integer colour) 469
Create_text(Text text,Real x,Real y,Real size,Integer colour,Real angle,Integer justif, Integer size_mode) 470
Create_text(Text text,Real x,Real y,Real size,Integer colour,Real angle,Integer justif,Integer size_mode,Real offset_distance,Real rise_distance) 470
Create_text(Text text,Real x,Real y,Real size,Integer colour,Real angle,Integer justif) 470
Create_tick_box(Message_Box message) 820
Create_tin_box(Text title,Message_Box message,Integer mode) 822
Create_view_box(Text title,Message_Box message,Integer mode) 825

Create_xyz_box(Text title,Message_Box message) 828
Cut_strings_with_nulls(Dynamic_Element seed,Dynamic_Element strings,Dynamic_Element &result) 897
Cut_strings(Dynamic_Element seed,Dynamic_Element strings,Dynamic_Element &result) 897

D

Date(Integer &d,Integer &m,Integer &y) 123
Date(Text &date) 123
Delete_hip(Element elt,Integer i) 622
Delete_vip(Element elt,Integer i) 626
Destroy_on_exit( ) 73
direction text 83, 344, 362
Directory_prompt(Text msg,Text &ret) 659
Display_relative(Menu menu, Integer &across_rel,Integer &down_rel,Text &reply) 153
Display(Menu menu,Integer &across_pos,Integer &down_pos,Text &reply) 153
drainage junction 487
drainage network 487
Drainage_pipe_attribute_debug (Element elt,Integer pipe) 550
Drainage_pipe_attribute_delete (Element elt,Integer pipe,Integer att_no) 549
Drainage_pipe_attribute_delete (Element elt,Integer pipe,Text att_name) 549
Drainage_pipe_attribute_delete_all (Element elt,Integer pipe) 550
Drainage_pipe_attribute_dump (Element elt,Integer pipe) 550
Drainage_pipe_attribute_exists (Element elt, Integer pipe,Text name,Integer &no) 549
Drainage_pipe_attribute_exists (Element elt,Integer pipe,Text att_name) 548
Drainage_pit_attribute_debug (Element elt,Integer pit) 528
Drainage_pit_attribute_delete (Element elt,Integer pit,Integer att_no) 528
Drainage_pit_attribute_delete (Element elt,Integer pit,Text att_name) 527
Drainage_pit_attribute_delete_all (Element elt,Integer pit) 528
Drainage_pit_attribute_dump (Element elt,Integer pit) 528
Drainage_pit_attribute_exists (Element elt,Integer pit,Text att_name) 527
Drainage_pit_attribute_exists (Element elt,Integer pit,Text name,Integer &no) 527
Drape(Tin tin,Dynamic_Element de, Dynamic_Element &draped_elts) 875
Drape(Tin tin,Model model,Dynamic_Element &draped_elts) 875
Draw_text (Draw_Box box,Real x,Real y,Real size,Real ht,Text text) 721
Draw_to (Draw_Box box,Real x,Real y) 720
Draw_triangle (Tin tin,Integer tri,Integer c) 281
Draw_triangles_about_point(Tin tin,Integer pt ,Integer c) 282
Drop_point(Element elt,Real xd,Real yd,Real zd,Real &xf,Real &yf, Real &zf,Real &ch,Real &inst_dir,Real &off,Segment &segment) 633
Drop_point(Element elt,Real xd,Real yd,Real zd,Real &xf,Real &yf, Real &zf,Real &ch,Real &inst_dir,Real &off) 633
Drop_point(Segment segment,Point pt_to_drop, Point &dropped_pt,Real &dist) 198
Drop_point(Segment segment,Point pt_to_drop, Point &dropped_pt) 198

E

Element_delete(Element elt) 261
Element_draw(Element elt, Integer colour) 630
Element_draw(Element elt) 630
Element_duplicate(Element elt,Element &dup_elt) 261
Element_exists(Element elt) 254
End_batch_draw (Draw_Box box) 719
endian 1098
endianness 1098
Error_prompt(Text msg) 653
Exit(Integer code) 73
Exit(Text msg) 73
Extend_string(Element elt,Real before,Real after,Element &newelt) 889

F

Face_drape(Tin tin,Dynamic_Element de,Dynamic_Element &face_draped_strings) 875
Face_drape(Tin tin,Model model, Dynamic_Element &face_draped_elts) 875
Factor(Dynamic_Element elements, Real xf,Real yf,Real zf) 898
Fence(Dynamic_Element data_to_fence,Integer mode,Dynamic_Element polygon_list,Dynamic_Element &ret_inside,Dynamic_Element &ret_outside) 899
Fence(Dynamic_Element data_to_fence,Integer mode,Element user_poly,Dynamic_Element &ret_inside,Dynamic_Element &ret_outside) 899
File_close(File file) 148
File_delete(Text file_name) 148
File_exists(Text file_name) 141
File_flush(File file) 144
File_open(Text file_name, Text mode,File &file) 142
File_prompt(Text msg,Text key,Text &ret) 653
File_read_line(File file,Text &text_in) 143
File_rewind(File file) 144
File_seek(File file,Integer pos) 143
File_tell(File file,Integer &pos) 143
File_write_line(File file,Text text_out) 143
Filter(Dynamic_Element in_de,Integer mode, Integer pass_others,Real tolerance,Dynamic_Element &out_de) 900
Find_system_file (Text new_file_name,Text old_file_name,Text env) 127
Find_text(Text text,Text tofind) 77
Fitarc(Point pt_1,Point pt_2,Point pt_3,Arc &fillet) 190
Fitarc(Segment seg_1,Segment seg_2,Point start_tp, Arc &fillet) 191
Fitarc(Segment seg_1,Segment seg_2,Real radius, Point cpt,Arc &fillet) 191
Flip_triangles(Tin tin,Integer t1,Integer t2) 283
From_text(Text text, Dynamic_Text &de) 80
From_text(Text text, Integer &value,Text format) 79
From_text(Text text, Integer &value) 79
From_text(Text text, Real &value,Text format) 79
From_text(Text text, Real &value) 79
From_text(Text text, Text &value,Text format) 79
Function_prompt(Text msg,Text &ret) 658
Function_rename(Text original_name,Text new_name) 912

G

Get_2d_data(Element elt,Integer i,Real &x,Real &y) 587
Get_2d_data(Element elt,Real &z) 588
Get_3d_data(Element elt,Integer i, Real &x,Real &y,Real &z) 592
Get_4d_angle(Element elt,Real &angle) 600
Get_4d_data(Element elt,Integer i, Real &x,Real &y,Real &z, Text &t) 597
Get_4d_height(Element elt,Real &height) 601
Get_4d_justify(Element elt,Integer &justify) 599
Get_4d_offset(Element elt,Real &offset) 600
Get_4d_rise(Element elt,Real &rise) 601
Get_4d_size(Element elt,Real &size) 599
Get_4d_slant(Element elt,Real &slant) 602
Get_4d_style(Element elt,Text &style) 603
Get_4d_units(Element elt,Integer &units_mode) 598
Get_4d_x_factor(Element elt,Real &xfact) 602
Get_4dmodel_version(Integer &major,Integer &minor,Text &patch) 127
Get_all_linestyles(Dynamic_Text &linestyles) 159
Get_all_textstyles(Dynamic_Text &textstyles) 159
Get_arc_centre(Element elt,Real &xc,Real &yc,Real &zc) 464
Get_arc_data(Element elt,Real &xc,Real &yc,Real &zc, Real &radius,Real &xs,Real &ys,Real &zs,Real &xe,Real &ye,Real &ze) 465

Get_arc_end(Element elt,Real &xe,Real &ye,Real &ze) 465
Get_arc_radius(Element elt,Real &radius) 464
Get_arc_start(Element elt,Real &xs,Real &ys,Real &zs) 464
Get_arc(Segment segment, Arc &arc) 185
Get_attribute_length(Element elt,Integer att_no,Integer &att_len) 269
Get_attribute_length(Element elt,Text att_name,Integer &att_len) 269
Get_attribute_name(Element elt,Integer att_no,Text &name) 268
Get_attribute_type(Element elt,Integer att_no,Integer &att_type) 269
Get_attribute_type(Element elt,Text att_name,Integer &att_type) 268
Get_attribute(Element elt,Integer att_no,Integer &att) 268
Get_attribute(Element elt,Integer att_no,Real &att) 268
Get_attribute(Element elt,Integer att_no,Text &att) 268
Get_attribute(Element elt,Text att_name,Integer &att) 267
Get_attribute(Element elt,Text att_name,Real &att) 267
Get_attribute(Element elt,Text att_name,Text &att) 267
Get_breakline(Element elt,Integer &break_type) 257
Get_centre(Arc arc) 169
Get_chainage(Element elt,Real &start_chain) 258
Get_char(Text t,Integer pos, Integer &c) 82
Get_circle_data(Element elt,Real &xc,Real &yc,Real &zc,Real &radius) 468
Get_colour(Element,Integer &colour) 255
Get_colour(Tin tin,Integer &colour) 286
Get_command_argument(Integer i,Text &argument) 71
Get_cursor_position(Integer &x,Integer &y) 666
Get_data (Screen_Text widget,Text &data) 776
Get_data (Text_Edit_Box widget,Text &data) 814
Get_data(Angle_Box box,Text &data) 693
Get_data(Choice_Box box,Text &data) 705
Get_data(Colour_Box box,Text &data) 710
Get_data(Directory_Box box,Text &data) 715
Get_data(Element elt,Integer i,Real &x,Real &y,Real &z) 254
Get_data(File_Box box,Text &data) 725
Get_data(Input_Box box,Text &data) 734
Get_data(Integer_Box box,Text &data) 736
Get_data(Justify_Box box,Text &data) 739
Get_data(Linestyle_Box box,Text &data) 743
Get_data(Map_File_Box box,Text &data) 747
Get_data(Message_Box box,Text &data) 833
Get_data(Model_Box box,Text &data) 751
Get_data(Name_Box box,Text &data) 754
Get_data(Named_Tick_Box box,Text &data) 756
Get_data(Plotter_Box box,Text &data) 766
Get_data(Real_Box box,Text &data) 771
Get_data(Report_Box box,Text &data) 774
Get_data(Select_Box select,Text &string) 779
Get_data(Select_Boxes select,Integer n,Text &string) 784
Get_data(Select_Button select,Text &string) 848
Get_data(Sheet_Size_Box box,Text &data) 789
Get_data(Template_Box box,Text &data) 804
Get_data(Text_Style_Box box,Text &data) 808
Get_data(Text_Units_Box box,Text &data) 810
Get_data(Tick_Box box,Text &data) 820
Get_data(Tin_Box box,Text &data) 823
Get_data(View_Box box,Text &data) 825
Get_data(XYZ_Box box,Text &data) 828
Get_directory (File_Box box,Text &data) 726
Get_distance_3d(Point p1,Point p2) 196
Get_distance(Point p1,Point p2) 196

Get_drainage_data(Element elt,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &f) 491
Get_drainage_float (Element,Integer &float) 494
Get_drainage_flow(Element elt,Integer &dir) 493
Get_drainage_fs_tin (Element,Tin &tin) 493
Get_drainage_hc_adopted_level(Element elt,Integer h,Real &level) 555
Get_drainage_hc_bush(Element elt,Integer h,Text &bush) 556
Get_drainage_hc_chainage(Element elt,Integer h,Real &chainage) 561
Get_drainage_hc_colour(Element elt,Integer h,Integer &colour) 556
Get_drainage_hc_depth(Element elt,Integer h,Real &depth) 557
Get_drainage_hc_diameter(Element elt,Integer h,Real &diameter) 557
Get_drainage_hc_grade(Element elt,Integer h,Real &grade) 557
Get_drainage_hc_hcb(Element elt,Integer h,Integer &hcb) 558
Get_drainage_hc_length(Element elt,Integer h,Real &length) 558
Get_drainage_hc_level(Element elt,Integer h,Real &level) 559
Get_drainage_hc_material(Element elt,Integer h,Text &material) 559
Get_drainage_hc_name(Element elt,Integer h,Text &name) 560
Get_drainage_hc_side(Element elt,Integer h,Integer &side) 560
Get_drainage_hc_type(Element elt,Integer h,Text &type) 561
Get_drainage_hc(Element elt,Integer h,Real &x,Real &y,Real &z) 555
Get_drainage_hcs(Element elt,Integer &no_hcs) 555
Get_drainage_ns_tin (Element,Tin &tin) 492
Get_drainage_outfall_height(Element elt,Real &ht) 492
Get_drainage_pipe_attribute (Element elt,Integer pipe,Integer att_no,Integer &att) 548
Get_drainage_pipe_attribute (Element elt,Integer pipe,Integer att_no,Real &att) 548
Get_drainage_pipe_attribute (Element elt,Integer pipe,Integer att_no,Text &att) 548
Get_drainage_pipe_attribute (Element elt,Integer pipe,Text att_name,Integer &att) 547
Get_drainage_pipe_attribute (Element elt,Integer pipe,Text att_name,Real &att) 547
Get_drainage_pipe_attribute (Element elt,Integer pipe,Text att_name,Text &att) 547
Get_drainage_pipe_attribute_length (Element elt,Integer pipe,Integer att_no,Integer &att_len) 551
Get_drainage_pipe_attribute_length (Element elt,Integer pipe,Text att_name,Integer &att_len) 550
Get_drainage_pipe_attribute_name (Element elt,Integer pipe,Integer att_no,Text &name) 551
Get_drainage_pipe_attribute_type (Element elt,Integer pipe,Integer att_name,Integer &att_type 552
Get_drainage_pipe_attribute_type (Element elt,Integer pipe,Text att_name,Integer &att_type) 551
Get_drainage_pipe_cover (Element,Integer pipe,Real &minc,Real &maxc) 534
Get_drainage_pipe_diameter(Element elt,Integer p,Real &diameter) 536
Get_drainage_pipe_flow(Element elt,Integer p,Real &flow) 540
Get_drainage_pipe_grade(Element elt,Integer p,Real &grade) 541
Get_drainage_pipe_hgls(Element elt,Integer p,Real &lhs,Real &rhs) 539
Get_drainage_pipe_inverts(Element elt,Integer p,Real &lhs,Real &rhs) 532
Get_drainage_pipe_length(Element elt,Integer p,Real &length) 540
Get_drainage_pipe_name(Element elt,Integer p,Text &name) 533
Get_drainage_pipe_number_of_attributes(Element elt,Integer pipe,Integer &no_atts) 550
Get_drainage_pipe_type(Element elt,Integer p,Text &type) 534
Get_drainage_pipe_velocity(Element elt,Integer p,Real &velocity) 540
Get_drainage_pit_angle (Element,Integer pit,Real &angle,Integer trunk) 505
Get_drainage_pit_angle(Element elt,Integer p,Real &angle) 505
Get_drainage_pit_attribute (Element elt,Integer pit,Integer att_no,Integer &att) 520
Get_drainage_pit_attribute (Element elt,Integer pit,Integer att_no,Real &att) 520
Get_drainage_pit_attribute (Element elt,Integer pit,Integer att_no,Text &att) 520
Get_drainage_pit_attribute (Element elt,Integer pit,Text att_name,Integer &att) 522
Get_drainage_pit_attribute (Element elt,Integer pit,Text att_name,Real &att) 521
Get_drainage_pit_attribute (Element elt,Integer pit,Text att_name,Text &att) 521
Get_drainage_pit_attribute_length (Element elt,Integer pit,Integer att_no,Integer &att_len) 519
Get_drainage_pit_attribute_length (Element elt,Integer pit,Text att_name,Integer &att_len) 519
Get_drainage_pit_attribute_name (Element elt,Integer pit,Integer att_no,Text &name) 520
Get_drainage_pit_attribute_type (Element elt,Integer pit,Integer att_name,Integer &att_type) 519
Get_drainage_pit_attribute_type (Element elt,Integer pit,Text att_name,Integer &att_type) 519
Get_drainage_pit_branches(Element,Integer pit,Dynamic_Element &branches) 511

Get_drainage_pit_chainage(Element elt,Integer p,Real &chainage) 506
Get_drainage_pit_depth(Element elt,Integer p,Real &depth) 511
Get_drainage_pit_diameter(Element elt,Integer p,Real &diameter) 499
Get_drainage_pit_drop(Element elt,Integer p,Real &drop) 512
Get_drainage_pit_float (Element,Integer pit,Integer &float) 508
Get_drainage_pit_hgls(Element elt,Integer p,Real &lhs,Real &rhs) 509
Get_drainage_pit_inverts(Element elt,Integer p,Real &lhs,Real &rhs) 505
Get_drainage_pit_name(Element elt,Integer p,Text &name) 498
Get_drainage_pit_number_of_attributes(Element elt,Integer pit,Integer &no_atts) 521
Get_drainage_pit_road_chainage(Element elt,Integer p,Real &chainage) 510
Get_drainage_pit_road_name(Element elt,Integer p,Text &name) 510
Get_drainage_pit_type(Element elt,Integer p,Text &type) 511
Get_drainage_pit(Element elt,Integer p,Real &x,Real &y,Real &z) 497
Get_drainage_pits(Element elt,Integer &npits) 497
Get_drainage_trunk (Element,Element &trunk) 494
Get_elements(Model model,Dynamic_Element &de, Integer &total_no) 230
Get_enable(Widget widget,Integer &mode) 677
Get_end_chainage(Element elt,Real &chainage) 258
Get_end(Arc arc) 170
Get_end(Line line) 167
Get_end(Segment segment,Point &point) 186
Get_extent_x(Element elt,Real &xmin,Real &xmax) 260
Get_extent_x(Model model,Real &xmin,Real &xmax) 233
Get_extent_y(Element elt,Real &ymin,Real &ymax) 260
Get_extent_y(Model model,Real &ymin,Real &ymax) 234
Get_extent_z(Element elt,Real &zmin,Real &zmax) 260
Get_extent_z(Model model,Real &zmin,Real &zmax) 234
Get_feature_centre(Element elt,Real &xc,Real &yc,Real &zc) 562
Get_feature_radius(Element elt,Real &radius) 563
Get_help (Widget widget,Integer &help) 686
Get_help (Widget widget,Text &help) 687
Get_hip_data(Element elt,Integer i,Real &x,Real &y,Real &radius,Real &left_spiral,Real &right_spiral) 620
Get_hip_data(Element elt,Integer i,Real &x,Real &y,Real &radius) 620
Get_hip_data(Element elt,Integer i,Real &x,Real &y) 619
Get_hip_geom(Element elt,Integer hip_no,Integer mode, Real &x,Real &y) 623
Get_hip_id (Element,Integer position,Integer &id) 628
Get_hip_points(Element elt,Integer &num_pts) 619
Get_hip_type(Element elt,Integer hip_no,Text &type) 622
Get_id(Element elt,Integer &id) 259
Get_id(Widget) 682
Get_interface_data(Element elt,Integer i,Real &x,Real &y,Real &z,Integer &f) 566
Get_item(Dynamic_Element &de,Integer i,Element &elt) 156
Get_item(Dynamic_Text &dt,Integer i,Text &text) 159
Get_length_3d(Element elt,Real &length) 632
Get_length_3d(Segment segment,Real &length) 189
Get_length(Element elt,Real &length) 631
Get_length(Segment segment,Real &length) 189
Get_line(Segment segment, Line &line) 185
Get_macro_name() 125
Get_model_attribute (Model model,Integer att_no,Integer &att) 241
Get_model_attribute (Model model,Integer att_no,Real &att) 241
Get_model_attribute (Model model,Integer att_no,Text &att) 241
Get_model_attribute (Model model,Text att_name,Integer &att) 240
Get_model_attribute (Model model,Text att_name,Real &att) 241
Get_model_attribute (Model model,Text att_name,Text &att) 240
Get_model_attribute_length (Model model,Integer att_no,Integer &att_len) 244
Get_model_attribute_length (Model model,Text att_name,Integer &att_len) 244
Get_model_attribute_name (Model model,Integer att_no,Text &name) 243

Get_model_attribute_type (Model model,Integer att_name,Integer &att_type) 243
Get_model_create(Text model_name) 229
Get_model_number_of_attributes(Model model,Integer &no_atts) 244
Get_model(Element elt,Model &model) 256
Get_model(Text model_name) 231
Get_module_license(Text module_name) 126
Get_name(Element elt,Text &elt_name) 255
Get_name(Function func,Text &name) 912
Get_name(Model model,Text &model_name) 231
Get_name(Tin tin,Text &tin_name) 275
Get_name(View view,Text &view_name) 245
Get_name(Widget widget,Text &text) 680
Get_number_of_attributes(Element elt,Integer &no_atts) 267
Get_number_of_command_arguements 71
Get_number_of_items(Dynamic_Element &de,Integer &no_items) 156
Get_number_of_items(Dynamic_Text &dt,Integer &no_items) 158
Get_number_of_items(Model model,&num) 230
Get_optional(Widget widget,Integer &mode) 678
Get_pipe_data(Element elt,Integer i, Real &x,Real &y,Real &z) 610
Get_pipe_diameter(Element elt, Real &diameter) 611
Get_pipe_justify(Element elt,Integer &justify) 611
Get_pipeline_diameter(Element pipeline,Real &diameter) 483
Get_pipeline_length (Element pipeline,Real &length) 484
Get_plot_frame_colour(Element elt,Integer &colour) 578
Get_plot_frame_draw_border(Element elt,Integer &draw_border) 577
Get_plot_frame_draw_title_file(Element elt,Integer &draw_title) 577
Get_plot_frame_draw_viewport(Element elt,Integer &draw_viewport) 577
Get_plot_frame_margins(Element elt,Real &l,Real &b,Real &r,Real &t) 576
Get_plot_frame_name(Element elt,Text &name) 575
Get_plot_frame_origin(Element elt,Real &x,Real &y) 576
Get_plot_frame_plot_file(Element elt,Text &plot_file) 578
Get_plot_frame_plotter_name(Element elt,Text &plotter_name) 578
Get_plot_frame_plotter(Element elt,Integer &plotter) 578
Get_plot_frame_rotation(Element elt,Real &rotation) 575
Get_plot_frame_scale(Element elt,Real &scale) 575
Get_plot_frame_sheet_size(Element elt,Real &w,Real &h) 576
Get_plot_frame_sheet_size(Element elt,Text &size) 576
Get_plot_frame_text_size(Element elt,Real &text_size) 577
Get_plot_frame_textstyle(Element elt,Text &textstyle) 578
Get_plot_frame_title_1(Element elt,Text &title) 579
Get_plot_frame_title_2(Element elt,Text &title) 579
Get_plot_frame_title_file(Element elt,Text &title_file) 579
Get_point(Segment segment, Point &point) 185
Get_points(Element elt,Integer &numpts) 254
Get_polyline_data(Element elt,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &f) 615
Get_position(Element elt,Real ch,Real &x,Real &y, Real &z,Real &inst_dir) 632
Get_position(Element elt,Real ch,Real &x,Real &y,Real &z,Real &inst_dir,Real &radius, Real &inst_grade) 633
Get_project_attribute (Integer att_no,Integer &att) 227
Get_project_attribute (Integer att_no,Real &att) 227
Get_project_attribute (Integer att_no,Text &att) 226
Get_project_attribute (Text att_name,Integer &att) 226
Get_project_attribute (Text att_name,Real &att) 225
Get_project_attribute (Text att_name,Text &att) 228
Get_project_attribute_length (Integer att_no,Integer &att_len) 224
Get_project_attribute_length (Text att_name,Integer &att_len) 225
Get_project_attribute_name (Integer att_no,Text &name) 224
Get_project_attribute_type (Integer att_no,Integer &att_type) 225
Get_project_attribute_type (Text att_name,Integer &att_type) 225

Get_project_colours(Dynamic_Text &colours) 202
Get_project_functions(Dynamic_Text &function_names) 219
Get_project_models(Dynamic_Text &model_names) 231
Get_project_name(Text &name) 219
Get_project_number_of_attributes(Integer &no_atts) 224
Get_project_templates(Dynamic_Text &template_names) 885
Get_project_tins(Dynamic_Text &tins) 274
Get_project_views(Dynamic_Text &view_names) 246
Get_radius(Arc arc) 169
Get_read_locks (Element elt,Integer &no_locks) 635
Get_segment(Element elt,Integer i,Segment &seg) 188
Get_segments(Element elt,Integer &nsegs) 188
Get_select_coordinate(Select_Box select,Real &x,Real &y,Real &z,Real &ch,Real &ht) 781
Get_select_coordinate(Select_Boxes select,Integer n,Real &x,Real &y,Real &z,Real &ch,Real &ht) 786
Get_select_coordinate(Select_Button select,Real &x,Real &y,Real &z,Real &ch,Real &ht) 850
Get_select_direction(Select_Box select,Integer &dir) 781
Get_select_direction(Select_Boxes select,Integer n,Integer &dir) 786
Get_select_direction(Select_Button select,Integer &dir) 849
Get_size (Draw_Box,Integer &x,Integer &y) 718
Get_size (Widget widget,Integer &x,Integer &y) 681
Get_sort (List_Box box,Integer &mode) 745
Get_start(Arc arc) 170
Get_start(Line line) 167
Get_start(Segment segment,Point &point) 186
Get_style(Element elt,Text &elt_style) 257
Get_subtext(Text text,Integer start,Integer end) 77
Get_super_2d_level (Element,Real &level) 309
Get_super_culvert (Element,Real &w,Real &h) 342
Get_super_data(Element,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &f) 302
Get_super_diameter (Element,Real &diameter) 340
Get_super_pipe_justify (Element,Integer &justify) 334
Get_super_segment_attribute (Element elt,Integer seg,Integer att_no,Integer &att) 427
Get_super_segment_attribute (Element elt,Integer seg,Integer att_no,Real &att) 427
Get_super_segment_attribute (Element elt,Integer seg,Integer att_no,Text &att) 426
Get_super_segment_attribute (Element elt,Integer seg,Text att_name,Integer &att) 426
Get_super_segment_attribute (Element elt,Integer seg,Text att_name,Real &att) 426
Get_super_segment_attribute (Element elt,Integer seg,Text att_name,Text &att) 425
Get_super_segment_attribute_length (Element elt,Integer seg,Integer att_no,Integer &att_len) 428
Get_super_segment_attribute_length (Element elt,Integer seg,Text att_name,Integer &att_len) 428
Get_super_segment_attribute_name (Element elt,Integer seg,Integer att_no,Text &name) 427
Get_super_segment_attribute_type (Element elt,Integer seg,Integer att_name,Integer &att_type) 428
Get_super_segment_attribute_type (Element elt,Integer seg,Text att_name,Integer &att_type) 427
Get_super_segment_colour (Element,Integer seg,Integer &colour) 397
Get_super_segment_culvert (Element,Integer seg,Real &w,Real &h) 343
Get_super_segment_diameter (Element,Integer seg,Real &diameter) 341
Get_super_segment_major (Element,Integer seg,Integer &major) 317
Get_super_segment_number_of_attributes(Element elt,Integer seg,Integer &no_atts) 425
Get_super_segment_radius (Element,Integer seg,Real &radius) 317
Get_super_segment_text_angle (Element,Integer vert,Real &a) 372
Get_super_segment_text_colour (Element,Integer vert,Integer &c) 371
Get_super_segment_text_justify (Element,Integer vert,Integer &j) 369
Get_super_segment_text_offset_height (Element,Integer vert,Real &o) 371
Get_super_segment_text_offset_width (Element,Integer vert,Real &o) 370
Get_super_segment_text_size (Element,Integer vert,Real &s) 372
Get_super_segment_text_slant (Element,Integer vert,Real &s) 373
Get_super_segment_text_style (Element,Integer vert,Text &s) 374
Get_super_segment_text_type (Element,Integer &type) 369
Get_super_segment_text_x_factor (Element,Integer vert,Real &x) 373

Get_super_segment_tinability (Element,Integer seg,Integer &tinability) 315
Get_super_segment_visibility (Element,Integer seg,Integer &visibility) 442
Get_super_use_2d_level (Element,Integer &use) 308
Get_super_use_3d_level (Element,Integer &use) 309
Get_super_use_culvert (Element,Integer &use) 331
Get_super_use_diameter (Element,Integer &use) 330
Get_super_use_pipe_justify (Element,Integer &use) 332
Get_super_use_segment_annotation_array(Element,Integer &use) 366
Get_super_use_segment_annotation_value(Element,Integer &use) 366
Get_super_use_segment_attribute (Element,Integer &use) 420
Get_super_use_segment_colour (Element,Integer &use) 397
Get_super_use_segment_culvert (Element,Integer &use) 332
Get_super_use_segment_diameter (Element,Integer &use) 330
Get_super_use_segment_radius (Element,Integer &use) 316
Get_super_use_segment_text_array (Element,Integer &use) 365
Get_super_use_segment_text_value (Element,Integer &use) 364
Get_super_use_symbol (Element,Integer &use) 322
Get_super_use_tinability (Element,Integer &use) 311
Get_super_use_vertex_annotation_array(Element,Integer &use) 348
Get_super_use_vertex_annotation_value(Element,Integer &use) 347
Get_super_use_vertex_attribute (Element,Integer &use) 409
Get_super_use_vertex_point_number (Element,Integer &use) 318
Get_super_use_vertex_symbol (Element,Integer &use) 322
Get_super_use_vertex_text_array (Element,Integer &use) 346
Get_super_use_vertex_text_value (Element,Integer &use) 345
Get_super_use_visibility (Element,Integer &use) 438
Get_super_vertex_attribute (Element elt,Integer vert,Integer att_no,Integer &att) 416
Get_super_vertex_attribute (Element elt,Integer vert,Integer att_no,Real &att) 416
Get_super_vertex_attribute (Element elt,Integer vert,Integer att_no,Text &att) 415
Get_super_vertex_attribute (Element elt,Integer vert,Text att_name,Integer &att) 415
Get_super_vertex_attribute (Element elt,Integer vert,Text att_name,Real &att) 415
Get_super_vertex_attribute (Element elt,Integer vert,Text att_name,Text &att) 414
Get_super_vertex_attribute_length (Element elt,Integer vert,Integer att_no,Integer &att_len) 417
Get_super_vertex_attribute_length (Element elt,Integer vert,Text att_name,Integer &att_len) 416
Get_super_vertex_attribute_name (Element elt,Integer vert,Integer att_no,Text &name) 416
Get_super_vertex_attribute_type (Element elt,Integer vert,Integer att_name,Integer &att_type) 417
Get_super_vertex_attribute_type(Element elt,Integer vert,Text att_name,Integer &att_type) 417
Get_super_vertex_coord (Element,Integer vert,Real &x,Real &y,Real &z) 301
Get_super_vertex_number_of_attributes(Element elt,Integer vert,Integer &no_atts) 414
Get_super_vertex_point_number (Element,Integer vert,Integer &point_number) 319
Get_super_vertex_symbol_colour (Element,Integer vert,Integer &c) 324
Get_super_vertex_symbol_offset_height(Element,Integer vert,Real &r) 326
Get_super_vertex_symbol_offset_width (Element,Integer vert,Real &o) 325
Get_super_vertex_symbol_rotation (Element,Integer vert,Real &a) 326
Get_super_vertex_symbol_size (Element,Integer vert,Real &s) 327
Get_super_vertex_symbol_style (Element,Integer vert,Text &s) 324
Get_super_vertex_text (Element,Integer vert,Text &text) 349
Get_super_vertex_text_angle (Element,Integer vert,Real &a) 353
Get_super_vertex_text_colour (Element,Integer vert,Integer &c) 352
Get_super_vertex_text_justify (Element,Integer vert,Integer &j) 351
Get_super_vertex_text_offset_height (Element,Integer vert,Real &o) 352
Get_super_vertex_text_offset_width (Element,Integer vert,Real &o) 351
Get_super_vertex_text_size (Element,Integer vert,Real &s) 353
Get_super_vertex_text_slant (Element,Integer vert,Real &s) 354
Get_super_vertex_text_style (Element,Integer vert,Text &s) 355
Get_super_vertex_text_type (Element,Integer &type) 350
Get_super_vertex_text_x_factor (Element,Integer vert,Real &x) 354
Get_super_vertex_tinability (Element,Integer vert,Integer &tinability) 313

Get_super_vertex_visibility (Element,Integer vert,Integer &visibility) 440
Get_text_angle(Element elt,Real &angle) 475
Get_text_data(Element elt,Text &text,Real &x,Real &y,Real &size,Integer &colour,Real &angle,Integer &justification,Integer &size_mode,Real &offset_dist,Real &rise_dist) 471
Get_text_height(Element elt,Real &height) 476
Get_text_justify(Element elt,Integer &justify) 474
Get_text_length(Element elt,Real &length) 472
Get_text_offset(Element elt,Real &offset) 475
Get_text_rise(Element elt,Real &rise) 476
Get_text_size(Element elt,Real &size) 474
Get_text_slant(Element elt,Real &slant) 476
Get_text_style(Element elt,Text &style) 477
Get_text_units(Element elt,Integer &units_mode) 473
Get_text_value(Element elt,Text &text) 471
Get_text_x_factor(Element elt,Real &xfact) 477
Get_text_xy(Element elt,Real &x, Real &y) 473
Get_time_created(Element elt,Integer &time) 259
Get_time_updated(Element elt,Integer &time) 259
Get_tin(Element elt) 274
Get_tin(Text tin_name) 274
Get_tooltip (Widget widget,Text &help) 686
Get_type (Function_Box box,Integer &type) 729
Get_type (Function_Box box,Text &type) 729
Get_type(Element elt,Integer &elt_type) 261
Get_type(Element elt,Text &elt_type) 257
Get_type(Segment segment) 185
Get_type(View view,Text &type) 246
Get_user_name(Text &name) 126
Get_view(Text view_name) 246
Get_vip_data(Element elt,Integer i, Real &ch,Real &ht,Real &parabolic) 624
Get_vip_data(Element elt,Integer i,Real &ch,Real &ht,Real &value,Integer &mode) 624
Get_vip_data(Element elt,Integer i,Real &ch,Real &ht) 624
Get_vip_geom(Element elt,Integer vip_no,Integer mode,Real &chainage,Real &height) 627
Get_vip_id (Element,Integer position,Integer &id) 628
Get_vip_points(Element elt,Integer &num_pts) 624
Get_vip_type(Element elt,Integer vip_no,Text &type) 627
Get_widget_position(Widget widget,Integer &x,Integer &y) 682
Get_widget_size(Widget widget,Integer &w,Integer &h) 681
Get_wildcard (File_Box box,Text &data) 725
Get_write_locks(Element elt,Integer &no_locks) 635
Get_x(Point pt) 165
Get_y(Point pt) 165
Get_z(Point pt) 165
Getenv (Text env) 127

H

Head_to_tail(Dynamic_Element in_list, Dynamic_Element &out_list) 901
Helmert(Dynamic_Element elements, Real rotate,Real scale,Real dx,Real dy) 902
Hide_widget(Widget widget) 681
Horizontal_Group Create_button_group() 670
Horizontal_Group Create_horizontal_group(Integer mode) 670

I

Insert_hip(Element elt,Integer i, Real x,Real y,Real radius,Real left_spiral,Real right_spiral) 622
Insert_hip(Element elt,Integer i, Real x,Real y,Real radius) 621
Insert_hip(Element elt,Integer i,Real x,Real y) 621

Insert_text(Text &text,Integer start,Text sub) 78
Insert_vip(Element elt,Integer i, Real ch,Real ht,Real parabolic) 626
Insert_vip(Element elt,Integer i,Real ch,Real ht,Real value,Integer mode) 626
Insert_vip(Element elt,Integer i,Real ch,Real ht) 626
Integer Null(Element elt) 260
Integer Set_size (Widget widget,Integer x,Integer y) 681
Integer Set_super_pipe_justify (Element,Integer justify) 334
Interface(Tin tin,Element string,Real cut_slope,Real fill_slope,Real sep,Real search_dist,Integer side, Element &interface_string,Dynamic_Element &tadpoles) 884
Interface(Tin tin,Element string,Real cut_slope,Real fill_slope,Real sep,Real search_dist,Integer side,Element &interface_string) 884
Intersect_extended(Segment seg_1,Segment seg_2,Integer &no_intersects,Point &p1,Point &p2) 193, 194
Intersect(Segment seg_1,Segment seg_2,Integer &no_intersects,Point &p1,Point &p2) 193
Is_null(Real value) 871
Is_practise_version() 128

J

Join_strings(Element string1,Real x1,Real y1,Real z1,Element string2,Real x2,Real y2,Real z2,Element &joined_string) 891
junction 487
justification point 83, 321, 344, 362
Justify_prompt(Text msg,Text &ret) 658

L

Linestyle_prompt(Text msg,Text &ret) 657
little endian 1098
Locate_point(Point from,Real angle,Real dist,Point &to) 197
Loop_clean(Element elt,Point ok_pt,Element &new_elt) 634

M

Map_file_add_key(Map_File file,Text key,Text name,Text model,Integer colour,Integer ptln,Text style) 646
Map_file_close(Map_File file) 645
Map_file_create(Map_File &file) 645
Map_file_find_key(Map_File file,Text key, Integer &number) 646
Map_file_get_key(Map_File file,Integer n,Text &key,Text &name,Text &model, Integer &colour,Integer &ptln,Text &style) 646
Map_file_number_of_keys(Map_File file,Integer &number) 645
Map_file_open(Text file_name, Text prefix, Integer use_ptline,Map_File &file) 645
Match_name(Dynamic_Element de,Text reg_exp, Dynamic_Element &matched) 870
Match_name(Text name,Text reg_exp) 870
Menu_delete(Menu menu) 152
Model_attribute_debug (Model model) 240
Model_attribute_delete (Model model,Integer att_no) 239
Model_attribute_delete (Model model,Text att_name) 239
Model_attribute_delete_all (Model model,Element elt) 240
Model_attribute_dump (Model model) 240
Model_attribute_exists (Model model,Text att_name) 239
Model_attribute_exists (Model model,Text name,Integer &no) 239
Model_delete(Model model) 236
Model_draw(Model model,Integer colour) 235
Model_draw(Model model) 235
Model_duplicate(Model model,Text dup_name) 234
Model_exists(Model model) 230
Model_exists(Text model_name) 230
Model_get_views(Model model, Dynamic_Text &view_names) 247

Model_prompt(Text msg,Text &ret) 654
Model_rename(Text original_name,Text new_name) 235
mouse buttons
    LB 11
    left 11
    MB 11
    middle 11
    RB 11
    right 11
Move_to (Draw_Box box,Real x,Real y) 720

N

Name_prompt(Text msg,Text &ret) 660
Null_by_angle_length (Tin tin,Real a1,Real l1,Real a2,Real l2) 285
Null_ht_range(Dynamic_Element elements,Real ht_min,Real ht_max) 871
Null_ht(Dynamic_Element elements,Real height) 871
Null_item(Dynamic_Element &de,Integer i) 157
Null_triangles(Tin tin, Element poly, Integer mode) 284
Null(Dynamic_Element &de) 156
Null(Dynamic_Text &dt) 158
Null(Model model) 235
Null(Real value) 871
Null(Tin tin) 284
Null(View view) 245
Numchr(Text text) 76

O

Offset_intersect_extended(Segment seg_1,Real off_1,Segment seg_2,Real off_2,Integer &no_intersects,Point &p1,Point &p2) 194

P

Parallel(Arc arc, Real distance, Arc &parallelled) 190
Parallel(Element elt, Real distance, Element &parallelled) 634
Parallel(Line line,Real distance,Line &parallelled) 190
Parallel(Segment segment, Real dist, Segment &parallelled) 190
Plan_area(Element elt, Real &plan_area) 632
Plan_area(Segment segment,Real &plan_area) 189
Plot_ppf_file(Text name) 951
Plotter_prompt(Text msg,Text &ret) 656
Print(Integer value) 138
Print(Real value) 138
Print(Text msg) 138
Project_attribute_debug () 224
Project_attribute_delete (Integer att_no) 223
Project_attribute_delete (Text att_name) 223
Project_attribute_delete_all (Element elt) 223
Project_attribute_dump() 224
Project_attribute_exists (Text att_name) 223
Project_attribute_exists (Text name,Integer &no) 223
Project_prompt(Text msg,Text &ret) 659
Projection(Segment segment,Point start_point, Real dist,Point &projected_pt) 199
Projection(Segment segment,Real dist,Point &projected_pt) 199
Prompt(Text msg,Integer &ret) 651
Prompt(Text msg,Real &ret) 651
Prompt(Text msg,Text &ret) 651

Prompt(Text msg) 650

R

Reset_colour_triangles(Tin tin,Element poly,Integer mode) 287
Reset_colour_triangles(Tin tin) 287
Reset_null_ht(Dynamic_Element elements,Real height) 872
Reset_null_triangles(Tin tin,Element poly, Integer mode) 284
Reset_null_triangles(Tin tin) 284
Retain_on_exit( ) 73
Retriangulate (Tin tin) 282
Reverse (Segment segment) 188
Reverse(Arc arc) 171
Reverse(Line line) 168
Rotate(Dynamic_Element elements, Real xorg,Real yorg,Real angle) 903

S

Select_string(Text msg,Element &string,Real &x,Real &y,Real &z,Real &ch,Real &ht) 629, 630
Select_string(Text msg,Element &string) 629
Set_2d_data(Element elt,Integer i,Real x, Real y) 589
Set_2d_data(Element elt,Real z) 589
Set_3d_data(Element elt,Integer i,Real x, Real y,Real z) 593
Set_4d_angle(Element elt,Real angle) 599
Set_4d_data(Element elt,Integer i,Real x, Real y,Real z,Text t) 596
Set_4d_height(Element elt,Real height) 601
Set_4d_justify(Element elt,Integer justify) 599
Set_4d_offset(Element elt,Real offset) 600
Set_4d_rise(Element elt,Real rise) 600
Set_4d_size(Element elt,Real size) 598
Set_4d_slant(Element elt,Real slant) 601
Set_4d_style(Element elt,Text style) 602
Set_4d_units(Element elt,Integer units_mode) 598
Set_4d_x_factor(Element elt,Real xfact) 602
Set_arc_centre(Element elt,Real xc,Real yc,Real zc) 463
Set_arc_data(Element elt,Real xc,Real yc,Real zc, Real radius,Real xs,Real ys,Real zs,Real xe,Real ye,Real ze) 465
Set_arc_end(Element elt,Real xe,Real ye,Real ze) 465
Set_arc_radius(Element elt,Real radius) 464
Set_arc_start(Element elt,Real xs,Real ys,Real zs) 464
Set_arc(Segment &segment, Arc arc) 187
Set_attribute(Element elt,Integer att_no,Integer att) 270
Set_attribute(Element elt,Integer att_no,Real att) 271
Set_attribute(Element elt,Integer att_no,Text att) 270
Set_attribute(Element elt,Text att_name,Integer att) 270
Set_attribute(Element elt,Text att_name,Real att) 270
Set_attribute(Element elt,Text att_name,Text att) 269
Set_border(Horizontal_Group group,Integer bx,Integer by) 671
Set_border(Horizontal_Group group,Text text) 671
Set_border(Vertical_Group group,Integer bx,Integer by) 674
Set_border(Vertical_Group group,Text text) 673
Set_breakline(Element elt,Integer break_type) 256
Set_chainage(Element elt,Real start_chain) 258
Set_char(Text t,Integer pos, Integer c) 82
Set_circle_data(Element e,Real xc,Real yc,Real zc,Real radius) 467
Set_colour (Draw_Box box,Integer colour) 719
Set_colour (Draw_Box box,Integer r,Integer g,Integer b) 719
Set_colour(Element elt,Integer colour) 255
Set_colour(Tin tin,Integer colour) 286

Set_cursor_position(Integer x,Integer y) 666
Set_cursor_position(Widget widget) 682
Set_data (Colour_Box box,Text data) 709
Set_data (Screen_Text widget,Text data) 776
Set_data (Text_Edit_Box widget,Text data) 814
Set_data(Angle_Box box,Real data) 693
Set_data(Choice_Box box,Text data) 705
Set_data(Colour_Box box,Integer data) 709
Set_data(Directory_Box box,Text data) 715
Set_data(File_Box box,Text data) 725
Set_data(Input_Box box,Text data) 734
Set_data(Integer_Box box,Integer data) 736
Set_data(Justify_Box box,Integer data) 740
Set_data(Linestyle_Box box,Text data) 743
Set_data(Map_File_Box box,Text data) 747
Set_data(Message_Box box,Text data) 833
Set_data(Model_Box box,Text data) 751
Set_data(Name_Box box,Text data) 754
Set_data(Named_Tick_Box box,Text data) 756
Set_data(Plotter_Box box,Text data) 766
Set_data(Real_Box box,Real data) 771
Set_data(Report_Box box,Text data) 775
Set_data(Select_Box select,Text string) 779
Set_data(Select_Boxes select,Integer n,Text string) 784
Set_data(Select_Button select,Text string) 848
Set_data(Sheet_Size_Box box,Text data) 789
Set_data(Template_Box box,Text data) 805
Set_data(Text_Style_Box box,Text data) 808
Set_data(Text_Units_Box box,Integer data) 810
Set_data(Tick_Box box,Text data) 820
Set_data(Tin_Box box,Text data) 823
Set_data(View_Box box,Text data) 826
Set_data(XYZ_Box box,Real x,Real y,Real z) 828
Set_directory (File_Box box,Text data) 726
Set_drainage_data(Element elt,Integer i,Real x,Real y,Real z,Real r,Integer f) 490
Set_drainage_float (Element,Integer float) 493
Set_drainage_flow(Element elt,Integer dir) 493
Set_drainage_fs_tin (Element,Tin tin) 492
Set_drainage_hc_adopted_level(Element,Integer hc,Real level) 555
Set_drainage_hc_bush (Element,Integer hc,Text bush) 555
Set_drainage_hc_colour (Element,Integer hc,Integer colour) 556
Set_drainage_hc_depth (Element,Integer hc,Real depth) 556
Set_drainage_hc_diameter (Element,Integer hc,Real diameter) 557
Set_drainage_hc_grade (Element,Integer hc,Real grade) 557
Set_drainage_hc_hcb (Element,Integer hc,Integer hcb) 558
Set_drainage_hc_length (Element,Integer hc,Real length) 558
Set_drainage_hc_level (Element,Integer hc,Real level) 559
Set_drainage_hc_material (Element,Integer hc,Text material) 559
Set_drainage_hc_name (Element,Integer hc,Text name) 559
Set_drainage_hc_side (Element,Integer hc,Integer side) 560
Set_drainage_hc_type (Element,Integer hc,Text type) 560
Set_drainage_ns_tin (Element,Tin tin) 492
Set_drainage_outfall_height(Element elt,Real ht) 492
Set_drainage_pipe_attribute (Element elt,Integer pipe,Integer att_no,Integer att) 553
Set_drainage_pipe_attribute (Element elt,Integer pipe,Integer att_no,Real att) 553
Set_drainage_pipe_attribute (Element elt,Integer pipe,Integer att_no,Text att) 553
Set_drainage_pipe_attribute (Element elt,Integer pipe,Text att_name,Integer att) 552
Set_drainage_pipe_attribute (Element elt,Integer pipe,Text att_name,Real att) 552

Set_drainage_pipe_attribute (Element elt,Integer pipe,Text att_name,Text att) 552
Set_drainage_pipe_cover (Element,Integer pipe,Real cover) 534
Set_drainage_pipe_diameter(Element elt,Integer p,Real diameter) 535
Set_drainage_pipe_flow(Element elt,Integer p,Real flow) 540
Set_drainage_pipe_hgls(Element elt,Integer p,Real lhs,Real rhs) 539
Set_drainage_pipe_inverts(Element elt,Integer p,Real lhs,Real rhs) 532
Set_drainage_pipe_name(Element elt,Integer p,Text name) 533
Set_drainage_pipe_type(Element elt,Integer p,Text type) 534
Set_drainage_pipe_velocity(Element elt,Integer p,Real velocity) 539
Set_drainage_pit_attribute (Element elt,Integer pit,Integer att_no,Integer att) 525
Set_drainage_pit_attribute (Element elt,Integer pit,Integer att_no,Real att) 525
Set_drainage_pit_attribute (Element elt,Integer pit,Integer att_no,Text att) 526
Set_drainage_pit_attribute (Element elt,Integer pit,Text att_name,Integer att) 526
Set_drainage_pit_attribute (Element elt,Integer pit,Text att_name,Real att) 526
Set_drainage_pit_attribute (Element elt,Integer pit,Text att_name,Text att) 526
Set_drainage_pit_diameter(Element elt,Integer p,Real diameter) 498
Set_drainage_pit_float (Element,Integer pit,Integer float) 507
Set_drainage_pit_hgls(Element elt,Integer p,Real lhs,Real rhs) 509
Set_drainage_pit_inverts(Element elt,Integer p,Real lhs,Real rhs) 504
Set_drainage_pit_name(Element elt,Integer p,Text name) 497
Set_drainage_pit_road_chainage(Element elt,Integer p,Real chainage) 510
Set_drainage_pit_road_name(Element elt,Integer p,Text name) 510
Set_drainage_pit_type(Element elt,Integer p,Text type) 511
Set_drainage_pit(Element elt,Integer p,Real x,Real y,Real z) 497
Set_enable(Widget widget,Integer mode) 677
Set_end(Arc &arc,Point end) 171
Set_end(Line &line, Point pt) 167
Set_end(Segment &segment,Point point) 187
Set_error_message(Widget widget,Text text) 680
Set_feature_centre(Element elt,Real xc,Real yc,Real zc) 563
Set_feature_radius(Element elt,Real radius) 563
Set_finish_button (Widget panel,Integer move_cursor) 845
Set_focus(Widget widget) 683
Set_height (Tin tin,Integer pt,Real ht) 283
Set_help (Widget widget,Integer help) 686
Set_help (Widget widget,Text help) 687
Set_hip_data(Element elt,Integer i, Real x,Real y,Real radius) 621
Set_hip_data(Element elt,Integer i,Real x,Real y,Real radius,Real left_spiral,Real right_spiral) 621
Set_hip_data(Element elt,Integer i,Real x,Real y) 620
Set_interface_data(Element elt, Integer i,Real x, Real y,Real z,Integer flag) 567
Set_item(Dynamic_Element &de,Integer i,Element elt) 157
Set_item(Dynamic_Text &dt,Integer i,Text text) 159
Set_line(Segment &segment, Line line) 187
Set_message_mode(Integer mode) 650
Set_message_text(Text msg) 650
Set_model_attribute (Model model,Integer att_no,Integer att) 242
Set_model_attribute (Model model,Integer att_no,Real att) 242
Set_model_attribute (Model model,Integer att_no,Text att) 242
Set_model_attribute (Model model,Text att_name,Integer att) 242
Set_model_attribute (Model model,Text att_name,Real att) 242
Set_model_attribute (Model model,Text att_name,Text att) 243
Set_model(Dynamic_Element de,Model model) 256
Set_model(Element elt,Model model) 256
Set_name(Element elt,Text elt_name) 255
Set_name(Widget widget,Text text) 679
Set_optional(Widget widget,Integer mode) 678
Set_origin (Draw_Box box,Real x,Real y) 718
Set_page(Widget_Pages pages,Integer page_no) 689

Set_pipe_data(Element elt,Integer i,Real x, Real y,Real z) 610
Set_pipe_diameter(Element elt, Real diameter) 611
Set_pipe_justify(Element elt,Integer justify) 611
Set_pipeline_diameter(Element pipeline,Real diameter) 483
Set_pipeline_length (Element pipeline,Real length) 483
Set_plot_frame_colour(Element elt,Integer colour) 582
Set_plot_frame_draw_border(Element elt,Integer draw_border) 582
Set_plot_frame_draw_title_file(Element elt,Integer draw_title) 582
Set_plot_frame_draw_viewport(Element elt,Integer draw_viewport) 582
Set_plot_frame_margins(Element elt,Real l,Real b,Real r,Real t) 581
Set_plot_frame_name(Element elt,Text name) 579
Set_plot_frame_origin(Element elt,Real x,Real y) 580
Set_plot_frame_plot_file(Element elt,Text plot_file) 583
Set_plot_frame_plotter_name(Element elt,Text plotter_name) 583
Set_plot_frame_plotter(Element elt,Integer plotter) 583
Set_plot_frame_rotation(Element elt,Real rotation) 580
Set_plot_frame_scale(Element elt,Real scale) 580
Set_plot_frame_sheet_size(Element elt,Real w,Real h) 581
Set_plot_frame_sheet_size(Element elt,Text size) 581
Set_plot_frame_text_size(Element elt,Real text_size) 581
Set_plot_frame_textstyle(Element elt,Text textstyle) 582
Set_plot_frame_title_1(Element elt,Text title_1) 583
Set_plot_frame_title_2(Element elt,Text title_2) 584
Set_plot_frame_title_file(Element elt,Text title_file) 584
Set_point(Segment &segment, Point point) 186
Set_polyline_data(Element elt,Integer i,Real x,Real y,Real z,Real r,Integer f) 616
Set_project_attribute (Integer att_no,Integer att) 227
Set_project_attribute (Integer att_no,Real att) 227
Set_project_attribute (Integer att_no,Text att) 226
Set_project_attribute (Text att_name,Integer att) 226
Set_project_attribute (Text att_name,Real att) 226
Set_project_attribute (Text att_name,Text att) 228
Set_radius(Arc &arc, Real radius) 170
Set_raised_button(Button button,Integer mode) 844
Set_scale (Draw_Box box,Real xs,Real ys) 718
Set_select_snap_mode(Select_Box select,Integer mode,Integer control,Text snap_text) 780
Set_select_snap_mode(Select_Box select,Integer snap_control) 780
Set_select_snap_mode(Select_Boxes select,Integer n,Integer control) 785
Set_select_snap_mode(Select_Boxes select,Integer n,Integer snap_mode,Integer snap_control,Text snap_text) 785
Set_select_snap_mode(Select_Button select,Integer mode,Integer control,Text text) 849
Set_select_snap_mode(Select_Button select,Integer snap_control) 849
Set_select_type(Select_Box select,Text type) 780
Set_select_type(Select_Boxes select,Integer n,Text type) 785
Set_select_type(Select_Button select,Text type) 849
Set_sort (List_Box box,Integer mode) 745
Set_start(Arc &arc, Point start) 170
Set_start(Line &line, Point pt) 167
Set_start(Segment &segment,Point point) 187
Set_style(Element elt,Text elt_style) 257
Set_subtext(Text &text,Integer start,Text sub) 78
Set_super_2d_level (Element,Real level) 310
Set_super_culvert (Element,Real w,Real h) 342
Set_super_data (Element,Integer i,Real x,Real y,Real z,Real r,Integer f) 301
Set_super_diameter (Element,Real diameter) 340
Set_super_segment_attribute (Element elt,Integer seg,Integer att_no,Integer att) 430
Set_super_segment_attribute (Element elt,Integer seg,Integer att_no,Real att) 430
Set_super_segment_attribute (Element elt,Integer seg,Integer att_no,Text att) 429
Set_super_segment_attribute (Element elt,Integer seg,Text att_name,Integer att) 429

Set_super_segment_attribute (Element elt,Integer seg,Text att_name,Real att) 429
Set_super_segment_attribute (Element elt,Integer seg,Text att_name,Text att) 428
Set_super_segment_colour (Element,Integer seg,Integer colour) 397
Set_super_segment_culvert (Element,Integer seg,Real w,Real h) 342
Set_super_segment_device_text (Element) 368
Set_super_segment_diameter (Element,Integer seg,Real diameter) 341
Set_super_segment_major (Element,Integer seg,Integer major) 317
Set_super_segment_radius (Element,Integer seg,Real radius) 316
Set_super_segment_text (Element,Integer seg,Text text) 367
Set_super_segment_text_angle (Element,Integer vert,Real a) 371
Set_super_segment_text_colour (Element,Integer vert,Integer c) 371
Set_super_segment_text_justify (Element,Integer vert,Integer j) 369
Set_super_segment_text_offset_height (Element,Integer vert,Real o) 370
Set_super_segment_text_offset_width (Element,Integer vert,Real o) 370
Set_super_segment_text_size (Element,Integer vert,Real s) 372
Set_super_segment_text_slant (Element,Integer vert,Real s) 373
Set_super_segment_text_style (Element,Integer vert,Text s) 374
Set_super_segment_text_type (Element,Integer type) 369
Set_super_segment_text_x_factor (Element,Integer vert,Real x) 373
Set_super_segment_tinability (Element,Integer seg,Integer tinability) 315
Set_super_segment_visibility (Element,Integer seg,Integer visibility) 442
Set_super_segment_world_text (Element) 368
Set_super_use_2d_level (Element,Integer use) 308
Set_super_use_3d_level (Element,Integer use) 308
Set_super_use_culvert (Element,Integer use) 331
Set_super_use_diameter (Element,Integer use) 329
Set_super_use_pipe_justify (Element,Integer use) 332
Set_super_use_segment_annotation_array(Element,Integer use) 366
Set_super_use_segment_annotation_value(Element,Integer use) 365
Set_super_use_segment_attribute (Element,Integer use) 420
Set_super_use_segment_colour (Element,Integer use) 397
Set_super_use_segment_culvert (Element,Integer use) 331
Set_super_use_segment_diameter (Element,Integer use) 330
Set_super_use_segment_radius (Element,Integer use) 316
Set_super_use_segment_text_array (Element,Integer use) 364
Set_super_use_segment_text_value (Element,Integer use) 364
Set_super_use_symbol (Element,Integer use) 321
Set_super_use_tinability (Element,Integer use) 311
Set_super_use_vertex_annotation_array(Element,Integer use) 348
Set_super_use_vertex_annotation_value(Element,Integer use) 347
Set_super_use_vertex_attribute (Element,Integer use) 409
Set_super_use_vertex_point_number (Element,Integer use) 318
Set_super_use_vertex_symbol (Element,Integer use) 322
Set_super_use_vertex_text_value (Element,Integer use) 345
Set_super_use_visibility (Element,Integer use) 438
Set_super_vertex_attribute (Element elt,Integer vert,Integer att_no,Integer att) 419
Set_super_vertex_attribute (Element elt,Integer vert,Integer att_no,Real att) 419
Set_super_vertex_attribute (Element elt,Integer vert,Integer att_no,Text att) 418
Set_super_vertex_attribute (Element elt,Integer vert,Text att_name,Integer att) 418
Set_super_vertex_attribute (Element elt,Integer vert,Text att_name,Real att) 418
Set_super_vertex_attribute (Element elt,Integer vert,Text att_name,Text att) 417
Set_super_vertex_coord (Element,Integer vert,Real x,Real y,Real z) 301
Set_super_vertex_device_text (Element) 349
Set_super_vertex_point_number (Element,Integer vert,Integer point_number) 318
Set_super_vertex_symbol_colour (Element,Integer vert,Integer c) 324
Set_super_vertex_symbol_offset_height(Element,Integer vert,Real r) 325
Set_super_vertex_symbol_offset_width (Element,Integer vert,Real o) 325
Set_super_vertex_symbol_rotation (Element,Integer vert,Real a) 326

Set_super_vertex_symbol_size (Element,Integer vert,Real s) 326
Set_super_vertex_symbol_style (Element,Integer vert,Text s) 324
Set_super_vertex_text (Element,Integer vert,Text text) 349
Set_super_vertex_text_angle (Element,Integer vert,Real a) 353
Set_super_vertex_text_colour (Element,Integer vert,Integer c) 352
Set_super_vertex_text_justify (Element,Integer vert,Integer j) 350
Set_super_vertex_text_offset_height (Element,Integer vert,Real o) 351
Set_super_vertex_text_offset_width (Element,Integer vert,Real o) 351
Set_super_vertex_text_size (Element,Integer vert,Real s) 353
Set_super_vertex_text_slant (Element,Integer vert,Real s) 354
Set_super_vertex_text_style (Element,Integer vert,Text s) 355
Set_super_vertex_text_type (Element,Integer type) 350
Set_super_vertex_text_x_factor (Element,Integer vert,Real x) 354
Set_super_vertex_tinability (Element,Integer vert,Integer tinability) 313
Set_super_vertex_visibility (Element,Integer vert,Integer visibility) 440
Set_super_vertex_world_text (Element) 349
Set_supertin (Tin_Box box,Integer mode) 283
Set_text_align (Draw_Box box,Integer mode) 721
Set_text_angle(Element elt,Real angle) 474
Set_text_colour (Draw_Box box,Integer r,Integer g,Integer b) 720
Set_text_data(Element elt,Text text,Real x,Real y,Real size,Integer colour,Real angle,Integer justif,Integer size_mode,Real offset_distance,Real rise_distance) 470
Set_text_font (Draw_Box box,Text font) 721
Set_text_height(Element elt,Real height) 476
Set_text_justify(Element elt,Integer justify) 474
Set_text_offset(Element elt,Real offset) 475
Set_text_rise(Element elt,Real rise) 475
Set_text_size(Element elt,Real size) 473
Set_text_slant(Element elt,Real slant) 476
Set_text_style(Element elt,Text style) 477
Set_text_units(Element elt,Integer units_mode) 473
Set_text_value(Element elt,Text text) 471
Set_text_weight (Draw_Box box,Integer weight) 721
Set_text_x_factor(Element elt,Real xfact) 477
Set_text_xy(Element elt,Real x, Real y) 472
Set_time_updated(Element elt,Integer time) 259
Set_tooltip (Widget widget,Text help) 686
Set_type (Function_Box box,Integer type) 729
Set_type (Function_Box box,Text type) 729
Set_vip_data(Element elt,Integer i, Real ch,Real ht,Real parabolic) 625
Set_vip_data(Element elt,Integer i,Real ch,Real ht,Real value,Integer mode) 625
Set_vip_data(Element elt,Integer i,Real ch,Real ht) 625
Set_width_in_chars(Widget widget,Integer chars) 680
Set_wildcard (File_Box box,Text data) 725
Set_x(Point &pt, Real x) 165
Set_y(Point &pt, Real y) 166
Set_z(Point &pt, Real z) 166
sewer junction 487
Sheet_size_prompt(Text msg,Text &ret) 657
Show_browse_button(Widget widget,Integer mode) 676
Show_widget(Widget widget,Integer x,Integer y) 681
Show_widget(Widget widget) 680
Split_string(Element string,Real chainage,Element &string1,Element &string2) 890
Start_batch_draw (Draw_Box box) 718
String_close(Element elt) 631
String_closed(Element elt, Integer &closed) 631
String_open(Element elt) 631
String_self_intersects(Element elt,Integer &intersects) 634

Super_segment_attribute_debug (Element elt,Integer seg) 425
Super_segment_attribute_delete (Element elt,Integer seg,Integer att_no) 424
Super_segment_attribute_delete (Element elt,Integer seg,Text att_name) 424
Super_segment_attribute_delete_all (Element elt,Integer seg) 424
Super_segment_attribute_dump (Element elt,Integer seg) 425
Super_segment_attribute_exists (Element elt,Integer seg,Text att_name) 423
Super_segment_attribute_exists (Element elt,Integer seg,Text name,Integer &no) 424
Super_vertex_attribute_debug (Element elt,Integer vert) 414
Super_vertex_attribute_delete (Element elt,Integer vert,Integer att_no) 413
Super_vertex_attribute_delete (Element elt,Integer vert,Text att_name) 413
Super_vertex_attribute_delete_all (Element elt,Integer vert) 413
Super_vertex_attribute_dump (Element elt,Integer vert) 414
Super_vertex_attribute_exists (Element elt,Integer vert,Text att_name) 413
Super_vertex_attribute_exists (Element elt,Integer vert,Text name,Integer &no) 412
Swap_xy(Dynamic_Element elements) 904
symbol
    justification point 321
symbol justification point 321
System(Text msg) 123

T

Tangent(Segment seg_1,Segment seg_2,Line &line) 192
Template_exists(Text template_name) 885
Template_prompt(Text msg,Text &ret) 654
Template_rename(Text original_name,Text new_name) 885
text
    direction 83, 344, 362
    justification point 83, 344, 362
Text_justify(Text text) 77
Text_length(Text text) 76
Text_lower(Text text) 77
Text_units_prompt(Text msg,Text &ret) 659
Text_upper(Text text) 76
Textstyle_prompt(Text msg,Text &ret) 657
Time(Integer &h,Integer &m,Real &sec) 124
Time(Integer &time) 124
Time(Text &time) 124
Tin_aspect(Tin tin,Real x, Real y, Real &aspect) 278
Tin_boundary(Tin tin,Integer colour_for_strings,Dynamic_Element &de) 278
Tin_colour(Tin tin,Real x, Real y,Integer &colour) 277
Tin_delete(Tin tin) 279
Tin_duplicate(Tin tin,Text dup_name) 278
Tin_exists(Text tin_name) 274
Tin_exists(Tin tin) 274
Tin_get_point(Tin tin, Integer point, Real &x, Real &y, Real &z) 279
Tin_get_triangle_colour(Tin tin, Integer triangle, Integer &colour) 286
Tin_get_triangle_from_point(Tin tin, Integer &triangle, Real x,Integer y, Integer z) 281
Tin_get_triangle_inside(Tin tin, Integer triangle, Integer &Inside) 280
Tin_get_triangle_neighbours(Tin tin, Integer triangle, Integer &n1, Integer &n2, Integer &n3) 279
Tin_get_triangle_points(Tin tin, Integer triangle, Integer &p1, Integer &p2,Integer &p3) 279
Tin_get_triangle(Tin tin, Integer triangle, Integer &p1, Integer &p2, Integer &p3, Integer &n1, Integer &n2, Integer &n3, Real &x1, Real &y1, Real &z1, Real &x2, Real &y2, Real &z2,Real &x3, Real &y3, Real &z3) 281
Tin_height(Tin tin,Real x, Real y, Real &height) 277
Tin_models (Tin tin,Dynamic_Text &models) 282
Tin_models(Tin tin, Dynamic_Text &models_used) 275
Tin_number_of_duplicate_points(Tin tin, Integer &notri) 276
Tin_number_of_points(Tin tin, Integer &notri) 276

Tin_number_of_triangles(Tin tin, Integer &notri) 276
Tin_prompt(Text msg,Integer mode,Text &ret) 655
Tin_prompt(Text msg,Text &ret) 655
Tin_rename(Text original_name,Text new_name) 278
Tin_slope (Tin tin,Real x, Real y, Real &slope) 277
Tin_tin_depth_contour(Tin original,Tin new,Integer cut_colour,Integer zero_colour,Integer fill_colour,Real interval,Real start_level,Real end_level,Integer mode,Dynamic_Element &de) 873
Tin_tin_intersect(Tin original,Tin new, Integer colour,Dynamic_Element &de) 873
Tin_tin_intersect(Tin original,Tin new,Integer colour,Dynamic_Element &de,Integer mode) 874
To_text(Integer value,Text format) 81
To_text(Integer value) 81
To_text(Real value, Integer no_dec) 81
To_text(Real value,Text format) 81
To_text(Text text,Text format) 82
Translate(Dynamic_Element elements, Real dx,Real dy,Real dz) 905
Triangulate (Dynamic_Text list,Text tin_name,Integer colour, Integer preserve,Integer bubbles,Tin &tin) 273
Triangulate(Dynamic_Element de,Text tin_name, Integer tin_colour,Integer preserve,Integer bubbles,Tin &tin) 273

U

UCS 1097
Unicode Transformation Format 1097
Universal Characters Set 1097
Use_browse_button(Widget widget,Integer mode) 676
UTF 1097

V

Validate (Select_Box select,Element &string,Integer silent) 778
Validate (Select_Boxes select,Integer n,Element &string,Integer silent) 783
Validate (Select_Button select,Element &string,Integer silent) 847
Validate(Angle_Box box,Real &result) 694
Validate(Choice_Box box,Text &result) 705
Validate(Colour_Box box,Integer &result) 709
Validate(Directory_Box box,Integer mode,Text &result) 714
Validate(File_Box box,Integer mode,Text &result) 724
Validate(Input_Box box,Text &result) 734
Validate(Integer_Box box,Integer &result) 736
Validate(Justify_Box box,Integer &result) 739
Validate(Linestyle_Box box,Integer mode,Text &result) 743
Validate(Map_File_Box box,Integer mode,Text &result) 747
Validate(Model_Box box,Integer mode,Model &result) 750
Validate(Name_Box box,Text &result) 754
Validate(Named_Tick_Box box,Integer &result) 756
Validate(Plotter_Box box,Text &result) 766
Validate(Real_Box box,Real &result) 771
Validate(Report_Box box,Integer mode,Text &result) 774
Validate(Select_Box select,Element &string) 778
Validate(Select_Boxes select,Integer n,Element &string) 783
Validate(Select_Button select,Element &string) 847
Validate(Sheet_Size_Box box,Real &w,Real &h,Text &code) 789
Validate(Template_Box box,Integer mode,Text &result) 804
Validate(Text_Style_Box box,Text &result) 808
Validate(Text_Units_Box box,Integer &result) 810
Validate(Tick_Box box,Integer &result) 820
Validate(Tin_Box box,Integer mode,Tin &result) 822
Validate(View_Box box,Integer mode,View &result) 825
Validate(XYZ_Box box,Real &x,Real &y,Real &z) 828

Vertical_Group Create_vertical_group(Integer mode) 673
View_add_model(View view, Model model) 247
View_exists(Text view_name) 245
View_exists(View view) 245
View_fit(View view) 248
View_get_models(View view, Dynamic_Text &model_names) 247
View_get_size(View view,Integer &width,Integer &height) 248
View_prompt(Text msg,Text &ret) 656
View_redraw(View view) 247
View_remove_model(View view, Model model) 247
Volume_exact(Tin tin_1,Element tin_2,Element poly,Real &cut,Real &fill,Real &balance) 883
Volume_exact(Tin tin_1,Real ht,Element poly,Real &cut,Real &fill, Real &balance) 883
Volume(Tin tin_1,Real ht,Element poly,Real ang,Real sep,Text report_name,Integer report_mode,Real &cut,Real &fill,Real &balance) 882
Volume(Tin tin_1,Tin tin_2,Element poly,Real ang,Real sep,Text report_name,Integer report_mode,Real &cut,Real &fill,Real &balance) 882

W

Wait_on_widgets(Integer &id,Text &cmd,Text &msg) 676
Widget_Pages Create_widget_pages() 689
Winhelp (Widget widget,Text helpfile,Integer helpid,Integer popup) 688
Winhelp (Widget widget,Text helpfile,Integer helpid) 688
Winhelp (Widget widget,Text helpfile,Integer table,Text key) 687
Winhelp (Widget widget,Text helpfile,Text key) 687

Y

Yes_no_prompt(Text msg,Text &ret) 656

12d Solutions Pty Ltd

Civil and Surveying Software

Course Notes

CIVIL AND

SURVEYING SOFTWARE

THE 12D PERSPECTIVE

12dModel

Programming Language

12D Solutions Pty Ltd

ACN 101 351 991

Phone: +61 (2) 9970 7117Fax: +61 (2) 9970 7118 Email training@12d.com Web www.12d.com

12d Model Programming Language Course Notes

These course notes assume that the trainee has the basic 12d Model skills usually obtained from the "**12d Model Training Manual**"

These notes are intended to cover basic 12d model programming language examples. For more information regarding training courses contact 12d Solutions training Manager.

These notes were prepared by
Robert Graham and Lee Gregory

Disclaimer

12d Model is supplied without any express or implied warranties whatsoever.
No warranty of fitness for a particular purpose is offered.
No liabilities in respect of engineering details and quantities produced by 12d Model are accepted.

Every effort has been taken to ensure that the advice given in these notes and the program 12d Model is correct, however, no warranty is expressed or implied by 12d Solutions Pty Ltd.

Course Introduction **5**

Getting Started **6**
- Names and Reserved Names 6
- White Space and Comments 6
- Variables, Assignments and Operators 6
  - Variables 7
  - Assignment Operator 8
  - Operators 8
- Statements and Blocks 9

Functions **11**
- General Information About Functions 11

Your First Program **13**
- Print(Text msg).... your first 12dPL function 13
- Creating Your First Program 14
- Compiling and Running the Program 15

Common Compile Error Messages **17**

Overloaded Functions **18**

Using Input and Output Functions **19**
- Output to the Macro Console 19
- Input via the Macro Console (quick and easy) 24

Using Flow Control **28**
- Logical Expressions 28
- 12dPL Flow Controls 28
- .“goto” and “label” Statements 29
- .“if” and “else” Statements 29
- .Error Checking Using “goto”, “label”, “if” and “else” Statements 30
- “for” loops 32
- “while” loops 33
- “switch” Statement 33
- “continue” Statement 35
- “break” Statement 35

Running Existing 12dPL Programs **36**

Unleashing the Power - 12d Database Handles **37**
- Locks 37
- Read In Some Data to use 12dPL Programs On 37
- Elements, Models and Uids 38
- Accessing Elements 39
- Exercises 1 and 2 41
  - Exercise 1 41
  - Exercise 2 41
- Accessing Models 42
- Dynamic_Elements 43
- Accessing Element in Models 44
- Getting Information about an Element 45
- Putting it All Together 45
- Exercises 3and 4 48
  - Exercise 3 48
  - Exercise 4 48

Infinite Loops **49**
- Killing a 12dPL Program 49
- Ending the Process 12d.exe 50

Writing to a Text File (Reports) **51**
- Writing a Simple Unicode and ANSI (Ascii) Files 52
- Writing 12d Model Data to a Text File 52

Checking if a File Exists 54
Exercise 5 54

Reading a Text File 55
What to Do with the Line Read from a File 55
Reading a Text File 56
Exercise 6 56
Using a Clipboard 57
Binary Files 57

Creating User Defined Functions 58
A Simple User Defined Function Example 58
Exercise 7 59
Exercise 8 59

User Menus, User Defined Function Keys and Toolbars 61

Panel Basics 63
Creating and Displaying a Panel 64
Adding Widgets to the Panel 65
Monitoring Events in the Panel 66
Events Produced by a Panel 67
Processing Events from a Panel 68
Set_Ups.h and #include 70
Creating a Model_Box 70
Creating a File_Box 72
More Events from Wait_on_widgets 72
Exercise 9 72
Horizontal and Vertical Groups 74
Exercise 10 75
Validating Boxes and Buttons 76
Model_Box Events 76
File_Box Events 76
Write Button 76
Exercise 11 79
CHECK and GET Modes 80
Ignored Events 80

Working with 12d Model Strings 81
Exercise 12 82
Types of Elements 83
Dimensions of a Super String 84
Exercise 13 85
Accessing (x,y,z) Data for a Super String 86
Exercise 14 86
Changing Element Header Properties 87
Exercise 15 88

Some Examples 90
Exercise_8.4dm 90
Eleven_1.4dm 92
Eleven_2.4dm 93
Eleven_3.4dm 94
Twelve_1.4dm 96
Thirteen.4dm 97
Fourteen.4dm 99
Fifteen.4dm 101

Not Used 103

12d Model Programming Language Course

1.0 Course Introduction

The ***12d Model*** programming Language (12dPL) is a powerful programming language designed to run from within 12d Solutions software ***12d Model***.

Its main purpose is to allow users to enhance the existing 12d Solutions package by writing their own programs (also known as ***12d Model*** macros).

12dPL is based on a subset of the C++ language with special extensions to allow easy manipulation of 12d Model data. A large number of intrinsic functions are supplied which cover most aspects of civil modelling.

12dPL has been designed to fit in with the ability of ***12d Model*** to "stack" an incomplete operation.

This training manual does not try to teach programming techniques. Instead this manual takes the user through the basics steps to get started with 12dPL.

This course intends to teach you:

1. How to use the 12dPL manual
2. The syntax for 12dPL programs
3. How to create/compile and run 12dPL code.
4. The basic 12dPL variable types and "handles" to 12d Elements (strings etc.).
5. How to retrieve and change basic Element properties.
6. File input/output (creating reports).
7. How to build 12d panels.
8. How to include your 12dPL programs in the 12d menu system, function keys and toolbars.

The course does not try and each you everything about 12dPL but builds up your knowledge in a structured, step by step approach, with many programming examples.

At first the going may appear slow but the pace accelerates once you have a good understanding of the basics, and how to effectively use the 12dPL manual.

COURSE NOTES

12d Model Programming Language

2.0 Getting Started

2.1 Names and Reserved Names

12dPL programs consists of names (also known as words) and names are broken in to ***reserved*** names and ***user defined*** names.

The ***reserved*** names (or reserved words or Key words) that have special purposes. For example goto, if, else, while, switch, Real, Text (For a more complete list, see Reserved Names).

Some of these reserved words are part of the language structure (for example goto, if, else, while, switch), others are 12dPL variable types (for example Real, Integer, Model, Element) and 12dPL supplied function names.

In many places a user defines their own names (user defined names) but a user defined name can not be the same as any ***reserved name***.

Example of ***user defined names*** are for variable names (see Variables, Assignments and Operators) and user defined function names.

2.2 White Space and Comments

Spaces, tabs, new lines (<Enter>), form feeds and comments are collectively known as **white space**.

White space is ignored except for the purpose of separating names, or in text between double quotes. Hence blank lines are ignored in the program code.

For example

```
goto      fred    ;
```

is the same as

goto fred

and “many spaces” remains as it is.

Comments are extremely important for writing any program.

12dPL supports two styles of comments:

(a) a line oriented comment

where all the characters after a double forward slash (//) and up to the end of the line are ignored.

(b) a block comment

where all characters between a starting /* and a terminating */ are ignored.

The following is an example of 12dPL code with single and multiple line comments.

```
void main()
{
Real   y = 1;   // the rest of this line is comment
/* this comment can carry
over many lines until
we get to the termination characters */
}
```

2.3 Variables, Assignments and Operators

Variables and constants are the basic data objects manipulated in a 12dPL program.

Variables have unique *user defined names* and a unique *type* which is specified in a *Variable Declaration*. All variables must be declared prior to use.

Operators specify what is to be done to variables.

Expressions combine variables and operators to produce new values.

The *type* of the variable determines the set of values it can have and what operations can be performed on it.

2.3.1 Variables

2.3.1.1 Variable Names

In 12dPL, variable names must start with an alphabetic character and can consist of upper and/or lower case alphabetic characters, numbers and underscores(_) and there is no restriction on the length of variable names.

12dPL variable names are case sensitive.

2.3.1.2 Variable Declarations

All variables must be declared before they are used.

A declaration consists of a ***variable type*** (which is a reserved name) and a list of variable names separated by commas and **ending the line with a semi-colon ";"**.

For example

```
Integer   fred, joe, tom;
```

where Integer is the variable type and fred, joe and tom are the names of variables of type Integer.

2.3.1.3 Variable Types

There are a wide variety of *12d Model* variable types supported in the 12dPL language. For example void, Integer, Real, Text, Arrays.

Important Note: unlike C and C++, array in 12dML start at position 1.

See Variables **.**

2.3.1.3.1 Void

This is a special type which is only used for functions which have no return value.

2.3.1.3.2 Integers, Real and Text

Integer - a 32-bit whole number. It can be positive or negative.

Real - a 64-bit decimal number. It can be positive or negative.

Text - a sequence of characters.

Examples of declarations:

```
Integer i;
Real x,y,z;
Text    ans, rep;
```

2.3.1.3.3 Arrays

Arrays may be allocated statically or dynamically.*See* Array Types **.**

BIG WARNING: array subscripts start at 1 and not 0 like in C and C++

Static Array

```
Real x[10];          // great for small arrays (created on the stack)
```

Dynamic Allocated Array

```
Integer n = 100;     // a must for large arrays (say greater than 10)
Real x[n];
```

2.3.2 Assignment Operator

An ***assignment*** gives a value to a variable.

In 12dPL, the assignment operator is a single equal sign (=).

An assignment consists of a

variable_name = expression

For example

```
x = y + 3
```

The Assignment is **NOT** a mathematical equality and in interpreted as:

the expression on the right hand side is evaluated and then the variable on the left is given that value.

For example

```
x = y + 3
```

means that **x** is given the value that is equal to the current value of **y** plus 3. The value of **y** does not change.

If the same variable occurs on both sides of the assignment operator, the current value is used in evaluating the expression on the right hand side of the "=" and then the variable on the left is given the value of the expression on the right.

For example,

```
x = x + 1;
```

means that x is given the new value that is equal to the original value of x, plus 1.

It is also allowable to use assignments to give constant values to a variable in the variable declaration.

```
Integer i=2;     // this is declaring the type and also assigning it the value 2.
```

2.3.3 Operators

Operators specify operations that are done to variables.

The other most common operators are

Binary Arithmetic Operators

+	addition
-	subtraction
*	multiplication
/	division - note that integer division truncates any fractional part

Increment and decrement operators

++	post and pre-increment	e.g. i++ which is shorthand for i = i + 1
--	post and pre-decrement	e.g. i-- which is shorthand for i = i -1

Assignment operators

+=	x += y is shorthand for x = x + y
-=	x -= y is shorthand for x = x - y
*=	x *= y is shorthand for x = x *y
/=	x /= y is shorthand for x = x /y

Logical Operators

==	equal to
!=	not equal to
||	inclusive or
&&	and
!	not

Relational operators

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

For more information see Assignment and Operators

2.4 Statements and Blocks

An expression such as x = 0 or i++ becomes a **statement** when it is followed by a semi-colon.

Curly brackets **{** and **}** (braces) are used to group declarations and statements together into a **compound statement**, or **block**, so that they are syntactically equivalent to a **single statement**.

There is no semi-colon after the right brace that ends a block.

Blocks can be nested but cannot overlap.

Examples of statements are

```
x = 0;
i++;
fred = 2 * joe + 9.0;
```

An example of a compound statement or **block** is

```
{
  x = 0;
  i++;
```

```
  fred = 2 * joe + 9.0;
}
```

COURSE NOTES

12d Model Programming Language

3.0 Functions

Functions can be used to break large computing tasks into smaller ones and allow users to build on software that already exists.

Basically a program is just a set of definitions of variables and functions. Communication between the functions is by function arguments, by values returned by the functions, and through global variables.

The 12dPL program file must contain a starting function called **main**, calls to 12dPL supplied functions as well as zero or more ***user defined*** functions.

(a) **main** function

The special function called **main** is the designated start of the program.

The main function is simply a header **void main ()** followed by the actual program code enclosed between a start brace { and an end brace }.

Hence the function called **main** is a header followed by a block of code:

```
void  main ()
      {
         declarations and statements
         i.e. program code
      }
```

For more information, see Main Function .

(b) 12dPL Supplied Functions

A large number of functions are supplied with 12dPL to make tasks easier for the program writer. These 12dPL supplied functions are predefined and nothing special is needed to use them. The 12dPL supplied functions are all given in the 12d Model Programming Language manual.

Note - All 12dPL supplied functions begin with a capital letter to help avoid clashes with any user variable names or user defined function names.

(c) User Defined Functions

As well as the *main* function, and 12dPL supplied functions, a program file can also contain ***user defined*** functions.

We will examine user defined functions later in the course (see Creating User Defined Functions).

3.1 General Information About Functions

A function performs a specific task using the variables (arguments) that are passed to it in brackets. After it has completed these tasks it can return a value. The returning value is often a result or answer from the function or it is a code indicating the success of the function.

The definition of a function would look like the following

```
Real calc_distance(Real x1, Real y1, Real x2, Real y2)
```

This says that the function called ***calc_distance*** has the Real values of x1,y1,x2,y2 passes to it. The function body (not shown) might calculate the distance between the two points (x1,y1) and (x2,y2) and return the distance as a Real number as the function return value.

When *calc_distance* is called inside a12dPL program, the code would look like the following.

```
Real distance, x1,y1,x2,y2          // defining distance,x1,y1,x2,y2 as Real variables
...
```

distance = calc_distance(x1,y1,x2,y2); // *distance* is given the return value of calc_distance

Note that when used, the types of the variables (Real in this case) are **not** included. They are only used in the function definition to specify what types the arguments and return funkiest value must be.

The arguments (constants or variables) of the function can be Passed by Value (a one way transfer) as in the above example calc_distance, or a variable can be Passed by Reference (a two way transfer) by including an & before the variable name in the argument list. The arguments in the following function definition for calc_distance are passed by reference.

Real calc_distance(Real &x1, Real &y1, Real &x2, Real& y2);

With ***passed by reference***, the argument variable in the calling routine can be changed by the function.

The **return** statement in a function is the mechanism for returning a value from the called function to its caller using the *return-type* of the function.

The general definition of the return statement is:

return expression;

For a function with a *void* return-type (a void function), the expression must be empty. That is, for a void return-type you can only have return and no expression since no value can be returned.

Thus for a void function the return statement is

return;

Also for a void function, the function will implicitly return if it reaches the end of the function without executing a return statement. The function *main* is an example of a void function.

For a function with a non-void return-type (a non-void function), the expression after the return must be of the same type as the return type of the function. Hence any function with a non-void return-type must have a return statement with the correct expression type.

The calling function is free to ignore the returned value.

Restrictions

Unlike C++, in 12dPL the last statement for a function with a non-void return type must be a *return* statement.

WARNING! Function named are case sensitive!

4.0 Your First Program

4.1 Print(Text msg).... your first 12dPL function

This is the first function from 12dPL that we will examine. If we search for Print in the Help system, we will find the following function.

void Print(Text msg)

and its definition in the manual is:

Print(Text msg)

Name

void Print(Text msg)

Description

Print the Text **msg** to the Output Window.

ID = 24

This is read as:

The function **Print(Text msg)** has no return value (void) and has a Text argument, *msg* say.

The function prints the value of the Text variable *msg* to the Output Window.

The Text argument is passed by value (as there is no ampersand & after Text).

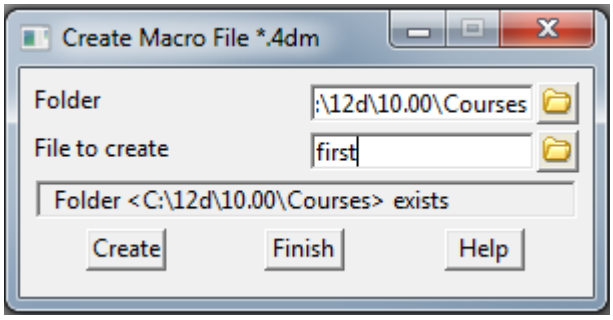
24 is the unique identification number given to this function. The identification number is the best way of identifying the function if there are a number of functions with the same, or similar, names.

4.2 Creating Your First Program

From the Main menu select

Utilities=>Macros=>Create

and the following panel will appear.

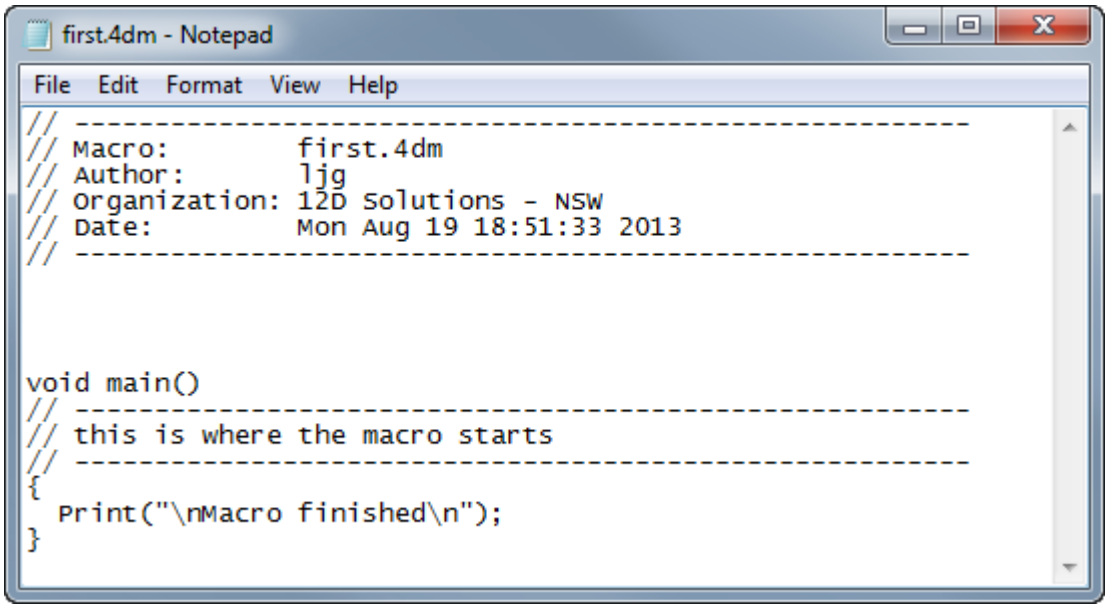


The directory is defaulted to your project directory.

Type **first** as the name of your first macro.

Select **Create** to create the macro and load it into your text editor.

You will now see the following



A file will be created with the name ***first.4dm***.

The first few lines are comments (beginning with the //). Following the comments and blank lines is the function *main*().

All programs must have the *main* function. It is always of type **void** and will have nothing in the parameter list (parameters for *main* are available but they will not be covered in this training manual). *See* Main Function .

You will note that the main function has one line of executable code and that includes the **Print(Text msg)** function. The **Print(Text msg)** function can have a text constant or text variable as its argument. In this case it is a text constant "\n Macro finished\n". Note the special line feed character "\n" that moves the printing to the next line.

When run, this program will write to the Output Window, a blank line, followed by the words **Macro finished** on the new lines, and then onto another new line, and then stop.

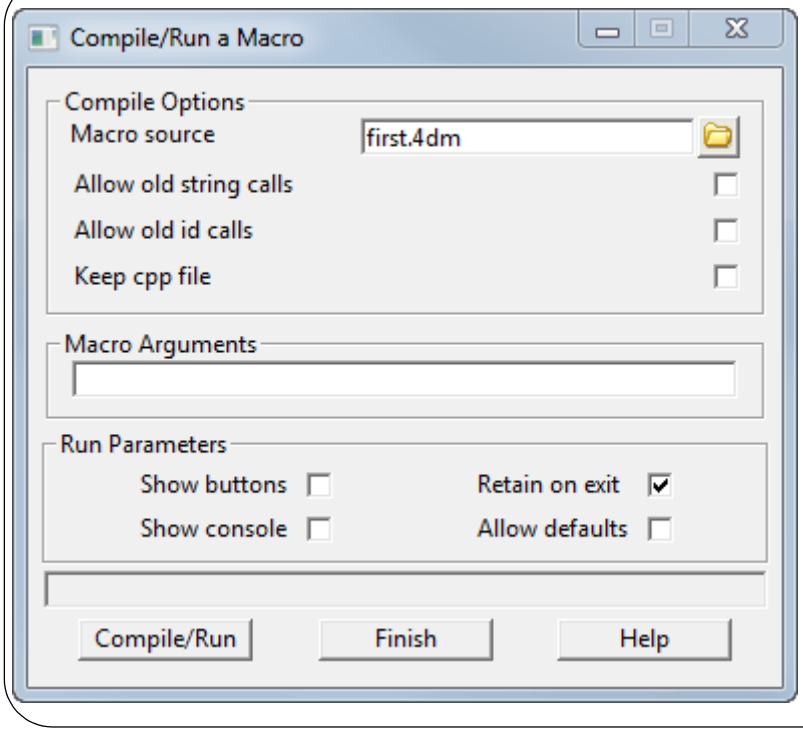
Save the program.

4.3 Compiling and Running the Program

From the Main menu select

Utilities=>Macros=>Compile/run

and the following panel will appear.



Select the **Browse** icon and then select the macro code text file *first.4dm* from the pop-up list.

Select **Retain on Exit** so that the prompt box will remain after the macro finishes.

Select **Compile/Run**.

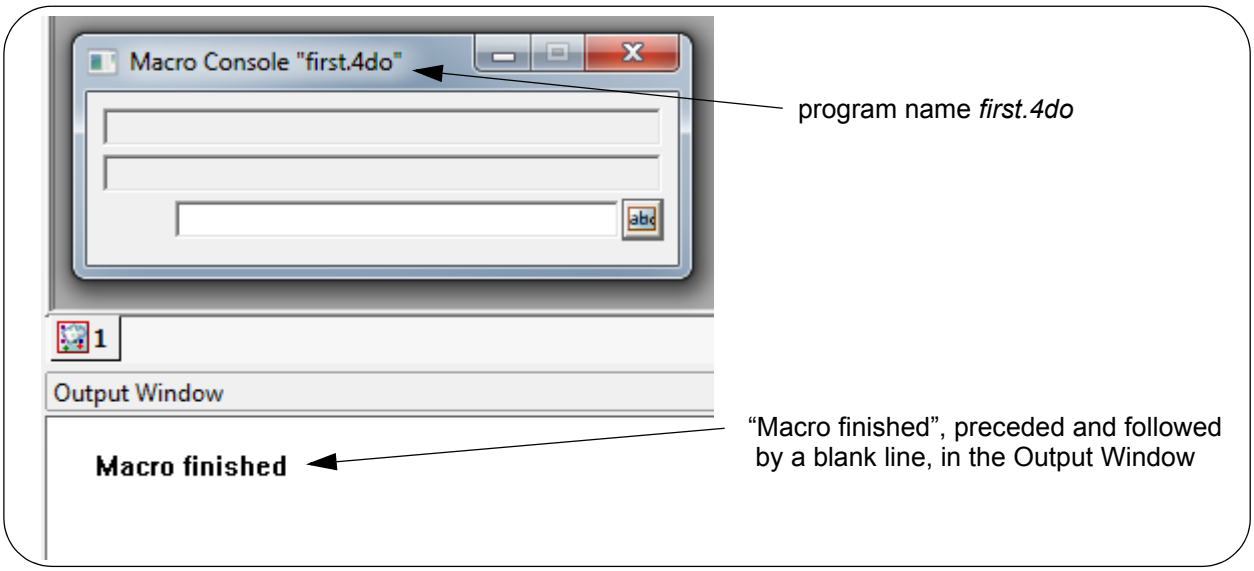
The file ***first.4dm*** will be compiled to create an object file called **first.4do.**

This compiled file that is then run by ***12d Model***.

The running program brings up the ***Macro Console*** and also writes *Macro finished* to the Output Window.

COURSE NOTES

12d Model Programming Language



Note that the *Macro Console* has the program name on the top and in the Output Window, the words **Macro finished** appear (preceded and followed by a blank line).

You have just created and run your first program!

5.0 Common Compile Error Messages

The most common typing error is to forget the semi colon at the end of a statement.

Try removing the semi colon at the end of the **Print** function and then *Compile/Run* the program. What do you notice about the line number that the compiler reports?

Because there was an error, an error log called first.4dl is produced (that is what is displayed in the editor) and no compiled object is produced (first.4do) and so isn't run.

Next put the semi colon back in and remove one of the quote marks **"** in the Print function.

Now *Compile/Run* this file and check the error messages.

6.0 Overloaded Functions

In our program, we used the function void Print(Text msg) but there are four functions with exactly the same name **Print.**

void Print(Text msg)

void Print(Integer value)

void Print(Real value)

void Print()

In 12dPL you can have functions with the same name as long as each one has a different number of argument and/or different argument types. This is called Overloading of Function Names .

In the above examples, each **Print** function has different argument types and there is a Print function for any of the argument types Integer, Real and Text, or with no argument at all.

We will see how each of the four Print functions are used in the programs we create.

7.0 Using Input and Output Functions

You have seen one method of output from the 12dPL. You may also create output by writing to the *Macro Console*, by placing text on the clipboard or by writing to files.

Input to the 12dPL may be via the *Macro Console* or via custom 12dPL panels with advanced error checking.

7.1 Output to the Macro Console

The Prompt(Text msg) function is used to print to the Macro Console. From the manual:

Prompt(Text msg)

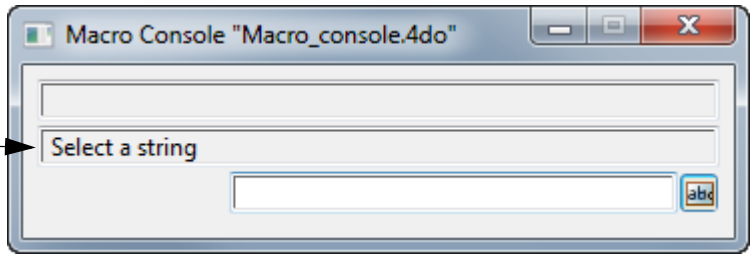
Name

void Prompt(Text msg)

Description

Print the message **msg** to the **prompt message area** of the macro console.

Prompt Message Area



If another message is written to the prompt message area then the previous message will be overwritten by the new message.

ID = 34

We will now create our second program that writes the message “Hello World” to the Macro Console.

Note: “Hello World” is known as a Text Constant which is a special case of a Text variable that the Prompt(Text msg) function requires as its argument.

Type in and then Compile/Run this second program.

COURSE NOTES

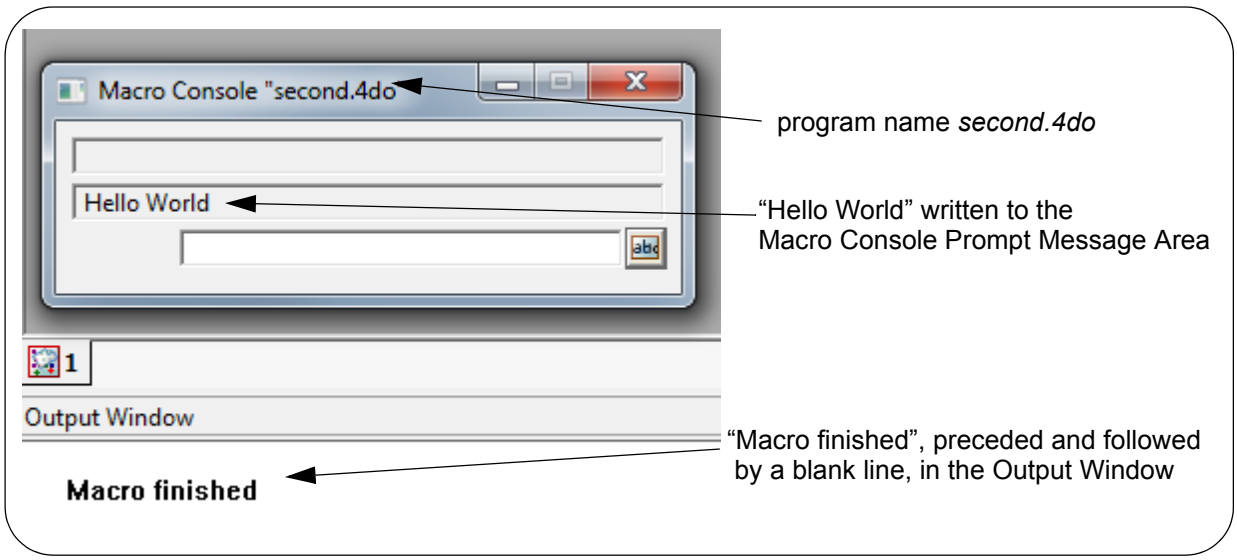
12d Model Programming Language

second.4dm - Notepad

File Edit Format View Help

```
// -------------------------------------------------------
// Macro:        second.4dm
// Author:       ljg
// Organization: 12D Solutions - NSW
// Date:         Mon Aug 19 19:00:03 2013
// -------------------------------------------------------



void main()
// -------------------------------------------------------
// this is where the macro starts
// -------------------------------------------------------
{
    Prompt("Hello World");       // write to the Macro Console

    Print("\nMacro finished\n"); // write to the Output Window
}
```

The running program *second.4do* brings up the ***Macro Console***, writes *Hello World* to the Macro Console and also writes *Macro finished* to the Output Window.



The Output Window is a scrolling window but the Prompt Message Area for he Macro Console contains only one line so if a second message is written to the Prompt Message Area then it will overwrite the first message.

So running the program

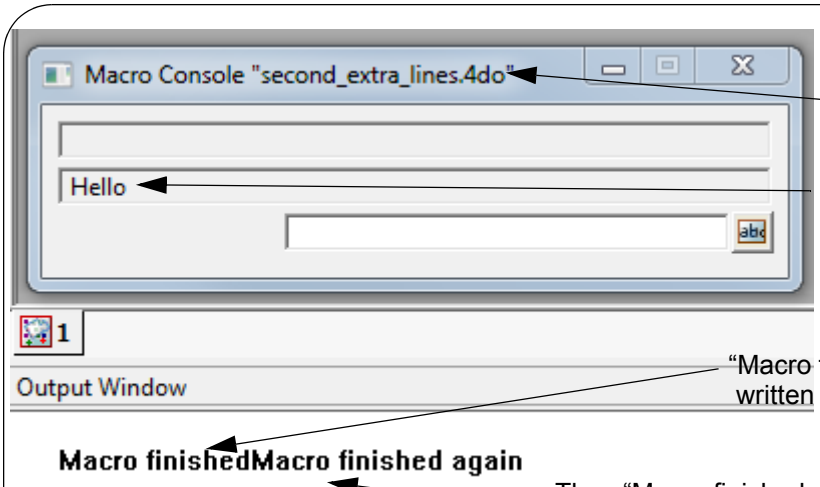
COURSE NOTES

12d Model Programming Language

macro second_exta_lines.4dm

```
void main()
// -------------------------------------------------------
// this is where the macro starts
// -------------------------------------------------------
{
    Prompt("Hello World");        // write to the Macro Console
    Prompt("Hello");               // write to the Macro Console

    Print("\nMacro finished");        // write to the Output Window
    Print("Macro finished again\n");  // write to the Output Window
}
```

produces



program name *second_extra_ines.4do*

"Hello World" and then "Hello" written to the Macro Console Prompt Message Area. Only "Hello" remains.

"Macro finished", preceded by a blank line, written to the Output Window

Then "Macro finished again", with **no** preceded by a blank line is written to the Output Window.

Note that with **Print** and the Output Window, the message continues to be written across the line of the Output Window and a "\n" is needed to scroll to the next line (or by calling **Print()** which is equivalent to **Print("\n")**).

In contrast, **Prompt** overwrites the message in the Macro Console Prompt Message Area.

Hint

Prior to using the **Print** function, you can use the function Clear_console() to clear the Output Window. This function does not have any arguments.

Yes I know, it should be *Clear_output_window* but the programmer must have been in a dream that day.

You will also note that the message "Hello World" flashed by in the Macro Console Prompt Message Area so fast that you never saw it. It was replaced by "Hello".

If you want the program to stop execution after the "Hello World", we'll use the function.

Integer Error_prompt(Text msg)

Even though this function has a return code, you do not have to do anything special. Return codes can just be ignored.

We'll now change

```
Prompt("Hello World");
```

to

```
Error_prompt("Hello World");
```

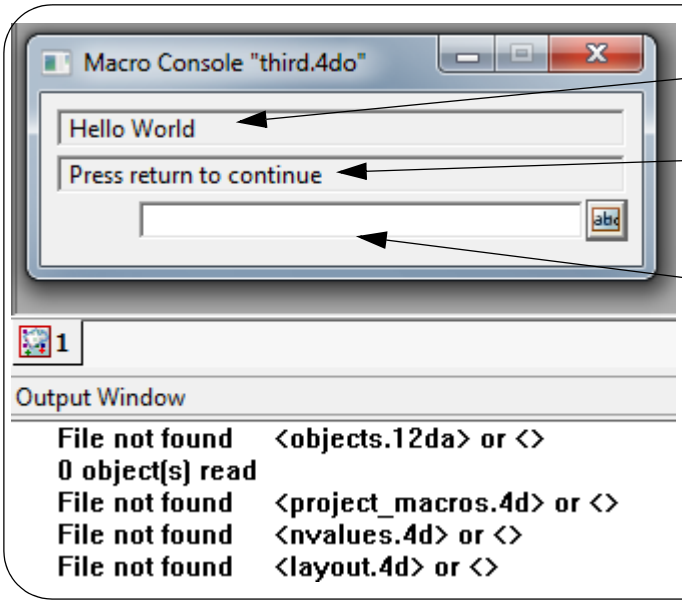
and also change the Print function back to the original and add a Clear_console() call.

The program is now:

macro third.4dm

```
void main()
// -------------------------------------------------------
// this is where the macro starts
// -------------------------------------------------------
{
    Error_prompt("Hello World");   // write to the Macro Console
    Prompt("Hello");               // write to the Macro Console

    Clear_console();
    Print("\nMacro finished\n"); // write to the Output Window

}
```

When running this program, it writes "Hello World" to the Macro Console information/error message area message area, and "Press return to continue" and then pauses.

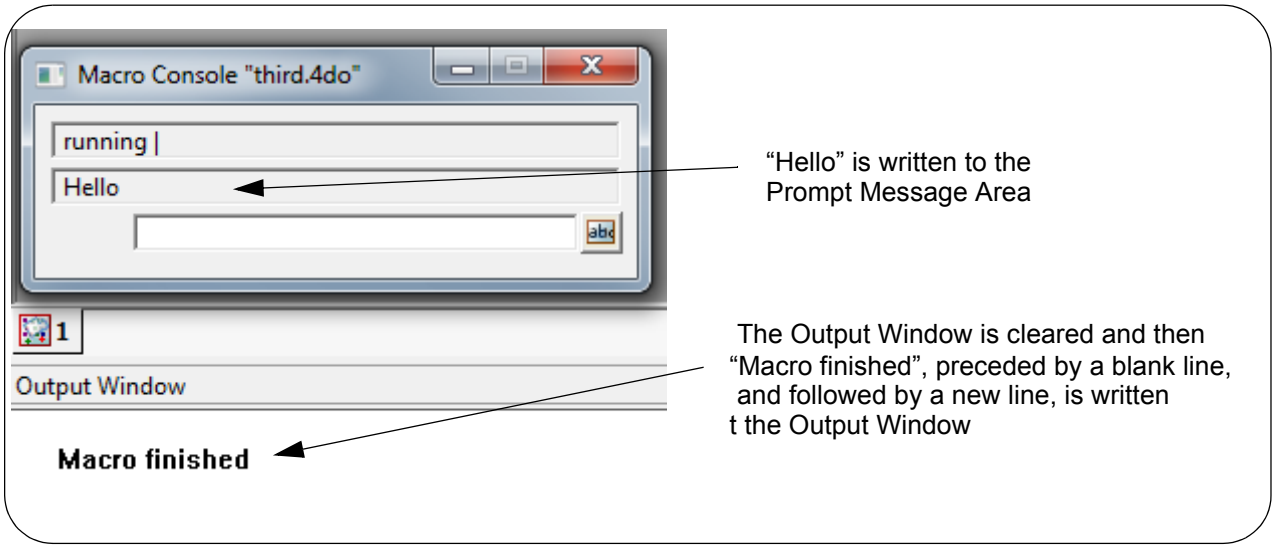


"Hello World" is written to the Information/Error Message Area

"Press return to continue" is written to the Prompt Message Area

The macro then waits until <Enter> is pressed whilst the cursor is focused n the User Reply Area.

When <Enter> is pressed whilst the cursor is focused on the User Reply Area, "Hello" is written to the Prompt Message Area, the Output Window is cleared, then a blank line, "Macro finished" followed by a new line is written to the Output Window.



Click on **X** to remove the Macro Console.

COURSE NOTES

12d Model Programming Language

7.2 Input via the Macro Console (quick and easy)

A simple method to input data is via the Macro Console.

There are three **Prompt** functions with two arguments that can be used to receive data from the Macro Console.

Integer Prompt(Text msg,Text &ret) - writes out **msg** and waits for a Text to be typed in

Integer Prompt(Text msg,Integer &ret) - writes out **msg** and waits for an Integer to be typed in

Integer Prompt(Text msg,Real &ret) - writes out **msg** and waits for a Real to be typed in

Note that the variable name of the second argument is preceded with a **&**. This indicates that the variable is Passed by Reference and so data can be passed back to the calling program via the second arguments.

We are now going to change our program so that it asks for Text, Inter and Real values and prints the values to the Output Window.

To print out the values, we will use the functions

void Print(Text msg) - prints out a Text variable

void Print(Integer value) - prints out an Integer variable

void Print(Real value) - prints out a real variable

void Print() - prints out a blank line

The program to type in is

macro four.4dm

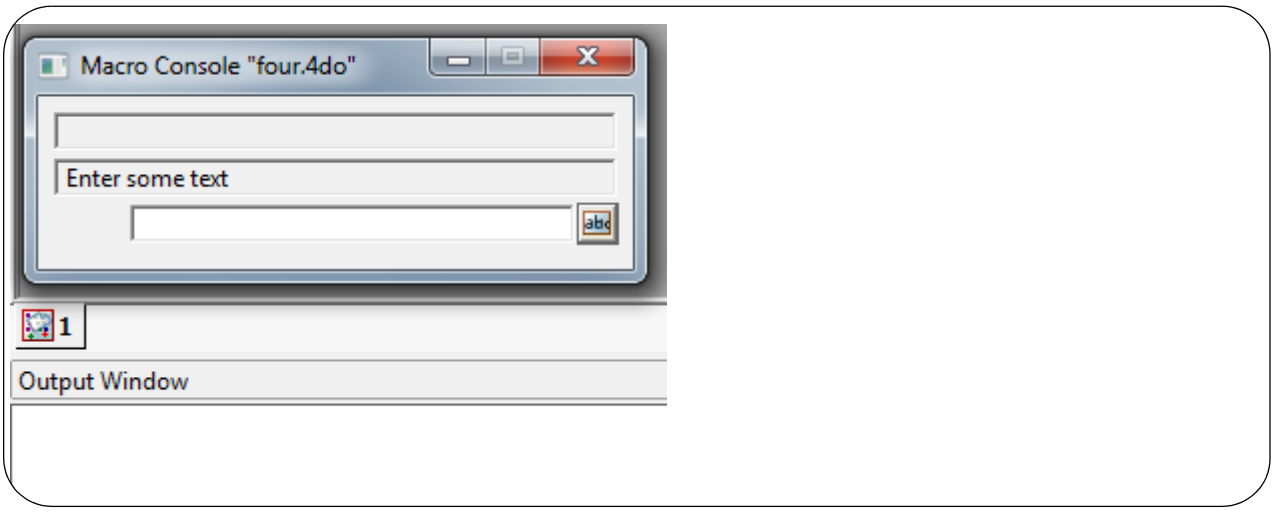
```
void main()
{
  Clear_console();

  Text input_text;            // input_text is a user defined name
  Prompt("Enter some text",input_text);
  Print(input_text+"\n");     // print out a Text variable
                              // + is used to append two Text's


  Integer input_integer;      // input_integer is a user defined name
  Prompt("Enter a positive integer",input_integer);
  Print(input_integer);      // print out an Integer variable
  Print();                   // print out a blank line

  Real input_real;           // input_real is a user defined name
  Prompt("Enter a real",input_real);
  Print(input_real);          // print out a Real variable
  Print("\n");                // print out a blank line

  Prompt("Macro finished");
  Print("\nMacro finished\n"); // write to the Output Window
}
```

Compile/run this program and the program starts by writing “Enter some text” to the Prompt

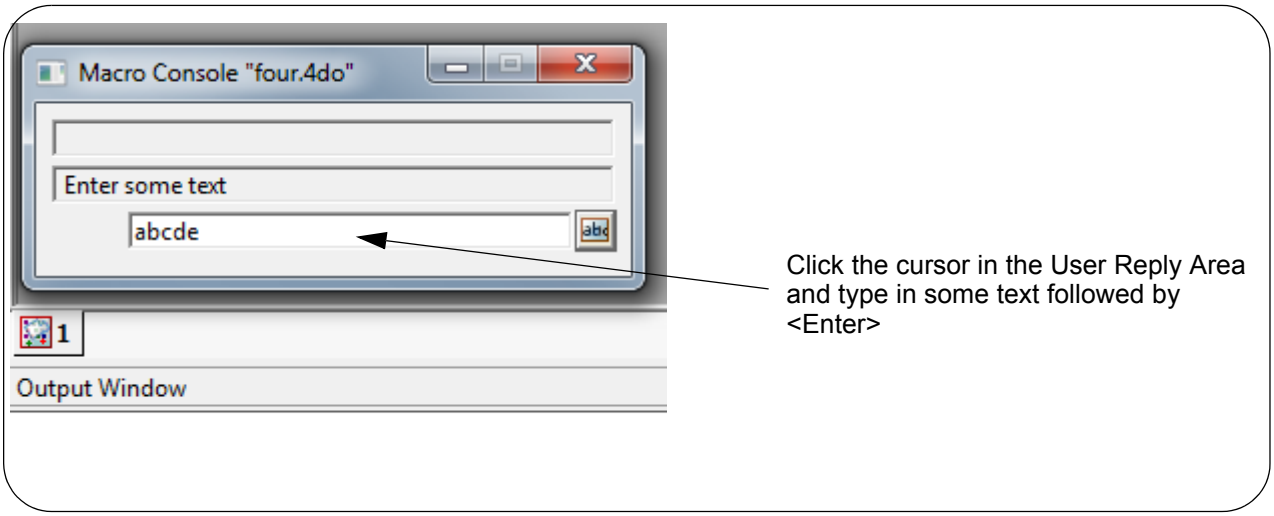
COURSE NOTES

12d Model Programming Language

Message Area

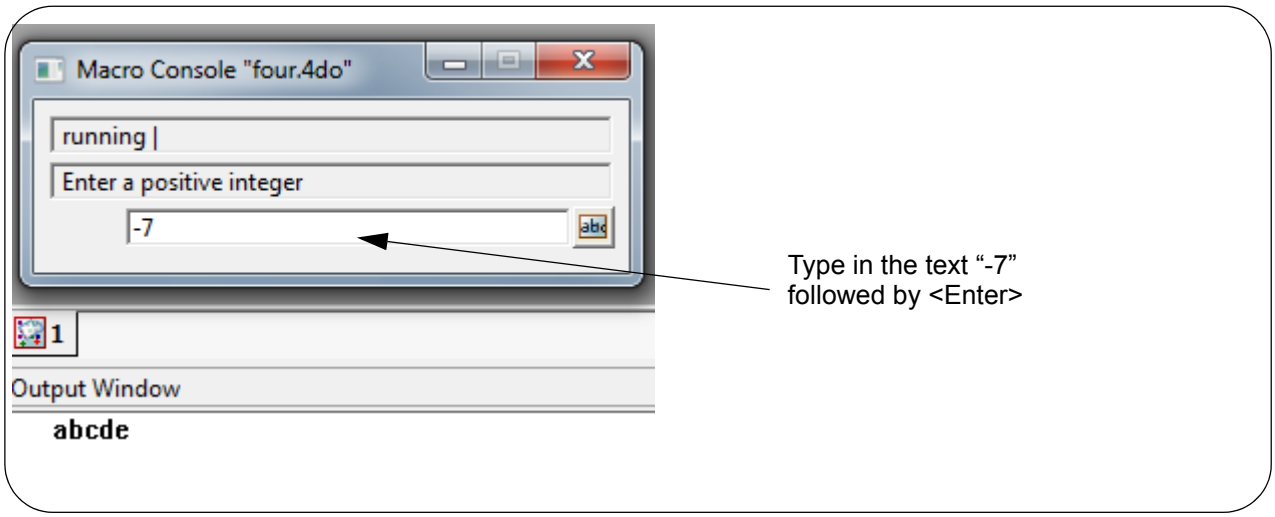


Click the cursor in the User Reply Area and type in some text followed by <Enter>.



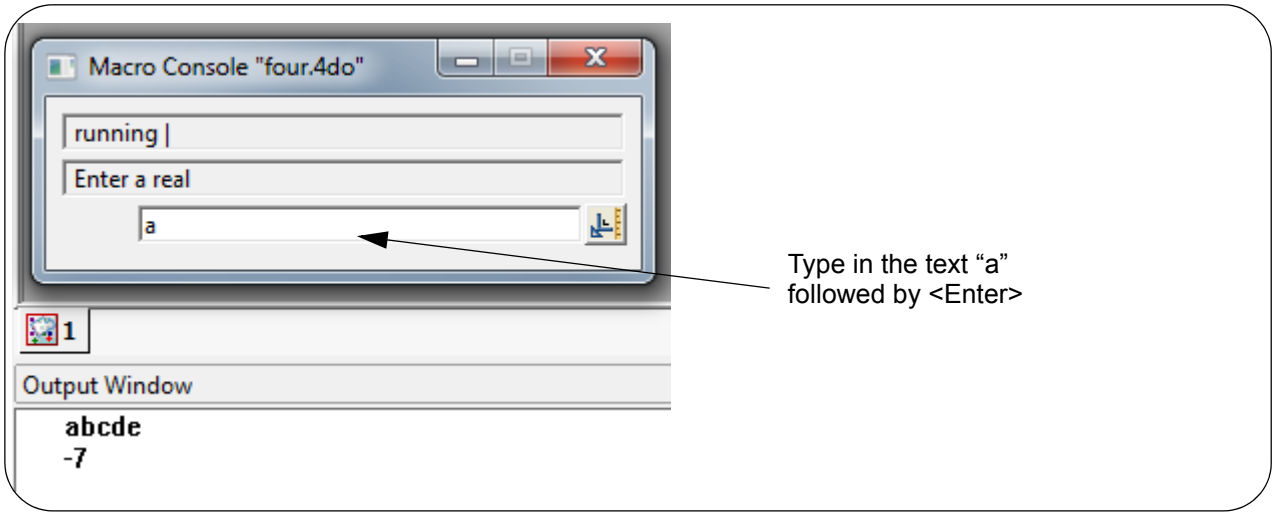
The text is then written to the Output Window and the message “Enter a positive integer” is written to the Prompt Message Area.

Type in the text “-7” followed by <Enter>...



The Integer “-7” is then written to the Output Window and the message “Enter a real” is written to the Prompt Message Area.

Type in the text “a” followed by <Enter>.

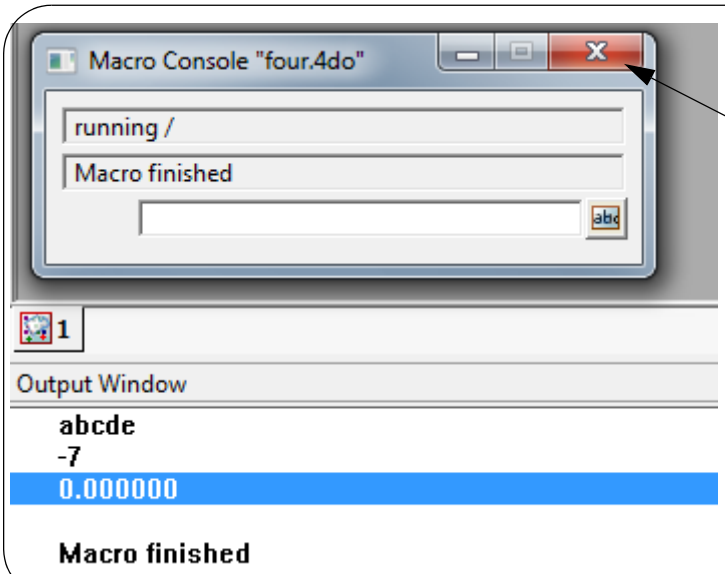


The Real “0.000000” or some other number is then written to the Output Window and the message “Macro Finished” is written to the Prompt Message Area.

COURSE NOTES

12d Model Programming Language

Type in the text “a” followed by <Enter>.



Click on X to remove the Macro Console

Click on **X** to remove the Macro Console.

Now you will notice a few strange things happened whilst running this program.

We were asked to type in some text which we did and everything was fine.

Next we were asked to type in a positive integer and we typed in “-7” which is not a positive integer. Then “-7” was written to the Output Window.

Finally we were then asked to type in a real and we typed in “a” which is not a real. Then “0.000000” (or some other strange number) was written to the Output Window.

So this program is a bit deficient.

To make the program do what we really intended it to do, we need to be able to check if the values we typed in are what we expected, and if not, get annoyed and go back and get new values typed in.

To do this we need to make tests and control the order in which the lines of the program are executed. That is, we need **flow control**.

COURSE NOTES

12d Model Programming Language

8.0 Using Flow Control

In a program, the normal processing flow is that a statement is processed and then the following statement is processed.

The ***flow control*** statements of a language change the **order** in which statements are processed.

12dPL supports a subset of the C++ flow control statements but before we start examining the flow controls, we need to look at logical expressions.

8.1 Logical Expressions

Many flow control statements include expressions that must be *logically evaluated*.

That is, the flow control statements use expressions that must be evaluated as being either ***true*** or ***false***.

For example,

a is equal to b	a == b
a is not equal to b	a != b
a is less than b	a < b

Following C++, 12dPL extends the expressions that have a truth value to any expression that can be evaluated arithmetically by the simple rule:

> **an expression is considered to be true if its value is non-zero, otherwise it is considered to be false**.

Hence the truth value of an arithmetic expression is equivalent to:

"value of the expression" is not equal to zero

For example, the expression

a + b

is true when the sum a+b is non-zero.

Any expression that can be evaluated logically (that is, as either true or false) will be called a **logical expression**.

8.2 12dPL Flow Controls

The flow control statements supported by 12dPL are listed below with links to for definitions for them. However we will only cover some of them in this course.

- if, else, else if
- Conditional Expression
- Switch
- While Loop
- For Loop
- Do While Loop
- Continue
- Goto and Labels

8.3 .“goto” and “label” Statements

12dPL supports the standard C++ **goto** and **labels**.

Although modern programming theory frowns upon goto’s and labels, they are very simple to understand and use.

A **label** has the same form as a variable name and is followed by a colon (**:**).

A label can be attached to any statement in a function. A label name must be unique within the function.

A **goto** is always followed by a **label** and then a semi-colon (**;**).

When a **goto** is executed in a program, control is immediately transferred to the statement with the appropriate **label** attached to it. The label must be in the same function as the goto.

There may be many gotos with the same label in the function.

8.4 .“if” and “else” Statements

If statements are used frequently to execute a statement or a block of statements only if a condition is true.

```
if (conditional) {
        // these statements are executed if the conditional is true
}
```

If else statements are used frequently to execute a statement or a block of statements if a condition is true, and a different statement or a block of statements if the condition is false.

```
if (conditional) {
     // these statements are executed if the conditional is true
} else {
     // these statements are executed if the conditional is false
}
```

If can follow **else**.

```
if (conditional_1) // these statements are executed if the
                   //conditional_1 is true
} else if (conditional_2){
                // these statements are executed if the
                // conditional_1 is false and conditional_2 is true
}
```

8.5 .Error Checking Using “goto”, “label”, “if” and “else” Statements

We will now change the previous program using flow control statements to try and fix up some of the problems.

program five.4dm

```
void main()
{
  Clear_console();

  Text input_text;
  Prompt("Enter some text",input_text);
  if (input_text == "some text") Print("good typing\n");
  else Print("typing error\n");

  Integer input_integer;

  get_integer:
  Prompt("Enter a positive integer",input_integer);
  if(input_integer > 0) {
    Print(input_integer);
    Print();
  } else {
    Print("The number is less than 1. Go and try again");
    Print();
    goto get_integer;
 }

  Integer ierr;
  Real input_real;

  get_real:
  ierr = Prompt("Enter a real",input_real);
  if(ierr!= 0){
    Print("Not a real. Go and try again\n");
    goto get_real;
  } else {
    Print(input_real);
    Print();
  }

  Prompt("Macro finished");
  Print("\nMacro finished\n"); // write to the Output Window
}
```

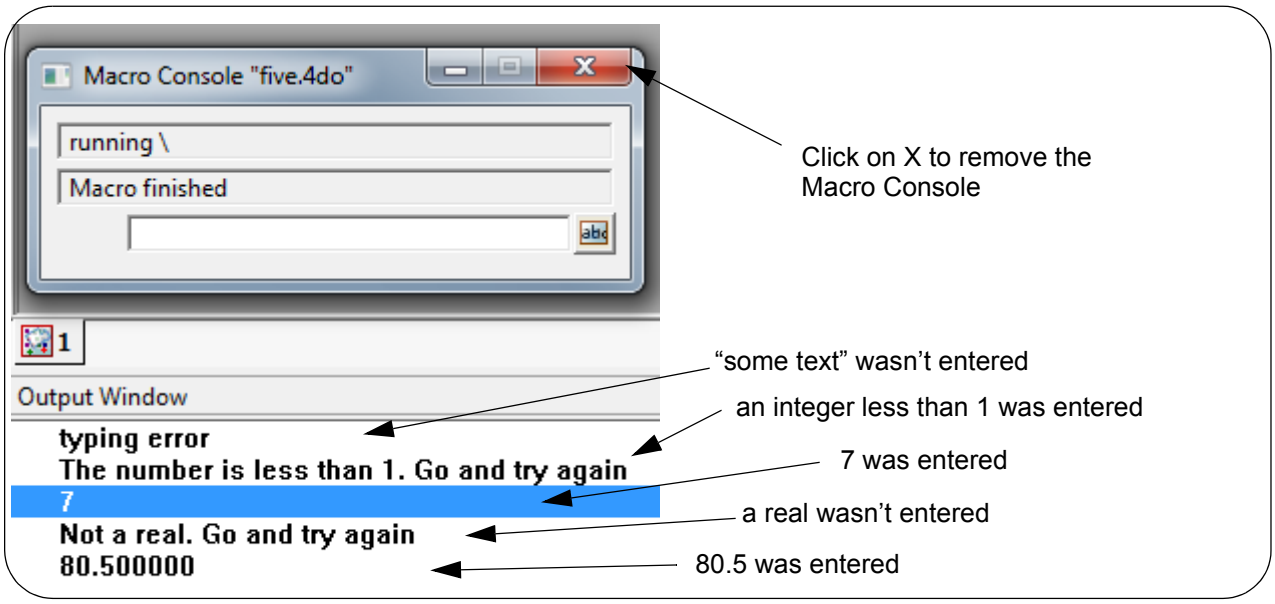
checking a value

indenting

checking a positive value

checking a function return code

indenting



A few things to note in the program ***five.4dm*** are:

1. **Indenting** - each line has been indented by an extra two spaces when inside a block. This is to make it easier to line up brackets etc.
2. **Checking a value** - for the code around "Enter some text", it expects the text "some text" to be entered to get the message "good typing". But if you don't type that in then you get the message "typing error" and the program moves on.
3. **Checking a value** - for the code around "Enter a positive integer", it tests to see if the entered integer is greater than zero and if not, it loops back and asks you to "Enter a positive integer" again. This will keep looping forever or until a positive integer is entered.
4. **Checking the function return code** - for the code around "Enter a real", the Integer variable ierr records the function return code

   ```
   ierr = Prompt("Enter a real",input_real);
   ```

   From the documentation on Prompt(Text msg,Real &ret) , if **ierr** is non zero then there was a error in the function. This would occur when "a" was typed in instead of a real number.

   If an error occurs then it loops back and asks you to "Enter a real" again. This will keep looping forever or until a real number is entered.

IMPORTANT NOTE

Always check function return codes or error codes to ensure that the function behaved correctly'. If an error has occurred, then the results of the function may be garbage.

8.6 "for" loops

A **for** loop is appropriate when a block has to be executed a fixed number of times.

12dPL supports the standard C++ **for** statement.

for (expression1;logical_expression;expression2) statement

This looks like gibberish but in long hand it means:

(a) first execute ***expression1***.

(b) if ***logical_expression*** is true, execute ***statement*** and ***expression2*** and then test ***logical_expression*** again.

(c) repeat (b) until the ***logical_expression*** is false.

This probably still seems like gibberish so an example might help.

```
j = 0;
for (i = 1; i <= 10; i++)
  j = j + i;
```

This actually sums the numbers 1 through to 10. To see that we'll step though it more carefully:

expression1 is i = 1.

logical_expression is i <= 10. That is, is less than or equal to 10.

expression2 is i++. That is, increase i by 1.

statement is j = j + i. That is, the new value for j is the current value of j plus the current value of i.

Start by setting j is to 0.

First execute expression1: **i** is set to1.

First pass:

1 <= 10 so j= j + i is executed so j = 0 + 1 = 1.

i is then incremented to 2 and 2 <= 10.

Second pass:

Now i = 2 and 2 <= 10 so j = j + 2 is executed so j = 1 + 2 = 3.

i is then incremented to 3 and 3<= 10.

Third pass:

Now i = 3 and 3<= 10 so j = j + 3 is executed so j = 1 + 2 + 3 = 6.

i is then incremented to 4 and 4<= 10.

…

Ninth pass:

Now i = 9 and 9<= 10 so j = j + 9 is executed so j = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9

i is then incremented to 10 and 10<= 10.

Tenth pass:

Now i = 10 and 10<= 10 so j = j + 10 is executed so j = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10

i is then incremented to 11 and 11 > 10 and so the loop stops.

8.7 “while” loops

while loops are convienent for executing a block of statements until a condition is reached.

12dPL supports the standard C++ **while** statement.

while (logical_expression) statement

Again this may look like gibberish but in long hand it means:

(a) If **logical_expression** is true, execute **statement** and then test the **logical_expression** again.

(b) repeat (a) until the **logical_expression** is false.

A simple example of a **while** loop is.

```
Text data;
data = “ ”;

while (data != "stop") {
  Prompt("Enter some text",data);
  Print(data+"\n");
}
```

This keeps prompting the user to enter some text and it keeps re asking until the text “stop” is entered. To see that we’ll step though it more carefully:

logical_expression is data != “stop”. That is, the Text data is not equal to “stop”

statement is Prompt(“Enter some test,data);

First pass

The data is “ ” so data does not equal “stop” and Prompt for some Text data to be entered.

Repeat Pass

Check if new data does equal “stop” then logical_expression is false and this ends the **while** loop.

If the entered data does not equal “stop”, then it prompts again for some Text data to be entered and the **Repeat Pass** is repeated.

8.8 “switch” Statement

12dPL supports a **switch** statement.

The **switch** statement is a multiway decision that tests a value against a set of constants and branches accordingly.

In its general form, the switch structure is:

```
switch (expression) {
    case constant_expression : { statements }
    case constant_expression : { statements }
    default  : { statements }
}
```

Each case is labelled by one of more constants.

When **expression** is evaluated, control passes to the case that matches the expression value.

The case labelled **default** is executed if the expression matches none of the cases.

A default is optional; if it isn't there and none of the cases match, no action takes place.

Once the code for one case is executed, *execution falls through to the next case* unless explicit action is taken to escape using **break**, **return** or **goto** statements.

A **break** statement transfers control to the end of the switch statement (see "break" Statement).

Warning

Unlike C++, in 12dPL the statements after the **case constant_expression**: must be enclosed in curly brackets ({}).

Switch Example

An example of a switch statement is:

```
switch (a) {
        case 1 : {
          x = y;
          break;
        }
        case 2: {
          x = y + 1;
          z = x * y;
        }
        case 3: case 4: {
          x = z + 1;
          break;
        }
        default : {
          y = z + 2;
          break;
        }
}
```

Note

If control goes to case 2, it will execute the two statements after the case 2 label and then continue onto the statements following the case 3 label.

Restrictions

1. Currently the switch statement only supports an **Integer**, **Real** or **Text** expression. All other expression types are not supported.
2. Statements after the **case** constant_expression: must be enclosed in curly brackets ({}).

8.9 “continue” Statement

Now that we are starting to use flow control statements, another useful statement is ***control***.

The **continue** statement causes the next iteration of the enclosing **for** or **while** loop to begin. It also applies to **do while** loops which we haven’t defined yet. See Do While Loop .

In the **while** and **do**, this means that the test part is executed immediately.

In the **for**, control passes to the evaluation of expression2, normally an increment step.

Important Note
The **continue** statement applies only to loops. A **continue** inside a **switch** inside a **loop** causes the next **loop iteration**.

8.10 “break” Statement

break is used to exit from a **do**, **for**, or **while** loop, bypassing the normal loop condition. It is also used to exit from a **switch** statement.

In a **switch** statement, **break** keeps program execution from "falling through" to the next **case**. A **break** statement transfers control to the **end** of the **switch** statement.

A **break** only terminates the **for**, **do** or **while** statement that contains it. It will not break out of any nested loops or **switch** statements.

9.0 Running Existing 12dPL Programs

For most of the work we have been doing so far we have used

Utilities =>Macros =>Compile/run

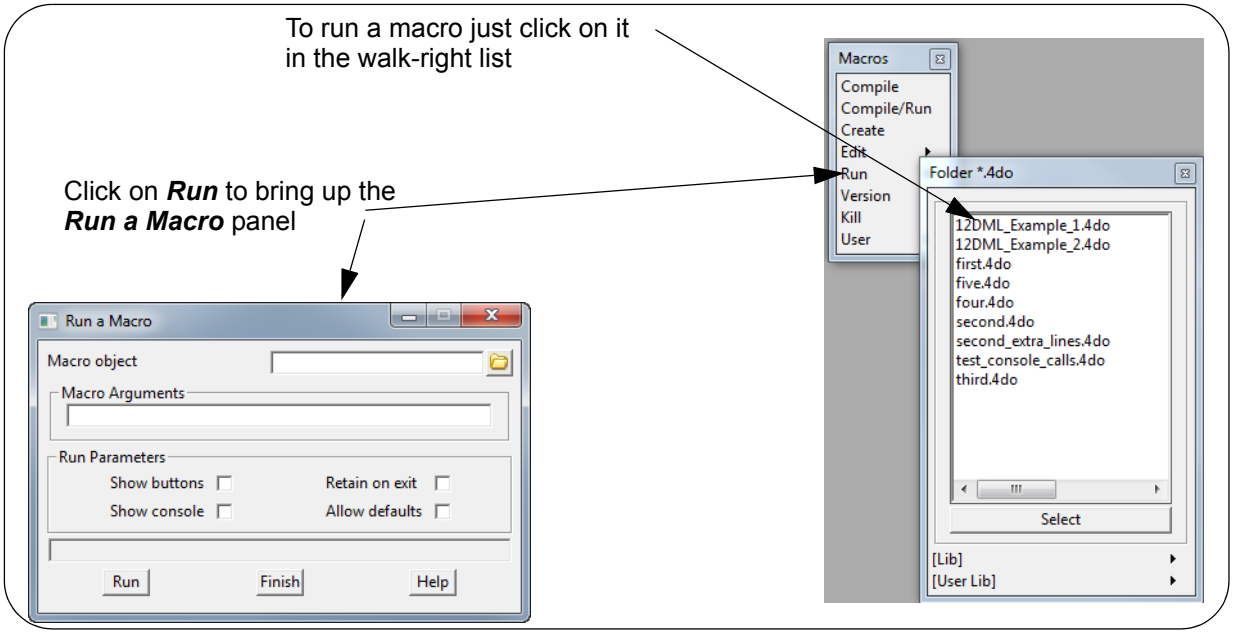
which is normal when you are writing and debugging your program.

However once the program is finished, you no longer need to compile it every time you run the program.

The option

Utilities =>Macros =>Run

has a walk right menu and a program can be run by clicking on it in the walk-right list.



However programs run this way will not have **Retain on exit** ticked on and so the *Macro Console* in the examples we have created will disappear as soon as the program finishes.

To bring up the ***Run a Macro*** panel which allows **Retain on exit** to be ticked on, don't walk right but click on click on

Utilities =>Macros =>Run

For regularly used program, we will later see how they can be added to user menus or toolbars, or bound to function keys (see User Menus, User Defined Function Keys and Toolbars).

10.0 Unleashing the Power - 12d Database Handles

The real power of the 12dPL comes with accessing the data inside the ***12d Model*** database. This database holds all of the entities for the project such as Views, Model, Strings, Tins, Functions etc.

An entity in the ***12d Model*** database is accessed by creating what is called a ***handle*** to the entity. The **handle** doesn't contain the actual database information but merely points to the appropriate database record for the entity.

The 12dPL variables **Element**, **Model**, **View**, and **Macro_Function** create and use handles.

Once a **handle** has been constructed to point to an entity, the properties of the entity may be obtained, printed in a report, changed etc via the **handle**.

Since the **handle** merely points to the Project data, the handle can be changed so that it points to a different record without affecting the data it originally pointed to.

Sometimes it is appropriate to set a handle so that it doesn't point to any data. This process is referred to as setting the handle to *null*.

Note that when setting a handle to null ("nulling" it), no **12d Model** data is changed - the handle simply points to nothing.

For more information, see 12d Model Database Handles .

As well as accessing existing entities, 12dPL can also create **new *12d Model*** database entities. For example, data can be read from reports and then strings created according to the information read in from the report.

10.1 Locks

Whenever an handle to an entity (string, model, tin etc.) in the database is created and assigned to a variable, the entity becomes locked to other processes. In order to remove the lock, the variable holding the handle must go out of scope. A variable defined inside a block goes out of scope when execution reaches the bottom of the block.

For this reason blocks are often defined solely to have variables go out of scope. Also it is good practice to obtain all of your handles after all user input is finished and have the variables go out of scope (or null them using the null() function) before requesting more input from a prompt box or dialogue. In this way the entities never remain locked while the program is in a user input mode.

For more information, see Locks .

10.2 Read In Some Data to use 12dPL Programs On

We need some ***12d Model*** data to use with the programs we will be creating.

Read in the 12da file *Barwon_data.4da* into your project and add the models *terrain* and *boundary* to a plan view.

10.3 Elements, Models and Uids

The variable type Element is used as a handle to all the data types that can be stored in a ***12d Model* model**. That is, Elements are used to refer to 12d Model strings, tins, super tins and plot frames.

Elements act as handles to the data in the *12d Model* database so that the data can be easily referred to and manipulated within a program.

For example, once we have an Element, we can call functions such as Get_points(Element elt,Integer &num_verts) :

Get_points(Element elt,Integer &num_verts)

Name

Integer Get_points(Element elt,Integer &num_verts)

Description

Get the number of vertices in the Element **elt**.

The number of vertices is returned as the Integer **num_verts**.

For Elements of type Alignment, Arc and Circle, Get_points gives the number of vertices when the Element is approximated using the 12d Model cord-to-arc tolerance.

A function return value of zero indicates the number of vertices was successfully returned.

ID = 43

The variable type Model is used as a handle to ***12d Model* models** which act as containers of Element data.

Elements and *Models* created within ***12d Model*** are given a unique identifier called a **Uid** (see Uid's). When a new element or model is created, it is given the next available Uid. Uid's are never reused so when an element or model is deleted, its Uid is not available for any other element or model.

10.4 Accessing Elements

When a string is requested by the user the first step is to create a handle to the string. Handles to strings are variables of type **Element**.

A simple way to allow the user to select a string from a program is with the *String_select* function

Select_string(Text msg,Element &string)

Name

Integer Select_string(Text msg,Element &string)

Description

Write the message **msg** to the 12d Model ***Output Window*** and wait until a selection is made.

If a pickable Element is selected, then return the Element picked by the user in **string** and the function return value is 1.

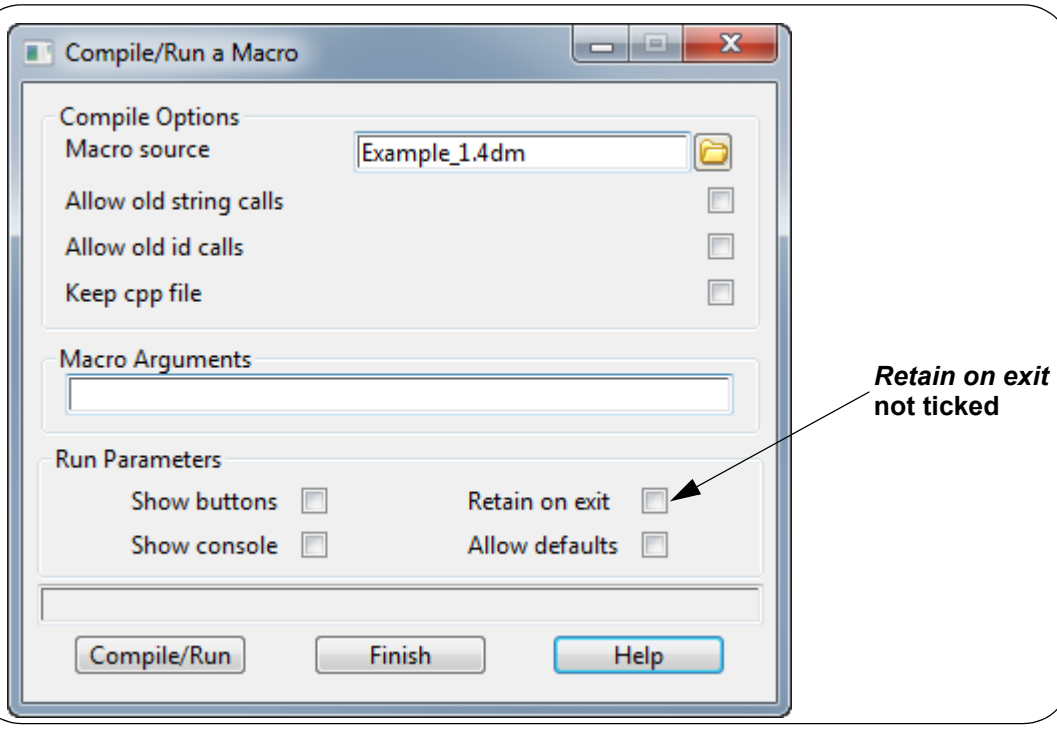
If no pickable Element is picked and the function returns, then the function returns codes are:

-1	indicates cancel was chosen from the pick-ops menu.
0	pick unsuccessful
1	pick was successful
2	a cursor pick

ID = 29

Now that we can select a string, we'll write a program to select a string and write out to the *Macro Console* how many vertices there are in the string.

This time we will **not** tick on **Retain on exit** on the *Compile/Run a Macro* panel. The *Macro Console* will then be removed as soon as the program terminates.



COURSE NOTES

12d Model Programming Language

Example 1

```
void main(){
  Element string;
  Integer ret,no_verts;
  Text text;

  Prompt("Select a string");// write message to console

ask:
  ret = Select_string("Select a string",string); //message to Output Window
  if(ret == -1) {
    Prompt("Macro finished - cancel selected");
    return;
  } else if (ret == 1) {
    if(Get_points(string,no_verts)!=0) goto ask;
    text = To_text(no_verts);
    text = "There are "+text+" vertices in the string. Select another string";
    Prompt(text);
    goto ask;
  } else {
    Prompt("Invalid pick. Select again");
    goto ask;
  }
}
```

handle to an Element

end running of macro at this point

converting an Integer to a Text

building up the Text message

A few things to note are:

1. The ***return*** statement, when executed, terminates the program. All the previous programs terminated because they reached the end of statements in the program.
2. The Integer no_verts was converted to Text so that is could be concatenated with other texts using the + operator.
3. Function return codes are important

   The function return code for *Select_string* gives important information about the select action not just if a string was successfully selected or not. For example if a string was not selected, the function return code supplies the extra information about if *Cancel* chosen, or a cursor pick was made.
4. Some Prompt messages may not be visible because another message may over write them.

10.5 Exercises 1 and 2

10.5.1 Exercise 1

Rewrite Example 1 so there are no goto's used.

See Example 1a .

10.5.2 Exercise 2

Modify Example 1 so that it asks if the selected string is to be deleted.

And if the answer is yes, then delete the string.

See Example 2 *and* Example 2a .

10.6 Accessing Models

When a model is requested by the user the first step is to create a handle to the model. Handles to models are variables of type **Model**.

A simple way to interact with the user regarding models is with the *Model_prompt* function

Model_prompt(Text msg,Text &ret)

Name

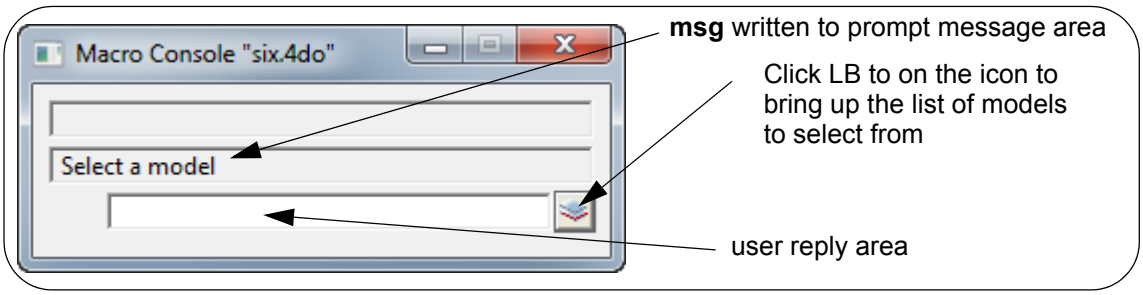
Integer Model_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the **prompt message area** and then read back a Text from the **user reply area** of the Macro Console.

If LB is clicked on the model icon at the right hand end of the **user reply area**, a list of all existing models is placed in a pop-up. If a model is selected from the pop-up (using LB), the model name is placed in the **user reply area**.

MB for "Same As" also applies. That is, If MB is clicked in the **user reply area** and then a string from a model on a view is selected, the name of the model containing the selected string is written to the **user reply area**.



The reply, either typed or selected from the model pop-up or Same As, must be terminated by pressing <Enter> for the macro to continue.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text **ret** is returned successfully.

ID = 401

From reading the *Model_prompt* documentation, all that is returned is the name of a model, **not** a handle to the model.

But there is a function to get a handle to a model when you have a model name - *Get_model*.

Get_model(Text model_name)

Name

Model Get_model(Text model_name)

Description

Get the Model model with the name **model_name**.

If the model exists, its handle is returned as the function return value.

If no model of name **model_name** exists, a null Model is returned as the function return value.

ID = 58

So *Get_model* will return a handle to the model of a given name.

Programs often need to operate on all of the elements in a model so a method is needed to obtain all the handles to each of the **Elements** in a model. And to easily do that, we need to know about *Dynamic_Elements*.

10.7 Dynamic_Elements

When we ask for a list of all the handles to elements in the model, or are creating lists of handles to elements, we may not know how many elements there are, or are required.

So to cope with these situations, there is a variable called a Dynamic_Element .

A **Dynamic_Element** is a *dynamic array* and can hold an arbitrary number of **handles** to elements. At any time, the number of items in a dynamic array is known but extra items can be added at any time.

Like fixed arrays, the items in dynamic arrays are accessed by their unique position number. It is equivalent to an array subscript for a fixed array.

But unlike fixed arrays, the items of a dynamic array can only be accessed through 12dPL function calls rather than by array subscripts enclosed in square brackets.

As for an array in 12dPL, the dynamic array positions go from **one** to the **number of items** in the dynamic array.

So for a model, the function

Integer Get_elements(Model model,Dynamic_Element &de,Integer &total_no)

gets all of the handles of the elements in the model and loads them into a Dynamic_Element (de say).

.

Get_elements(Model model,Dynamic_Element &de,Integer &total_no)

Name

Integer Get_elements(Model model,Dynamic_Element &de,Integer &total_no)

Description

Get all the Elements from the Model model and add them to the Dynamic_Element array, **de**.

The total number of Elements in **de** is returned by **total_no**.

Note: whilst this Dynamic_Element exists, all of the elements with handles in the Dynamic_Element are locked.

A function return value of zero indicates success.

ID = 132

While this Dynamic_Element exists, all of the elements it refers to will be locked.

10.8 Accessing Element in Models

We will now look at a program using the variable types **Model**, **Element** and **Dynamic_Element**.

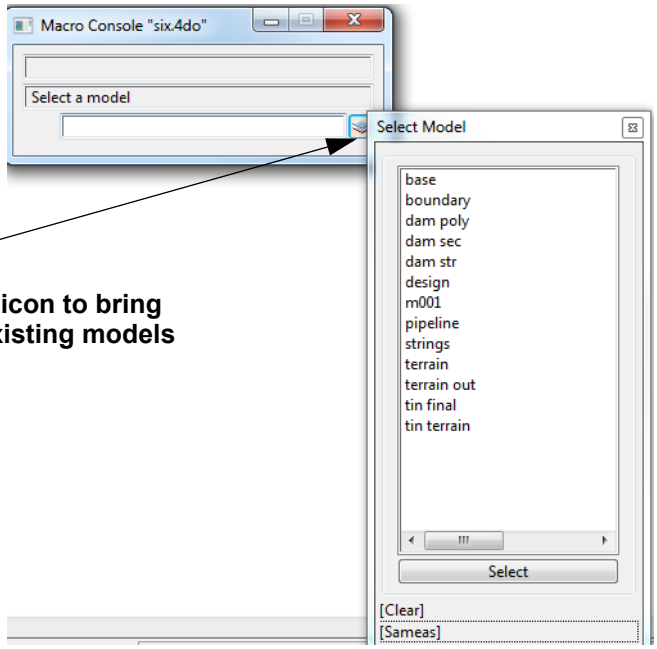
macro six.4dm

```
void main()
{
Text my_model_name;
  Model my_model;

  while(!Model_exists(my_model)) {
    Model_prompt("Select a model",my_model_name);
    my_model = Get_model(my_model_name);
  }

  Uid model_uid;

  Get_id(my_model,model_uid);
  Print("Model uid ");
  Print(model_uid);
  Print("\n");

  Dynamic_Element model_elts;
  Integer num_elts;

  Get_elements(my_model,model_elts,num_elts);
  Print("There are ");
  Print(num_elts);
  Print(" elements in the model: " + my_model_name + "\n");

  Prompt("Macro finished");
  Print("\nMacro finished\n"); // write to the Output Window
}
```

using the Integer return code as a logical value

get the Uid of a model

get handles to all the elements in a model and load them into a Dynamic_Element



click on model icon to bring up the list of existing models to select from

10.9 Getting Information about an Element

Once we have a element handle, there are numerous 12dPL functions to get information about the element such as *Get_points* which we used before, and the new call *Get_id*.

..

Get_id(Element elt,Uid &uid)

Name

Integer Get_id(Element elt,Uid &uid)

Description

Get the unique Uid of the Element **elt** and return it in **uid**.

If **elt** is null or an error occurs, **uid** is set to zero.

A function return value of zero indicates the Element Uid was successfully returned.

ID = 1908

10.10 Putting it All Together

Now we will add the flow control **for** to retrieve and for each element in the selected model, print the element's name, Uid, type and the number of vertices in the element.

This program will use most of the concepts we have introduced.

COURSE NOTES

12d Model Programming Language

macro seven.4dm

```
void main()
{
  Text my_model_name;
  Model my_model;

  while(!Model_exists(my_model)) {
    Model_prompt("Select a model",my_model_name);
    my_model = Get_model(my_model_name);
  }

  Uid model_uid;
  Get_id(my_model,model_uid);
  Print("Model uid ");
  Print(model_uid);
  Print("\n");

  Dynamic_Element model_elts;
  Integer num_elts;

  Get_elements(my_model,model_elts,num_elts);
  Print("There are ");
  Print(num_elts);
  Print(" elements in the model: " + my_model_name + "\n");

  for(Integer i=1;i<=num_elts;i++) {
    Element element;
    Get_item(model_elts,i,element);

    Text element_name;
    Get_name(element,element_name);
    Print("Name: "+ element_name +" Uid: ");

    Uid element_uid;
    Get_id(element,element_uid);
    Print(element_uid);

    Text element_type;
    Get_type(element,element_type);
    Print(" Type: " + element_type + Num vertices: ");

    Integer num_verts;
    Get_points(element,num_verts);
    Print(num_verts);
    Print("\n\n");
  }
  Prompt("Macro finished");
  Print("\nMacro finished\n"); // write to the Output Window
}
```

Compile and Run the program.

COURSE NOTES

12d Model Programming Language

A few things to note are:

1. **It is important to read the 12dPL function documentation carefully**

   Every function call is different and the function return value and its meaning can be different.

2. **The type of the function return code varies**

   The variable type of the function return codes varies. For *Model_prompt* it is an **Integer** but for *Get_model* it is a **Model**.

3. **Function return codes are not always for errors**

   Sometimes the function return code is for indicating an error BUT NOT ALWAYS.

   Sometimes a return code of zero indicates the function ran successfully, and sometimes zero indicates the function didn't run successfully.

10.11 Exercises 3and 4

10.11.1 Exercise 3

The program *six.4dm* finishes after reporting the number of elements for one model.

How can the program be modified so that after reporting the number of elements for one model, that it repeats the process. That is, it keeps asking for a new model and printing the number of elements out for the new model.

How will the program finish?

Hint

What does the following piece of code do?

```
while(!Model_exists(my_model)) {
  Model_prompt("Select a model",my_model_name);
  my_model = Get_model(my_model_name);
  Print("Entered name = <");
  Print(my_model_name); Print(">\n");
  my_model = Get_model(my_model_name);  }
}
```

An Aside
Notice that it is legal to have more than one statement on the one line.

Question

Why was the “<“ and “>” included in the piece of code?

10.11.2 Exercise 4

The program *seven.4dm* finishes after reporting the number of elements and some information for each string in the model.

Modify s*even.4dm* so that after reporting the information about one model, that it repeats the process. That is, it keeps asking for a new model and prints out the information for the new model.

11.0 Infinite Loops

When writing programs it is possible to put the program into a loop so that the program never finishes (infinite loops).

Some program loops can be stopped gracefully (see Killing a 12dPL Program), others require ***12d Model*** itself to be stopped (see Ending the Process 12d.exe).

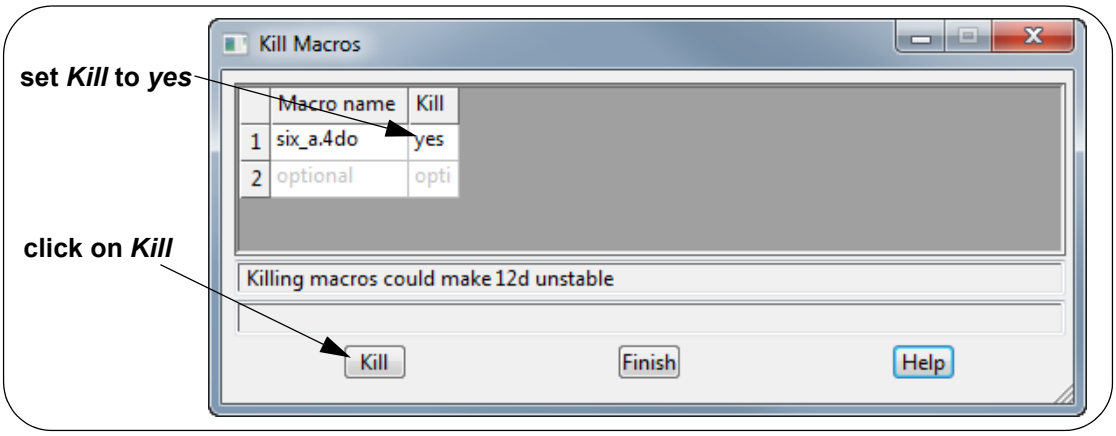
So it is important to thoroughly test your programs on data and projects that are not important before using them on critical data.

11.1 Killing a 12dPL Program

Some looping programs pause whilst waiting for further information. These programs can usually be stopped by clicking on the **X** on the Macro Console, or if there is no Macro Console, by the option

Utilities =>Macro => Kill.

which lists the running programs and allows them to be stopped (killed).



Set the Kill column to ***yes*** for the programs to be killed and then click on ***Kill***.

The selected programs will then be terminated.

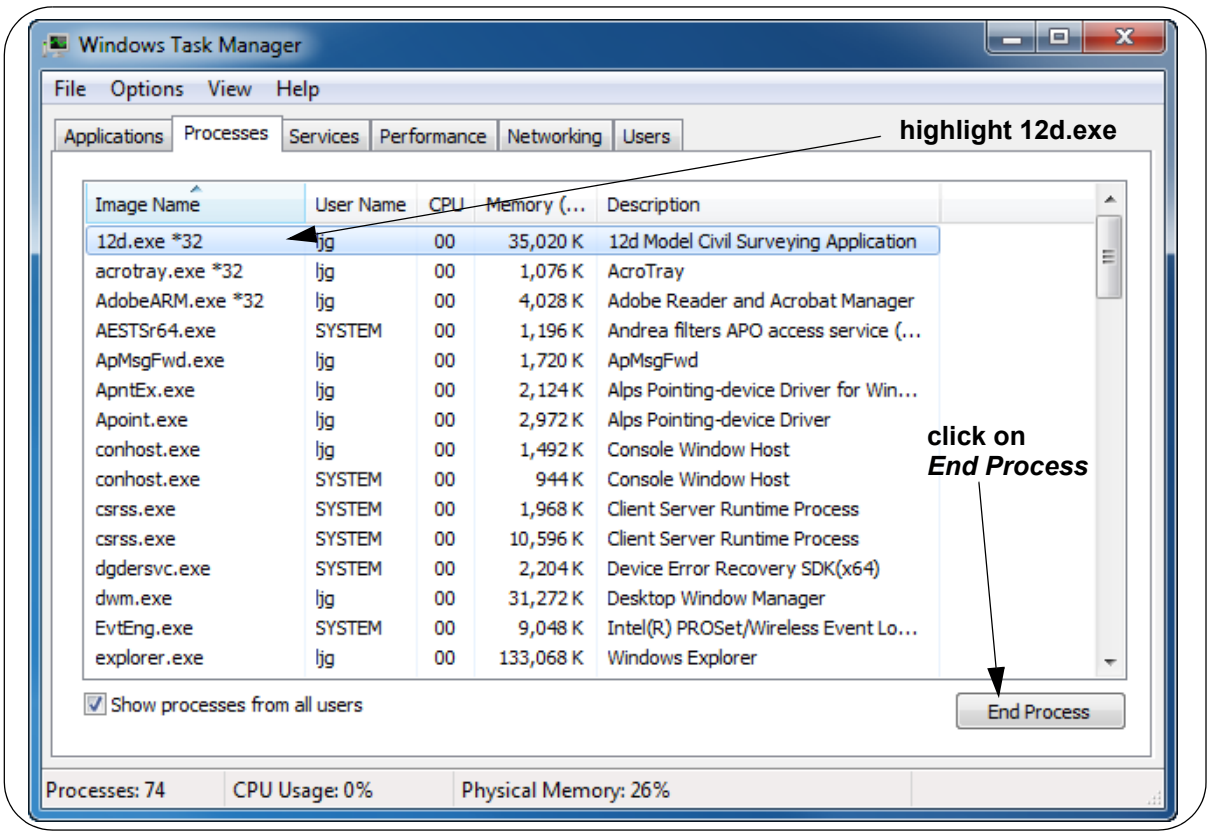
Note: after killing any program, it is a good procedure to restart ***12d Model***. A save may or may not be appropriate depending on what the killed programs did.

11.2 Ending the Process *12d.exe*

Some looping programs do not pause waiting for further information and so totally lock up ***12d Model***.

These programs can only be stopped by stopping the Process *12d.exe* itself.

This is done by holding the Ctrl, Alt and Delete keys down together (<Ctrl>+<Alt>+<Delete>) and selecting ***Start Task Manager*** to bring up the ***Windows Task Manager.***



Highlight ***12d.exe*** and then click on ***End Process***.

This will totally stop ***12d Model*** and any data that has not been saved will be lost.

12.0 Writing to a Text File (Reports)

The previous example *seven.4dm* can be quickly modified to write the data to a text file rather than to the Output Window. For example, if a report is needed.

Text files, both ANSI (ASCII) or UNICODE, can be created and read via 12dPL functions.

To write a text file, four 12dPL functions are required.

(a) Open a Text File for Writing

Integer File_open(Text file_name, “w”,””, File &file)

to write a new file with ANSI encoding (ASCII)

or

Integer File_open(Text file_name, “w”,”ccs=UNICODE”, File &file)

to write a new file with UNCODE encoding

or

Integer File_open(Text file_name, “a”,””,File &file)

to append to an existing file.

Opening a file accesses the file and returns a **handle** to the file of variable type **File**.

Note that if the file already exists and it has a BOM (Byte Order Mark), the Unicode coding specified by the BOM takes precedence over that specified by the ccs flag. The ccs encoding is only used when no BOM is present or the file is a new file.

For all the File_open choices, see File_open(Text file_name,Text mode,Text ccs_text,File &file) .

(b) Write to a Text File

Integer File_write_line(File file,Text text_out)

This is used to write data to the file, line by line.

(c) Flush the File

Integer File_flush(File file)

This is used to make certain all the data has been written out to the file.

and finally

(d) Closing a File

Integer File_close(File file)

The file must be closed once writing has been finished. If a file is not closed, then some of the data might not get written out to the file. Also other processes will not be able to access the file.

12.1 Writing a Simple Unicode and ANSI (Ascii) Files

The default file type in ***12d Model*** is now Unicode files. However some older software may not be able to read Unicode files and you may be required to write out an ANSI (Ascii) file.

Example 5a creates both an Unicode and an Ascii file.

Example 5a

```
void main()
  File file;
  Text file_name, file_type;
  Integer file_start;
  Clear_console();

  file_name = "test_unicode.rpt";
  file_type = "ccs=UNICODE";
  if(File_exists(file_name)) File_delete(file_name);
  File_open(file_name,"w",file_type,file);
  File_tell(file,file_start);     // record the beginning of the file
  File_write_line(file,"one line");
  Print("File <"+file_name+"> Start pos = "+To_text(file_start)+"\n");
  File_close(file);

  file_name = "test_ansi.rpt";
  file_type = "";
  if(File_exists(file_name)) File_delete(file_name);
  File_open(file_name,"w",file_type,file);
  File_tell(file,file_start);     // record the beginning of the file
  File_write_line(file,"one line");
  Print("File <"+file_name+"> Start pos = "+To_text(file_start)+"\n");
  File_close(file);

  Print("\nMacro finished\n"); // write to the Output Window
}
```

the text file is to be UNICODE

the text file is to be Ascii

Compile and Run *Example 5a*.

Look at the files *test_unicode.rpt* and *test_ansi.rpt* to check that they are of the correct type.

12.2 Writing 12d Model Data to a Text File

In the following example, ***eight.4dm***, the user is asked for a model and then information about the model, and information about each element in the model, is written to a Unicode file.

Compile and Run *eight.4dm*.

COURSE NOTES

12d Model Programming Language

.

macro eight.4dm

```
void main()
{
  Text my_model_name;
  Model my_model;
  Clear_console();

  while(!Model_exists(my_model)) {
    Model_prompt("Select a model",my_model_name);
    my_model = Get_model(my_model_name);
  }

  Text file_name;
  File_prompt("Enter the file name","*.rpt",file_name);

  File my_file;
  File_open(file_name,"w",”ccs=UNICODE”,my_file);

  Uid model_uid;
  Get_id(my_model,model_uid);
  File_write_line(my_file,"Model uid "+To_text(model_uid));

  Dynamic_Element model_elts;
  Integer num_elts;

  Get_elements(my_model,model_elts,num_elts);
  File_write_line(my_file,"There are "+To_text(num_elts)+" elements in
the model: "+ my_model_name);

  for(Integer i=1;i<=num_elts;i++) {
    Element element;
    Get_item(model_elts,i,element);

    Text line_out;
    Text element_name;
    Get_name(element,element_name);
    line_out = element_name+"\t";

    Uid element_uid;
    Get_id(element,element_uid);
    line_out += To_text(element_uid)+"\t";

    Text element_type;
    Get_type(element,element_type);
    line_out += element_type+"\t";

    Integer num_verts;
    Get_points(element,num_verts);
    line_out += To_text(num_verts);
    File_write_line(my_file,line_out);
  }
  File_flush(my_file);
  File_close(my_file);
}
```

wild_card_key

open the text file as UNICODE

open the text file for writing
any existing contents are destroyed

tab character

A few things to note are:

1. **wild_card_key in File_prompt**

   With the File_prompt, if a name is entered without a dot ending (e.g. fred and not fred.csv say) then the ending after the dot in the **wild_card_key** is automatically added to the name.

   For example, if **wild_card_key** = "*.rpt" and "fred" is type in as the file name, then **ret** will be returned as **ret** = "fred.rpt".

12.3 Checking if a File Exists

Looking at the documentation on using the “w” flag to open a file, it say:

w opens a file for writing. If the files exists, its current contents are destroyed.

So unless you want the contents of the file destroyed, it is a good idea to check that the file exists before opening the file for writing.

To check if a file exist, we use the function:

Integer File_exists(Text file_name)

Note that *File_exists* returns a ***non-zero*** value if the file exists. Why?

12.3.1 Exercise 5

Modify program *eight.4dm* so that it only writes information out to a ***new*** file.

13.0 Reading a Text File

Text files, both ANSI (ASCII) or UNICODE, can be ***read*** as well as written via 12dPL functions.

To read a file, three 12dPL functions are required.

(a) Open a Text File for Reading

Integer File_open(Text file_name, “r”,””, File &file)

to read a text file with ANSI encoding (ASCII)

Opening a file accesses the file and returns a **handle** to the file of variable type **File**.

Note that if the file already exists and it has a BOM (Byte Order Mark), the Unicode coding specified by the BOM takes precedence over that specified by the ccs flag. The ccs encoding is only used when no BOM is present or the file is a new file.

For all the File_open choices, see File_open(Text file_name,Text mode,Text ccs_text,File &file) .

(b) Reading from a File

Integer File_read_line(File file,Text &text_in)

This is used to read data from the file, line by line.

and finally

(c) Close a File

Integer File_close(File file)

The file must be closed once reading has been finished. If a file is not closed, then other processes will not be able to access the file.

13.1 What to Do with the Line Read from a File

We now have a line of information read from the file but what can we do with it?

Unlike writing a file, to do anything sensible with the information in the file, you need to know how that information in the file is structured. What you think the data represents may not be correct.

For example the text “ 1235.235436235781” could represent the real number “1235.235436235781” but it is possible the data was written to the file to a specification that states that starting form the beginning of the line, that each 10 characters (including spaces) is a separate number. It would then represents two numbers: ”1235.23” and “5436235781” (there were three spaces before the first “1”). This is not unusual and is known as a ***fixed format.***

And if the numbers had to be Integers only (whole numbers) then the first number is invalid.

Text Conversion functions are used to covert a Text into items such as Integers and Reals, and also for the reverse process, to convert Integers and Reals into Text.

To start with, we will break the line of text into individual ***words*** where a ***word*** is defined as the grouping of one or more non-blank characters between blank characters.

For example, in

This is an example

there are four words “This”, “is”, “an” and “example”. Notice that there can be more than one space separating the words.

The function

Integer From_text(Text text,Dynamic_Text &dtext)

breaks a Text into separate words and returns the individual words in a Dynamic_Text.

13.2 Reading a Text File

We'll now look at Example 4 which opens an existing file, reads it in line by line and counts the number of words that are separated by spaces.

Example 4

```
void main()
{
  Text file_name; File file;

  while (1) {
    File_prompt("Enter the file name","*.rpt",file_name);
    if(!File_exists(file_name)) continue;
    File_open(file_name,"r","ccs=UNICODE",file);
    break;
  }
  Integer eof,count = 0 word_count = 0;
  Text line;

  while(1) {
    if(File_read_line(file,line)!= 0) break;
    ++count;

// break line into words
    Dynamic_Text words;
    Integer no_words = From_text(line,words);
    word_count = word_count + no_words;//
//                        this could be written as word_count +=no_words
    Get_number_of_items(words,no_words);
    for(Integer i=1;i<=no_words;i++) {
      Text t;
      Get_item(words,i,t);
      Print(t); Print();
    }
  }
  File_close(file);

// display the number of lines and words read
  Text out;
  out = To_text(count)+" lines & " +To_text(word_count) + "words read";
  Prompt(out); Print(out);
  Print("\nMacro finished\n"); // write to the Output Window
```

this is always true so the while loop would continue forever unless a break or goto transfers control

break out of the while loop

13.2.1 Exercise 6

Compile **Example 4** and then run it on the file produced by ***eight.4dm***.

What is strange about the results?

Why it is so?

What can be done about it?

Can you modify *Example 4* so the break up into words is correct?

13.3 Using a Clipboard

Text data can be written to and read from the Windows clipboard using the following 12dPL functions.

Integer Console_to_clipboard() ;

Integer Set_clipboard_text(Text txt) ;

Integer Get_clipboard_text(Text &txt) ;

13.4 Binary Files

We have only been reading and writing text files but it is also possible to read and write binary files which contain Real, Integer and Text variables, and Real and Integer arrays.

Reading and writing binary files will not be covered in this course.

14.0 Creating User Defined Functions

As well as the *main* function, and 12dPL supplied functions, a program file can also contain ***user defined*** functions.

User defined functions allow re-use of code and generally make programs easier to follow.

Like the *main* function, *user defined functions* consist of a header followed by the program code enclosed in braces. However the header for a user defined function must include a ***return type*** for the function and the ***order*** and ***variable types*** for each of the ***parameters of the function.***

Hence each user defined function definition has the form

return-type function-name(argument declarations)

```
{
    declarations and statements
}
```

User defined function names must start with an alphabetic character and can consist of upper and/ or lower case alphabetic characters, numbers and underscores (_). There is no restriction on the length of user defined function names. User defined function names are case sensitive.

User defined function names cannot be the same as any of the 12dPL keywords or variable names in the program, or any of the 12dPL supplied functions.

User defined functions must occur in the file before they are used in the program file unless a ***Function Prototype*** is included before the function is used. If this occurs then the user defined function can be defined anywhere in the file. *See* Function Prototypes .

For more information, see User Defined Functions .

14.1 A Simple User Defined Function Example

In Example 5a , the code to check if a file exist, creating the file and writing information to the file is repeated in two places - once with file_name = ”test_unicode.rpt and file_type = “ccs=UNICODE”, and the other time with file_name = ”test_ansi.rpt and file_type = “”.

```
if(File_exists(file_name)) File_delete(file_name);
File_open(file_name,"w",file_type,file);
File_tell(file,file_start);     // record the beginning of the file
File_write_line(file,"one line");
Print("File <"+file_name+"> Start pos = "+To_text(file_start)+"\n");
File_close(file);
```

And if we wanted to also create two extra files with file_type = “ccs=UFT-8” and file_type = “ccs=UFT-16LE”, then the piece of code would repeated two more times.

This is the perfect situation for creating a ***user defined function***.

The information that changes is the file_name and the file_type so they would need to be passed as arguments to the user defined function. There is no information that needs to be returned.

So we’ll define a user defined function called *create_new_file* which has two Text arguments:

```
Integer create_new_file(Text file_name,Text file_type)
{
  File file;
  Integer file_start,file_end;

  if(File_exists(file_name)) File_delete(file_name);
  File_open(file_name,"w",file_type,file);
  File_tell(file,file_start);     // record the beginning of the file
  File_write_line(file,"one line");
  File_tell(file,file_end);     // record after writing a line
  Print("File <" + file_name + "> Start pos = " + To_text(file_start) +
          " End pos = " + To_text(file_end) + "\n");
  File_close(file);
  return(0);
}
```

function return type

function arguments

return with this function return value

We'll now use this function and rewrite Example 5a to give:

```
Integer create_new_file(Text file_name,Text file_type)
{
  File file;
  Integer file_start,file_end;

  if(File_exists(file_name)) File_delete(file_name);
  File_open(file_name,"w",file_type,file);
  File_tell(file,file_start);     // record the beginning of the file
  File_write_line(file,"one line");
  File_tell(file,file_end);     // record after writing a line
  Print("File <" + file_name + "> Start pos = " + To_text(file_start) +
          " End pos = " + To_text(file_end) + "\n");
  File_close(file);
  return(0);
}
void main()
{
  Clear_console();

  create_new_file("test_unicode.4dm","ccs=UNICODE");
  create_new_file("test_ansi.4dm","");

  Print("\nMacro finished\n"); // write to the Output Window
}
```

14.1.1 Exercise 7

Modify this example so it also creates a file with file_name = "test_utf_8.rpt and file_type = "ccs=UTF-8", and a fourth file file_name = "test_utf_16.rpt and file_type = "ccs=UTF-16LE".

If you get stuck, see Example 5b .

14.1.2 Exercise 8

For program ***eight.4dm***, create a function called

Integer write_out_model(Model model,File file)

that does the writing out of the data to the file, up to and including closing the file. Is is assumed that the handles to the Model and the File have already been created and are passed as arguments to the user defined function.

If you get stuck, see Exercise_8.4dm .

Notice that in the *User Defined Function* write_out_model, the variable names can be different from what they were in ***eight.4dm***.

COURSE NOTES

12d Model Programming Language

15.0 User Menus, User Defined Function Keys and Toolbars

12dPL programs can be added to the ***12d Model*** *User* menus, toolbars and also hooked to function keys. The best place to put such 12dPL programs is in the *User_Lib* folder.

(a) 12dPL Programs on ***12d Model*** User menus

To add 12dPL programs to the 12d Model User menu, you need to add the entries to the ***usermenu.4d*** file which is the *User* folder. Unless someone has already added 12dPL programs to Usermenu.4d, you will need to create it for the first time.

An example of an entry in *usermenu.4d* is.

```
Menu "User Reports" {
  Button "Info on strings in model" {
  Command "macro   -close_on_exit   $USER_LIB/Exercise_8.4do"
  }
}
```

Exercise_8.4do must then be in *User_Lib*.

The menu name (`"User String Create"` in the example) must correspond to the name on the top of the 12d Model User menu that you wish to attach your program to.

The other macro options that can be used with, or in place of, ***-close_on_exit*** are:

-no_console	// don't display macro console
-close_on_exit	// remove console when macro terminates
-buttons	// have buttons for finish, restart and quit on console
-allow_defaults	// allow default answers for console questions

The default when there are no macro options is to run the macro with a console but without buttons, and to leave the macro console on the screen when the macro terminates.

Buttons and sub menus may also be created and the syntax is given in the ***12d Reference*** manual.

A good example to look at is the 12d supplied file ***xtramenu.4d*** which is in the folder Set_ups.

Important Notes

1. the entire command `"macro   -close_on_exit   $USER_LIB/Exercise_8.4do"` has quotes around it.

2. usermenu.4d is only read in when a ***12d Model*** project is opened so if your project is already open, you need to do a ***Project =>Restart*** to see the results of any changes to *usermenu.4d*.

(b) 12dPL Programs on User Defined Function Keys

To add 12dPL programs to user defined function keys, you need to add the entries to the ***userkeys.4d*** file which, if it has not been added to, is in Set_Ups, or if it has been modified, should be in User. If you add to the *userkeys.4d* file, place the modified *userkeys.4d* file in User.

An example of an entry in *userkeys.4d* is:

```
shift f5 macro   -close_on_exit   $USER_LIB/Exercise_8.4do
```

Exercise_8.4do must then be in *User_Lib*.

The other macro options that can be used with, or in place of, ***-close_on_exit*** are:

-no_console // don't display macro console

-close_on_exit	// remove console when macro terminates
-buttons	// have buttons for finish, restart and quit on console
-allow_defaults	// allow default answers for console questions

The default when there are no macro options is to run the macro with a console but without buttons, and to leave the macro console on the screen when the macro terminates.

Important Notes

1. unlike in the User Menus, `macro -close_on_exit $USER_LIB/Exercise_8.4do` does not have quotes around it.

2. userkeys.4d is only read in when a ***12d Model*** project is opened so if your project is already open, you need to do a ***Project =>Restart*** to see the results of any changes to *userkeys.4d*.

(c) 12dPL Programs on User Defined Toolbars

To add 12dPL programs to user defined toolbars, you need to add the entries to the ***user_toolbars.4d*** file in the folder *User*. If the file *user_toolbars.4d* does not exist, then create it.

```
Toolbar "User Reports" {
  Button "Info on strings in model" {
  Command "macro  -close_on_exit   $USER_LIB/Exercise_8.4do"
  Icon "Tin_Contour.bmp"
  }
}
```

Exercise_8.4do must then be in *User_Lib*.

Obviously the icon Tin_Contour.bmp is not the correct one and you would need to create a suitable icon for the option. If the `Icon` line in missing, then there will just be a black square in its place on the toolbar.

The other macro options that can be used with, or in place of, ***-close_on_exit*** are:

-no_console	// don't display macro console
-close_on_exit	// remove console when macro terminates
-buttons	// have buttons for finish, restart and quit on console
-allow_defaults	// allow default answers for console questions

The default when there are no macro options is to run the macro with a console but without buttons, and to leave the macro console on the screen when the macro terminates.

Toolbar Flyouts may also be created and the syntax for them is given in the ***12d Reference*** manual.

A good example to look at is the 12d supplied file ***toolbars.4d*** which is in the folder Set_ups. In that file you will see that user_toolbars.4d has been included in toolbars.4d with the command #include_silent "user_toolbars.4d".

Important Notes

1. the entire command `"macro -close_on_exit $USER_LIB/Exercise_8.4do"` has quotes around it.

2. user_toolbars.4d is only read in when a ***12d Model*** project is opened so if your project is already open, you need to do a ***Project =>Restart*** to see the results of any changes to *user_toolbars.4d*.

COURSE NOTES

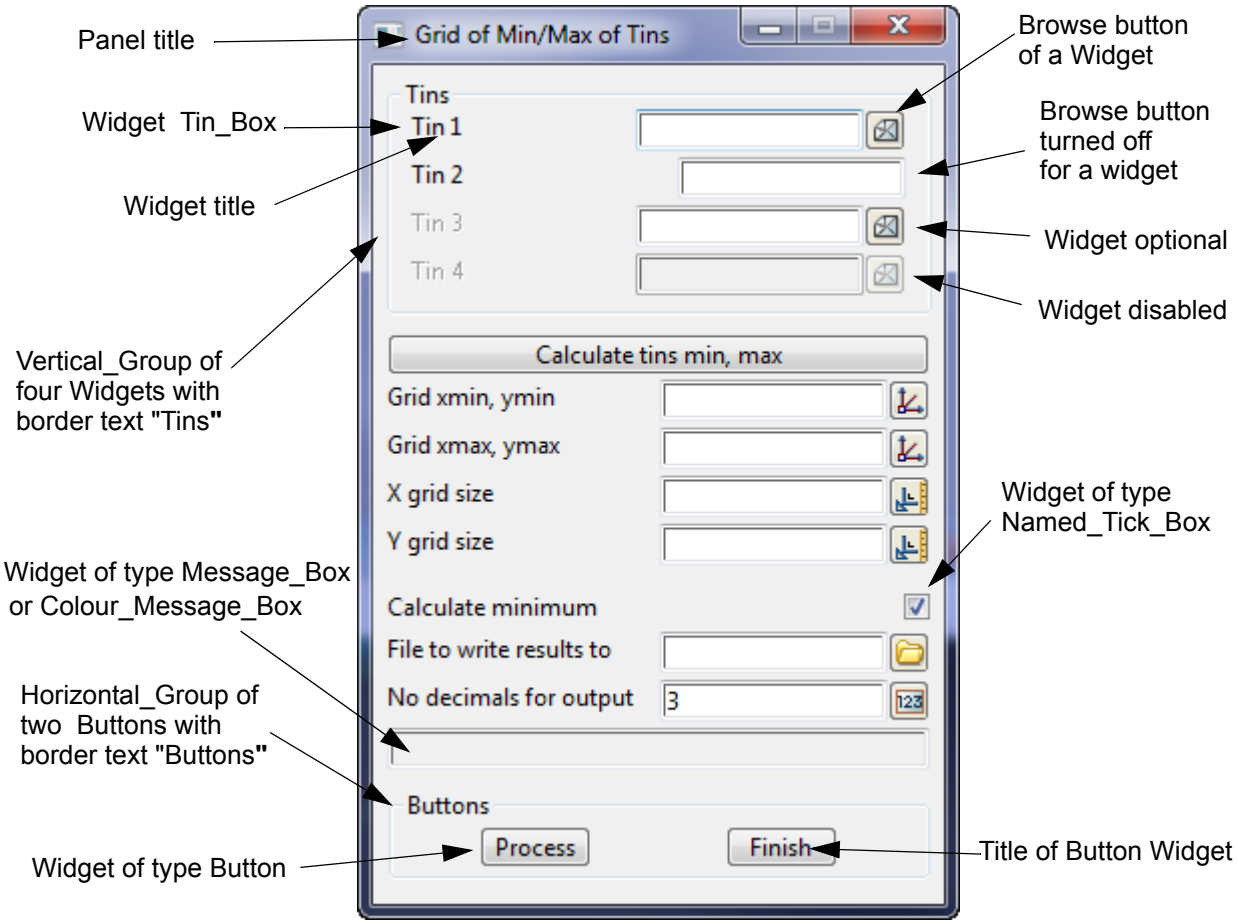
12d Model Programming Language

16.0 Panel Basics

So far all the examples have used the Macro Console and hence have been of a sequential nature. That is, the user is only asked for one thing at a time.

We will now look at building and using Panels in 12dPL that replicates the look and feel, and much of the functionality, of standard *12d Model* panels.

Panels consist of zero or more items called Widgets. And Widgets include such things as panel fields, message boxes and buttons.



The user can usually type/enter/push things in any order on the Panel. That is, it is **event driven**. This makes life much more complicated because you have to program to catch everything that a user may do. And I mean everything.

The basic structure of ***12d Panel*** code is as follows.

(a) Create and display the panel

(b) Create a loop that monitors events for the panel - this is usually a while loop.

(c) Process each event as it occurs.

For example, an event may be clicking on a Button.

A switch statement is regularly used in the event monitoring.

(d) Hopefully there is an event that terminates the program.

COURSE NOTES

12d Model Programming Language

The easiest way to learn to code and work with Panels is to look at some simple examples and build up from there.

16.1 Creating and Displaying a Panel

Panel is a variable type in 12dPL and an individual panel is create by the call

Panel Create_panel(Text title_text)

So in your code you would have say:

```
Panel  panel = Create_panel("Training Panel");
```

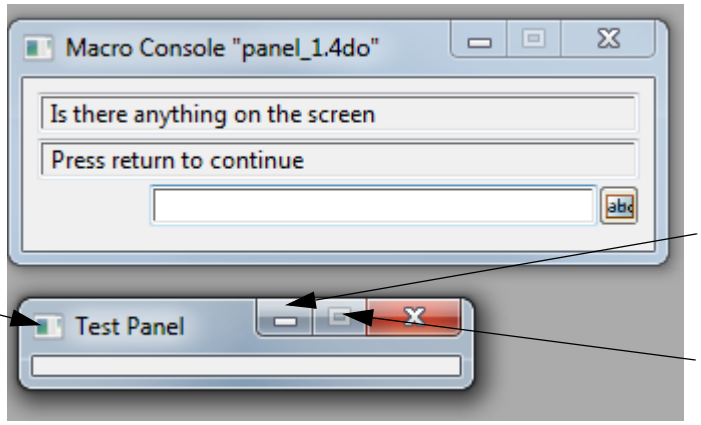
Note that this does not show a panel, it just defines an object that is a Panel. To display the panel, we use the call

Integer Show_widget(Widget widget)

So type in and run this small program to define and display a Panel with the title "Test Panel".

panel_1.4dm

```
void main()
{
  Panel panel = Create_panel("Test Panel");
  Show_widget(panel);

  Error_prompt("Is there anything on the screen");
}
```



Not an exciting program but it shows how to create and display a panel. The minimise, restore and Windows buttons work but that is all. Everything else in a panel has to be controlled by the program, even the **X** on the panel.

If the *Error_prompt* call was missing, the panel would be displayed but then removed when the program finished and it would have been so fast that you wouldn't have seen it.

16.2 Adding Widgets to the Panel

There are many different **Widgets** we can add to a panel and which ones we use depends on the what the application.

For example, we usually want a **Message_Box** so that we can write messages out to the panel (see Create_message_box(Text message_text)).

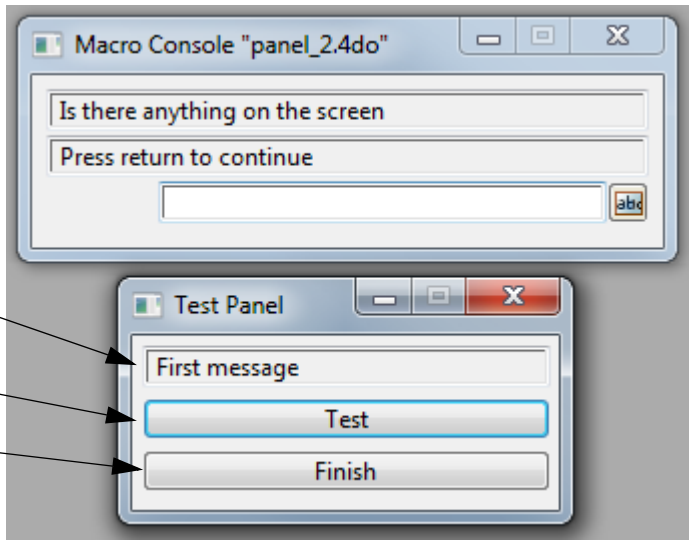
A **Finish** button is useful (see Create_finish_button(Text title_text,Text reply)) and we'll also add a **Button** with the name "Test" (see Create_button(Text title_text,Text reply)).

The order that things must be done is that the **Panel** and **Widgets** are created (and then the **Widgets** are added to the **Panel** using the Append(Widget widget,Panel panel) call.

The creation order for the **Panel** and **Widgets** is not important but the **Panel** must be created before any **Widgets** are appended to it. The order of the Widgets in the **Panel** is the order that they are appended to the **Panel**.

panel_2.4dm

```
void main()
{
  Panel panel = Create_panel("Test Panel");

  Message_Box msg_box = Create_message_box("First message");
  Button finish_button,test_button;

  test_button   = Create_button("Test","test_reply");
  finish_button = Create_finish_button("Finish","finish_reply");

  Append(msg_box,panel);
  Append(test_button,panel);
  Append(finish_button,panel);

  Show_widget(panel);
  Error_prompt("Is there anything on the screen");
}
```



Message_Box appended first

Button appended second

Finish Button appended third

16.3 Monitoring Events in the Panel

The next step is to start monitoring and then acting on the events in the Panel. For example, monitoring that the **Finish** button was clicked on, and then terminating the program.

The function that monitors events in a panel is

Integer Wait_on_widgets(Integer &id,Text &cmd,Text &msg)

and when the user activates a Widget displayed on the screen (for example by clicking on a Button Widget), the **id**, **cmd** and **msg** from the Widget is passed back to *Wait_on_widgets*.

id is the id of the Widget that has been activated - this is a unique number set by 12d Model when the Widget is created.

cmd is the command text that is returned from the Widget - this is dependent on the type of Widget.

msg is the message text that is returned from the Widget - this is dependent on the type of Widget.

For example, for a Button and a Finish Button, pressing and releasing LB or RB whilst highlighting the Button send the Text ***reply*** (set by the programmer when creating the Button) as ***cmd*** with nothing in ***msg***. Pressing and releasing MB does nothing.

To monitor *Wait_on_widgets*, we put the call inside a **while** loop and then *test* the values *id*, *cmd* and *msg* returned by *Wait_on_widgets*.

For example, a snippet of code to monitor a Panel is

```
Integer doit = 1;
while(doit) {
  Integer id;
  Text cmd,msg;

  Integer ret = Wait_on_widgets(id,cmd,msg);

// Process events from any of the Widgets on the panel
// somewhere in here doit must be set to 0
// or a jump made to outside the loop
// or the while loop will go on forever

  }
```

16.4 Events Produced by a Panel

What sort of events are monitored by *Wait_on_widgets*?

One easy way to find out is to put Print statements inside the **while** loop and print out the values of **id**, **cmd** and **msg** returned by *Wait_on_widgets*.

macro nine.4dm

```
void main()
{
  Panel panel = Create_panel("Test Panel");

  Message_Box msg_box = Create_message_box("First message");
  Button finish_button,test_button;

  test_button   = Create_button("Test","test_reply");
  finish_button = Create_finish_button("Finish","finish_reply");

  Append(msg_box,panel);
  Append(test_button,panel);
  Append(finish_button,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1;

  while(doit) {
    Integer id;
    Text cmd,msg;

    Integer ret = Wait_on_widgets(id,cmd,msg);

// Process events from any of the Widgets on the panel

    Print("id= " + To_text(id));
    Print(" cmd=<" + cmd + ">");
    Print("msg=<" + msg + ">\n");
  }
}
```

Type in the code for ***nine.4dm***, compile and run the program.

Click and press on the widgets in *Test Panel* and see what messages are written to the Output Window.

Note in particular what happens when you click on the **X** on the top right hand corner of the panel, and also when you click on the *Test* and *Finish* buttons.

You will also notice that there is no *Macro Console* panel (because we made no *Macro Console* calls) and also that the program will not stop. It is in an infinite loop.

Luckily the **while** loop is sitting waiting for events so whilst it is waiting, we can go and start other ***12d Model*** options. So we can get to the option

Utilities =>Macro =>Kill

to kill the program ***nine.4do*** (see Killing a 12dPL Program).

16.5 Processing Events from a Panel

The final step is to start processing the events returned from a panel.

What events we look for and how we process it of course depends on the purpose of the program.

From running program ***nine.4do***, you will have noticed that clicking on **X** returns with

cmd = "Panel Quit"

So testing for **cmd** equal to "Panel Quit" would give us a way to trap the **X** and end the program.

Looking further at the messages produced by ***nine.4do***, clicking on the **Finish** buttons returns with

cmd = "finish_reply"

which is the Text **reply** we set when creating the **Finish** button.

Similarly clicking on the **Test** button returns with **cmd** = "test_reply" which is the reply we set for that button.

Also note that the **id** that is returned is always the same for the same **Widget**. That is, clicking on **X** always returns the same id and it is different from the **id** you get when sicking on **Test** or **Finish**.

This is because every Widget is given a unique **id** when it is created.

And there is function to get the **id** for a Widget.

Integer Get_id(Widget widget)

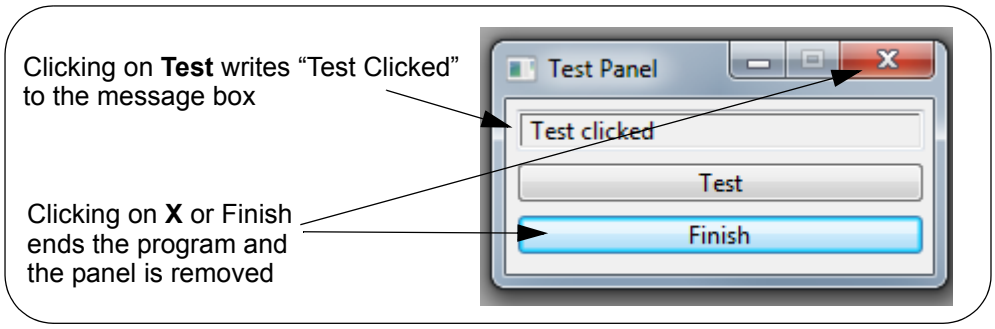
The Integer function return value is the **id** of the **Widget**.

We will now modify ***nine.4dm*** so that it

(a) Ends the program if **X** is clicked.

(b) Ends the program if **Finish** is clicked.

(c) Writes the message "Test clicked" to the Message_Box when **Test** is clicked.



Compile, run and test the program ***ten.4dm***.

COURSE NOTES

12d Model Programming Language

macro ten.4dm

```
void main()
{
Panel panel = Create_panel("Test Panel");

  Message_Box msg_box = Create_message_box("First message");
  Button finish_button,test_button;

  test_button   = Create_button("Test","test_reply");
  finish_button = Create_finish_button("Finish","finish_reply");

  Append(msg_box,panel);
  Append(test_button,panel);
  Append(finish_button,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1;

  while(doit) {
    Integer id;
    Text cmd,msg;

    Integer ret = Wait_on_widgets(id,cmd,msg);

// Process events from any of the Widgets on the panel

    Print("id= "+To_text(id)+" cmd=<"+cmd+"> msg=<"+msg+">\n");

    switch(id) {
      case Get_id(panel): {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button): {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(test_button): {
        Set_data(msg_box,"Test clicked");
        break;
      }
    }
  }
}
```

get the id of the Widget

16.6 Set_Ups.h and #include

In our earlier program ***eight.4dm*** and its rewrite using a user defined function Exercise_8.4dm , we selected a model and then wrote out information about all the elements in the model to a file. We used a *Model_prompt* and a File_prompt (see Writing 12d Model Data to a Text File).

We will now write a program similar to *eight.4dm* but using a **Panel** instead of a *Macro_Console.* So we need the equivalent of a Model_prompt and a File_prompt for a panel, and they are the Widgets **Model_Box** and **File_Box**.

We will first look at how to create a *Model_Box* and a *File_Box* but that gives us no clue as how to use them in a panel, and how use them when it is time to write out the information on elements in the model out to a file.

So after learning how to create a Model_Box and a File_Box, we will build a panel containing them and a **Write** button, and finally look at processing the evens inside the panel and writing the data out to a file.

16.6.1 Creating a Model_Box

Create_model_box(Text title_text,Message_Box message,Integer mode)

Name

Model_Box Create_model_box(Text title_text,Message_Box message,Integer mode)

Description

Create an input Widget of type **Model_Box** for inputting and validating Models.

The **Model_Box** is created with the title **title_text** (*see* Model_Box).

The Message_Box **message** is normally the message box for the panel and is used to display Model_Box validation messages.

If <enter> is typed into the Model_Box automatic validation is performed by the Model_Box according to **mode**. What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

For example,

```
CHECK_MODEL_MUST_EXIST     7   // if the model exists, the message says "exists".
                               // if it doesn't exist, the messages says "ERROR"
```

The values for **mode** and their actions are listed in Appendix A (*see* Model Mode).

If LB is clicked on the icon at the right hand end of the **Model_Box**, a list of all existing models is placed in a pop-up. If a model is selected from the pop-up (using LB), the model name is placed in the **information area** of the Model_Box and validation performed according to **mode**.

MB for "Same As" also applies. That is, If MB is clicked in the **information area** and then a string from a model on a view is selected, then the name of the model containing the selected string is written to the **information area** and validation performed according to **mode**.

The function return value is the created **Model_Box**.

Special Note:

#include "set_ups.h" must be in the macro code to define CHECK_MODEL_MUST_EXIST etc.

ID = 848

Notice that the *Create_model_box* requires a **Message_Box** - this is where error and other messages generated by the Model_Box are written to. So a *Message_Box* must be created

COURSE NOTES

12d Model Programming Language

BEFORE we create the *Model_Box*.

Also *Create_model_box* has a Integer **mode** and the value of **mode** determines the behaviour of the *Model_Box*. In the description for *Create_model_box* there is the example of **mode** = CHECK_MODEL_MUST_EXIST and this mode means you get an error message written to the Message_Box if the model does not exist.

CHECK_MODEL_MUST_EXIST has the value 7 but where is that defined?

CHECK_MODEL_EXIST and its value 7 is defined in a file called ***Set_ups.h*** and the file is put in the folder *Set_Ups* when ***12d Model*** is installed on your computer.

To include definitions such as CHECK_MODEL_EXISTS in the program without having to type it all in, we use the **#include** preprocessing command.

The command **#include**

```
#include "file_name"
```

in the program code tells the compile to include the "file_name" in the program code **before** the compile takes place (see Preprocessing).

Important Note

For any files mentioned in the #include preprocessing command, *12dPL* looks locally but also in the folder ***User*** and then ***Set_Ups*** for the file so all you need is the program is

```
#include "set_ups.h"
```

After looking at creating the File_Box and building the panel, we'll then look at Validating and getting information out of the Model_Box.

16.6.2 Creating a File_Box

Create_file_box(Text title_text,Message_Box message,Integer mode,Text wild)

Name

File_Box Create_file_box(Text title_text,Message_Box message,Integer mode,Text wild)

Description

Create an input Widget of type **File_Box** for inputting and validating files.

The **File_Box** is created with the title **title_text** (*see* File_Box).

The Message_Box **message** is normally the message box for the panel and is used to display File_Box validation messages.

If <enter> is typed into the File_Box, automatic validation is performed by the File_Box according to **mode**. What the validation is, what messages are written to Message_Box, and what actions automatically occur, depend on the value of **mode**.

For example,

```
CHECK_FILE_NEW        20   // if the file doesn't exists, the message says "will be created"
                           // if it exist, the messages says "ERROR"
```

The values for **mode** and their actions are listed in Appendix A (*see* File Mode).

If LB is clicked on the icon at the right hand end of the **File_Box**, a list of the files in the current area which match the wild card text **wild** (for example, *.dat) Is placed in a pop-up. If a file is selected from the pop-up (using LB), the file name is placed in the **information area** of the File_Box and validation performed according to **mode**.

The function return value is the created **File_Box**.

Special Note:

#include "set_ups.h" must be in the macro code to define CHECK_FILE_NEW etc.

ID = 906

The first thing you will notice is that the description for *Create_file_box* is very similar to *Create_model_box*.

Again there is a *Message_Box* which must be created BEFORE we create the *File_Box*.

There is also a **mode** and *set_ups.h* must again be included for CHECK_FILE_NEW etc to be valid but you only need include *set_ups.h* once.

16.6.3 More Events from Wait_on_widgets

16.6.4 Exercise 9

Start with program ***ten.4dm*** and make a copy as ***eleven.4dm.***

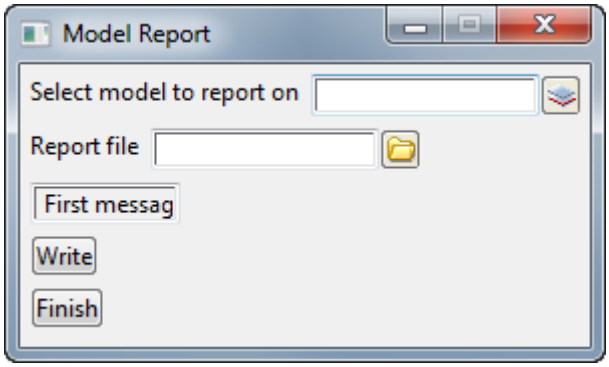
Add a *Model_Box* and a *File_Box* to the panel in the program *eleven.4dm*.

Change the name of the panel to “Model Report”. Also change the button labelled “Test” to the label “Write” and give it the reply “write_reply”.

Compile and run ***eleven.4dm***.

Click and press on the widgets in the panel *“Model Report”* and see what messages are written to the Output Window. In particular, type some text into the Model_Box and File_Box.

Once you get all the compile errors out), you will get something like



If you are having any problems, see Eleven_1.4dm .

A few things to note are:

1. **Strange sizes for Model_Box, File_Box and Message_Box**

   Nowhere in the definition of the Model_Box, File_Box and Message_Box was there a parameter to give the size of each box. Instead ***12d Model*** automatically sizes each box for you. This is done because any hard wired sizes would not respond to changing screen resolution or screen font sizes.

   However the above widths and layout of the Boxes is not ideal, and we shortly look at using Horizontal and Vertical Groups to control the panel layout.

2. **Typing into the Model_Box**

   When you type “a” into the Model_Box, the message printed to the Output Window is:

   id= 131275432 cmd=<keystroke> msg=<a>

   In fact, just clicking in and typing in the Model_Box creates a steady stream events returned by *Wait_on_widgets.*

   ```
   id= 131275432 cmd=<left_button_up> msg=<>
   id= 131275432 cmd=<keystroke> msg=<a>
   id= 131275432 cmd=<keystroke> msg=< >
   id= 131275432 cmd=<keystroke> msg=<f>
   id= 131275432 cmd=<keystroke> msg=<i>
   id= 131275432 cmd=<keystroke> msg=<
   >
   id= 131275432 cmd=<model selected> msg=<a fi>
   id= 131275432 cmd=<kill_focus> msg=<>
   id= 131315488 cmd=<set_focus> msg=<>
   id= 131315488 cmd=<kill_focus> msg=<>
   ```

   None of these events are currently checked for inside the **while** loop but they could be checked for and acted upon if there was a need.

3. **Write Button**

   In our program the **Write** button is going to be the trigger for the user to say that the panel has been filled in and it is time to write out the report. That is, processing is only done when **Write** is pressed.

16.7 Horizontal and Vertical Groups

Before looking at how we make the program do the work when the **Write** button is pressed, we'll first get the panel looking better.

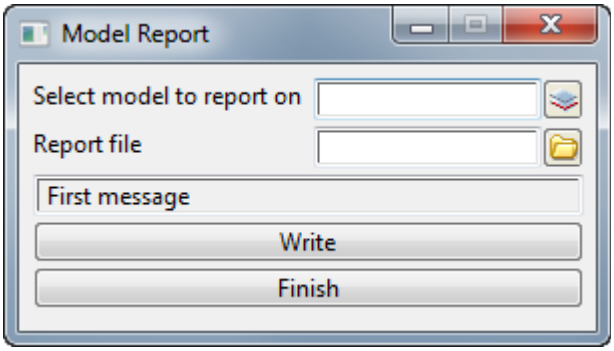
Nowhere in the definition of the Boxes and Buttons were there parameters giving the size of each Widget. Instead ***12d Model*** automatically sizes things for you.This is done because any hard wired sizes would not respond to changing screen resolution or screen font sizes.

To size and set out the Widgets in the panel the way we want them, before adding the Widgets to the panel we place them in Horizontal_Groups or Vertical_Groups to control the sizing and positioning algorithms for the Widgets.

Working from the top, we would like to Model_Box, File_Box and Message_Box to be the same widths. To do that, we first add them into a Vertical Group before adding them to the panel.

```
Vertical_Group vgroup = Create_vertical_group(0);

  Append(model_box,vgroup);
  Append(file_box,vgroup);
  Append(message_box,vgroup);
  Append(write_button,vgroup);
  Append(finish_button,vgroup);
  Append(vgroup,panel);
```

This will give you

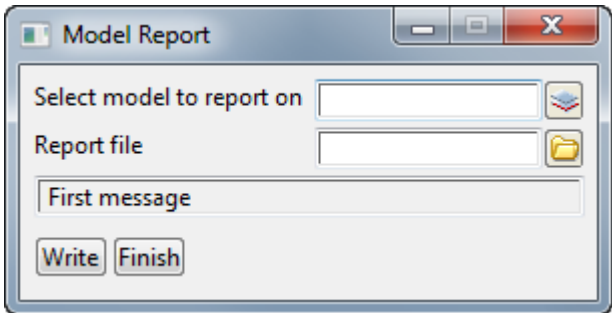


This is fine if you want very wide **Write** and **Finish** buttons but normally we like to have them on the same line. For this we will use a Horizontal Group .

So we'll take the Write and Finish buttons out of the Vertical_Group and add them to a Horizontal_Group, and then add the Horizontal_Group to the panel.

```
Vertical_Group vgroup = Create_vertical_group(0);

  Append(model_box,vgroup);
  Append(file_box,vgroup);
  Append(message_box,vgroup);
  Append(vgroup,panel);

  Horizontal_Group hgroup = Create_button_group();

  Append(write_button,hgroup);
  Append(finish_button,hgroup);
  Append(hgroup,panel);
```

This will give you

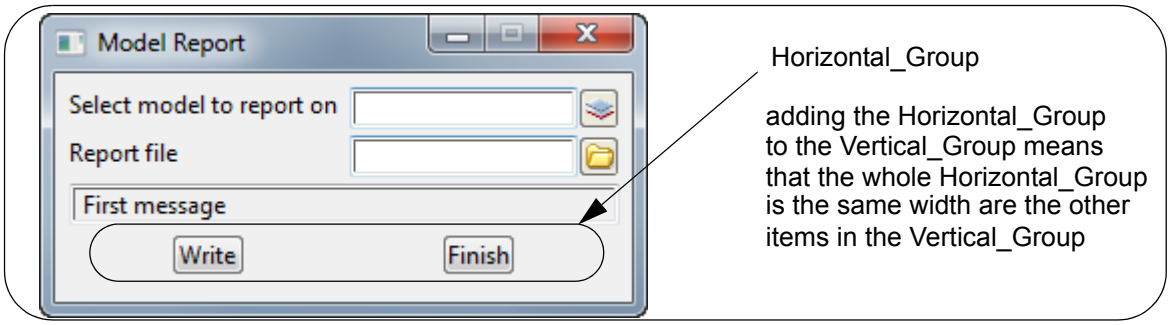


Close, but not the best look.

What we really want is the line containing the Write and Finish buttons to be as wide as the previous three lines. So we want the Horizontal_Group to be sized width wise, the same as the first three Widgets.

So to do that, we simply add the Horizontal_Group containing the **Write** and **Finish** buttons, to the Vertical_Group rather than straight to the panel. That way the Horizontal_Group will be given the same width as the other widgets in the Vertical Group, but unlike before, it is the entire Horizontal_Group and not the individual buttons that is given the width.

```
Vertical_Group vgroup = Create_vertical_group(0);
Append(model_box,vgroup);
Append(file_box,vgroup);
Append(message_box,vgroup);

Horizontal_Group hgroup = Create_button_group();
Append(write_button,hgroup);
Append(finish_button,hgroup);
Append(hgroup,vgroup);

Append(vgroup,panel);
```



16.7.1 Exercise 10

Compile and test your ***eleven.4dm*** code to make user you get the above panel (see Eleven_2.4dm if are having problems).

Although at first it may appear confusing, once you have used Horizontal and Vertical Groups are couple of times it becomes easy and creates good looking panels without you having to do any sizing calculations.

Now that you have a larger Message_Box, test the panel to see what messages you get in the Message_Box and again what events are monitored by *Wait_on_widgets*.

16.8 Validating Boxes and Buttons

16.8.1 Model_Box Events

Typing characters into the *Model_Box* creates Widget events but these can be ignored. The important event to track is when the <Enter> key is pressed, or a model is selected from the pop-up list.

In both these cases for the Widget event, **cmd** = “model selected” and **msg** is the model name.

id= 131275432 cmd=<model selected> msg=<boundary>

What messages are written to the *Message_Box* depends on the **mode** set when the *Model_Box* was created.

So if you wanted to do something special when a name is entered into the *Model_Box*, you only need to check for the **id** of the *Model_Box* in the switch statement, and when that occurs, check for **cmd** equal to “model selected”

Otherwise you can simply ignore the events for the *Model_Box*.

Note that although the *Model_Box* is right there in front of user in the panel, at this stage there is nothing forcing the user to do anything with the *Model_Box*. The user may simply go and click on the **Write** button.

16.8.2 File_Box Events

Typing characters into the *File_Box* also creates many Widget events that can be ignored. The important event to track is when the <Enter> key is pressed, or a file is selected from the pop-up list.

In both these cases for the Widget event, **cmd** = “file selected” and **msg** is the file name.

id= 131077296 cmd=<file selected> msg=<model.rpt>

What messages are written to the *Message_Box* depends on the **mode** set when the *File_Box* was created.

So if you wanted to do something special when a name is entered into the *File_Box*, you only need to check for the **id** of the *File_Box* in the switch statement, and when that occurs, check for **cmd** equal to “file selected”.

Otherwise you can simply ignore the events for the File_Box.

Note that just like the *Model_Box*, the *File_Box* is right there in front of user in the pane but the user may not touch it and just click on the **Write** button.

16.8.3 Write Button

The **Write** button is the trigger to say it is time to write out the report of all the strings in the selected model.

Currently in **eleven.4dm** we are capturing clicking on the **Write** button but all we do is write out the message “Write clicked” to the Message_Box. So we will look at the steps need to replace this with writing the data out to the file.

Looking back at Exercise_8.4dm , we have already extracted the file writing code in ***eight.4dm*** and turned it into the user defined function

Integer write_out_model(Model model,File file)

so we will simply reuse that function so we don’t have to create it again.

But before we can call *write_out_model*, we need to create the handles for model and file.

12d Model Programming Language

Now there is a ***Model_Box*** and a ***File_Box*** in the panel but not only do we NOT know if the user entered anything sensible into ***Model_Box*** or ***File_Box***, we have no idea if the user ever went to the two boxes. So even though in the code we may have checked things when the user clicked on the *Model_Box* and *File_Box*, we still have to ***check everything again*** after the **Write** button is clicked.

This is where panels are different, and a bit trickier and slightly more difficult to code than when using a Macro Console. But the power of panels quickly makes up for the extra development time.

So after the **Write** button is clicked, we have to:

(a) Get the model details from the *Model_Box* and check that it exists otherwise we have no elements to report on. If it doesn't exist we need to write an error message out to the *Message_Box* and stop further processing for the **Write** button.

To do this we use the Validate(Model_Box box,Integer mode,Model &result) call for the *Model_Box* with the **mode** GET_MODEL_ERROR = 13 .

With Validate and this *mode*, if the model exists then the return code is MODEL_EXISTS and the handle to the selected model is returned as the argument Model **result**.

If the model does not exist, then an error message "Error no model specified" is written to the Message_Box and the return code is NO_MODEL.

So by just checking the return code you know if an existing model was selected, or no existing model was selected and so you need to go back ask for an existing model.

So in the **switch** statement in the **while** loop, you would have in the **case Get_id(write_button):**

```
// check that the model exists for the name in the model box
  Model model;

  if(Validate(model_box,GET_MODEL_ERROR,model)!= MODEL_EXISTS)break;
```

This says if the model does not exist (!= MODEL_EXISTS), break out of the **switch** statement to go back and wait for further events with *Wait_on_widgets*.

If the model exists, the we have the handle to it returned as Model **model**.

(b) Get the file details from the *File_Box*.

If the file already exists then the person defining the behaviour of the program needs to tell us what to do.

Do we say it must be a new file and stop further processing for the Write button?

Do we delete the existing file so we write a new file with that name?

Do we append to the end of the existing file?

There are File ***modes*** to help do each of these but we need to know in advance what is required.

For this exercise, the requirements will be that if the file exists, it is alright to let the user say to delete the file, or ask for a new file. We won't allow the user to Append to an existing file.

To do this we use the Validate(File_Box box,Integer mode,Text &result) call for the *File_Box* with the **mode** GET_FILE_CREATE = 15 .

With Validate and this *mode*, if the file does not exist then the return code is NO_FILE and the text in the File_Box is returned in the Text **result**. Note that for the *File_Box*, no file handle was returned but just the file name.

If no text is typed into the File_Box then the return code is NO_NAME

If the file exists, then a *Replace or Cancel* panel is placed on the screen and if *Replace* is selected, then the file is ***deleted*** and the return code is NO_FILE.

If *Cancel* is selected, then the message “overwrite aborted by user” the return code is NO_FILE_ACCESS.

Once again, just checking the return code lets you know that the file doesn’t exist (NO_FILE), or the user cancelled and needs to go back and give another file name (NO_FILE_ACCESS), or nothing was typed into the file box (NO_NAME).

This time the only valid return we are looking for is NO_FILE.

So in the **switch** statement in the **while** loop, have in the **case Get_id(write_button):**

```
// check the file does not exist
  Text result; File file; Integer validate_return;

  validate_return = Validate(file_box,GET_FILE_CREATE,result);

  if(validate_return == NO_FILE){  //file doesn't exist
    File_open(result,"w","ccs=UNICODE",file);   // create the file
  } else {
    Set_data(msg_box,"Choose another file name");
    break;
  }
```

This says that if the file with the name given the *File_Box* does not exist, then it is created.

For anything else, the message “Choose another file” is written to the *Message_Box* and then a **break** out of the **switch** statement goes back to wait for further events with *Wait_on_widgets*.

(c) Write out the information about each element in the model to the file.

If we are still in the **case Get_id(write_button)** for the **switch** statement after the code above then we have an existing model with Model handle **model** and a file to write the data to with the File handle **file**.

The code to then write out the report is simply:

```
  write_out_model(model,file);  // write out data
  Set_data(msg_box,"Data written out");
```

We really should also be checking the function return code for *write_out_model* just in case there was an error in writing out the report. If an error is found, we could then write out an error message like “Error writing out the data to the file ...”.

```
case Get_id(write_button): {
// check that the model exists for the name in the model box
  Model model;
  if(Validate(model_box,GET_MODEL_ERROR,model)!= MODEL_EXISTS) break;

// check that the file does not exist
  Text result; File file; Integer validate_return;
  validate_return = Validate(file_box,GET_FILE_CREATE,result);

  if(validate_return == NO_FILE) {  // file doesn't exist so can create it
    File_open(result,"w","ccs=UNICODE",file);
  } else {
    Set_data(msg_box,"Choose another file");
    break;
  }
  write_out_model(model,file);  // write out data
  Set_data(msg_box,"Data written out");
  break
}
```

if the model does not exist then an error message is written to the message box

if the file exists you are asked if you want to replace it and if yes then the fie is deleted and NO_FILE is returned

16.8.4 Exercise 11

Copy the user defined function *write_out_model* from ***eight.4dm*** and put it into your ***eleven.4dm***, and also the above additions for the switch case *Get_id(write_button)*.

Now compile and test your new ***eleven.4dm*** code. See Eleven_3.4dm if you are having problems.

Try the different combinations of when file does and does not exit and when the model does and does not exist.

16.9 CHECK and GET Modes

In the *Create* and *Validate* calls for the *Model_Box* and *File_Box* there **modes** for controlling and reporting on what the Boxes did (*see* Model Mode *and* File Mode).

The modes used in the *Create* calls determine what automatically happens when you enter information into the created Box (for example, the File_Box) and so they are always used. Whereas you may never use Validate calls in your code.

Some of these modes were CHECK modes and others GET modes.

The major difference between them is that the CHECK modes only *check* things and write messages out to the *Message_Box*.

On the other hand, the GET modes may actually create and even delete things. We saw that with

```
Validate(file_box,GET_FILE_CREATE,result)
```

where using GET_FILE_CREATE allows the user to **delete** an existing file.

Because users may click all over the place in a panel, and may even quit out of the panel without ever pushing a *Process* button (the *Write* button in our eleven.4dm), when creating boxes you should only use the **CHECK** modes.

An example of how problems could arise in ***eleven.4dm*** by using GET_FILE_CREATE mode when creating the *File_Box* (Create_file_box("Report file",msg_box,GET_FILE_CREATE,"*.rpt")) rather than CHECK_FILE_CREATE as we are now doing, is that when the user picks a file in the *File_Box* and the file already exists, the GET_FILE_CREATE means at that time they would be asked about overwriting the file and if they said yes, the file would be deleted. But the user may then do the same thing and delete a number of files before they ever push the **Write** button. Worse still is that they may simply finish the panel and never click on the **Write** button but the files will of course still be deleted.

Although the same problem may occur with *Validates*, *Validates* usually only occur in the associated with *Process* button and so the actual processing is happening.

16.10 Ignored Events

From the information being written to the Output Window after the *Wait_on_widgets* call, you will notice lots of events that we are not processing.

Some of them are general events such as “kill focus”, “set focus”, left_button_up” and others are events such as “model selected” and “file selected” that are generated by the Widgets we placed in the panel.

Currently these events are not being processed in the **while** loop surrounding the *Wait_on_widgets* call but it is good to know they exist in case you do need to use them in future 12dPL programs.

17.0 Working with 12d Model Strings

In Example 1 using a Macro_Console, we selected a string and wrote out how many vertices there were in the string. We will now repeat this but with a Panel.

So we need to be able to select a string and there are two possible Boxes to use - the Select_Box and the New_Select_Box.

Create_select_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Name

Select_Box Create_select_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Description

Create an input Widget of type **Select_Box**.

The Select_Box is created with the title **title_text**.

The Select title displayed in the screen message area is **select_title**.

The value of **mode** is listed in the Appendix A - Select mode. *See* Select Mode.

The Message_Box **message** is normally the message box for the panel and is used to display string select validation messages.

The function return value is the created Select_Box.

ID = 882

Create_new_select_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Name

New_Select_Box Create_new_select_box(Text title_text,Text select_title,Integer mode,Message_Box message)

Description

Create an input Widget of type **New_Select_Box**. *See* New_Select_Box.

The New_Select_Box is created with the title **title_text**.

The Select title displayed in the screen message area is **select_title**.

The value of mode is listed in the Appendix A - Select mode. *See* Select Mode.

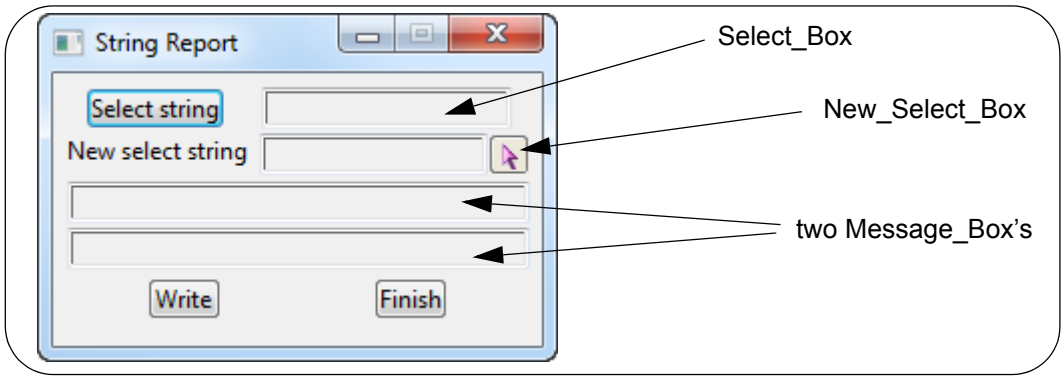
The Message_Box **message** is normally the message box for the panel and is used to display New_Select_Box validation messages.

The function return value is the created New_Select_Box.

ID = 2240

To see the difference between the two Boxes, we'll add them to a Panel. Also we'll use two Message_Box's with the messages going to different Message_Box's.

See Twelve_1.4dm.



Using either Box, we get many new Widget events like “motion event”, “pick select” and “accept select”.

```
id= 200878304 cmd=<motion select> msg=<42496.017276556 37297.013453461 null null null "2">
id= 200878304 cmd=<motion select> msg=<42496.017276556 37297.505793414 null null null "2">
id= 200878304 cmd=<motion select> msg=<42496.017276556 37297.998133367 null null null "2">
id= 200878304 cmd=<motion select> msg=<42496.841013635 37297.188194465 159.284034406 1966.826348328 159.284034406 "2">
id= 200878304 cmd=<pick select> msg=<"2">
id= 200878304 cmd=<motion select> msg=<42496.841013635 37297.188194465 159.284034406 1966.826348328 159.284034406 "2">
id= 200878304 cmd=<accept select> msg=<"2">
```

view name

The “motion select” event occurs after a Select button is activated and then the cursor is over the drawing area of a ***12d Model*** View. Notice that the “motion select” event does not occur when you are over a menu or panel that is covering the drawing area of a View. So the “motion select” only occurs when you are able to pick something in a model on a View.

Create a **Section View** and profile a string and then move over the view with a Select running.

Also create a **Perspective View**, add some data to it and do a Fit, and move over the view with a Select running.

At this stage we are not interested in the “motion select” and it is hard to see what other events are being written to the Output Window so we will stop writing out the “motion select” events. To do this, simply add a test for “motion select” before the Print statement.

```
if(cmd == "motion select") continue;
Print("id= "+To_text(id)+" cmd=<"+cmd+"> msg=<"+msg+">\n");
```

Now use the two selects for cursor picks, and also see what happens when Cancel is chosen from the **Pick Ops** menu (click RB when in the ***12d Model*** View to bring up the **Pick Ops** menu).

17.0.1 Exercise 12

Create a new 12dPL program called ***twelve.4dm*** by modifying ***twelve_1.4dm*** so that there is just the *New_Select_Box*, and when a string is selected, the number of vertices in the string is written out to the message box.

See Example 1b if you are having problems.

17.1 Types of Elements

We have been selecting string but there are more than string Elements. For example, there are Tin, SuperTin, Plot Frame Elements. And even for strings, there is more than one type of string. For example, string types include Super, Arc, Circle, Text, Super_Alignment, Drainage and Pipeline.

Some information is common to all the Element types such as name and colour but other information will depend on the Element type.

The full list of Element types is given in Types of Elements and the type is found by the call Get_type(Element elt,Text &elt_type) .

17.2 Dimensions of a Super String

The Super String is a very general string which was introduced to not only replace the string types 2d, 3d, 4d, interface, face, pipe and polyline, but also to allow for combinations that were never allowed in the old strings. For example, to have a polyline string but with a pipe diameter, or a 2d string with text at each vertex.

Different strings to cover every possible combination would have required hundreds of different string types. A better solution was to have one string type that has information to cover all of the properties of the other strings, and the ability to more easily add other properties now and in the future. This flexible string is the ***Super String***.

Having all possible combinations defined for every Super String would be very inefficient for computer storage and processing speed, so the Super String uses the concept of ***dimensions*** to refer to the different types of information that ***could*** be stored in the Super String.

Each **dimension** is well defined and is also ***optional*** so that no unnecessary information is required to be stored.

A Super String **always** has an (x,y) value for each vertex but what other information exists for a particular Super String depends on what optional dimensions are defined for that Super String.

For example, there are *two* Height dimensions called Att_ZCoord_Value and Att_ZCoord_Array. If Att_ZCoord_Value is set then the super string has a constant height value for the entire string (2d super string), and if Att_ZCoord_Array is set, then there is a z value for each vertex (3d super string). If **both** are set then Att_ZCoord_Array takes precedence.

So the two Height dimensions cover the functionality of both the old 2d string (one height for the entire string) and the old 3d string (different z value at each vertex). Plus the 2d super string only requires the storage of one height like the old 2d string and not the additional storage required for a z value at every vertex that the 3d string needs.

For each super string dimension, there are calls to check if a super string has that dimension set or not set.

Note
If both Att_ZCoord_Array and Att_ZCoord_Value exist then Att_ZCoord_Array takes precedence but it is also possible that NEITHER of them exist.

Get_super_use_2d_level(Element super,Integer &use)

Name

Integer Get_super_use_2d_level(Element super,Integer &use)

Description

Query whether the dimension height dimension Att_ZCoord_Value exists for the super string **super**.

See Height Dimensions for information on Height dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists, or 0 if the dimension doesn't exist.

If the Element **super** is not a super string, then a non zero function return value is returned.

A return value of 0 indicates the function call was successful.

ID = 701

Get_super_use_3d_level(Element super,Integer &use)

Name

Integer Get_super_use_3d_level(Element super,Integer &use)

Description

Query whether the height dimension Att_ZCoord_Array exists for the super string **super**.

See Height Dimensions for information on Height dimensions or Super String Dimensions for information on all dimensions.

use is returned as 1 if the dimension exists, or 0 if the dimension doesn't exist.

If the Element **super** is not a super string, then a non zero function return value is returned.

A return value of 0 indicates the function call was successful.

ID = 731

17.2.1 Exercise 13

Create a new 12dPL program called ***thirteen.4dm*** by modifying ***twelve.4dm*** so that the program not only writes out the number of vertices in the selected string but also writes out if the selected string has dimension Att_ZCoord_Array and if not, does it have the dimension Att_ZCoord_Value.

Contour the tin and then check what dimension the contours have.

What happens when the Super Alignment m001 is selected?

See Thirteen.4dm if you are having problems.

17.3 Accessing (x,y,z) Data for a Super String

There are a number of ways of getting coordinate data from a Super String, but the simplest is the Get_super_vertex_coord(Element super,Integer i,Real &x,Real &y,Real &z)

Get_super_vertex_coord(Element super,Integer i,Real &x,Real &y,Real &z)

Name

Integer Get_super_vertex_coord(Element super,Integer i,Real &x,Real &y,Real &z)

Description

Get the coordinate data (x,y,z) for **i**'th vertex (the vertex with index number **i**) of the super Element **super**.

The x coordinate is returned in Real **x**.
The y coordinate is returned in Real **y**.
The z coordinate is returned in Real **z**.

If the Element **super** is not of type **Super**, then the function return value is set to a non zero value.

A return value of 0 indicates the function call was successful.

ID = 733

So we can simply use Get_points(Element elt,Integer &num_verts) to get the number of vertices in the string and then Get_super_vertex_coord(Element super,Integer i,Real &x,Real &y,Real &z) to the coordinates of any of the string vertices.

17.3.1 Exercise 14

Create a new 12dPL program called ***fourteen.4dm*** by modifying ***thirteen.4dm*** so that it only looks at Super Strings of type 2d and 3d and then

(a) writes out the same information to the message box.

(b) **plus** writes the name and model of the string to the Output Window, followed by the same information as (a) except to the Output Window

(c) **plus** writes out the vertex index and the x,y and z coordinates of the string (one set per line) to the Output Window.

See Fourteen.4dm if you are having problems.

17.4 Changing Element Header Properties

To date we have obtained Element handles to strings so could inquire on string properties such as name, model containing the string and number of vertices. This type of information is often referred to as the *header information* or *header properties* for an Element because such information is common to all Elements. The functions we used to obtain the Element header information were mainly in the section Element Header Functions .

So far we have used Get_name, Get_model, Get_id, Get_type and Get_points but there are other routines such as

Get_colour(Element elt,Integer &colour) to get the Element colour

Get_style(Element elt,Text &elt_style) to get the Element style

Get_chainage(Element elt,Real &start_chain) to get the start chainage of the Element

For most of these functions, there is an equivalent *Set_* call that modifies that Element property. For example Set_name:

Set_name(Element elt,Text elt_name)

Name

Integer Set_name(Element elt,Text elt_name)

Description

Set the name of the Element **elt** to the Text **elt_name**.

A function return value of zero indicates the Element name was successfully set.

Note

This will not set the name of an Element of type Tin.

ID = 45

One exception is Get_points, which returns the number of vertices in an Element, and there is no simple Set_points.

We will now look at the tools required to write a12dPL program that changes the name and the colour of a Super String. But we will add the twist that if either the name or colour is left blank then that property is not changed. So we don't have to supply a name or a colour - that is ***optional***.

In ***12d Model***, optional Boxes are identified by the title text being greyed out but the information area and Browse button are **not** greyed out. And in 12dPL, you can easily do the same thing for most Boxes.

To get the new name and colour, we use a Colour_Box and a Name_Box . And to indicate that they are options, we use the Set_optional(Widget widget,Integer mode) call.

Set_optional(Widget widget,Integer mode)

Name

Integer Set_optional(Widget widget,Integer mode)

Description

Set the optional **mode** for the Widget **widget**.

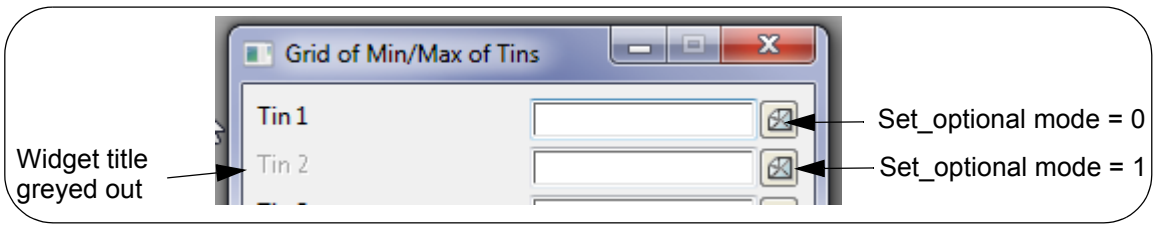
That is, if the Widget field is blank, the title text to the left is greyed out, signifying that this Widget is optional.

If **mode** = 1 the widget is optional
mode = 0 the widget is not optional.

The default value for a Widget is mode = 0.

If this mode is used (i.e. 1), the widget must be able to accept a blank response for the field, or assume a reasonable value.

A function return value of zero indicates the **mode** was successfully set.



ID = 1324

And you can easily tell if nothing has been entered into an optional Box with the *Validate* call.

Validate(Name_Box box,Text &result)

Name

Integer Validate(Name_Box box,Text &result)

Description

Validate the contents of Name_Box **box** and return the Text **result.**

The function returns the value of:

NO_NAME if the Widget Name_Box is optional and the box is left empty
TRUE (1) if no other return code is needed and *result* is valid.
FALSE (0) if there is an error.

So a function return value of zero indicates that there is an error.

Warning this is the opposite of most 12dPL function return values

ID = 931

NO_NAME is returned if the Box is optional and the box is left empty.

17.4.1 Exercise 15

Create a new 12dPL program that allows the user to change the name, colour and model of a selected string. If no new name is given then the name is not changed. if no new colour is given, then the colour is not changed.

COURSE NOTES

12d Model Programming Language

IMPORTANT NOTE

What happened when you changed the colour of a string?

Did it change straight away or only on a view redraw?

If only on a view redraw then you will want to know about the function Element_draw:

Element_draw(Element elt)

Name

Integer Element_draw(Element elt)

Description

Draw the Element **elt** in its natural colour on all the views that **elt** is displayed on.

A function return value of zero indicates that **elt** was drawn successfully.

ID = 371

If you weren't using this in your program then add it in now and try changing colours again.

See Fifteen.4dm if you are having problems.

18.0 Some Examples

18.1 Exercise_8.4dm

```
// -------------------------------------------------------
// Macro:        Exercise_8.4dm
// Author:       ljg
// Organization: 12D Solutions - NSW
// Date:         Wed Aug 21 00:59:41 2013
// -------------------------------------------------------
 Integer write_out_model(Model model,File file) {
// -------------------------------------------------------
// User Defined Function to write information about the
// elements in a model to a file
// -------------------------------------------------------
  Text model_name;
  Dynamic_Element model_elts;
  Integer num_elts,ierr;

  ierr = Get_name(model,model_name);
  if(ierr != 0) return(ierr);

  Uid model_uid;
  Get_id(model,model_uid);
  File_write_line(file,"Model uid "+To_text(model_uid));

  Get_elements(model,model_elts,num_elts);
  File_write_line(file,"There are "+To_text(num_elts)+" elements in the model: "+ model_name);

  for(Integer i=1;i<=num_elts;i++) {
    Element element;
    Get_item(model_elts,i,element);

    Text line_out;
    Text element_name;
    Get_name(element,element_name);
    line_out = element_name+"\t";

    Uid element_uid;
    Get_id(element,element_uid);
    line_out += To_text(element_uid)+"\t";

    Text element_type;
    Get_type(element,element_type);
    line_out += element_type+"\t";

    Integer num_verts;
    Get_points(element,num_verts);
    line_out += To_text(num_verts);
    File_write_line(file,line_out);
  }
  File_flush(file);
  File_close(file);
  return(0);
}

void main(){
// -------------------------------------------------------
// this is where the macro starts
// -------------------------------------------------------
  Clear_console();
  Text my_model_name;
  Model my_model;
```

```
  while(!Model_exists(my_model)) {
    Model_prompt("Select a model",my_model_name);
    my_model = Get_model(my_model_name);
  }

  Text file_name;
  File_prompt("Enter the file name","*.rpt",file_name);

  File my_file;
  File_open(file_name,"w","ccs=UNICODE",my_file);

  Integer ierr;

  ierr = write_out_model(my_model,my_file);
}
```

18.2 Eleven_1.4dm

```
//-------------------------------------------------------------
// Partially completed macro to write out a report on a model.
// -------------------------------------------------------------
#include "set_ups.h"

void main() {
  Panel         panel   = Create_panel("Model Report");
  Message_Box   msg_box = Create_message_box("First message");
  Model_Box model_box = Create_model_box("Select model to report on",msg_box,CHECK_MODEL_EXISTS);
  File_Box file_box = Create_file_box("Report file",msg_box,CHECK_FILE_NEW,"*.rpt");
  Button  write_button   = Create_button("Write","write_reply");
  Button  finish_button  = Create_finish_button("Finish","finish_reply");

  Append(model_box,panel);
  Append(file_box,panel);
  Append(msg_box,panel);
  Append(write_button,panel);
  Append(finish_button,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1;

  while(doit) {
    Integer id; Text cmd,msg;

    Integer ret = Wait_on_widgets(id,cmd,msg);

// Process events from any of the Widgets on the panel

    Print("id= "+To_text(id)+" cmd=<"+cmd+"> msg=<"+msg+">\n");

    switch(id) {
      case Get_id(panel): {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button): {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(write_button): {
        Set_data(msg_box,"Write clicked");
        break;
      }
    }
  }
}
```

18.3 Eleven_2.4dm

```
//-------------------------------------------------------------
// Partially completed macro to write out a report on a model.
// -------------------------------------------------------------
#include "set_ups.h"

void main() {
  Panel         panel   = Create_panel("Model Report");
  Message_Box   msg_box = Create_message_box("First message");
  Model_Box model_box = Create_model_box("Select model to report on",msg_box,CHECK_MODEL_EXISTS);
  File_Box file_box = Create_file_box("Report file",msg_box,CHECK_FILE_NEW,"*.rpt");
  Button  write_button   = Create_button("Write","write_reply");
  Button  finish_button = Create_finish_button("Finish","finish_reply");

  Vertical_Group vgroup = Create_vertical_group(0);
  Append(model_box,vgroup);
  Append(file_box,vgroup);
  Append(msg_box,vgroup);

  Horizontal_Group hgroup = Create_button_group();
  Append(write_button,hgroup);
  Append(finish_button,hgroup);

  Append(hgroup,vgroup);
  Append(vgroup,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1;

  while(doit) {
    Integer id; Text cmd,msg;

    Integer ret = Wait_on_widgets(id,cmd,msg);

// Process events from any of the Widgets on the panel

    Print("id= "+To_text(id)+" cmd=<"+cmd+"> msg=<"+msg+">\n");


    switch(id) {
      case Get_id(panel): {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button): {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(write_button): {
        Set_data(msg_box,"Write clicked");
        break;
      }
    }
  }
}
```

18.4 Eleven_3.4dm

```
//-------------------------------------------------------------
// Partially completed macro to write out a report on a model.
// -------------------------------------------------------------
#include "set_ups.h"

// -------------------------------------------------------
 Integer write_out_model(Model model,File file) {
// -------------------------------------------------------
// User Defined Function to write information about the
// elements in a model to a file
// -------------------------------------------------------
  Text model_name;
  Dynamic_Element model_elts;
  Integer num_elts,ierr;

  ierr = Get_name(model,model_name);
  if(ierr != 0) return(ierr);

  Uid model_uid;
  Get_id(model,model_uid);
  File_write_line(file,"Model uid "+To_text(model_uid));

  Get_elements(model,model_elts,num_elts);
  File_write_line(file,"There are "+To_text(num_elts)+" elements in the model: "+ model_name);

  for(Integer i=1;i<=num_elts;i++) {
    Element element;
    Get_item(model_elts,i,element);

    Text line_out;
    Text element_name;
    Get_name(element,element_name);
    line_out = element_name+"\t";

    Uid element_uid;
    Get_id(element,element_uid);
    line_out += To_text(element_uid)+"\t";

    Text element_type;
    Get_type(element,element_type);
    line_out += element_type+"\t";

    Integer num_verts;
    Get_points(element,num_verts);
    line_out += To_text(num_verts);
    File_write_line(file,line_out);
  }
  File_flush(file);
  File_close(file);
  return(0);
}

void main() {
  Panel         panel   = Create_panel("Model Report");
  Message_Box   msg_box = Create_message_box("First message");
  Model_Box model_box = Create_model_box("Select model to report on",msg_box,CHECK_MODEL_EXISTS);
  File_Box file_box = Create_file_box("Report file",msg_box,CHECK_FILE_NEW,"*.rpt");
  Button  write_button   = Create_button("Write","write_reply");
  Button  finish_button = Create_finish_button("Finish","finish_reply");

  Vertical_Group vgroup = Create_vertical_group(0);
  Append(model_box,vgroup);
  Append(file_box,vgroup);
  Append(msg_box,vgroup);
```

```
  Horizontal_Group hgroup = Create_button_group();
  Append(write_button,hgroup);
  Append(finish_button,hgroup);

  Append(hgroup,vgroup);
  Append(vgroup,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1;

  while(doit) {
    Integer id; Text cmd,msg;

    Integer ret = Wait_on_widgets(id,cmd,msg);

// Process events from any of the Widgets on the panel

    Print("id= "+To_text(id)+" cmd=<"+cmd+"> msg=<"+msg+">\n");


    switch(id) {
      case Get_id(panel): {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button): {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(write_button): {
// check that the model exists for the name in the model box
        Model model;
        if(Validate(model_box,GET_MODEL_ERROR,model)!= MODEL_EXISTS) break;

// check that the file does not exist
        Text result; File file; Integer validate_return;
        validate_return = Validate(file_box,GET_FILE_CREATE,result);

        if(validate_return == NO_FILE) {           // file doesn't exist so can create it
          File_open(result,"w","ccs=UNICODE",file);
        } else {
          Set_data(msg_box,"Choose another file");
          break;
        }
        write_out_model(model,file);             // write out data
        Set_data(msg_box,"Data written out");
        break;
      }           // end of case write_button
    }
  }
}
```

18.5 Twelve_1.4dm

```
//-------------------------------------------------------------------
// Partially completed macro to look at Select_Box and New_Select_Box
// -------------------------------------------------------------------
#include "set_ups.h"

void main() {
  Panel           panel          = Create_panel("String Report");
  Message_Box     msg_box        = Create_message_box("");
  Message_Box     new_msg_box    = Create_message_box("");
  Select_Box      select_box     =Create_select_box("Select string","Select a string",
                                       SELECT_STRING,msg_box);
  New_Select_Box   new_select_box = Create_new_select_box("New select string",
                                         "New select a string",SELECT_STRING,new_msg_box);
  Button           write_button   = Create_button("Write","write_reply");
  Button           finish_button  = Create_finish_button("Finish","finish_reply");

  Vertical_Group vgroup = Create_vertical_group(0);
  Append(select_box,vgroup);
  Append(new_select_box,vgroup);
  Append(msg_box,vgroup);
  Append(new_msg_box,vgroup);

  Horizontal_Group hgroup = Create_button_group();
  Append(write_button,hgroup);
  Append(finish_button,hgroup);

  Append(hgroup,vgroup);
  Append(vgroup,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1;

  while(doit) {
    Integer id; Text cmd,msg;
    Integer ret = Wait_on_widgets(id,cmd,msg);

// Process events from any of the Widgets on the panel

    Print("id= "+To_text(id)+" cmd=<"+cmd+"> msg=<"+msg+">\n");

    switch(id) {
      case Get_id(panel): {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button): {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(write_button): {
        Set_data(msg_box,"Write clicked");
        break;
      }
    }
  }
```

18.6 Thirteen.4dm

```
//-------------------------------------------------------------
// Programmer   Lee Gregory
// Date         22/9/13
// Description of Macro
// Macro using a panel to select a string and when a string is
// selected, write out to the message box, the
// number of vertices there are in the string.
// Also write out if Att_ZCoord_Value or Att_ZCoord_Array is
// set for the selected string.
// The macro terminates when the Finish button, or X is selected.
//-------------------------------------------------------------
#include "set_ups.h"

void main() {
  Pane  panel= Create_panel("Number of Vertices Report");
  Message_Box new_msg_box = Create_message_box("");
  New_Select_Box new_select_box = Create_new_select_box("Select string",
                                  "Select a string",SELECT_STRING,new_msg_box);
  Button  finish_button = Create_finish_button("Finish","finish_reply");

  Vertical_Group vgroup = Create_vertical_group(BALANCE_WIDGETS_OVER_HEIGHT);
  Append(new_select_box,vgroup);
  Append(new_msg_box,vgroup);

  Horizontal_Group hgroup = Create_button_group();
  Append(finish_button,hgroup);

  Append(hgroup,vgroup);
  Append(vgroup,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1,id; Text cmd,msg;

  while(doit) {
    Integer ret = Wait_on_widgets(id,cmd,msg);

    switch(id) {
      case Get_id(panel): {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button): {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(new_select_box): {
        Set_data(new_msg_box,"");
        if(cmd == "accept select") {
          Element string; Integer ierr,no_verts;
          ierr = Validate(new_select_box,string);
          if(ierr != TRUE) {
            Set_data(new_msg_box,"Invalid pick.");
            break;
          }
          if(Get_points(string,no_verts)!=0) {
            Set_data(new_msg_box,"error in string");
            break;
          }
          Integer use;
          ierr = Get_super_use_3d_level(string,use);//check 3d first in case both 2d & 3d are set
          if(ierr != 0) {
```

```
          Set_data(new_msg_box,To_text(no_verts) + " vertices in the string");
          break;
        }
        if(use ==1) {
          Set_data(new_msg_box,To_text(no_verts) +
                     " vertices in the string - Att_ZCoord_Array");
          break;
        }

        ierr = Get_super_use_2d_level(string,use);
        if(ierr != 0) {
          Set_data(new_msg_box,To_text(no_verts) + " vertices in the string");
          break;
        }
        if(use == 1) {
          Set_data(new_msg_box,To_text(no_verts)+
                 " vertices in the string - Att_ZCoord_Value");
          break;
        }
        Set_data(new_msg_box,To_text(no_verts) +
                     " vertices in the string - no Att_ZCoord");
      }
      break;
    }
  }
 }
}
```

18.7 Fourteen.4dm

```
//-------------------------------------------------------------
// Programmer   Lee Gregory
// Date         22/9/13
// Description of Macro
// Macro using a panel to select a string and when a string is
// selected, write out to the message box, the
// number of vertices there are in the string.
// Also write out if Att_ZCoord_Value or Att_ZCoord_Array is
// set for the selected string.
// Also writes all this information and the string name and model,
// to the Output Window, plus the vertex index and the
// corresponding (x,y,z) for each vertex in the string
// The macro terminates when the Finish button, or X is selected.
//-------------------------------------------------------------
// -------------------------------------------------------------
#include "set_ups.h"

void main() {
  Panel panel = Create_panel("Number of Vertices Report");
  Message_Box new_msg_box = Create_message_box("");
  New_Select_Box new_select_box = Create_new_select_box("Select string",
                  "Select a string",SELECT_STRING,new_msg_box);
  Button finish_button =Create_finish_button("Finish","finish_reply");

  Vertical_Group vgroup = Create_vertical_group(BALANCE_WIDGETS_OVER_HEIGHT);
  Append(new_select_box,vgroup);
  Append(new_msg_box,vgroup);

  Horizontal_Group hgroup = Create_button_group();
  Append(finish_button,hgroup);

  Append(hgroup,vgroup);
  Append(vgroup,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1,id; Text cmd,msg;

  while(doit) {
    Integer ret = Wait_on_widgets(id,cmd,msg);

    switch(id) {
      case Get_id(panel): {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button): {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(new_select_box): {
        Set_data(new_msg_box,"");
        if(cmd == "accept select") {
          Element string; Integer ierr,num_verts;
          ierr = Validate(new_select_box,string);
          if(ierr != TRUE) {
            Set_data(new_msg_box,"Invalid pick.");
            break;
          }
          Text string_type;
          Get_type(string,string_type);
          if(string_type != "Super") {
```

```
          Set_data(new_msg_box,"not a Super String");
          continue;
        }
        if(Get_points(string,num_verts)!=0) {
          Set_data(new_msg_box,"error in string");
          break;
        }
        Integer use_2d,use_3d; Text out;
        ierr = Get_super_use_3d_level(string,use_3d);
        if(ierr != 0) continue;
        ierr = Get_super_use_2d_level(string,use_2d);
        if(ierr != 0) continue;

        if((use_2d == 0)&&(use_3d == 0)){
          Set_data(new_msg_box,"not the correct string dimensions");
          continue;
        }

        out = To_text(num_verts) + " vertices in the string - ";
        if(use_3d == 1) {
          out = out + "Att_ZCoord_Array";
       } else if(use_2d == 1) {
         out = out + "Att_ZCoord_Value";
        }

        Text string_name,model_name; Model model;
        Get_name(string,string_name);
        Get_model(string,model);
        Get_name(model,model_name);

        Print("\nString name <" + string_name + "> Model name <" + model_name + ">\n");
        Set_data(new_msg_box,out);
        Print(out+"\n");

        Real x,y,z;
        for (Integer i=1;i<=num_verts;i++) {
          Get_super_vertex_coord(string,i,x,y,z);
          Print("vert index " + To_text(i) + " x = " + To_text(x) +
                   " y = " + To_text(y) + " z = " + To_text(z) + "\n");
        }
      }
      break;
    }
  }
 }
}
```

18.8 Fifteen.4dm

```
//-------------------------------------------------------------
// Programmer   Lee Gregory
// Date         22/9/13
// Description of Macro
// Macro using a panel to have an optional Name and Colour Box
// Select a string and when a string is selected
// change the name and/or colour of the string
// -------------------------------------------------------------
#include "set_ups.h"

void main() {
  Panel panel = Create_panel("Change String Name and Colour");
  Message_Box msg_box = Create_message_box("");
New_Select_Box new_select_box = Create_new_select_box("Select string",
                            "Select a string",SELECT_STRING,msg_box);
  Button finish_button = Create_finish_button("Finish","finish_reply");

  Name_Box name_box = Create_name_box("New name",msg_box);
  Set_optional(name_box,1);

  Colour_Box colour_box = Create_colour_box("New colour",msg_box);
  Set_optional(colour_box,1);

  Vertical_Group vgroup = Create_vertical_group(BALANCE_WIDGETS_OVER_HEIGHT);
  Append(name_box,vgroup);
  Append(colour_box,vgroup);
  Append(new_select_box,vgroup);
  Append(msg_box,vgroup);

  Horizontal_Group hgroup = Create_button_group();
  Append(finish_button,hgroup);

  Append(hgroup,vgroup);
  Append(vgroup,panel);

  Show_widget(panel);
  Clear_console();

  Integer doit = 1,id; Text cmd,msg;

  while(doit) {
    Integer ret = Wait_on_widgets(id,cmd,msg);

    switch(id) {
      case Get_id(panel): {
        if(cmd == "Panel Quit") doit = 0; // will end while loop
        break;
      }
      case Get_id(finish_button): {
        if(cmd == "finish_reply") doit = 0; // will end while loop
        break;
      }
      case Get_id(new_select_box): {
        Set_data(msg_box,"");
        if(cmd == "accept select") {
          Element string; Integer ierr,num_verts;
          ierr = Validate(new_select_box,string);
          if(ierr!= TRUE) {
            Set_data(msg_box,"Invalid pick.");
            break;
          }
          Text string_type,new_name; Integer new_colour;
// check string is a Super String
```

```
          Get_type(string,string_type);
          if(string_type!= "Super") {
            Set_data(msg_box,"not a Super String");
            continue;
          }
// check for errors in Name_Box
          Integer val_name_box   = Validate(name_box,new_name);
          if(val_name_box == FALSE) {
            Set_data(msg_box,"error in new name");
            continue;
          }
// check for errors in Colour_Box
          Integer val_colour_box = Validate(colour_box,new_colour);
          if(val_colour_box == FALSE) {
            Set_data(msg_box,"error in new colour");
            continue;
          }
// modify the string
          if(val_name_box!= NO_NAME) Set_name(string,new_name);
          if(val_colour_box!= NO_NAME) {
            Set_colour(string,new_colour);
            Element_draw(string);
          }
          Set_data(msg_box,"changes made");
        }
        break;
      }
    }
  }
}
```

19.0 Not Used

```
case Get_id(write_button): {
// check that the model exists for the name in the model box
Model model;

  if(Validate(model_box,GET_MODEL_ERROR,model)!= MODEL_EXISTS) break;

// check that the file does not model exist for the name in the file box
  Text result; File file; Integer validate_return;



  validate_return = Validate(file_box,GET_FILE_CREATE,result);

  if(validate_return == NO_FILE) {  // file doesn't exist so can create it
    File_open(result,"w","ccs=UNICODE",file);
    ierr = write_out_model(model,file);  // write out data
    Set_data(msg_box,"Data written out");
  } else if (validate_return == NO_FILE_ACCESS) {
    Set_data(msg_box,"Chose another file name");
  } else if (validate_return == NO_NAME){
    Set_data(msg_box,"No file name given");
  } else {
    Set_data(msg_box,"Give a file name");
  }
  break;
}
```

if the model does not exist then an error message is written to the message box

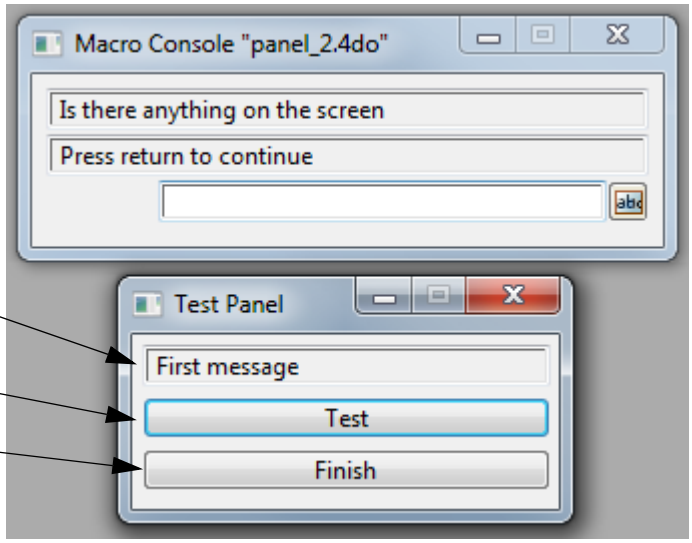
if the file exists you are asked if you want to replace it and if yes then the fie is deleted and NO_FILE is returned

if no file is given then NO_NAME is returned

if the file exists and you say no to replacing it then NO_FILE_ACCESS is returned

panel_3.4dm

```
#include "set_ups.h"

void main()
{
  Panel panel = Create_panel("Test Panel");

  Message_Box msg_box = Create_message_box("");
  Button finish_button,write_button;

  write_button   = Create_button("Write","write_reply");
  finish_button = Create_finish_button("Finish","finish_reply");

  Model_Box model_box = Create_model_box("Select model",
                        msg_box,CHECK_MODEL_MUST_EXIST);

  Append(model_box,panel);
  Append(msg_box,panel);
  Append(write_button,panel);
  Append(finish_button,panel);

  Show_widget(panel);
  Error_prompt("Is there anything on the screen");
}
```



Message_Box appended first

Button appended second

Finish Button appended third