12d® Model™

12d Model Macro Language Programming Manual

Version 9.0

July 2011

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 V9.00

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

1 Introduction

The 12D Solutions Macro Language (4DML) is a powerful programming language designed to run from within 12d Model.

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

4DML 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.

4DML 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 4DML.

Examples of usage are given for many of the 4DML supplied functions.

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

The Mouse

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

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

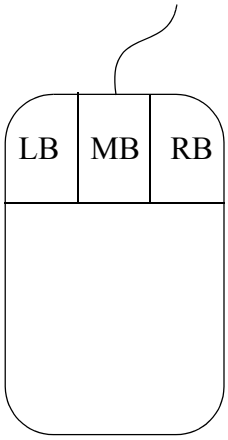
12d Model can be operated with either a two or a three button mouse but a three button mouse is preferred.

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.

Compiling and Running a 4DML Macro

A 12d Model Macro Language program or **macro** 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 macro must end in **.4dm**.

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

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

To compile a 4DML macro, use either

(a) inside 12d Model: the **compile** or **compile and run** options

Utilities =>Macros =>Compile

Utilities =>Macros =>Compile/run

or

(b) outside 12d Model: the **compile_4d** command.

For example

compile_4d macro-file.4dm

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

For example, the **compile_4d** command

compile_4d *macro-file.4dm*

will check the macro *macro-file.4dm* and produce a run-time object called

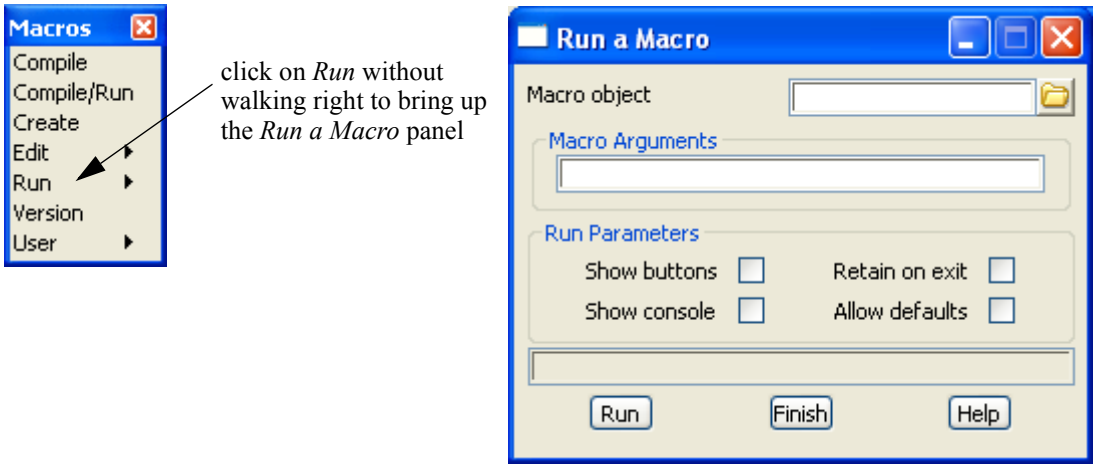
macro-file.4do

To run the run-time macro from within 12d Model, walk-right on the menu option

Utilities =>Macros =>Run

and select the macro from the list of available macros.

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 macro 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 macro, and then selecting the button **Run**.

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

2 Basic Language Structure

Basic Concepts

A name 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.

Keywords

The following keywords 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 4DML variable types and 4DML 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 macro.

For example,

```
goto        fred    ;
```

is the same as

```
goto fred;
```

Comments

4DML 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 4DML 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 */
}
```

Variable Types

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 and Types

In 4DML, variable 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 variable names.

4DML variable names are **case sensitive**.

In 4DML, 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.

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

(a) void

This is a special type which is only used for function 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 4DML macro 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 4DML macros for geometric calculations. They are only temporary objects and only last for the duration of the macro.

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 macro 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 macro 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 acessing 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 4DML variables, go to Summary of 4DML 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 4DML macros for geometric calculations but they are temporary objects and only last for the duration of the macro.

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 the 12d Model macro language, 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 4DML Library Calls

In the 12d Model macro language, a Spiral is a construction entity and is not stored in **12d Model** models.

Parabola

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.

In the 12d Model macro language, 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.

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 4DML macro 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.

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
View
Macro_Function or Function
Undo_List

Element

The variable type **Element** is used to refer to the standard *12d Mode*l strings and tin entities. That is, Elements are handles to data that can be stored in *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

2d	string with (x,y) at each pt but constant z. *See* 2d Strings
3d	string with (x,y,z) at each point. *See* 3d Strings
4d	string with (x,y,z,text) at each point. *See* 4d Strings
Alignment	string with separate horizontal and vertical geometry defined only by using the intersection point methods. *See* Interface String
Arc	an arc in the (x,y) plane with linear interpolated z values (i.e. a helix). *See* Arc Strings
Circle	a circle in the (x,y) plane with a constant z value. *See* Circle Strings
Feature	a circle with a z-value at the centre but only null values on the circumference. *See* Feature String
Drainage	string for drainage or sewer elements. *See* Drainage Strings
Interface	string with (x,y,z,cut/fill flag) at each point. *See* Interface String
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 Frames
Polyline	string with (x,y,z,radius) at each point. *See* Pipeline Strings
Pipeline	an Alignment string with a diameter. *See* Pipeline Strings
Super	general string with at least (x,y,z,radius) at each point.

See Super String Dimensions and Flags

Text	string with text at a point. *See* Text Strings
Tin	triangulated irregular network - a triangulation *See* Tins
SuperTin	a list of Tins that acts as one Tin
Super Alignment	a string with separate horizontal geometry defined by using the intersection point methods and other construction methods such as fixed and floating.

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 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 macro terminates.

See

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

Uid

A unique number 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 macro calls, see Panels

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 .

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* Widget Controls

Horizontal_Group

Used for formatting a panel.

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

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 coloours. *See* Colour_Box .

Colour_Message_Box

A box on a Panel for writing messages to. Different bbackground 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 Name_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 4DML 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, 4DML 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 4DML 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 4DML Variable Types

The 4DML 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 number 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, justication

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_Box es -
Sheet_Size_Box -
Source_Box -
Symbol_Box -
Tab_Box -
Target_Box - // not yet implimented
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 Constants
Real Constants
Text Constants

Integer Constants

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 Constants

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 Constants

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	\	\\

Operators and Assignments

See

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

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

^ dot 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

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

Bitwise Operators

The bitwise operators are

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

Assignment Operators

assignment

=	assignment e.g. x = y

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;
}
```

Flow Control

The flow control statements of a language specify the order in which computations are performed.

Many of the 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 less than b a < b

Following C++, 4DML 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**.

If, Else, Else If

4DML 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

else if

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

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

4DML 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

4DML 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.

Note

Switch Note

Note

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

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;
  }
}
```

Notes

(a) More that one statement can follow the case statement without the statements being enclosed in braces.

(b) 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

4DML supports the standard C++ **while** statement.

```
while (logical_expression)
    statement
```

is interpreted as:

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

This cycle continues 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

4DML 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

4DML 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** loop to begin.

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.

Note

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

Goto and Labels

4DML 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

4DML 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.

Reprocessing

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

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 macro 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 on Blocks and Scope).

The 4DML macro file must contain a starting function called **main** as well as zero or more user defined functions. These functions can occur in any order in the macro file. The syntax for the functions will be described in the following sections.

A large number of functions are supplied with 4DML to make tasks easier for the macro writer. These 4DML supplied functions are predefined and nothing special is needed to use them. The 4DML supplied functions will all be specified later in the manual.

In 4DML, 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 4DML keywords or variable names in the macro.

4DML function names are case sensitive.

Note

All 4DML supplied functions begin with a capital letter to help avoid clashes with any user variable names.

Main Function

A 4DML macro must contain a special function called **main**. This function is the designated start of the macro.

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
       ie program code
    }
```

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

Hence every macro 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 macro 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 functions.

Hence each 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
    }
```

Array Variable

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.

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.

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++, the last statement for a function with a non-void return type must be a return statement.

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 the function name, the function return type and the order and type of all the function parameters.

A function prototype looks like the function header. Note 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 type and arguments of a function so that a function can be used in a macro before the code for the function has been found in the file.

Notes

(a) The function main and any 4DML 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

4DML follows C++ in that function arguments can be passed "by value" or "by reference".

In "pass by value", the called function is passed the values of its arguments in temporary variables which are not connected with the original variables in the calling routine.

Hence, if an argument is passed by value to a function, any modifications of the variable inside the function will not affect the original argument in the calling routine.

In 4DML, the default for non-array arguments is "pass by value".

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

If an 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.

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 "pass by value" and "pass 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 4DML, if you have a number of functions that perform the same operation but with different argument types, there is no need to give each function a different name.

As long as the argument types differ in some way, 4DML 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
}
```

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 4DML, 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 4DML for functions with **named arguments**.

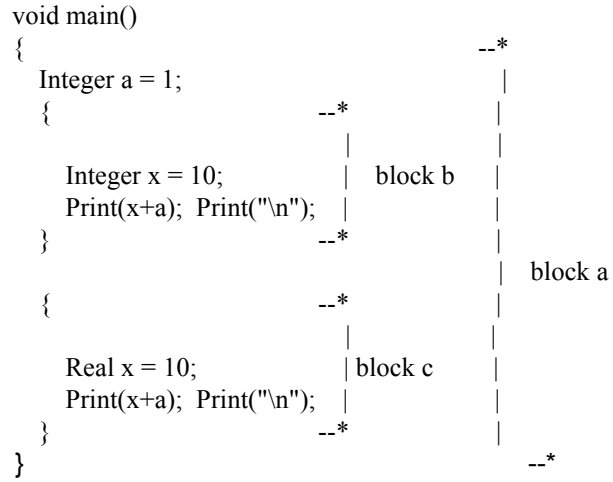
Blocks and Scopes

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**.



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

In 4DML, 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 4DML, 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;
    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 4d/12d Model operation.

To prevent data corruptions, **locks** are automatically used within 4d/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 4DML Library Calls

The 4DML Library Calls section consists of descriptions of all the supplied 4DML 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.

Function Argument Promotions

Because 4DML has automatic variable type promotions and function overloading, many of the 4DML 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 4DML 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 4DML 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:

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 macro is invoked, command-line arguments (parameters) can be passed down and accessed from within the macro.

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 macro.

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 macro.

Get_number_of_command_arguements()

Name

Integer Get_number_of_command_arguments()

Description

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

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

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**.

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

The arguments start from 1.

Exit

Macro 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 macro simply terminates.

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

However, 4DML also has special exit routines that will immediately stop the execution of the macro 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 macro and write the message

macro exited with code **exit_code**

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

Exit(Text msg)

Name

void Exit(Text msg)

Description

Immediately exit the macro and write the message

macro exited with message **msg**

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

Destroy_on_exit( )

Name

void Destroy_on_exit( )

Description

Destroy current macro console panel when exit the macro.

Retain_on_exit( )

Name

void Retain_on_exit( )

Description

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

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
Real Half_pi()	the value of half pi
Real Two_pi()	the value of 2 * pi

Types of Angles

In **4DML**, 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 **4DML**, 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)
Integer Degrees_to_radians(Real deg,Real &rad)
Integer Radians_to_hp_degrees(Real rad,Real &hp_deg)
Integer Hp_degrees_to_radians(Real hp_deg,Real &rad)
Integer Degrees_to_hp_degrees(Real deg,Real &hp_deg)
Integer Hp_degrees_to_degrees(Real hp_deg,Real &deg)
Integer Degrees_to_dms(Real deg,Integer &dd,Integer &mm,Real &ss)
Integer Dms_to_degrees(Integer dd,Integer mm,Real ss,Real &deg)
Integer Angle_to_bearing(Real angle,Real &bearing)
```

Integer Bearing_to_angle(Real bearing,Real &angle)

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".

4DML 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**.

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.

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.

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.

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.

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**.

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.

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.

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.

Text Conversions

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.

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.

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.

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.

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.

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. That is, except for characters between matching double quotes ", spaces are the separators between individual words.

Any characters (including blanks) between matching double quotes are considered to be one word.

For example, in

This is "an example"

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

The function return value is zero if the break up is successful.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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**.

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

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**.

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

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**.

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

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.

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

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.

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.

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.

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.

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 br **style**.

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

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.

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 type be **type**.

LJG? what is type

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

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.

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**.

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

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**.

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

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**.

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

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.

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

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.

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.

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.

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.

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.

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

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.

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

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.

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

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.

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.

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

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.

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

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.

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

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.

Maths

Most of the standard C++ mathematical functions are supported in 4DML.

The angles for the trigonometric functions are expressed in radians

Real Sin(Real x)	sine of x
Real Cos(Real x)	cosine of x
Real Tan(Real x)	tangent of x
Real Asin(Real x)	arcsine(x) in range [-pi/2,pi/2], -1<= x <= 1
Real Acos(Real x)	arccosine(x) in range [-pi/2,pi/2], -1<= value <= 1
Real Atan(Real x)	arctan(x) in range [-pi/2,pi/2]
Real Atan2(Real y, Real x)	Arctan(y/x) in range [-pi,pi]
Real Sinh(Real x)	hyperbolic sine of x
Real Cosh(Real x)	hyperbolic cosine of x
Real Tanh(Real x)	hyperbolic tangent of x
Real Exp(Real x)	exponential function
Real Log(Real x)	natural logarithm ln(x), x > 0
Real Log10(Real x)	base 10 logarithm log(x), x> 0
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.
Real Sqrt(Real x)	square root of x, x >= 0
Real Ceil(Real x)	smallest integer not less than x, as a Real
Real Floor(Real x)	largest integer not greater than x, as a Real
Real Absolute(Real x)	absolute value of x
Integer Absolute(Integer i)	absolute value of x
Real Ldexp(Real x,Integer n)	x*(2 to the power n)
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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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)

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)

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)

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.

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.

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.

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

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

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

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.

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.

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.

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))

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))

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))

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.

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.

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)

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)

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.

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.

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.

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.

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.

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.

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**.

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**.

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.

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.

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.

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**) /.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Get_macro_name()

Name

Text Get_macro_name()

Description

Get the name of the macro file.

A function return value is the macro name.

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** for licensed, **0** for not licensed.

Getenv(Text env)

Name

Text Getenv(Text env)

Description

Get the temporary directory for Windows.

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

<no description>

Find_system_file(file_name,"colour_map.def","COLOUR_4D")

Name

Text Find_system_file(file_name,"colour_map.def","COLOUR_4D")

Description

<no description>

Get_4dmodel_version(Integer &major,Integer &minor,Text &patch)

Name

void Get_4dmodel_version(Integer &major,Integer &minor,Text &patch)

Description

<no description>

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 4DML function return values

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

<no description>

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

<no description>

Uid's

Elements created within 12d Model are given a unique identifier called a Uid.

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 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.

For documentation on Uid calls, go to the next section Uid Functions

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.

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.

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.

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() .

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.

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.

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.

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.

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.

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.

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.

Null(Uid &uid)

Name

void Null(Uid &uid)

Description

Set the UID uid to be a **null** Uid.

There is no function return value.

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 4DML function return values

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 4DML function return values

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 4DML function return values

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 4DML function return values

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 4DML 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.

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 4DML function return values

Is_unknown(Uid uid)

Name

Integer Is_unknown(Uid uid)

Description

<no description>

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 4DML function return values

Input/Output

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 window.

A function return value of zero indicates success.

Print(Integer value)

Name

void Print(Integer value)

Description

Print the Integer **value** out in text to the window.

A function return value of zero indicates success.

Print(Real value)

Name

void Print(Real value)

Description

Print the Real **value** out in text to the window.

A function return value of zero indicates success.

void Print()

Name

void Print()

Description

Print the text "\n" (a new line) to the window.

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.

4DML provides a wide range of functions to allow the user to easily read and write files within macros.

For reading in data, 4DML only provides the File_read_line function which reads only read one line of text. However, the powerful 4DML Text functions can then be used on the text line to "pull the line apart" and extract the relevant information.

Similarly, the File_write_line function only outputs one text line but again the powerful Text functions are used to build up any complex line of text required.

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 4DML function return values

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.

File_open(Text file_name,Text mode,File &file)

Name

Integer File_open(Text file_name,Text mode,File &file)

Description

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.

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.

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 zero indicates the text was successfully read in.

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.

File_rewind(File file)

Name

Integer File_rewind(File file)

Description

Rewind the File **file** to its beginning.

A function return value of zero indicates the file was successfully rewound.

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.

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.

A function return value of zero indicates the file position was successfully found.

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.

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.

Menus

Menus with the same look and feel as 12d Model menus can be easily created within 4DML.

A 4DML 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, 4DML 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, 4DML has this special variable type called **Menu**.)

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.

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.

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.

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");
}
```

Dynamic Arrays

The 4DML 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 4DML 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 4DML 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 4DML 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.

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.

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.

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.

Append(Element &elt,Dynamic_Element delt)

Name

Integer Append(Element &elt,Dynamic_Element delt)

Description

Append the Element **elt** to the end of the contents of the Dynamic_Element **elt**. This will increase the size of the Dynamic_Element by one.

A function return value of zero indicates the append was successful.

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.

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.

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.

Dynamic Text Arrays

The 4DML 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 4DML Dynamic_Text can be thought of as a dynamic array of Texts.

The following functions are used to access and modify Dynamic_Text's.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Dynamic Real Arrays

The 4DML 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 4DML Dynamic_Real can be thought of as a dynamic array of Reals.

The following functions are used to access and modify Dynamic_Real's.

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.

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.

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.

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 that are added will be set to null (LJG? or zero?).

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

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.

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.

Dynamic Integer Arrays

The 4DML 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 4DML Dynamic_Integer can be thought of as a dynamic array of Integers.

The following functions are used to access and modify Dynamic_Integer's.

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.

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.

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.

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 that are added will be set to zero (LJG? or zero?).

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

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.

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.

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**.

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**.

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**.

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**.

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.

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**.

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**.

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**.

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**.

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**.

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.

Arcs

A 4DML 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 4DML 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 4DML 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.

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.

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.

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.

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.

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.

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.

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.

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**.

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.

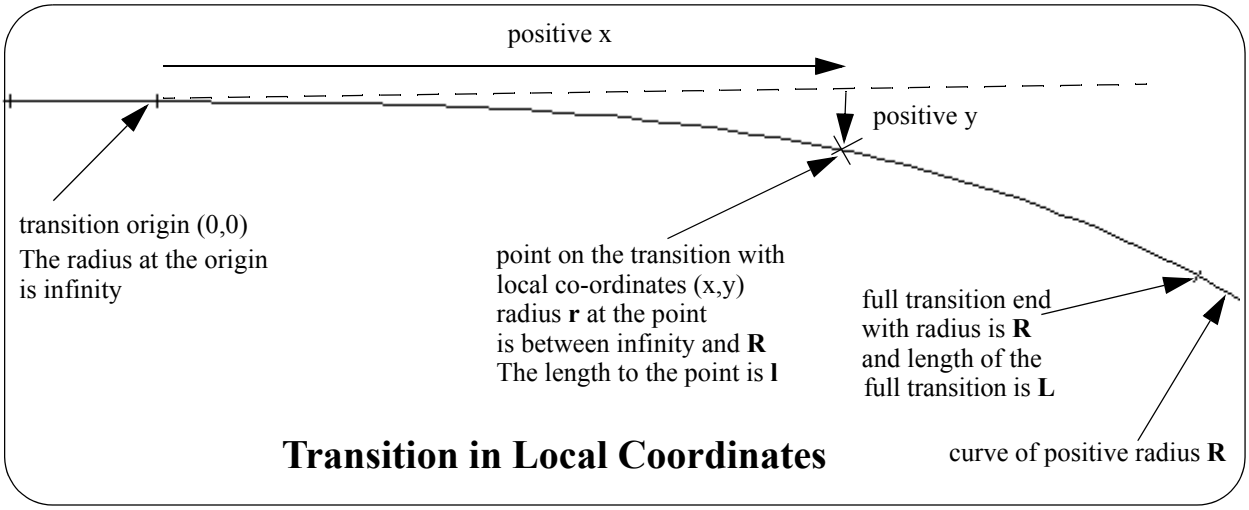
Examples of a common transitions used (mainly for rail) are:

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

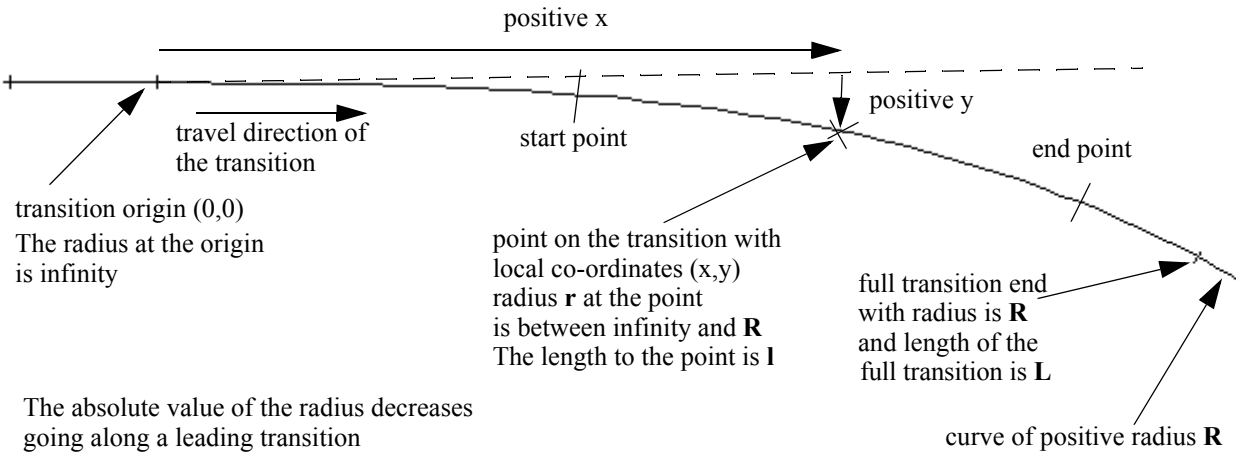
Sometimes the full transition curve is not required and only a part of the transition is used. The transition is only used from a **start point** (at transition length **start length** from the beginning of the full transition), to and **end point** (at transition length **end length** from the beginning 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.

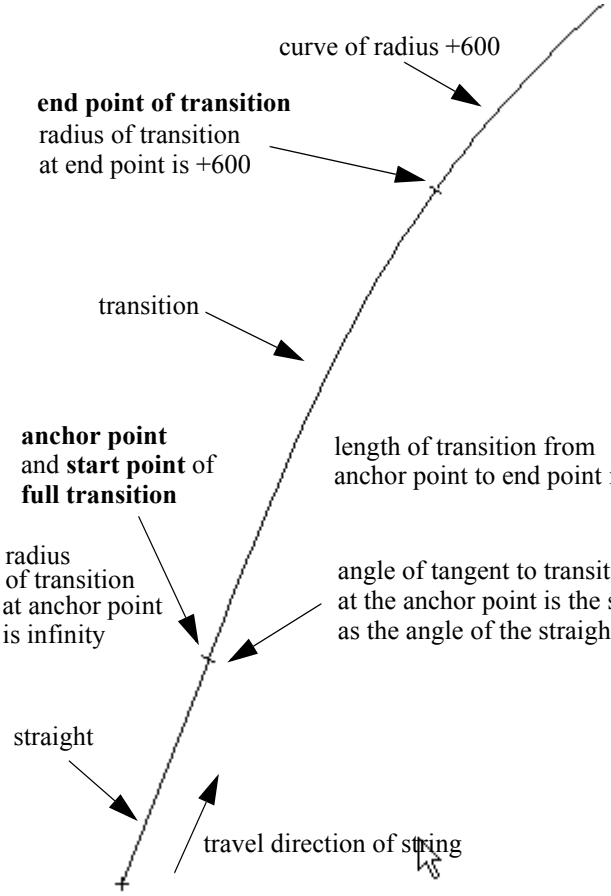
trailing transition starts on a curve of absolute radius R and ends on a straight. The absolute value of the radius of curvature goes from infinity to a value R



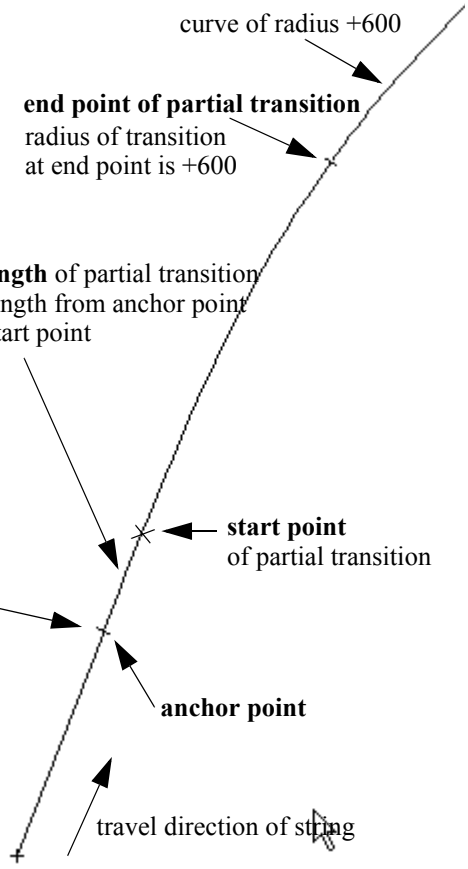
A Leading Transition in Local Coordinates

Finally the transition needs to be placed in world coordinates.

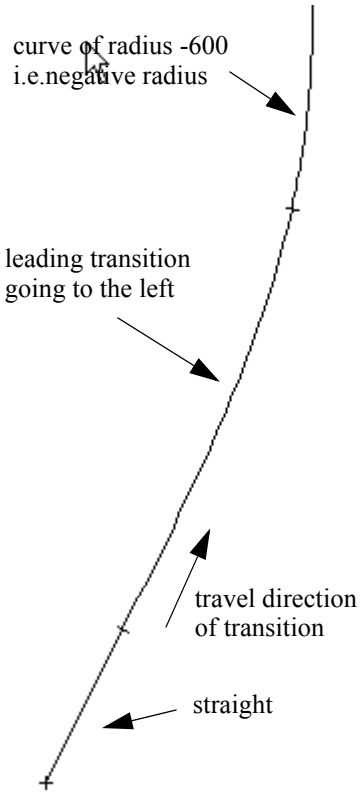
So to position the transition in world coordinates, the local transition origin (0,0) is translated to the position (x,y) (called the **anchor point** of the transition) and the transition is rotated about the anchor point though the angle **direction** (the angle is measure in a counterclockwise direction from the positive x axis). So the at the anchor point will be at the angle **direction**.

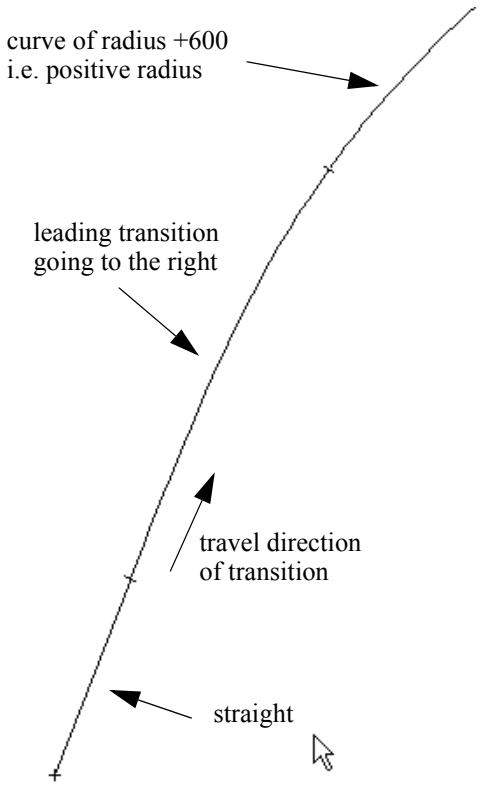


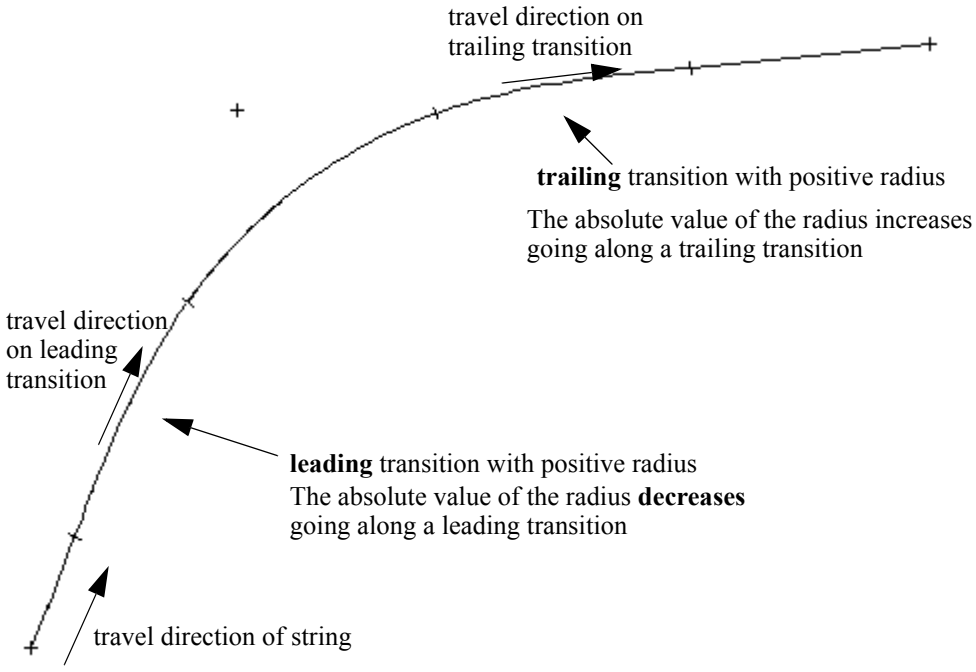
Full Leading Transition



Partial Leading Transition







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 4DML 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

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.

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.

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.

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.

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.

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.

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**.

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.

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.

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.

Get_type(Spiral trans)

Name

Integer Get_type(Spiral trans)

Description

LJG? yes what are they?

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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)

Get_shift_y(Spiral trans)

Name

Real Get_shift_y(Spiral trans)

Description

shift at end point of transition **trans**

Get_shift(Spiral trans)

Name

Real Get_shift(Spiral trans)

Description

shift

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.

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**.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 change of angle between the 3 points.

Point 1 is defined by Real **x1** and Real **y1**.
Point 2 is defined by Real **x2** and Real **y2**.
Point 3 is defined by Real **x3** and Real **y3**.
The angle value is returned in Real **angle**.

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

Change_of_angle(Line l1,Line l2,Real &angle)

Name

Integer Change_of_angle(Line l1,Line l2,Real &angle)

Description

Calculate the change of angle between the 2 lines.

Line 1 is defined by Line **l1**.
Line 2 is defined by Line **l2**.
The angle value is returned in Real **angle**.

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

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 4DML 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 4DML.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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.

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**.

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.

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.

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.

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.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

Attribute_dump(Attributes attr)

Name

Integer Attribute_dump(Attributes attr)

Description

Write out information about the Attributes **attr** to the Output Window.

A function return value of zero indicates the function was successful.

Attribute_debug(Attributes attr)

Name

Integer Attribute_debug(Attributes attr)

Description

Write out even more information about the Attributes **attr** to the Output Window.

A function return value of zero indicates the function was successful.

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 4DML function return values

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.

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.

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.

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.

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.

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*

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*

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***

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**

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.

Get_program_special_build_name()

Name

Text Get_program_special_build_name()

Description

The function return value is a special build description for pre-release versions and it is written after the 12d Model version information. It is blank for release versions.

Get_program_patch_version_name()

Name

Text Get_program_patch_version_name()

Description

<no description>

Get_program_full_title_name()

Name

Text Get_program_full_title_name()

Description

<no description>

Get_program()

Name

Text Get_program()

Description

The function return value is the full path to where the 12d.exe is on disk.

For example "C:\Program Files\12d\12dmodel\10.00\nt.x86"

Get_program_name()

Name

Text Get_program_name()

Description

The function return value is the name of the 12d Model executable. That is, "12d.exe".

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"

LJG the folder where 12d.exe is

LJG ?? what is the difference with Get_program(). Or does Get_program() include 12d.exe

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"

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.

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.

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.

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.

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 linestyles, textstyles, fonts and colours.

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.

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.

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.

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.

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.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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.

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.

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.

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.

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)**.

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)**.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Models

The variable type **Model** is used to refer to 12d Model models.

Model variables act as ***handles*** to the actual model so that the model can be easily referred to and manipulated within a macro.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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**.

LJG? Units of time?

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

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**.

LJG? Units of time?

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

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**.

LJG? Units of time?

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

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.

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.

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.

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.

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.

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 **model_id** then a non-zero function return value is returned.

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

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.

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.

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.

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**.

A function return value of zero indicates success.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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**.

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**.

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**.

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.

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**.

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**.

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**.

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**.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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)**.

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)**.

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.

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.

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 4DML function return values

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 4DML function return values

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**.

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

If **view** is null, the function return value is non-zero.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Tins

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.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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**.

LJG? Units of time?

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

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**.

LJG? Units of time?

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

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**.

LJG? Units of time?

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

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.

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.

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.

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

<no description>

Tin_models(Tin tin,Dynamic_Text &models)

Name

Integer Tin_models(Tin tin,Dynamic_Text &models)

Description

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.

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.

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.

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.

Set_supertin(Tin_Box box,Integer mode)

Name

Integer Set_supertin(Tin_Box box,Integer mode)

Description

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.

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.

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.

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.

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.

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 4DML 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.

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.

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.

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.

Elements

The variable type Element is used to refer to the standard 12d Model strings and tin entities. That is, Elements are used as handles to data that can be stored in 12d Model models.

Elements act as ***handles*** on the data so that the data can be easily referred to and manipulated within a macro.

See Types of Elements
See Parts of 12d Elements
See Element Header
See Element Body
See 2d Strings
See 3d Strings
See 4d Strings
See Interface String
See Alignment Strings
See Arc Strings
See Circle Strings
See Text Strings
See Pipeline Strings
See Polyline Strings
See Drainage Strings
See Pipe Strings
See Face Strings
See Plot Frames
See Feature String
See Super String Element

Types of Elements

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
Super_Alignment	for a Super Alignment string - a string with separate horizontal and vertical geometry
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
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
Feature	a circle with a z-value at the centre but only null values on the circumference.
Drainage	string for drainage and sewer elements
Interface	string with (x,y,z,cut-fill flag) at each point
Text	string with text at a point
Tin	triangulated irregular network - a triangulation
SuperTin	a SuperTin of tins
Plot Frame	for a plot frame - an element used for production of plan plots
Pipeline	a string with separate horizontal and vertical geometry defined by Intersection Points only, and one diameter for the entire string.

2d	for a 2d string - a string with (x,y) at each pt but constant z value. An old string type replaced by super strings.
3d	for a 3d string - a string with (x,y,z) at each point An old string type replaced by super strings.
4d	for a 4d string - a string with (x,y,z,text) at each point An old string type replaced by super strings.
Pipe	for a pipe string - a string with (x,y,z) at each point and a diameter An old string type replaced by super strings.
Polyline	for a polyline string - a string with (x,y,z,radius) at each point An old string type replaced by super strings.
Alignment	for an Alignment string - a string with separate horizontal and vertical geometry defined by Intersection Points only. An old string type replaced by a Super Alignment string.

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 two 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.

(b) element-type specific information (the body of the Element) such as the z value for an Element of type **2d**.

The functions for manipulating the header information are given first, followed by the specific functions for each type of element.

Element Header

When an Element is created, its type is given by the Element creation function.

All new Elements are given the default header information:

id	unique id for the Element
model	none
colour	magenta
name	none
chainage	0
style	1

For all Element types, inquiries and modifications to the Element header information can be made by the following 4DML 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.

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.

Get_id(Element elt,Uid &id)

Name

Integer Get_id(Element elt,Uid &id)

Description

Get the unique Uid 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 Uid was successfully returned.

Get_points(Element elt,Integer &numpts)

Name

Integer Get_points(Element elt,Integer &numpts)

Description

Get the number of points in the Element **elt**.

The number of points is returned as the Integer **numpts**.

For Elements of type Alignment, Arc and Circle, Get_points gives the number of points when the Element is approximated using the 12d Model cord-to-arc tolerance.

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

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 4DML functions to convert the colour number to a colour name and vice-versa.

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.

Get_type(Element elt,Integer &elt_type)

Name

Integer Get_type(Element elt,Integer &elt_type)

Description

Not yet 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.

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.

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.

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.

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.

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.

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 point 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.

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.

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.

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.

Get_tin(Element elt)

Name

Tin Get_tin(Element elt)

Description

If the Element **elt** is of type **Tin**, a Tin handle for the tin will be returned.

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.

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Element Attributes

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.

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.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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 4DML function return values.

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 4DML function return values

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.**

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.**

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.

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.

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.

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.

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.

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.

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.

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.

Element Body

Strings of type 2d, 3d, 4d and Interface 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.

Strings of type Alignment, Arc, Circle and Text do not have simple arrays to define them.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

4d Strings

A 4d string consists of (x,y,z,text) values at each point of the string.

All the texts in a 4d string have the same text parameters - text units, size, angle, justification, offset and rise.

The size and distances are specified in either world units or pixels and are set by a units_mode where units_mode equals

0 for pixel units (the default)

1 for world units.

The justification point (default 1) can be one of nine positions defined in relation to the (x,y) position of the point of the 4d string:

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

The angle (default 0) of the base line of the text is measured from the horizontal axis and is in radians.

The offset distance is measured along the base line of the text (which will be at a given angle) and the rise distance is measured perpendicular to the base line of the text. The defaults for the offset and rise distances are zero.

The following functions are used to create new 4d strings and make inquiries and modifications to existing 4d strings.

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.

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.

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.

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.

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.

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.

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**.

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

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.

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**.

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

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**.

The angle is returned as Real **angle**.

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

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**.

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

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**.

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

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.

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.

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**.

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.

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.

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.

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.

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.

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.

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.

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.

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**.

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.

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.

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.

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.

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**.

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

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.

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**.

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

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**.

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

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**.

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

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**.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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**.

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.

Interface String

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.

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.

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.

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.

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.

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.

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.

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.

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.

Alignment Strings

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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**.

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.

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.

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

<no description>

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.

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.

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.

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

<no description>

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.

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.

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

<no description>

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 ith 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.

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.

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

<no description>

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.

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.

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.

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**.

Get_hip_id(Element elt,Integer position,Integer &id)

Name

Integer Get_hip_id(Element elt,Integer position,Integer &id)

Description

Get_vip_id(Element elt,Integer position,Integer &id)

Name

Integer Get_vip_id(Element elt,Integer position,Integer &id)

Description

Arc Strings

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Circle Strings

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.

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.

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.

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.

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.

Text Strings

A text string consists of a Text value at a point (x,y).

Text strings have a height which can be measured in either world units or pixels, an angle, a justification point and an offset distance and rise distance.

The unit for size is be given by a size_mode where size_mode equals

0	for pixel units (the default)
1	for world units

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 angle (default 0) of the base line of the text is measured from the horizontal axis and is in radians.

The offset distance is measured along the base line of the text (which will be at a given angle) and the rise distance is measured perpendicular to the base line of the text. The defaults for the offset and rise distances are zero.

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.

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.

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.

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.

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.

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.

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.

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**.

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

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.

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**.

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

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**.

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

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**.

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

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**.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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**.

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.

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.

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.

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**.

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

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.

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**.

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

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**.

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

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**.

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

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**.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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**.

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.

Pipeline Strings

Integer Create_pipeline()

Name

Integer Create_pipeline()

Description

Create a pipeline.

A function return value of zero indicates the pipeline was created successfully.

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.

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.

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.

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.

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.

Polyline Strings

A polyline string consists of (x,y,z,radius,flag) values at each point of the string.

For a given point, (**x,y,z**) defines the co-ordinates of the point, and (**radius,flag**) 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 **flag** specifies whether the arc is a minor or

major arc (0 for a minor arc; 1 for a major arc).

The following functions are used to create new polyline strings and make inquiries and modifications to existing polyline strings.

Create_polyline(Real x[],Real y[],Real z[],Real r[],Integer f[],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 **f[]**.

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 **f[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.

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,f) points.

The actual x, y, z, r, and f 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.

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,f) 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 f 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.

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.

Get_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer max_pts,Integer &num_pts)

Name

Integer Get_polyline_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z,r,f) data for the first **max_pts** points of the polyline Element **elt**.

The (x,y,z,r,f) values at each string point are returned in the Real arrays **x[], y[], z[], r[]** and **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 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.

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 drainage 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.

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.

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.

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.

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.

Drainage Strings

A **drainage** string is based on a **Polyline** string for defining its plan geometry but also contains information about pit locations, pipe invert levels, pipe sizes etc. See the documentation on Polyline strings for further information about polyline geometry.

Pits can be located anywhere on the string, not just at the polyline vertex points.

The drainage string is used in the ***Drainage*** modules (Drainage, Drainage Analysis and Dynamic Drainage Analysis) and also in the ***Sewer*** (Waste Water) module.

The following functions are used to create new drainage strings and make inquiries and modifications to existing drainage strings.

See Drainage String Functions
See Drainage String Pipes
See Drainage String Pipe Attributes
See Drainage String Pits
See Drainage String Pit Attributes
See Drainage String House Connections - Only Available for the Sewer Module

Drainage String Functions

Create_drainage(Integer num_pts,Integer num_pits)

Name

Element Create_drainage(Integer num_pts,Integer num_pits)

Description

Create an Element of type Drainage with room for num_pts points and room for Integer num_pits pits.

The actual data of the drainage string are set after the string is created.

If the drainage string could not be created, then the returned Element will be null.

Create_drainage(Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts, Integer num_pits)

Name

Element Create_drainage(Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts, Integer num_pits)

Description

Create an Element of type drainage.

The Element has **num_pts** points with (x,y,z) values given in the Real arrays **x[], y[] and z[]**, and arcs between each point given by the Real array **r[]** and the Integer array **f[]** (see Polyline string).

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.

Get_drainage_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer max_pts,Integer &num_pts)

Name

Integer Get_drainage_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer max_pts,Integer &num_pts)

Description

Get the (x,y,z,r,f) data for the first max_pts points of the drainage Element elt.

The (x,y,z,r,f) values at each string point are returned in the Real arrays **x[], y[], z[], r[] and 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 returned by Integer **num_pts**

num_pts <= max_pts

If the Element **elt** 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.

Set_drainage_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts)

Name

Integer Set_drainage_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 drainage 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 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.

Get_drainage_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_drainage_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 drainage 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 drainage 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 Drainage, 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_drainage_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts,Integer start_pt)

Name

Integer Set_drainage_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts,Integer start_pt)

Description

For the drainage 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 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_pt** of one gives the same result as the previous function.

(b) This function can not create new Drainage Elements but only modify existing Drainage Elements.

Get_drainage_data(Element elt,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &f)

Name

Integer Get_drainage_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 **i**th point of the 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**.
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.

Set_drainage_data(Element elt,Integer i,Real x,Real y,Real z,Real r,Integer f)

Name

Integer Set_drainage_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. if f = 0, arc < 180 degrees; if **f** = 1, arc >180 degrees.

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

Get_drainage_float(Element elt,Integer &float)

Name

Integer Get_drainage_float(Element elt,Integer &float)

Description

Set_drainage_float(Element elt,Integer float)

Name

Integer Set_drainage_float(Element elt,Integer float)

Description

Get_drainage_ns_tin(Element elt,Tin &tin)

Name

Integer Get_drainage_ns_tin(Element elt,Tin &tin)

Description

Set_drainage_ns_tin(Element elt,Tin tin)

Name

Integer Set_drainage_ns_tin(Element elt,Tin tin)

Description

Get_drainage_fs_tin(Element elt,Tin &tin)

Name

Integer Get_drainage_fs_tin(Element elt,Tin &tin)

Description

Set_drainage_fs_tin(Element elt,Tin tin)

Name

Integer Set_drainage_fs_tin(Element elt,Tin tin)

Description

Get_drainage_outfall_height(Element elt,Real &ht)

Name

Integer Get_drainage_outfall_height(Element elt,Real &ht)

Description

Get the outfall height of the drainage Element **elt**

The outfall height is returned as Real **ht**.

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

Set_drainage_outfall_height(Element elt,Real ht)

Name

Integer Set_drainage_outfall_height(Element elt,Real ht)

Description

Set the outfall height of the drainage Element **elt**.

The outfall height is given as Real **ht**.

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

Get_drainage_flow(Element elt,Integer &dir)

Name

Integer Get_drainage_flow(Element elt,Integer &dir)

Description

Get the flow direction of the drainage Element **elt**.

The flow direction is returned as Integer **dir**.

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

Note

Not implemented (maybe never)

Set_drainage_flow(Element elt,Integer dir)

Name

Integer Set_drainage_flow(Element elt,Integer dir)

Description

Set the flow direction of the drainage Element **elt**

The flow direction is given as Integer **dir**.

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

Note

Not implemented (maybe never)

Get_drainage_trunk(Element elt,Element &trunk)

Name

Integer Get_drainage_trunk(Element elt,Element &trunk)

Description

Drainage String Pipes

Get_drainage_pipe_cover(Element elt,Integer pipe,Real &minc,Real &maxc)

Name

Integer Get_drainage_pipe_cover(Element elt,Integer pipe,Real &minc,Real &maxc)

Description

Set_drainage_pipe_cover(Element elt,Integer pipe,Real cover)

Name

Integer Set_drainage_pipe_cover(Element elt,Integer pipe,Real cover)

Description

Get_drainage_pipe_diameter(Element elt,Integer p,Real &diameter)

Name

Integer Get_drainage_pipe_diameter(Element elt,Integer p,Real &diameter)

Description

Get the pipe diameter for the pth pipe 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.

Set_drainage_pipe_diameter(Element elt,Integer p,Real diameter)

Name

Integer Set_drainage_pipe_diameter(Element elt,Integer p,Real diameter)

Description

Set the pipe diameter for the **p**th pipe 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.

Get_drainage_pipe_inverts(Element elt,Integer p,Real &lhs,Real &rhs)

Name

Integer Get_drainage_pipe_inverts(Element elt,Integer p,Real &lhs,Real &rhs)

Description

Get the pipe invert levels for the **p**th pipe of the string Element **elt**.

The downstream invert level of the pipe is returned in Real **lhs**.

The upstream invert level of the pipe is returned in Real **rhs**.

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

Set_drainage_pipe_inverts(Element elt,Integer p,Real lhs,Real rhs)

Name

Integer Set_drainage_pipe_inverts(Element elt,Integer p,Real lhs,Real rhs)

Description

Set the pipe invert levels for the **p**th pipe of the string Element **elt**.

The downstream invert level of the pipe is given as Real **lhs**.

The upstream invert level of the pipe is given as Real **rhs**.

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

Get_drainage_pipe_hgls(Element elt,Integer p,Real &lhs,Real &rhs)

Name

Integer Get_drainage_pipe_hgls(Element elt,Integer p,Real &lhs,Real &rhs)

Description

Get the pipe HGL levels for the **p**th pipe of the string Element **elt**.

The downstream hgl level of the pipe is returned in Real **lhs**.

The upstream hgl level of the pipe is returned in Real **rhs**.

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

Set_drainage_pipe_hgls(Element elt,Integer p,Real lhs,Real rhs)

Name

Integer Set_drainage_pipe_hgls(Element elt,Integer p,Real lhs,Real rhs)

Description

Set the pipe hgl levels for the **p**th pipe of the string Element elt.

The downstream hgl level of the pipe is given as Real **lhs**.

The upstream hgl level of the pipe is given as Real **rhs**.

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

Get_drainage_pipe_name(Element elt,Integer p,Text &name)

Name

Integer Get_drainage_pipe_name(Element elt,Integer p,Text &name)

Description

Get the pipe name for the **p**th pipe of the string Element **elt**.

The pipe name is returned in Text **name**.

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

Set_drainage_pipe_name(Element elt,Integer p,Text name)

Name

Integer Set_drainage_pipe_name(Element elt,Integer p,Text name)

Description

Set the pipe name for the **p**th pipe of the string Element **elt**.

The pipe name is given as Text **name**.

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

Get_drainage_pipe_type(Element elt,Integer p,Text &type)

Name

Integer Get_drainage_pipe_type(Element elt,Integer p,Text &type)

Description

Get the pipe type for the **p**th pipe of the string Element **elt**.

The pipe type is returned in Text **type**.

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

Set_drainage_pipe_type(Element elt,Integer p,Text type)

Name

Integer Set_drainage_pipe_type(Element elt,Integer p,Text type)

Description

Set the pipe type for the **p**th pipe of the string Element **elt**.

The pipe type is given as Text **type**.

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

Get_drainage_pipe_velocity(Element elt,Integer p,Real &velocity)

Name

Integer Get_drainage_pipe_velocity(Element elt,Integer p,Real &velocity)

Description

Get the flow velocity for the **p**th pipe of the string Element elt.

The velocity is returned in Real **velocity**.

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

Set_drainage_pipe_velocity(Element elt,Integer p,Real velocity)

Name

Integer Set_drainage_pipe_velocity(Element elt,Integer p,Real velocity)

Description

Get the pipe flow velocity for the **p**th pipe of the string Element **elt**.

The velocity of the pipe is returned in Real **velocity**.

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

Get_drainage_pipe_flow(Element elt,Integer p,Real &flow)

Name

Integer Get_drainage_pipe_flow(Element elt,Integer p,Real &flow)

Description

Get the flow volume for the **p**th pipe of the string Element **elt**.

The volume is returned in Real **velocity**.

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

Set_drainage_pipe_flow(Element elt,Integer p,Real flow)

Name

Integer Set_drainage_pipe_flow(Element elt,Integer p,Real flow)

Description

Get the pipe flow volume for the **p**th pipe of the string Element **elt**.

The velocity of the pipe is returned in Real **flow**.

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

Get_drainage_pipe_length(Element elt,Integer p,Real &length)

Name

Integer Get_drainage_pipe_length(Element elt,Integer p,Real &length)

Description

Get the pipe length for the **p**th pipe of the string Element **elt**.

The length of the pipe is returned in Real **length**.

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

Get_drainage_pipe_grade(Element elt,Integer p,Real &grade)

Name

Integer Get_drainage_pipe_grade(Element elt,Integer p,Real &grade)

Description

Get the pipe grade for the **p**th pipe of the string Element **elt**.

The grade of the pipe is returned in Real **grade**.

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

Get_drainage_pipe_ns(Element elt,Integer p,Real ch[],Real ht[],Integer max_pts,Integer &npts)

Name

Integer Get_drainage_pipe_ns(Element elt,Integer p,Real ch[],Real ht[],Integer max_pts,Integer &npts)

Description

For the drainage string **elt,** 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**.

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

Get_drainage_pipe_fs(Element elt,Integer p,Real ch[],Real ht[],Integer max_pts,Integer &npts)

Name

Integer Get_drainage_pipe_fs(Element elt,Integer p,Real ch[],Real ht[],Integer max_pts,Integer &npts)

Description

For the drainage string **elt,** 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**.

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

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.

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.

LJG ?? type is the name of the ith pipe type

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.

Drainage String Pipe Attributes

Get_drainage_pipe_attributes(Element drain,Integer pipe,Attributes &att)

Name

Integer Get_drainage_pipe_attributes(Element elt,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.

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.

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.

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.

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.

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.

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**.

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.

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.

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.

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**.

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**.

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**.

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**.

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**.

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**.

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 4DML function return values.

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 4DML function return values.

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.

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.

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.

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.

Drainage_pipe_attribute_debug (Element elt,Integer pipe)

Name

Integer Drainage_pipe_attribute_debug (Element elt,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.

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.

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.**

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.**

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.

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.

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.

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.

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**.

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**.

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.

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.

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**.

Drainage String Pits

Get_drainage_pit(Element elt,Integer p,Real &x,Real &y,Real &z)

Name

Integer Get_drainage_pit(Element elt,Integer p,Real &x,Real &y,Real &z)

Description

Get the x,y & z for the **p**th pit of the string Element **elt**.

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).

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

Set_drainage_pit(Element elt,Integer p,Real x,Real y,Real z)

Name

Integer Set_drainage_pit(Element elt,Integer p,Real x,Real y,Real z)

Description

Set the x,y & z for the **p**th pit of the string Element **elt**.

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**.

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

Get_drainage_pit_angle(Element elt,Integer p,Real &ang)

Name

Integer Get_drainage_pit_angle(Element elt,Integer p,Real &ang)

Description

Get the angle between pipes for the **p**th pit of the string Element **elt**.

The angle between points of the pit is returned in Real **ang**.

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

Get_drainage_pit_angle (Element elt,Integer pit,Real &ang,Integer trunk)

Name

Integer Get_drainage_pit_angle(Element elt,Integer pit,Real &ang,Integer trunk)

Description

Get_drainage_pit_diameter(Element elt,Integer p,Real &diameter)

Name

Integer Get_drainage_pit_diameter(Element elt,Integer p,Real &diameter)

Description

Get the diameter for the **p**th pit of the string Element **elt**.

The diameter of the pit is returned in Real **diameter**.

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

Set_drainage_pit_diameter(Element elt,Integer p,Real diameter)

Name

Integer Set_drainage_pit_diameter(Element elt,Integer p,Real diameter)

Description

Set the diameter for the **p**th pit of the string Element **elt**.

The diameter of the pit is given as Real **diameter**.

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

Get_drainage_pit_float(Element elt,Integer pit,Integer &float)

Name

Integer Get_drainage_pit_float(Element elt,Integer pit,Integer &float)

Description

Set_drainage_pit_float(Element elt,Integer pit,Integer float)

Name

Integer Set_drainage_pit_float(Element elt,Integer pit,Integer float)

Description

Get_drainage_pit_inverts(Element elt,Integer p,Real &lhs,Real &rhs)

Name

Integer Get_drainage_pit_inverts(Element elt,Integer p,Real &lhs,Real &rhs)

Description

Get the invert levels for the **p**th pit of the string Element **elt**.

The downstream invert level of the pit is returned in Real **lhs**.

The upstream invert level of the pit is returned in Real **rhs**.

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

Set_drainage_pit_inverts(Element elt,Integer p,Real lhs,Real rhs)

Name

Integer Set_drainage_pit_inverts(Element elt,Integer p,Real lhs,Real rhs)

Description

Set the invert levels for the pth pit of the string Element **elt**.

The downstream invert level of the pit is given as Real **lhs**.

The upstream invert level of the pit is given as Real **rhs**.

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

Get_drainage_pit_hgls(Element elt,Integer p,Real &lhs,Real &rhs)

Name

Integer Get_drainage_pit_hgls(Element elt,Integer p,Real &lhs,Real &rhs)

Description

Get the hgl levels for the pth pit of the string Element **elt**.

The hgl level of the left side of the pit is returned in Real **lhs**.

The hgl level of the right side of the pit is returned in Real **rhs**.

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

Set_drainage_pit_hgls(Element elt,Integer p,Real lhs,Real rhs)

Name

Integer Set_drainage_pit_hgls(Element elt,Integer p,Real lhs,Real rhs)

Description

Set the hgl levels for the pth pit of the string Element **elt**.

The hgl level of the left side of the pit is given as Real **lhs**.

The hgl level of the right side of the pit is given as Real **rhs**.

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

Get_drainage_pit_hgl(Element elt,Integer pit,Real &hgl)

Name

Integer Get_drainage_pit_hgl(Element elt,Integer pit,Real &hgl)

Description

Get the hgl level for centre of the pth pit of the string Element **elt**.

The hgl level of the centre of the pit is returned in Real **hgl**.

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

Set_drainage_pit_hgl(Element element,Integer pit,Real hgl)

Name

Integer Set_drainage_pit_hgl(Element element,Integer pit,Real hgl)

Description

Get_drainage_pit_name(Element elt,Integer p,Text &name)

Name

Integer Get_drainage_pit_name(Element elt,Integer p,Text &name)

Description

Get the name for the **p**th pit of the string Element **elt**.

The name of the pit is returned in Text **name**.

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

Set_drainage_pit_name(Element elt,Integer p,Text name)

Name

Integer Set_drainage_pit_name(Element elt,Integer p,Text name)

Description

Set the name for the **p**th pit of the string Element **elt**.

The name of the pit is given as Text **name**.

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

Get_drainage_pit_road_chainage(Element elt,Integer p,Real &chainage)

Name

Integer Get_drainage_pit_road_chainage(Element elt,Integer p,Real &chainage)

Description

Get the road chainage for the **p**th pit of the string Element **elt**.

The road chainage of the pit is returned in Real **chainage**.

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

Set_drainage_pit_road_chainage(Element elt,Integer p,Real chainage)

Name

Integer Set_drainage_pit_road_chainage(Element elt,Integer p,Real chainage)

Description

Set the road chainage for the pth pit of the string Element **elt**.

The road chainage of the pit is given as Real **chainage**.

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

Get_drainage_pit_road_name(Element elt,Integer p,Text &name)

Name

Integer Get_drainage_pit_road_name(Element elt,Integer p,Text &name)

Description

Get the road name for the **p**th pit of the string Element **elt**.

The road name of the pit is returned in Text **name**.

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

Set_drainage_pit_road_name(Element elt,Integer p,Text name)

Name

Integer Set_drainage_pit_road_name(Element elt,Integer p,Text name)

Description

Set the road name for the **p**th pit of the string Element **elt**.

The road name of the pit is given as Text **name**.

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

Get_drainage_pit_type(Element elt,Integer p,Text &type)

Name

Integer Get_drainage_pit_type(Element elt,Integer p,Text &type)

Description

Get the type for the **p**th pit of the string Element **elt**.

The type of the pit is returned in Text **type**.

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

Set_drainage_pit_type(Element elt,Integer p,Text type)

Name

Integer Set_drainage_pit_type(Element elt,Integer p,Text type)

Description

Set the type for the **p**th pit of the string Element **elt**.

The type of the pit is given as Text **type**.

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

Get_drainage_pit_branches(Element elt,Integer pit,Dynamic_Element &branches)

Name

Integer Get_drainage_pit_branches(Element elt,Integer pit,Dynamic_Element &branches)

Description

Get_drainage_pit_chainage(Element elt,Integer p,Real &chainage)

Name

Integer Get_drainage_pit_chainage(Element elt,Integer p,Real &chainage)

Description

Get the chainage for the **p**th pit of the string Element **elt**.

The chainage of the pit is returned in Real **chainage**.

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

Get_drainage_pit_depth(Element elt,Integer p,Real &depth)

Name

Integer Get_drainage_pit_depth(Element elt,Integer p,Real &depth)

Description

Get the depth of the **p**th pit of the string Element **elt**.

The depth of the pit is returned in Real **depth**.

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

Get_drainage_pit_drop(Element elt,Integer p,Real &drop)

Name

Integer Get_drainage_pit_drop(Element elt,Integer p,Real &drop)

Description

Get the drop through the pth pit of the string Element **elt**.

The drop through the pit is returned in Real **drop**.

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

Get_drainage_pits(Element elt,Integer &npits)

Name

Integer Get_drainage_pits(Element elt,Integer &npits)

Description

Get the number of pits for the string Element **elt**.

The number of pits is returned in Integer **npits**.

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

Get_drainage_pit_ns(Element elt,Integer n,Real &ht)

Name

Integer Get_drainage_pit_ns(Element elt,Integer n,Real &ht)

Description

For the drainage string **elt,** get the height from the natural surface tin at the location of the centre of the **n**th pit.

The height of the natural surface is returned in **ht**.

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

Get_drainage_pit_fs(Element elt,Integer n,Real &ht)

Name

Integer Get_drainage_pit_fs(Element elt,Integer n,Real &ht)

Description

For the drainage string **elt,** get the height from the finished surface tin at the location of the centre of the **n**th pit.

The height of the finished surface is returned in **ht**.

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

Get_drainage_number_of_manhole_types(Integer &n)

Name

Integer Get_drainage_number_of_manhole_types(Integer &n)

Description

Get the number of manhole (pit) types from the drainage.4d file and return the number in *n*.

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

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.

Get_drainage_manhole_length(Text type,Real &length)

Name

Integer Get_drainage_manhole_length(Text type,Real &length)

Description

Get the *length* of the manhole of type **type** from the drainage.4d file and return the length in ***length***.

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.

Get_drainage_manhole_width(Text type,Real &width)

Name

Integer Get_drainage_manhole_width(Text type,Real &width)

Description

Get the *width* of the manhole of type **type** from the drainage.4d file and return the width in ***width***.

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.

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.

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.

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.

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

type is the name of the manhole type.

values from attributes cap_multi, cap_fixed, cap_percent, cap_coeff, cap_power

if undefined, defaults are 1, 0,0,0,1

0 is ok

Get_drainage_number_of_sag_curves(Text type,Integer &n)

Name

Integer Get_drainage_number_of_sag_curves(Text type,Integer &n)

Description

What ever is in drainage.4d.

What is type - manhole type in drainage 4d.

It is the number of sag curves. in drainge.4d cap_curve_sag

0 is ok

Get_drainage_sag_curve_name(Text type,Text &name)

Name

Integer Get_drainage_sag_curve_name(Text type,Text &name)

Description

??maybe there is only one sag curve allowed ??

0 is ok

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

type is the name of the manhole type.

values from attributes cap_multi, cap_fixed, cap_percent, cap_ceoff, cap_power

if undefined, defaults are 1, 0,0,0,1

0 is ok

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

type is the name of the manhole type.

get back the number of coords

0 is ok

Get_drainage_sag_curve_coords(Text type,Real Depth[],Real Qin[],Integer nmax,Integer &n)

Name

Integer Get_drainage_sag_curve_coords(Text type,Real Depth[],Real Qin[],Integer nmax,Integer &n)

Description

type is the name of the manhole type.

get back the coords

Get_drainage_number_of_grade_curves(Text type,Integer &n)

Name

Integer Get_drainage_number_of_grade_curves(Text type,Integer &n)

Description

get number of grade curves

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

for manhole named type, and ith curve, get name of curve

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

for manhole named type, name of name, get road_grade, road_crossfall,

road_grade keyword in darainage.4d. if there then by_grade are set and to be used

road_crossfall keyword in darainage.4d. if there then by_cross are set and to be used.

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

for manhole named type, and for name of name, get cap_ etc,

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

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

Get_drainage_manhole_config(Text type,Text &cap_config)

Name

Integer Get_drainage_manhole_config(Text type,Text &cap_config)

Description

for manhole of type type, ca_config is "g" - on grade pit, "s" - ag pit, or "m" manole sealed pit.

if not g, s, m then it returns an error.

Get_drainage_manhole_diam(Text type,Real &diameter)

Name

Integer Get_drainage_manhole_diam(Text type,Real &diameter)

Description

for manhole type, it sets a diameter.

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.**

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.

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.

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.

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.

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.

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.

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**.

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.

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.

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.

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.

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.

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.

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.

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.

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**.

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.

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**.

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.

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.

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**.

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**.

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**.

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**.

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**.

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.

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 4DML function return values.

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 4DML function return values.

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.

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.

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.

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.

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.

Drainage String House Connections - Only Available for the Sewer Module

Get_drainage_hcs(Element elt,Integer &no_hcs)

Name

Integer Get_drainage_hcs(Element elt,Integer &no_hcs)

Description

Get the number of house connections for the string Element **elt**.

The number of house connection is returned in Integer **no_hcs**.

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

Get_drainage_hc(Element elt,Integer h,Real &x,Real &y,Real &z)

Name

Integer Get_drainage_hc(Element elt,Integer h,Real &x,Real &y,Real &z)

Description

Get the x,y & z for the **h**th house connection of the string Element **elt**.

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.

Get_drainage_hc_adopted_level(Element elt,Integer h,Real &level)

Name

Integer Get_drainage_hc_adopted_level(Element elt,Integer h,Real &level)

Description

Get the adopted level for the **h'**th house connection of the string Element elt.

The adopted level of the house connection is returned in Real **level**.

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

Set_drainage_hc_adopted_level(Element elt,Integer hc,Real level)

Name

Integer Set_drainage_hc_adopted_level(Element elt,Integer hc,Real level)

Description

Get_drainage_hc_bush(Element elt,Integer h,Text &bush)

Name

Integer Get_drainage_hc_bush(Element elt,Integer h,Text &bush)

Description

Get the bush type for the **h'**th house connection of the string Element **elt**.

The bush type of the house connection is returned in Text **bush**.

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

Set_drainage_hc_bush(Element elt,Integer hc,Text bush)

Name

Integer Set_drainage_hc_bush(Element elt,Integer hc,Text bush)

Description

Get_drainage_hc_colour(Element elt,Integer h,Integer &colour)

Name

Integer Get_drainage_hc_colour(Element elt,Integer h,Integer &colour)

Description

Get the colour for the **h'**th house connection of the string Element **elt**.

The colour of the house connection is returned in Integer **colour**.

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

Set_drainage_hc_colour(Element elt,Integer hc,Integer colour)

Name

Integer Set_drainage_hc_colour(Element elt,Integer hc,Integer colour)

Description

Get_drainage_hc_depth(Element elt,Integer h,Real &depth)

Name

Integer Get_drainage_hc_depth(Element elt,Integer h,Real &depth)

Description

Get the depth for the **h'**th house connection of the string Element **elt**.

The depth of the house connection is returned in Real **depth**.

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

Set_drainage_hc_depth(Element elt,Integer hc,Real depth)

Name

Integer Set_drainage_hc_depth(Element elt,Integer hc,Real depth)

Description

Get_drainage_hc_diameter(Element elt,Integer h,Real &diameter)

Name

Integer Get_drainage_hc_diameter(Element elt,Integer h,Real &diameter)

Description

Get the diameter for the **h'**th house connection of the string Element **elt**.

The diameter of the house connection is returned in Real **diameter**.

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

Set_drainage_hc_diameter(Element elt,Integer hc,Real diameter)

Name

Integer Set_drainage_hc_diameter(Element elt,Integer hc,Real diameter)

Description

Get_drainage_hc_grade(Element elt,Integer h,Real &grade)

Name

Integer Get_drainage_hc_grade(Element elt,Integer h,Real &grade)

Description

Get the grade for the **h'**th house connection of the string Element **elt**.

The grade of the house connection is returned in Real **grade**.

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

Set_drainage_hc_grade(Element elt,Integer hc,Real grade)

Name

Integer Set_drainage_hc_grade(Element elt,Integer hc,Real grade)

Description

Get_drainage_hc_hcb(Element elt,Integer h,Integer &hcb)

Name

Integer Get_drainage_hc_hcb(Element elt,Integer h,Integer &hcb)

Description

Get the hcb for the **h'**th house connection of the string Element **elt**.

The hcb of the house connection is returned in Integer **hcb**.

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

Set_drainage_hc_hcb(Element elt,Integer hc,Integer hcb)

Name

Integer Set_drainage_hc_hcb(Element elt,Integer hc,Integer hcb)

Description

Get_drainage_hc_length(Element elt,Integer h,Real &length)

Name

Integer Get_drainage_hc_length(Element elt,Integer h,Real &length)

Description

Get the length for the **h'**th house connection of the string Element **elt**.

The length of the house connection is returned in Real **length**.

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

Set_drainage_hc_length(Element elt,Integer hc,Real length)

Name

Integer Set_drainage_hc_length(Element elt,Integer hc,Real length)

Description

Get_drainage_hc_level(Element elt,Integer h,Real &level)

Name

Integer Get_drainage_hc_level(Element elt,Integer h,Real &level)

Description

Get the level for the **h'**th house connection of the string Element **elt**.

The level of the house connection is returned in Real **level**.

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

Set_drainage_hc_level(Element elt,Integer hc,Real level)

Name

Integer Set_drainage_hc_level(Element elt,Integer hc,Real level)

Description

Get_drainage_hc_material(Element elt,Integer h,Text &material)

Name

Integer Get_drainage_hc_material(Element elt,Integer h,Text &material)

Description

Get the material for the **h'**th house connection of the string Element **elt**.

The material of the house connection is returned in Text **material**.

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

Set_drainage_hc_material(Element elt,Integer hc,Text material)

Name

Integer Set_drainage_hc_material(Element elt,Integer hc,Text material)

Description

Get_drainage_hc_name(Element elt,Integer h,Text &name)

Name

Integer Get_drainage_hc_name(Element elt,Integer h,Text &name)

Description

Get the name for the **h'**th house connection of the string Element **elt**.

The name of the house connection is returned in Text **name**.

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

Set_drainage_hc_name(Element elt,Integer hc,Text name)

Name

Integer Set_drainage_hc_name(Element elt,Integer hc,Text name)

Description

Get_drainage_hc_side(Element elt,Integer h,Integer &side)

Name

Integer Get_drainage_hc_side(Element elt,Integer h,Integer &side)

Description

Get the side for the **h'**th house connection of the string Element **elt**.

The side of the house connection is returned in Integer **side**.

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

Note:

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.

Set_drainage_hc_side(Element elt,Integer hc,Integer side)

Name

Integer Set_drainage_hc_side(Element elt,Integer hc,Integer side)

Description

Get_drainage_hc_type(Element elt,Integer h,Text &type)

Name

Integer Get_drainage_hc_type(Element elt,Integer h,Text &type)

Description

Get the type for the **h'**th house connection of the string Element **elt**.

The type of the house connection is returned in Text **type**.

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

Set_drainage_hc_type(Element elt,Integer hc,Text type)

Name

Integer Set_drainage_hc_type(Element elt,Integer hc,Text type)

Description

Get_drainage_hc_chainage(Element elt,Integer h,Real &chainage)

Name

Integer Get_drainage_hc_chainage(Element elt,Integer h,Real &chainage)

Description

Get the chainage for the **h'**th house connection of the string Element **elt**.

The chainage of the house connection is returned in Real **chainage**.

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

Get_drainage_hc_ip(Element elt,Integer h,Integer &ip)

Name

Integer Get_drainage_hc_ip(Element elt,Integer h,Integer &ip)

Description

Get the intersect point for the **h'**th house connection of the string Element **elt**.

The intersection point of the house connection is returned in Integer **ip**.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Face Strings

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Plot Frames

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Feature String

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.

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.

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.

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.

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.

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.

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.

Super String Element

See Super String Dimensions and Flags
See Flags and Dimension Combinations

See Super String Functions
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 Vertex Symbol Functions
See Super String Solid/Bitmap/Hatch/Fill/Pattern/ACAD Pattern Functions
See Super String Hole Functions
See Super String Vertex Text Functions
See Super String Vertex Annotation Functions
See Super String Segment Text Functions
See Super String Segment Annotation Functions
See Super String Tinability Functions
See Super String Point Id 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 Visibility Functions

Super String Dimensions and Flags

The super string is intended as a replacement of the following string types:

2d, 3d, 4d, interface, face, pipe and polyline.

The super string covers all these string types and many more combinations that were never allowed for the other strings.

For example, users wanted to be able to have a polyline string but with a pipe diameter, or a 2d string with text at each vertex. To cover every combination that the users required would mean thousands of different string types.

The solution is to offer a string that has optional dimensions to cover all of the properties of the other strings. For example, the super string has two mutually exclusive dimensions called 2d level and 3d level. This can cover the functionality offered by both the 2d string and the 3d string.

When a dimension does not exist in the super string, there is no storage used and hence a 2d super string only requires the same memory as a 2d string.

The super string supports over 50 different dimensions, of which only two are mandatory dimensions (the x & y coordinates). Every other dimension is optional.

Before using any functionality, the super string must be told that a particular dimension is required and there are function calls to set each dimension (*use* calls).

The list of dimensions follows below with the names that are defined, and the actual number that the name has. Either the name or the number can be used in calls requiring a super string dimension.

Please note that where two dimensions are listed on one line, this means that only one or no dimension may exist, but not both. (Strictly speaking, they can both exist but the array dimension takes precedence over the value dimension, and the super string may compress or remove the value dimension.)

Note - although there are calls to set each of the dimensions individually, it is possible to set more than one dimension at once using flags (see Flags and Dimension Combinations)

a

The calls for each dimension are grouped together. There are also general super string creation and data setting calls documented in Super String Functions and Element Operations .

For information on the Super String Dimensions:

See Height Dimensions
See Segment Radius Dimension
See Pipe/Culvert Dimensions
See Solid/Bitmap/Hatch/ Fill/Pattern/ACAD Pattern Dimensions
See Hole Dimension
See Text Dimensions
See Text Annotation Dimensions
See Vertex Symbol Dimensions
See User Defined Attributes Dimensions
See Tinability Dimensions
See Visibility Dimensions
See Colour Dimension
See Point Id Dimension
See Vertex Image Dimensions
See Segment Geometry Dimension
See Matrix Dimension
See UID Dimensions
See Att Database Point Dimensions
See Extrude Dimensions
See Att Null Levels Dimensions

See information on setting more than one dimension at once, Flags and Dimension Combinations

For information on the Super String function calls:

See Super String Dimensions and Flags
See Flags and Dimension Combinations
See Super String Functions
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 Vertex Symbol Functions
See Super String Solid/Bitmap/Hatch/Fill/Pattern/ACAD Pattern Functions
See Super String Hole Functions
See Super String Vertex Text Functions
See Super String Vertex Annotation Functions
See Super String Segment Text Functions
See Super String Segment Annotation Functions
See Super String Tinability Functions
See Super String Point Id 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 Visibility Functions

Height Dimensions

Att_ZCoord_Value 1 or Att_ZCoord_Array 2

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

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.

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

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 has a diameter for each segment.

If Att_Diameter_Value is set and Att_Diameter_Array not set, then the super string has one diameter value for the entire string.

Att_Culvert_Value 24 or Att_Culvert_Array 25

If Att_Culvert_Array is set, then the super string has a width and height for each segment.

If Att_Att_Culvert_Value is set and Att_Att_Culvert_Array not set, then the super string has one width and height for the entire string.

If none of the Pipe/Culvert dimensions exist, then the string has no thickness. 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

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 Solid/Bitmap/Hatch/Fill/Pattern/ACAD Pattern Functions

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

Text Dimensions

Att_Vertex_Text_Value 10 or Att_Vertex_Text_Array 7

If Att_Vertex_Array is set, then the super string can have different text at each vertex.

If Att_Vertex_Value is set and Att_Vertex_Array not set, then the super string has the one text for each vertex of the string.

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 one text for each segment of the string.

Note that it is possible to have text associated with a vertex/segment but it is not visible. To be able to draw the text, see **Text Annotation Dimensions** below.

See Super String Segment Text Functions
See Super String Vertex Text Functions

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 text on 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 Annotation Functions

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 text on 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 Annotation Functions

Vertex Symbol Dimensions

Att_Symbol_Value 17 or Att_Symbol_Array 18

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

User Defined 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.

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 Vertex Attributes Functions
See Super String Segment Attributes Functions

Tinability Dimensions

Att_Contour_Array 3 This dimension applies for both vertex and segment tinability.

Att_Vertex_Tinable_Value 37 or Att_Vertex_Tinable_Array 38

If Att_VertexTinable_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

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

Colour Dimension

Att_Colour_Array 9 LJG? For a colour for each segment (what about vertex?)

See Super String Segment Colour Functions

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

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

Segment Geometry Dimension

Att_Geom_Array 32 allow transitions for segments

If Att_Geom_Array is set, then the super string then each segment can be a line, arc or a transition.

See Super String Segment Geometry Functions

Matrix Dimension

Att_Matrix_Value 53 ?

UID Dimensions

Att_Vertex_UID_Array 35

Att_Segment_UID_Array 36

See Super String Uid Functions

Att Database Point Dimensions

Att_Database_Point_Array 47

Extrude Dimensions

Att_Extrude_Value 48

Att_Interval_Value 50

See Super String Extrude Functions

Att 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 Flags and Dimension

Combinations .

Flags and Dimension Combinations

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.

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 once 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. ZCoord_Value to Geom_Array)

then the definition

```
  Integer flags1 = String_Super_Bit(ZCoord_Value) | String_Super_Bit(Radius_Array);
```

means that **flag1** represents having the two dimensions ZCoord_Value and Radius_Array

// If **flag2** holds dimensions 32 to 64 (i.e. 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 Pattern_Value and Vertex_Tinable_Array

As an 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);
```

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. As for other string types, there are a number of calls to create and load bulk data into a super string.

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.

Create_super(Integer flag1,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts)

Name

Element Create_super(Integer flag1,Real x[],Real y[],Real z[],Real r[],Integer f[],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)**, f** (segment is bulged or not).

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, f** arrays may or may not be used, but the arrays must still be supplied. See Super String Dimensions and Flags 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 f[],Integer num_pts).*

Create_super(Integer flag1,Integer flag2,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts)

Name

Element Create_super(Integer flag1,Integer flag2,Real x[],Real y[],Real z[],Real r[],Integer f[],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)**, f** (segment is bulged or not).

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, f** arrays may or may not be used, but the arrays must still be supplied. See Super String Dimensions and Flags 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.

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 and Flags 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).*

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 and Flags 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.

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.

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 and Flags 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).*

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 and Flags 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.

Get_super_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer max_pts,Integer &num_pts)

Name

Integer Get_super_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],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 **elt**.

The (x,y,z,r,f) values at each string vertex are returned in the Real arrays **x[]**, **y[],z[],r[]** and **f[]**

(the arrays are x values**,** y values**,** z values**,** radius of segments**, f** segment is bulged or not).

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 **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.

Get_super_data(Element super,Real x[],Real y[],Real z[],Real r[],Integer f[], 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 f[], Integer max_pts,Integer &num_pts,Integer start_pt)

Description

For a super string Element **super**, get the (x,y,z,r,f) data for **max_pts** vertices starting at vertex number **start_pt** (the arrays are x values**,** y values**,** z values**,** radius of segments**, f** segment is bulged or not).

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,f) values at each string vertex are returned in the Real arrays **x[]**, **y[],z[],r[]** and **f[]**.

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.

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_super_data(Element super,Integer i,Real &x,Real &y,Real &z,Real &r, Integer &f)

Name

Integer Get_super_data(Element super,Integer i,Real &x,Real &y,Real &z,Real &r,Integer &f)

Description

Get the (x,y,z,r,f) data for the ith 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 bulge flag value is returned in Integer **f**.

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

Set_super_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[], Integer num_pts)

Name

Integer Set_super_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** vertices of the string 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,r,f) values for each string vertex are given in the Real arrays **x[]**, **y[],z[],r[]** and **f[]**.

The number of vertices to be set is given by Integer **num_pts**

If the Element **elt** 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.

Note

This function can not create new super Elements but only modify existing super Elements.

Set_super_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts,Integer start_pt)

Name

Integer Set_super_data(Element elt,Real x[],Real y[],Real z[],Real r[],Integer f[],Integer num_pts,Integer start_pt)

Description

For the super Element **elt**, set the (x,y,z,r,f) 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 **f[]**.

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 **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 3d Elements but only modify existing 3d Elements.

Set_super_data(Element elt,Integer i,Real x,Real y,Real z,Real r,Integer f)

Name

Integer Set_super_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 vertex of the super 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 bulge flag value is returned in Integer **f (**0 for no bulge, non zero for a bulge).

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

Get_super_vertex_coord(Element super,Integer vert,Real &x,Real &y,Real &z)

Name

Integer Get_super_vertex_coord(Element super,Integer vert,Real &x,Real &y,Real &z)

Description

A return value of 0 indicates the function call was successful.

Set_super_vertex_coord(Element super,Integer vert,Real x,Real y,Real z)

Name

Integer Set_super_vertex_coord(Element super,Integer vert,Real x,Real y,Real z)

Description

A return value of 0 indicates the function call was successful.

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.

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.

Set_super_segment_world_text(Element)

Name

Integer Set_super_segment_world_text(Element)

Description

A return value of 0 indicates the function call was successful.

<no description>

Set_super_segment_device_text(Element)

Name

Integer Set_super_segment_device_text(Element)

Description

A return value of 0 indicates the function call was successful.

<no description>

Super String Height Functions

For definitions of the height dimensions, see Height Dimensions

Get_super_use_2d_level(Element elt,Integer &use)

Name

Integer Get_super_use_2d_level(Element elt,Integer &use)

Description

Query whether the dimension Att_ZCoord_Value exists for the super string **elt**.

See Super String Dimensions and Flags for information on dimensions and Height Dimensions for information on Height dimensions.

use is returned as 1 if the dimension exists, or 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

Set_super_use_2d_level(Element elt,Integer use)

Name

Integer Set_super_use_2d_level(Element elt,Integer use)

Description

For the super string Element **elt**, define whether the dimension Att_ZCoord_Value is used.

See Super String Dimensions and Flags for information on dimensions and Height Dimensions for information on Height dimensions.

If **use** is 1, the dimension is set. If **use** is 0, the dimension is removed.

Note that if the dimension Att_ZCoord_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

Get_super_use_3d_level(Element elt,Integer &use)

Name

Integer Get_super_use_3d_level(Element elt,Integer &use)

Description

Query whether the dimension Att_ZCoord_Array exists for the super string **elt**.

See Super String Dimensions and Flags for information on dimensions and Height Dimensions for information on Height dimensions.

use is returned as 1 if the dimension exists, or 0 if the dimension doesn't exist.

A return value of 0 indicates the function call was successful.

Set_super_use_3d_level(Element elt,Integer use)

Name

Integer Set_super_use_3d_level(Element elt,Integer use)

Description

For the super string Element **elt**, define whether the dimension Att_ZCoord_Array is used.

See Super String Dimensions and Flags for information on dimensions and Height Dimensions

for information on Height dimensions.

If **use** is 1, the dimension is set. If **use** is 0, the dimension is removed.

Note that if the dimension Att_ZCoord_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

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 dimension Att_ZCoord_Value is set, then the z-value for the entire string is returned in **level**.

See Super String Dimensions and Flags for information on dimensions and Height Dimensions for information on Height 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.

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, then the z-value for the entire string is set to **level**.

See Super String Dimensions and Flags for information on dimensions and Height Dimensions for information on Height 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.

Super String Segment Colour Functions

For definitions of the Colour dimension, see Colour Dimension

Get_super_use_segment_colour(Element super,Integer &use)

Name

Integer Get_super_use_segment_colour(Element super,Integer &use)

Description

Query whether the dimension Att_Colour_Array exists for the super string. A value for **use** of 1 indicates the dimension exists.

See Super String Dimensions and Flags for information on dimensions and Colour Dimension for information on the Colour dimension.

A return value of 0 indicates the function call was successful.

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 the dimension Att_Colour_Array. A value for **use** of 1 sets the dimension and 0 removes it.

See Super String Dimensions and Flags for information on dimensions and Colour Dimension for information on the Colour dimension.

A return value of 0 indicates the function call was successful.

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 dimension Att_Colour_Array set.

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

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 dimension Att_Colour_Array set.

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

Super String Segment Radius Functions

For definitions of the Segment Radius dimensions, see Segment Radius Dimension

Get_super_use_segment_radius(Element super,Integer &use)

Name

Integer Get_super_use_segment_radius(Element super,Integer &use)

Description

Query whether the dimension Att_Radius_Array exists for the super string. A value for **use** of 1 indicates the dimension exists.

See Super String Dimensions and Flags for information on dimensions and Segment Radius Dimension for information on the Segment Radius dimension.

A return value of 0 indicates the function call was successful.

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 **elt**, define whether the dimension Att_Radius_Array is used.

See Super String Dimensions and Flags for information on dimensions and Segment Radius Dimension for information on Segment Radius dimensions.

If **use** is 1, the dimension is set. If **use** is 0, the dimension is removed.

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.

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 Super String Dimensions and Flags for information on dimensions and Segment Radius Dimension for information on the Segment Radius dimension.

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.

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 Super String Dimensions and Flags for information on dimensions and Segment Radius Dimension for information on the Segment Radius dimension.

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.

Get_super_segment_major(Element super,Integer seg,Integer &major)

Name

Integer Get_super_segment_major(Element super,Integer seg,Integer &major)

Description

For the super string **super**, get the major value of segment number **seg** and return the value in **major**.

See Super String Dimensions and Flags for information on dimensions and Segment Radius Dimension for information on the Segment Radius dimension.

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.

Set_super_segment_major(Element super,Integer seg,Integer major)

Name

Integer Set_super_segment_major(Element super,Integer seg,Integer major)

Description

For the super string **super**, set the major value of segment number **seg** to the value **major**.

See Super String Dimensions and Flags for information on dimensions and Segment Radius Dimension for information on the Segment Radius dimension.

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.

Super String Pipe/Culvert Functions

For definitions of the Pipe and Culvert dimensions, see Pipe/Culvert Dimensions
A super string can be a super pipe string or a super culvert string. It can't be both.

As a super pipe string, it can have either one diameter for all segments of the string, or it can have different diameters for each segment of the string.

As a super culvert string, it can have either one width and one height for all segments of the string, or it can have different heights and widths for each segment of the string.

See Super String Pipe/Culvert Justify Functions
See Super String Pipe Functions
See Super String Culvert Functions

Super String Pipe/Culvert Justify Functions

Get_super_use_pipe_justify(Element super,Integer &use)

Name

Integer Get_super_use_pipe_justify(Element super,Integer &use)

Description

Query whether the dimension Att_Pipe_Justify exists for the Element **super** of type **Super**.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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 pipe or a culvert.

A return value of 0 indicates the function call was successful.

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 dimension Att_Pipe_Justify is used.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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 pipe or a culvert.

A return value of 0 indicates the function call was successful.

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**.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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.

A return value of 0 indicates the function call was successful

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**.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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.

A return value of 0 indicates the function call was successful

Super String Pipe Functions

Get_super_use_diameter(Element elt,Integer &use) for V9

Get_super_use_pipe(Element elt,Integer &use) for V10 onwards

Name

Integer Get_super_use_diameter(Element elt,Integer &use)

Integer Get_super_use_pipe(Element elt,Integer &use)

Description

This function has the new name for V10 onwards. The old call will still work.

Query whether the dimension Att_Diameter_Value exists for the super string **elt**.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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).

A return value of 0 indicates the function call was successful.

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.

Set_super_use_diameter(Element elt,Integer use) for V9

Set_super_use_pipe(Element elt,Integer use) for V10 onwards

Name

Integer Set_super_use_diameter(Element elt,Integer use)

Integer Set_super_use_pipe(Element elt,Integer use)

Description

This function has the new name for V10 onwards. The old call will still work.

For the super string Element **elt**, define whether the dimension Att_Diameter_Value is used.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions.

If **use** is 1, the dimension is set That is, the pipe has one diameter 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.

A return value of 0 indicates the function call was successful.

Get_super_use_segment_diameter(Element elt,Integer &use) for V9

Get_super_use_segment_pipe(Element elt,Integer &use) for V10 onward

Name

Integer Get_super_use_segment_diameter (Element elt,Integer &use)

Integer Get_super_use_segment_pipe (Element elt,Integer &use)

Description

This function has the new name for V10 onwards. The old call will still work.

Query whether the dimension Att_Diameter_Array exists for the super string **elt**.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions.

use is returned as 1 if the dimension exists.
use is returned as 0 if the dimension doesn't exist, or it has a constant diameter (i.e. Att_Diameter_Value exists).

A return value of 0 indicates the function call was successful.

Set_super_use_segment_diameter(Element elt,Integer use) for V9

Set_super_use_segment_pipe(Element elt,Integer use) for V10 onwards

Name

Integer Set_super_use_segment_diameter(Element elt,Integer use)

Integer Set_super_use_segment_pipe(Element elt,Integer use)

Description

This function has the new name for V10 onwards. The old call will still work.

For the super string Element **elt**, define whether the dimension Att_Diameter_Array is used.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions.

If **use** is 1, the dimension is set. That is, each pipe segment can have a different diameter.
If **use** is 0, the dimension is removed.

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.

Get_super_diameter(Element super,Real &diameter) for V9

Get_super_pipe(Element super,Real &diameter) for V10 onwards

Name

Integer Get_super_diameter(Element super,Real &diameter)

Integer Get_super_pipe(Element super,Real &diameter)

Description

This function has the new name for V10 onwards. The old call will still work.

For the Element **super** of type **Super** which is a constant diameter pipe string (i.e. Att_Diameter_Value has been set), get the pipe diameter and return it in **diameter**.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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_pipe can be called to make sure it is constant diameter pipe string.

Set_super_diameter(Element elt,Real diameter) for V9

Set_super_pipe(Element elt,Real diameter) for V10 and above

Name

Integer Set_super_diameter (Element elt,Real diameter)

Integer Set_super_pipe (Element elt,Real diameter)

Description

This function has the new name for V10 onwards. The old call will still work.

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), set the diameter to **diameter**.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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_pipe can be called to make sure it is constant diameter pipe string.

Get_super_segment_diameter(Element elt,Integer seg,Real &diameter) for V9

Get_super_segment_pipe(Element elt,Integer seg,Real &diameter) for V10 onward

Name

Integer Get_super_segment_diameter(Element elt,Integer seg,Real &diameter)

Integer Get_super_segment_pipe(Element elt,Integer seg,Real &diameter)

Description

This function has the new name for V10 onwards. The old call will still work.

For the super Element **elt**, get the pipe diameter for segment number **seg** and return it in **diameter**.

For V10, if **elt** 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.

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.

Set_super_segment_diameter(Element elt,Integer seg,Real diameter) for V9

Set_super_segment_pipe(Element elt,Integer seg,Real diameter) for V10 onwards

Name

Integer Set_super_segment_diameter(Element elt,Integer seg,Real diameter)

Integer Set_super_segment_pipe(Element elt,Integer seg,Real diameter)

Description

This function has the new name for V10 onwards. The old call will still work.

For the super Element **elt**, set the pipe diameter for segment number **seg** to **diameter**.

For V10, if **elt** 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.

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.

Super String Culvert Functions

Get_super_use_culvert(Element super,Integer &use)

Name

Integer Get_super_use_culvert(Element super,Integer &use)

Description

Query whether the dimension Att_Culvert_Value exists for the super string.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 the dimension Att_Culvert_Value.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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.

Get_super_use_segment_culvert(Element super,Integer &use)

Name

Integer Get_super_use_segment_culvert(Element super,Integer &use)

Description

Query whether the dimension Att_Culvert_Array exists for the super string.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 the dimension Att_Culvert_Array.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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.

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 dimension flag Att_Culvert_Value has been set), get the culvert width and height and return them in **w** and **h** respectively.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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 constant culvert string.

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 dimension flag Att_Culvert_Value has been set), set the culvert width to **w** and the height to **h**.

See Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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.

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 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 Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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 variable segment culvert string.

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 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 Super String Dimensions and Flags for information on dimensions and Pipe/Culvert Dimensions for information on the Pipe/Culvert 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 variable segment culvert string.

Super String Vertex Symbol Functions

For definitions of the Vertex Symbols dimensions, see Vertex Symbol Dimensions

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

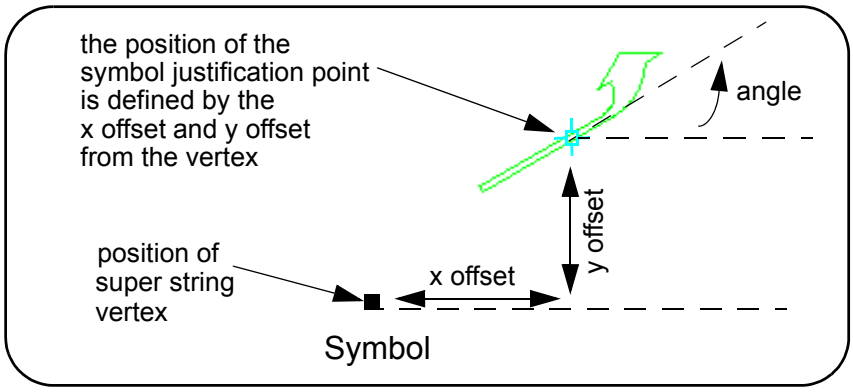
(c) angle of rotation of the symbol

(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.

Get_super_use_symbol(Element super,Integer &use)

Name

Integer Get_super_use_symbol(Element super,Integer &use)

Description

Query whether the dimension Att_Symbol_Value exists for the Element **super** of type **Super**.

See Super String Dimensions and Flags for information on dimensions and Vertex Symbol Dimensions for information on the Vertex Symbol 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.

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 dimension Att_Symbol_Value is used.

See Super String Dimensions and Flags for information on dimensions and Vertex Symbol Dimensions for information on the Vertex Symbol 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.

Get_super_use_vertex_symbol(Element super,Integer &use)

Name

Integer Get_super_use_vertex_symbol(Element super,Integer &use)

Description

Query whether the dimension Att_Symbol_Array exists for the super string.

See Super String Dimensions and Flags for information on dimensions and Vertex Symbol Dimensions for information on the Vertex Symbol dimensions.

A value **for** use of 1 indicates the dimension exists. That is, the super string has a different symbol on each vertex.

A return value of 0 indicates the function call was successful.

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 dimension Att_Symbol_Array is used.

See Super String Dimensions and Flags for information on dimensions and Vertex Symbol Dimensions for information on the Vertex Symbol 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.

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**.

A return value of 0 indicates the function call was successful.

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**.

A return value of 0 indicates the function call was successful.

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**.

A return value of 0 indicates the function call was successful.

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**.

A return value of 0 indicates the function call was successful.

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**.

A return value of 0 indicates the function call was successful.

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**.

A return value of 0 indicates the function call was successful.

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.

A return value of 0 indicates the function call was successful.

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.

A return value of 0 indicates the function call was successful.

Get_super_vertex_symbol_size(Element super,Integer vert,Real &s)

Name

Integer Get_super_vertex_symbol_size(Element super,Integer vert,Real &s)

Description

For the super Element **super**, return as **s** the size of the symbol on vertex number **vert**.

A return value of 0 indicates the function call was successful.

Set_super_vertex_symbol_size(Element super,Integer vert,Real s)

Name

Integer Set_super_vertex_symbol_size(Element super,Integer vert,Real s)

Description

For the super Element **super**, set the size of the symbol on vertex number **vert** to be **s**.

A return value of 0 indicates the function call was successful.

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**.

A return value of 0 indicates the function call was successful.

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**.

A return value of 0 indicates the function call was successful.

Super String Solid/Bitmap/Hatch/Fill/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 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. See Super String Dimensions and Flags for information on 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.

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 Super String Dimensions and Flags for information on 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Super String Solid 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. See Super String Dimensions and Flags for information on 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 0 indicates the function call was successful.

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 Super String Dimensions and Flags for information on 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 0 indicates the function call was successful.

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 0 indicates the function call was successful.

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 0 indicates the function call was successful.

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. See Super String Dimensions and Flags for information on 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 0 indicates the function call was successful.

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 Super String Dimensions and Flags for information on 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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. See “Super String Dimensions and Flags” for information on 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.

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 “Super String Dimensions and Flags” for information on 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.

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. See “Super String Dimensions and Flags” for information on 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.

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 “Super String Dimensions and Flags” for information on 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.

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. See Super String Dimensions and Flags for information on 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.

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 Super String Dimensions and Flags for information on 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.

Get_super_holes(Element super,Integer &no_holes)

Name

Integer Get_super_holes(Element super,Integer &no_holes)

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.

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.

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.

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.

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.

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.

Super String Vertex Text Functions

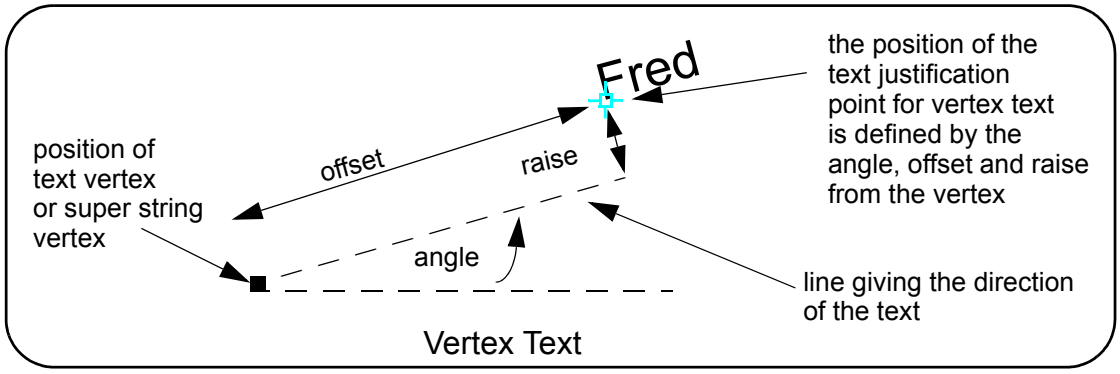
For definitions of the Vertex Text dimensions, see Text Dimensions

Super String Vertex text refers to the text at a super string vertex.

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

(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.



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

Set_super_vertex_world_text(Element super)

Name

Integer Set_super_vertex_world_text(Element)

Description

Tell the super string whether to use the dimension Att_Vertex_World_Annotate.

A return value of 0 indicates the function call was successful.

<no description>

Set_super_vertex_device_text(Element super)

Name

Integer Set_super_vertex_device_text(Element)

Description

A return value of 0 indicates the function call was successful.

<no description>

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.

See Super String Dimensions and Flags for information on dimensions and Text Dimensions for information on the Text dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 whether to use, or not use, the dimension Att_Vertex_Text_Value. If Att_Vertex_Text_Value is used, then there is one text for all the string vertices.

See Super String Dimensions and Flags for information on dimensions and Text Dimensions for information on the Text dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

Note if the dimension Att_Vertex_Text_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

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 for the super string

See Super String Dimensions and Flags for information on dimensions and Text Dimensions for information on the Text dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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, or not use, the dimension Att_Segment_Text_Array. If Att_Vertex_Text_Array is used, then there can be a different text for each string vertex.

See Super String Dimensions and Flags for information on dimensions and Text Dimensions for information on the Text 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.

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 Element **super**, return in **txt** the text on vertex number **vert**.

A return value of 0 indicates the function call was successful.

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 text at vertex number **vert** to be **txt**.

A return value of 0 indicates the function call was successful.

Get_super_vertex_text_type(Element super,Integer &type)

Name

Integer Get_super_vertex_text_type(Element super,Integer &type)

Description

A return value of 0 indicates the function call was successful.

<no description>

Set_super_vertex_text_type(Element super,Integer type)

Name

Integer Set_super_vertex_text_type(Element super,Integer type)

Description

A return value of 0 indicates the function call was successful.

<no description>

Get_super_vertex_text_justify(Element super,Integer vert,Integer &j)

Name

Integer Get_super_vertex_text_justify(Element super,Integer vert,Integer &j)

Description

A return value of 0 indicates the function call was successful.

<no description>

Set_super_vertex_text_justify(Element super,Integer vert,Integer j)

Name

Integer Set_super_vertex_text_justify(Element super,Integer vert,Integer j)

Description

A return value of 0 indicates the function call was successful.

<no description>

Get_super_vertex_text_offset_width(Element super,Integer vert,Real &off)

Name

Integer Get_super_vertex_text_offset_width(Element super,Integer vert,Real &off)

Description

For the super Element **super**, return as **off** the offset of the text from vertex number **vert**.

A return value of 0 indicates the function call was successful.

Set_super_vertex_text_offset_width(Element super,Integer vert,Real off)

Name

Integer Set_super_vertex_text_offset_width(Element super,Integer vert,Real o)

Description

For the super Element **super**, set the offset of the text from vertex number **vert** to **off**.

A return value of 0 indicates the function call was successful.

Get_super_vertex_text_offset_height(Element super,Integer vert,Real &ra)

Name

Integer Get_super_vertex_text_offset_height(Element super,Integer vert,Real &ra)

Description

For the super Element **super**, return as **ra** the raise of the text from vertex number **vert**.

A return value of 0 indicates the function call was successful.

Set_super_vertex_text_offset_height(Element super,Integer vert,Real ra)

Name

Integer Set_super_vertex_text_offset_height(Element super,Integer vert,Real ra)

Description

For the super Element **super**, set the raise of the text from vertex number **vert** to **ra**.

A return value of 0 indicates the function call was successful.

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 Element **super**, return as **col** the colour number of the text on vertex number **vert**.

A return value of 0 indicates the function call was successful.

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 Element **super**, set the colour number of the text on the vertex number **vert** to be **col**.

A return value of 0 indicates the function call was successful.

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 Element **super**, return the angle of rotation in **ang** of the text on vertex number **vert**. **ang** is in radians and is measured counterclockwise from the x-axis.

A return value of 0 indicates the function call was successful.

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 Element **super**, set the angle of rotation of the text on vertex number **vert** to **ang**. **ang** is in radians and is measured counterclockwise from the x-axis.

A return value of 0 indicates the function call was successful.

Get_super_vertex_text_size(Element super,Integer vert,Real &s)

Name

Integer Get_super_vertex_text_size(Element super,Integer vert,Real &s)

Description

For the super Element **super**, return as **s** the size of the text on vertex number **vert**.

A return value of 0 indicates the function call was successful.

Set_super_vertex_text_size(Element super,Integer vert,Real s)

Name

Integer Set_super_vertex_text_size(Element super,Integer vert,Real s)

Description

For the super Element **super**, set the size of the text on vertex number **vert** to **s**.

A return value of 0 indicates the function call was successful.

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 Element **super**, return as **xf** the x factor of the text on vertex number **vert**.

A return value of 0 indicates the function call was successful.

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 Element **super**, set the x factor of the text on vertex number **vert** to **xf**.

A return value of 0 indicates the function call was successful.

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 Element **super**, return as **sl** the slant of the text on vertex number **vert**.

A return value of 0 indicates the function call was successful.

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 Element **super**, set the slant of the text on vertex number **vert** to **sl**.

A return value of 0 indicates the function call was successful.

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 Element **super**, return as **ts** the textstyle of the text on vertex number **vert**.

A return value of 0 indicates the function call was successful.

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 Element **super**, set the textstyle of the text on vertex number **vert** to **ts**.

A return value of 0 indicates the function call was successful.

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 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.

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 set.

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

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 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.

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 set.

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

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 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.

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 set.

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

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 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.

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 set.

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

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 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.

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 set.

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

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 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.

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 set.

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

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 number **vert** to be **weight**.

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**, or if **super** does not have the dimension Att_Vertex_Text_Array set.

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

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 number **vert** and return it as **weight**.

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**, or if **super** does not have the dimension Att_Vertex_Text_Array set.

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

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 number **vert** to be **d**.

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.

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 number **vert** and return it as **d**.

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 returned.

Super String Vertex Annotation Functions

For definitions of the Vertex Annotation dimensions, see Text Annotation Dimensions

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. A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 whether to use the dimension Att_Vertex_Annotate_Value. 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.

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. A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 whether to use the dimension Att_Vertex_Annotate_Array. A value for **use** of 1 sets the dimension and 0 removes it. Note if the dimension Att_Vertex_Annotate_Value exists, this call is ignored.

A return value of 0 indicates the function call was successful.

Super String Segment Text Functions

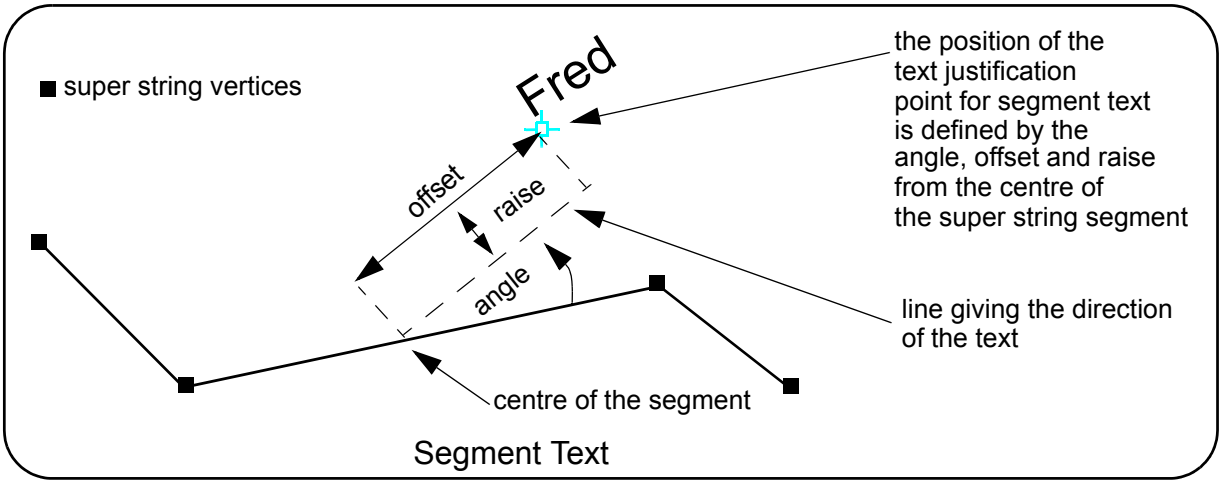
For definitions of the Segment Text dimension, see Text Dimensions

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 mostly the text is required to be parallel, or related to the segment direction.

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.

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.

See Super String Dimensions and Flags for information on dimensions and Text Dimensions for information on the Text dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 whether to use the dimension Att_Segment_Text_Value. If Att_Segment_Value_Array is used then there is one text for all the segments of the super string.

See Super String Dimensions and Flags for information on dimensions and Text Dimensions for information on the Text dimensions.

A value for **use** of 1 sets the dimension and 0 removes it.

Note if the dimension Att_Segment_Text_Array exists, this call is ignored.

A return value of 0 indicates the function call was successful.

Get_super_use_segment_text_array(Element element,Integer &use)

Name

Integer Get_super_use_segment_text_array(Element element,Integer &use)

Description

Query whether the dimension Att_Segment_Text_Array exists for the super string.

See Super String Dimensions and Flags for information on dimensions and Text Dimensions for information on the Text dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 whether to use, or not use, the dimension Att_Segment_Text_Array. If Att_Segment_Text_Array is used then there can be different text on every segment of the super string.

See Super String Dimensions and Flags for information on dimensions and Text Dimensions for information on the Text 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.

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 text on segment number **seg**.

A return value of 0 indicates the function call was successful.

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 text at segment number **seg** to be **txt**.

A return value of 0 indicates the function call was successful.

Get_super_segment_text_type(Element super,Integer &type)

Name

Integer Get_super_segment_text_type(Element super,Integer &type)

Description

A return value of 0 indicates the function call was successful.

<no description>

Set_super_segment_text_type(Element super,Integer type)

Name

Integer Set_super_segment_text_type(Element super,Integer type)

Description

A return value of 0 indicates the function call was successful.

<no description>

Get_super_segment_text_justify(Element super,Integer seg,Integer &j)

Name

Integer Get_super_segment_text_justify(Element super,Integer seg,Integer &j)

Description

A return value of 0 indicates the function call was successful.

<no description>

Set_super_segment_text_justify(Element super,Integer seg,Integer j)

Name

Integer Set_super_segment_text_justify(Element super,Integer seg,Integer j)

Description

A return value of 0 indicates the function call was successful.

<no description>

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 Element **super**, return as **off** the offset of the text on segment number **seg**.

A return value of 0 indicates the function call was successful.

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 Element **super**, set the offset of the text on segment number **seg** to **off**.

A return value of 0 indicates the function call was successful.

Get_super_segment_text_offset_height(Element super,Integer seg,Real &ra)

Name

Integer Get_super_segment_text_offset_height(Element super,Integer seg,Real &ra)

Description

For the super Element **super**, return as **ra** the raise of the text on segment number **seg**.

A return value of 0 indicates the function call was successful.

Set_super_segment_text_offset_height(Element super,Integer seg,Real ra)

Name

Integer Set_super_segment_text_offset_height(Element super,Integer seg,Real ra)

Description

For the super Element **super**, set the raise of the text on segment number **seg** to **ra**.

A return value of 0 indicates the function call was successful.

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 Element **super**, return as **col** the colour number of the text on segment number **seg**.

A return value of 0 indicates the function call was successful.

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 Element **super**, set the colour number of the text on segment number **seg** to **col**.

A return value of 0 indicates the function call was successful.

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 Element **super**, return the angle of rotation in **ang** of the text on segment number **seg**. **ang** is measured in radians and is measured counterclockwise from the direction of the segment.

A return value of 0 indicates the function call was successful.

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 Element **super**, set the angle of rotation of the text on segment number **seg** to **ang**. **ang** is measured in radians and is measured counterclockwise from the direction of the segment.

A return value of 0 indicates the function call was successful.

Get_super_segment_text_size(Element super,Integer seg,Real &s)

Name

Integer Get_super_segment_text_size(Element super,Integer seg,Real &s)

Description

For the super Element **super**, return as **s** the size of the text on segment number **seg**.

A return value of 0 indicates the function call was successful.

Set_super_segment_text_size(Element super,Integer seg,Real s)

Name

Integer Set_super_segment_text_size(Element super,Integer seg,Real s)

Description

For the super Element **super**, set the size of the text on segment number **seg** to **s**.

A return value of 0 indicates the function call was successful.

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 Element **super**, return as **xf** the x factor of the text on segment number **seg**.
A return value of 0 indicates the function call was successful.

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 Element **super**, set the x factor of the text on segment number **seg** to **xf**.
A return value of 0 indicates the function call was successful.

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 Element **super**, return as **sl** the slant of the text on segment number **seg**.
A return value of 0 indicates the function call was successful.

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 Element **super**, set the slant of the text on segment number **seg** to **sl**.
A return value of 0 indicates the function call was successful.

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 Element **super**, return as **ts** the textstyle of the text on segment number **seg**.
A return value of 0 indicates the function call was successful.

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 Element **super**, set the textstyle of the text on segment number **seg** to **ts**.
A return value of 0 indicates the function call was successful.

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 Element **super** of type **Super,** set the underline state for the segment number **seg** 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.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Segment_Text_Array set.

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

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 Element **super** of type **Super,** get the underline state for the segment number **seg** 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.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Segment_Text_Array set.

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

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 Element **super** of type **Super,** set the strikeout state for the segment number **seg** 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.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Segment_Text_Array set.

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

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 Element **super** of type **Super,** get the strikeout state for the segment number **seg** 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.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Segment_Text_Array set.

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

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 Element **super** of type **Super,** set the italic state for the segment number **seg** 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.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Segment_Text_Array set.

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

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 Element **super** of type **Super,** get the italic state for the segment number **seg** 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.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Segment_Text_Array set.

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

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 Element **super** of type **Super,** set the weight for the segment number **seg** to be **weight**.

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**, or if **super** does not have the dimension Att_Segment_Text_Array set.

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

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 Element **super** of type **Super,** get the weight for the segment number **seg** and return it as **weight.**

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**, or if **super** does not have the dimension Att_Segment_Text_Array set.

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

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 Element **super** of type **Super,** set the Textstyle_Data for the segment number **seg** to be **d**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Segment_Text_Value set.

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

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 Element **super** of type **Super,** get the Textstyle_Data for the segment number **seg** and return it as **d**.

A non-zero function return value is returned if **super** is not of type **Super**, or if **super** does not have the dimension Att_Segment_Text_Value set.

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

Super String Segment Annotation Functions

For definitions of the Segment Annotation dimensions, see Text Annotation Dimensions

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. A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 the dimension Att_Segment_Annotate_Value. 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 return value of 0 indicates the function call was successful.

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. A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 the dimension Att_Segment_Annotate_Array. A value for **use** of 1 sets the dimension and 0 removes it. Note if the dimension Att_Segment_Annotate_Value exists, this call is ignored.

A return value of 0 indicates the function call was successful.

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

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.

See Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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.

A value for **use** of 1 sets the dimension and 0 removes it.

See Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility dimensions.

A return value of 0 indicates the function call was successful.

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. If Att_Vertex_Tinable_Value is set then the tinability is the same for all vertices in **super**.

See Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility 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.

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 Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility 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.

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 in **super**.

See Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility 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.

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 Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility 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.

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 Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility dimensions.

A return value of 0 indicates the function call was successful.

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 Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility dimensions.

A return value of 0 indicates the function call was successful.

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. If Att_Segment_Tinable_Value is set then the tinability is the same for all vertices in **super**.

See Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility 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.

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**.

See Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility 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.

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 used. If Att_Segment_Tinable_Array is set then there can be a different tinability defined for each segment in **super**.

See Super String Dimensions and Flags for information on dimensions and Tinability Dimensions

for information on the Tinablility 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.

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**.

See Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility 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.

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 Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility dimensions.

A return value of 0 indicates the function call was successful.

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 Super String Dimensions and Flags for information on dimensions and Tinability Dimensions for information on the Tinablility dimensions.

A return value of 0 indicates the function call was successful.

Super String Point Id Functions

For definitions of the Point Id dimension, see Point Id Dimension

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 Point Ids for each vertex.

A value for **use** of 1 indicates the dimension exists.

See Super String Dimensions and Flags for information on dimensions and Point Id Dimension for information on the Point Id dimension.

A return value of 0 indicates the function call was successful.

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, or not use, the dimension Att_Point_Array.

A value **for** use of 1 sets the dimension and 0 removes it.

See Super String Dimensions and Flags for information on dimensions and Point Id Dimension for information on the Point Id dimension.

A return value of 0 indicates the function call was successful.

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

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 Super String Dimensions and Flags for information on dimensions and Point Id Dimension for information on the Point Id dimension.

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.

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 Super String Dimensions and Flags for information on dimensions and Point Id Dimension for information on the Point Id dimension.

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

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,** then a non-zero return code is returned.

LJG? what happens if the correct dimension is not set? Is it automatically done?

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.

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,** then a non-zero return code is returned.

LJG? what happens if the correct dimension is not set? Is it automatically done?

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

Super String Segment Geometry Functions

For definitions of the Segment Geometry dimension, see Segment Geometry Dimension

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.

See Super String Dimensions and Flags for information on dimensions and Segment Geometry Dimension for information on the Segment Geometry dimension.

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.

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.

See Super String Dimensions and Flags for information on dimensions and Segment Geometry Dimension for information on the Segment Geometry dimension.

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.

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 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 Super String Dimensions and Flags for information on dimensions and Segment Geometry Dimension for information on the Segment Geometry dimension.

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

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 Super String Dimensions and Flags for information on dimensions and Segment Geometry Dimension for information on the Segment Geometry dimension.

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

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 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 Super String Dimensions and Flags for information on dimensions and Segment Geometry Dimension for information on the Segment Geometry dimension.

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

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 Super String Dimensions and Flags for information on dimensions and Segment Geometry Dimension for information on the Segment Geometry dimension.

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

Super String Extrude Functions

For definitions of the Extrude dimensions, see Extrude Dimensions

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. If Att_Extrude_Value is set then an extrusion on the super string is allowed.

See Super String Dimensions and Flags for information on dimensions and Extrude Dimensions for information on the Extrude 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.

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**.

See Super String Dimensions and Flags for information on dimensions and Extrude Dimensions for information on the Extrude 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.

Set_super_extrude(Element super,Element elt)

Name

Integer Set_super_extrude(Element super,Element elt)

Description

For the Element **super** of type **Super** which has the dimension Att_Extrude_Value set**,** set **elt** to be the Element that defines the 2d profile that is extruded along **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 Super String Dimensions and Flags for information on dimensions and Extrude Dimensions for information on the Extrude dimensions.

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

Get_super_extrude(Element super,Element &elt)

Name

Integer Get_super_extrude(Element super,Element &elt)

Description

For the Element **super** of type **Super** and has the dimension Att_Extrude_Value set**,** get the Element **elt** that defines the 2d profile that is extruded along **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 Super String Dimensions and Flags for information on dimensions and Extrude Dimensions for information on the Extrude dimensions.

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

Super String Vertex Attributes Functions

For definitions of the Vertex Attributes dimensions, see User Defined Attributes Dimensions

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 an Attributes for each vertex.

See Super String Dimensions and Flags for information on dimensions and User Defined Attributes Dimensions for information on the Attributes dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 not use, the dimension Att_Vertex_Attribute_Array.

See Super String Dimensions and Flags for information on dimensions and User Defined Attributes Dimensions for information on the Attributes 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.

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 Super String Dimensions and Flags for information on dimensions and User Defined Attributes Dimensions for information on the Attributes dimensions.

A function return value of zero indicates the attribute is successfully returned.

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 Super String Dimensions and Flags for information on dimensions and User Defined Attributes Dimensions for information on the Attributes dimensions.

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

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 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**.

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 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**.

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 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**.

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 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**.

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**.

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**.

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**.

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**.

Super_vertex_attribute_exists(Element elt,Integer vert,Text name,Integer &no)

Name

Integer Super_vertex_attribute_exists(Element elt,Integer vert,Text name,Integer &no)

Description

<no description>

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 4DML function return values

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.

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.

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.

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.

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.

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.

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**.

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**.

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**.

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.

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.

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.

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.

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**.

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.

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.

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.

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**.

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**.

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**.

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**.

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**.

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**.

Super String Segment Attributes Functions

For definitions of the Segment Attributes dimensions, see User Defined Attributes Dimensions

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 Super String Dimensions and Flags for information on dimensions and User Defined Attributes Dimensions for information on the Attributes dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 the dimension Att_Segment_Attribute_Array.

See Super String Dimensions and Flags for information on dimensions and User Defined Attributes Dimensions for information on the Attributes 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.

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 Super String Dimensions and Flags for information on dimensions and User Defined Attributes Dimensions for information on the Attributes dimensions.

A function return value of zero indicates the attribute is successfully returned.

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 Super String Dimensions and Flags for information on dimensions and User Defined Attributes Dimensions for information on the Attributes dimensions.

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

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**.

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**.

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**.

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**.

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**.

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.

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.

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.

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

<no description>

A return value of 0 indicates the function call was successful.

Super_segment_attribute_exists(Element elt,Integer seg,Text name,Integer &no)

Name

Integer Super_segment_attribute_exists(Element elt,Integer seg,Text name,Integer &no)

Description

<no description>

A return value of 0 indicates the function call was successful.

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.

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.

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.

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.

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.

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.

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**.

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**.

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**.

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**.

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**.

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.

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.

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.

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.

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.**

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**.

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**.

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**.

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**.

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**.

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.

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**.

Super String Uid Functions

For definitions of the Visibility dimensions, see UID Dimensions

See Super String Vertex Uid
See Super String Segment Uid

Super String Vertex Uid

Set_super_use_vertex_uid(Element elt,Integer use)

Name

Integer Set_super_use_vertex_uid(Element elt,Integer use)

Description

Allows another dimension. Used in functions to allow backtracking. For experienced 12d staff only.

Used in survey data reduction and in underlying super string in super alignments.

Super String Segment Uid

Super String Vertex Image Functions

For definitions of the Visibility dimensions, see Vertex Image Dimensions

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. See “Super String Dimensions and Flags” for information on 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.

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. See “Super String Dimensions and Flags” for information on 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.

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 exists for the super string super. See "Super String Dimensions and Flags" for information on 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.

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. See "Super String Dimensions and Flags" for information on 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.

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

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 Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility dimensions.

A value for **use** of 1 indicates the dimension exists.

A return value of 0 indicates the function call was successful.

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 not use, the dimension Att_Visible_Array.

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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. If Att_Vertex_Visible_Value is set then there is one visibility value for all vertices in **super.**

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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**.

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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. If Att_Vertex_Visible_Array is set then there can be a different visibility defined for each vertex in **super**.

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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**.

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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 Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility dimensions.

A return value of 0 indicates the function call was successful.

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 Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility dimensions.

A return value of 0 indicates the function call was successful.

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. If Att_Segment_Visible_Value is set then the visibility is the same for all segments in **super**.

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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**.

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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. If Att_Segment_Visible_Array is set then there can be a different visibility defined for each segment in **super**.

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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**.

See Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility 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.

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 Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility dimensions.

A return value of 0 indicates the function call was successful.

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 Super String Dimensions and Flags for information on dimensions and Visibility Dimensions for information on the Visibility dimensions.

A return value of 0 indicates the function call was successful.

Element Operations

Selecting

Select_string(Text msg,Element &string)

Name

Integer Select_string(Text msg,Element &string)

Description

Write the message **msg** to the 12d Model message area and then return the Element picked by the user.

The picked Element is returned in the Element **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

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 message area 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

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 message area 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

Drawing

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.

Element_draw(Element elt)

Name

Integer Element_draw(Element elt)

Description

Draw the Element **elt** in its natural colour.

A function return value of zero indicates that elt

was drawn successfully.

Open and Close

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.

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.

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.

Length and Area

Get_length(Element elt,Real &length)

Name

Integer Get_length(Element elt,Real &length)

Description

Get the **plan** length of the string (which equals end chainage minus the start chainage)

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

Get_length_3d(Element elt,Real &length)

Name

Integer Get_length_3d(Element elt,Real &length)

Description

Get the 3d length of the string.

A function return value of zero indicates the 3d length was successfully returned.

Plan_area(Element elt, Real &plan_area)

Name

Integer Plan_area(Element elt,Real &plan_area)

Description

Calculate the plan area of an Element. 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 area is returned in the Real **plan_area**.

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

Position and Drop Point

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

Get the (**x,y,z**) position and instantaneous direction (**inst_dir** - as an angle, measured in radians) of a point at chainage **ch** on the Element **elt**.

A function return value of zero indicates success.

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.

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 in 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.

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 in 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.

Parallel

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.

Self Intersection

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.

Loop Clean Up

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

Locks

Get_read_locks(Element elt,Integer &num_locks)

Name

Integer Get_read_locks(Element elt,Integer &num_locks)

Description

<no description>

Get_write_locks(Element elt,Integer &num_locks)

Name

Integer Get_write_locks(Element elt,Integer &num_locks)

Description

<no description>

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.

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.

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.

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.

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.

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.

XML

The XML macro calls allow the user to read or write xml files from the macro language 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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**.

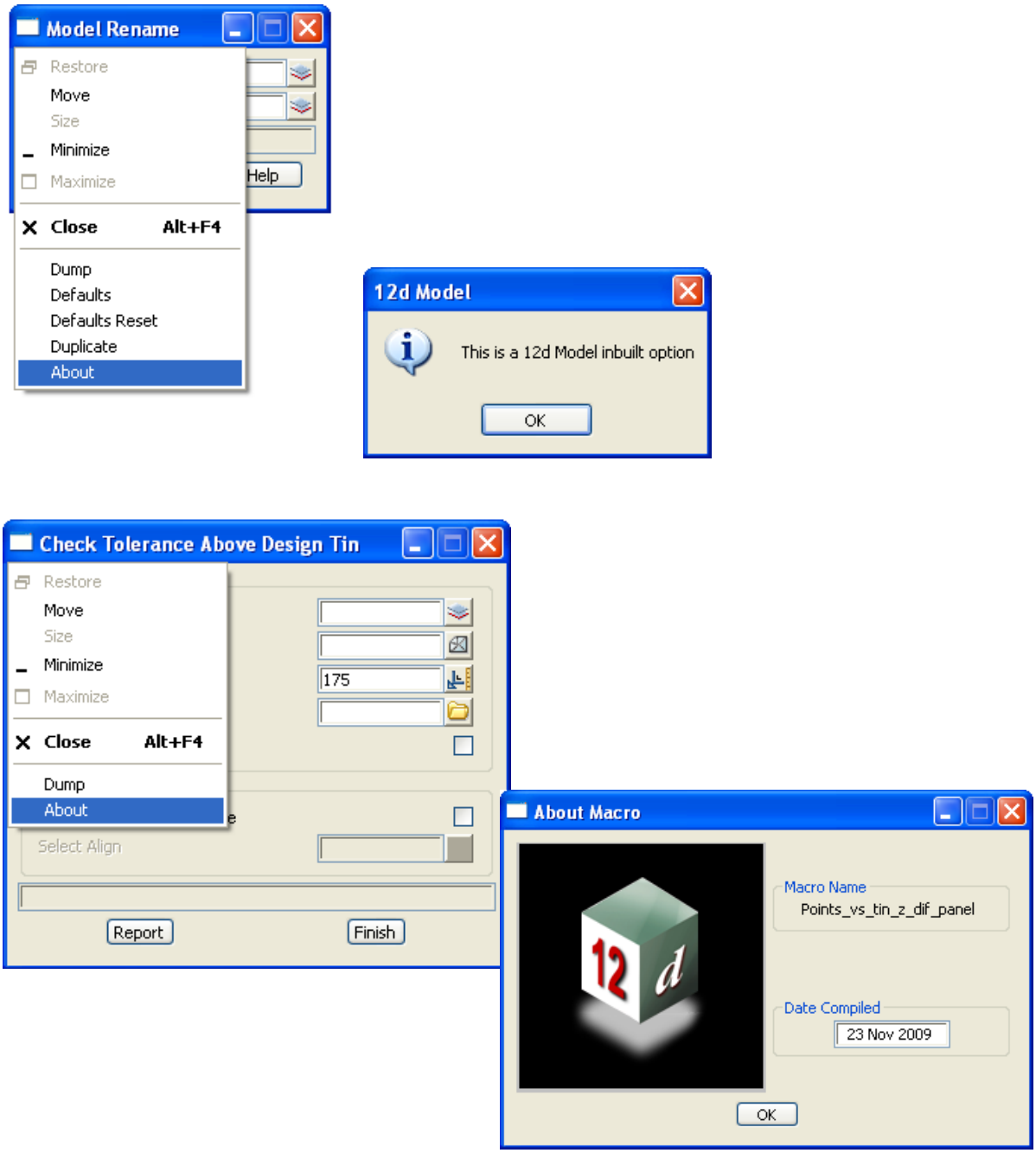
The file unit is given in Map_file **file**.

The record number is returned in Integer **number**.

A function return value of zero indicates the key was find successfully.

Panels

The user can build panels in the *12d Model* Macro Language 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 the *12d Model* Macro Language and in most cases, the only way to tell if a panel is an inbuilt 12d Model panel or is a 12dML 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** which can be Input widgets such a *Model_Box and Named_Tick_Box*, or Buttons such as *Report* or *Finish,* or *Trees* or *Grids*.

The Widgets can be built up in horizontal or vertical groups. Widgets inside are group are automatically spaced out by 12d Model.

One the Panel is constructed, it is displayed on screen by calling *Show_widget(Panel panel)*.

See Widget Controls
See Horizontal Group
See Vertical Group
See Panel Help and Tooltip Calls
See Panel Page
See Input Widgets
See Buttons
See GridCtrl_Box
See Tree Box Calls

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.

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.

Widget Controls

Create_panel(Text title_text)

Name

Panel Create_panel(Text title_text)

Description

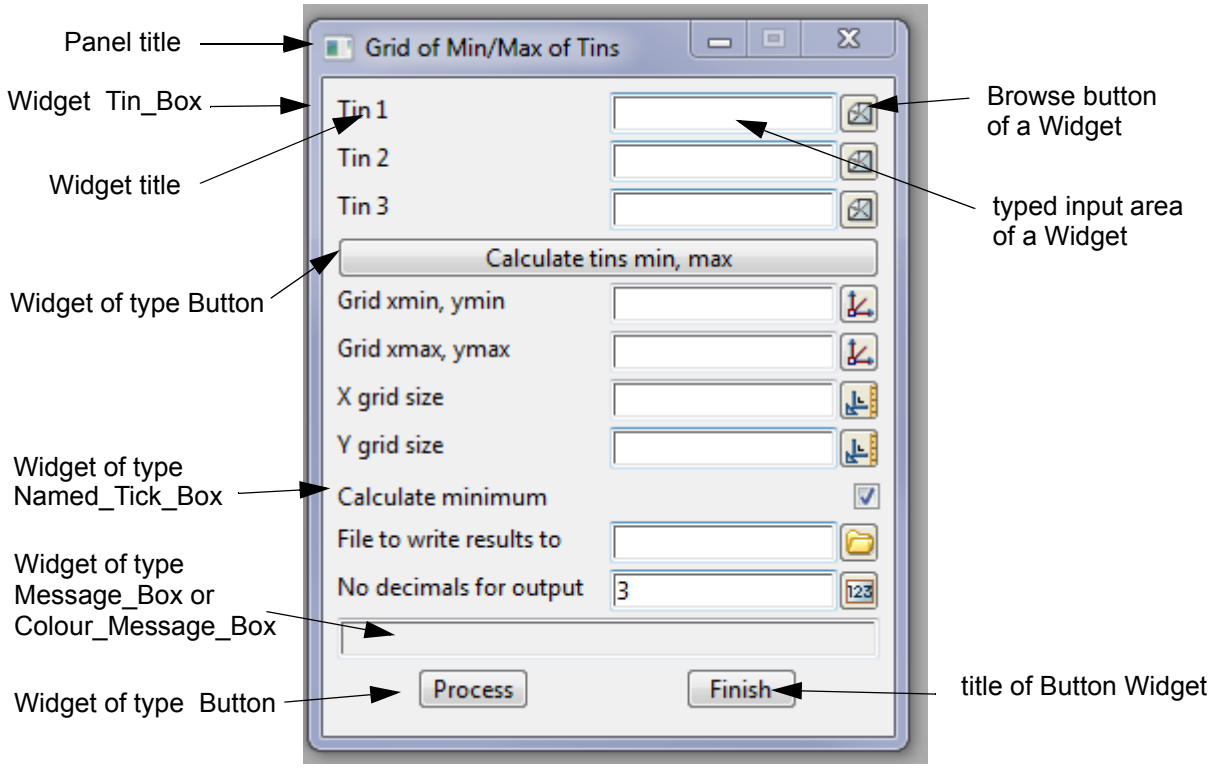
Create a panel with the title **title_text**.

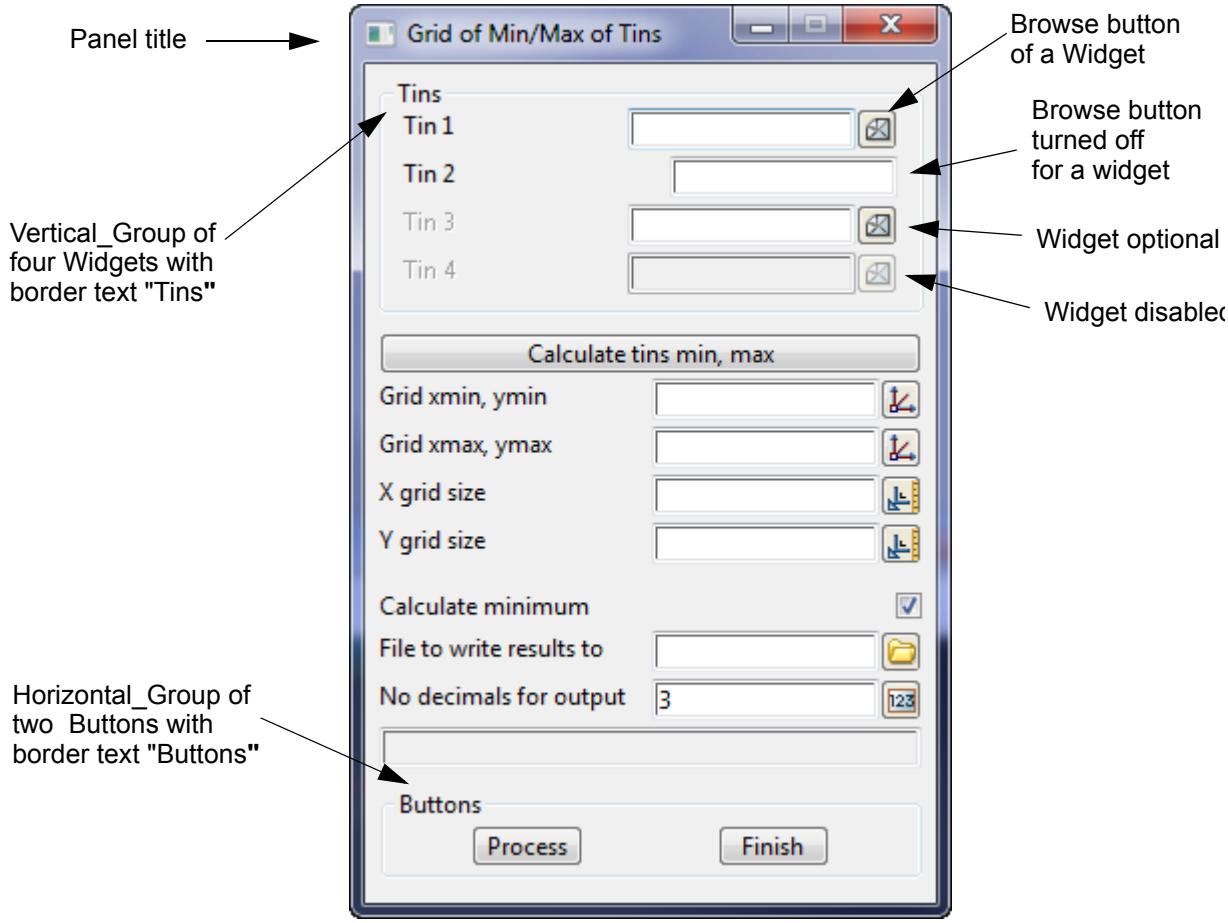
The function return value is the created Panel.

Note: the *Show_widget(Panel panel) call must be made to* display the panel on the screen. For example:

Panel Example:

```
 Panel    panel   = Create_panel("Grid of Min/Max of Tins");
Show_widget(panel);
```


Panel title
Widget Tin_Box
Widget title
Widget of type Button
Widget of type Named_Tick_Box
Widget of type Message_Box or Colour_Message_Box
Widget of type Button
Grid of Min/Max of Tins
Tin 1
Tin 2
Tin 3
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
Process
Finish
Browse button of a Widget
typed input area of a Widget
title of Button Widget


Panel title
Vertical_Group of four Widgets with border text "Tins"
Horizontal_Group of two Buttons with border text "Buttons"
Grid of Min/Max of 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
Buttons
Process
Finish
Browse button of a Widget
Browse button turned off for a widget
Widget optional
Widget disable

Append(Widget widget,Panel panel)

Name

Integer Append(Widget widget,Panel panel)

Description

Append the Widget **widget** to the Panel **panel**.

A function return value of zero indicates the widget was appended successfully.

Note: the panel is built up of Widgets in the order that they are Appended.

For an example of a panel with Widgets Tin_Box, Buttons, Message_Box etc, see Panel Example:

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**.

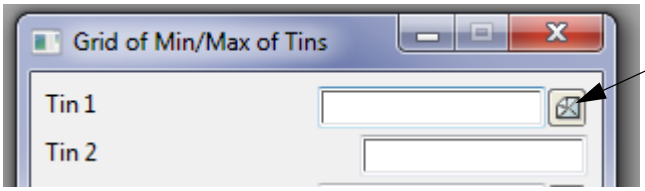
If **mode** = 1 use the browse button
if **mode** = 0 don't use the browse button.

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.



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.**

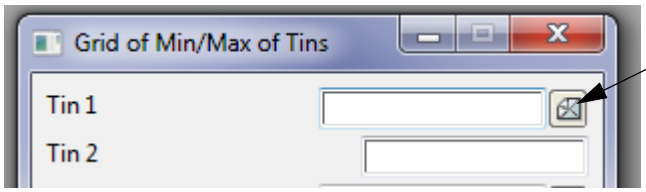
If **mode** = 1 show the browse button
if **mode** = 0 don't show the browse button.

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.



Set_enable(Widget widget,Integer mode)

Name

Integer Set_enable(Widget widget,Integer mode)

Description

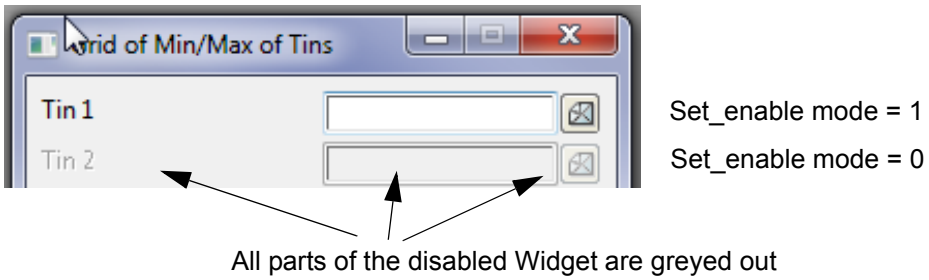
Set the enabled **mode** for the Widget **widget**.

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.



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.

Set_optional(Widget widget,Integer mode)

Name

Integer Set_optional(Widget widget,Integer mode)

Description

Set the optional **mode** for the Widget **widget**.

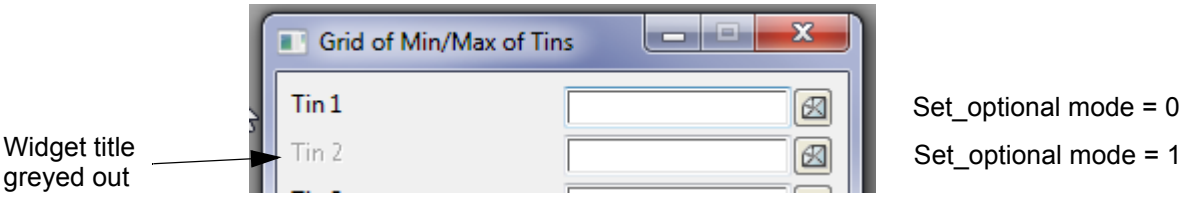
That is, if the Widget field is blank, the title text to the left is greyed out, signifying that this Widget is optional.

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.



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.

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.

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.

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.

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.

Show_widget(Widget widget)

Name

Integer Show_widget(Widget widget)

Description

Show the Widget **widget** at the cursor's current position.

A function return value of zero indicates the **widget** was shown successfully.

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 (pixels) x, y.

A function return value of zero indicates the **widget** was shown successfully.

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.

A function return value of zero indicates the **widget** was hidden successfully.

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.

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.

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.

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.

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.

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.

Get_id(Widget)

Name

Integer Get_id(Widget)

Description

Get the id of the Widget **widget**.

The function return value is the **id**.

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.

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*.

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

Horizontal Group

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.

The **mode** is always set to 0.

The function return value is the created **Horizontal_Group**.

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 **mode** is always set to 0.

The function return value is the created **Horizontal_Group**.

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.

Set_border(Horizontal_Group group,Text text)

Name

Integer Set_border(Horizontal_Group group,Text text)

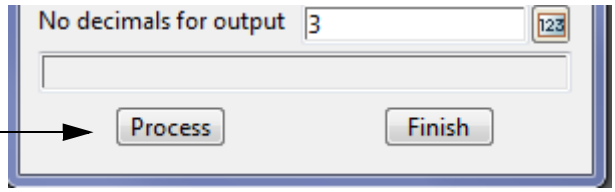
Description

Set a border for the Horizontal_Group **group** with Text **text**.on the top left side of the border.

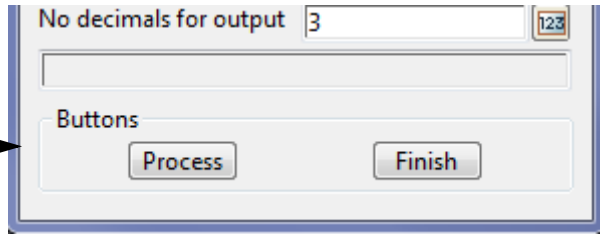
If text is blank, the border is removed.

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

Horizontal_Group of two Buttons with no border



Horizontal_Group of two Buttons with border and text "Buttons"



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.**

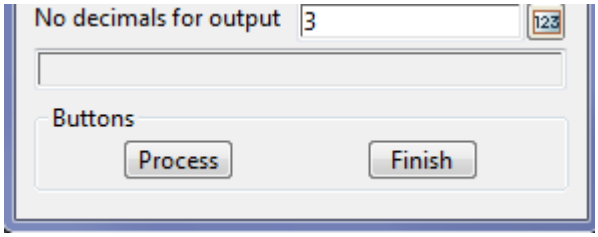
bx sets the left and right side gap around the border.

by sets the top and bottom side gap around of the border.

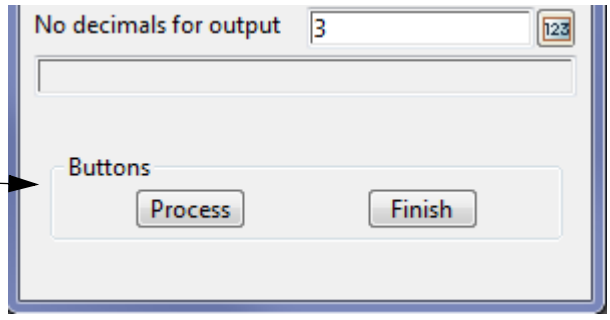
The units of bx and by are screen units (pixels).

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

Horizontal_Group of two Buttons with default border gaps and text "Buttons"



Horizontal_Group of two Buttons with border gaps bx =10 and by = 20 and text "Buttons"



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.

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.

The **mode** is always set to 0.

The function return value is the created Vertical_Group.

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.

Set_border(Vertical_Group group,Text text)

Name

Integer Set_border(Vertical_Group group,Text text)

Description

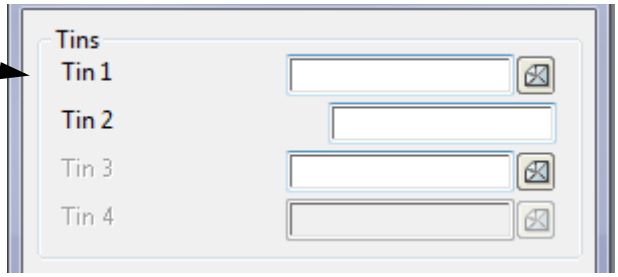
Set a border of the Vertical_Group **group** with Text text.on the top left side of the border. If text is blank, the border is removed.

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

The tins are a Vertical_Group of 4 Widgets with no border

The same Vertical_Group of 4 Widgets with border and text "Tins"

Note that for the left and right gaps that the width of the panel doesn't change but the gap from the sides of the panel to the box is increased



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.**

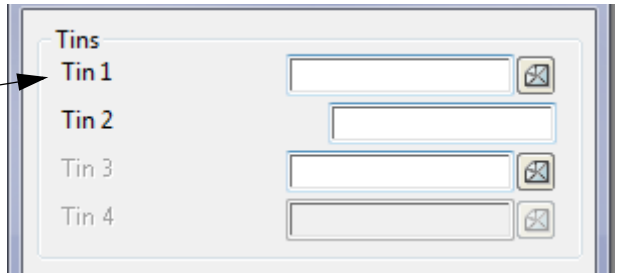
bx sets the left and right side gap around the border.

by sets the top and bottom side gap around of the border.

The units of bx and by are screen units (pixels).

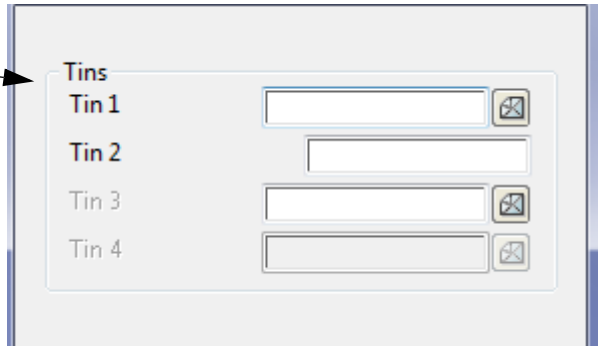
A function return value of zero indicates the border gap was successfully set.

Vertical_Group of 4 Widgets with default border gaps and text "Tins"



Vertical_Group of 4 Widgets with border gaps **bx** =10 and **by** = 20 and text "Tins"

Note that for the left and right gaps that the width of the panel doesn't change but the gap from the sides of the panel to the box is increased



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.

Panel Help and Tooltip Calls

Set_tooltip(Widget widget,Text tip)

Name

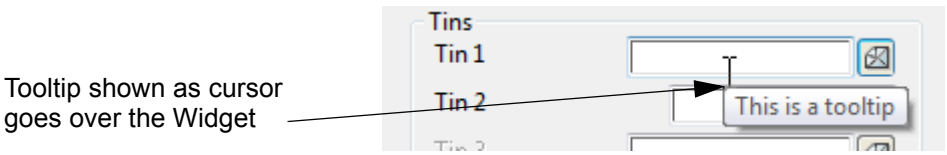
Integer Set_tooltip(Widget widget,Text tip)

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.



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.

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.

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.

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.

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.

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.

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.

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.

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.

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**.

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.

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.

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.

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.

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 Colour_Message_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 Message_Box
See Model_Box
See Name_Box
See New_Select_Box
See Name_Tick_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 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

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.

The **Angle_Box** is created with the title **title_text**.

The Message_Box **message** is used to display information.

The function return value is the created **Angle_Box**.

Set_data(Angle_Box box,Real angle)

Name

Integer Set_data(Angle_Box box,Real angle)

Description

Set the Real data for the Angle_Box **box** as the Real **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.

Set_data(Angle_Box box,Text text_data)

Name

Integer Set_data(Angle_Box box,Text text_data)

Description

Set the data of type Text for the Angle_Box **box** to **text_data**.

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

Get_data(Angle_Box box,Text &text_data)

Name

Integer Get_data(Angle_Box box,Text &text_data)

Get the data of type Text from the Angle_Box **box** and return it in **text_data**.

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

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 **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 4DML function return values

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**.

The Attributes_Box is created with the title **title_text**.

The Message_Box **message** is used to display information.

The function return value is the created Attributes_Box.

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.

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.

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.

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 4DML function return values

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**.

The Billboard_Box is created with the title **title_text**.

The Message_Box message is used to display information.

The function return value is the created Billboard_Box.

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.

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.

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 4DML function return values

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**.

The Bitmap_Fill_Box is created with the title **title_text**.

The Message_Box message is used to display information.

The function return value is the created Bitmap_Fill_Box.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Chainage_Box

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**.
The Chainage_Box is created with the title **title_text**.
The Message_Box message is used to display information.
The function return value is the created Chainage_Box.

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 4DML function return values

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Choice_Box

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**.
The **Choice_Box** is created with the title **title_text**.
The Message_Box **message** is used to display the choice information.
The function return value is the created **Choice_Box**.

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
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 4DML function return values

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.

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.

Set_data(Choice_Box box,Integer nc,Text choices[])

Name

Integer Set_data(Choice_Box box,Integer nc,Text choices[])

Description

Set the available choice list. There are nc items in the **choices** list for the Choice_Box **box**.

The data type in the **choices** list must be Text.

A function return value of zero indicates the nc'th data in the **choices** list was successfully set.

For information on the other Input Widgets, go to Input Widgets

Colour_Box

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**.

The **Colour_Box** is created with the title **title_text**.

The Message_Box message is used to display the colour information.

The function return value is the created **Colour_Box**.

Validate(Colour_Box box,Integer &result)

Name

Integer Validate(Colour_Box box,Integer &result)

Description

Validate the contents of Colour_Box **box** and return the colour Integer in **result.**

The function returns the value of:

NO_NAME if the Widget Colour_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 4DML function return values

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**.

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

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**.

colour_num must be **Integer**.

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

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**.

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

For information on the other Input Widgets, go to Input Widgets

Colour_Message_Box

Text messages can be sent and displayed in a Colour_Message_Box.

However unlike a Message_Box, the background colour of the display area can be modified for a

Colour_Message_Box.

This is useful for differentiating between different types of messages such as errors, warnings etc. The levels for the Colour_Message_Box are:

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)

Create_colour_message_box(Text title_text)

Name

Colour_Message_Box Create_colour_message_box(Text title_text)

Description

Create a box of type **Colour_Message_Box** for writing out messages.

The Colour_Message_Box is created with the title **title_text**.

The background colour of the display area is set using *Set_level (Colour_Message_Box, level)*.

The function return value is the created Colour_Message_Box.

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**.

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

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**.

The background colour of the box is set as **level**.

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

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 of the display area.

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)

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

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**.

The Date_Time_Box is created with the title **title_text**.

The Message_Box message is used to display information.

The function return value is the created Date_Time_Box.

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 4DML function return values

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.

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Directory_Box

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**.

The Directory_Box is created with the title **title_text**.

The Message_Box **message** is used to display the directory information.

The value of **mode** is listed in the Appendix A - Directory mode

The function return value is the created Directory_Box.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Draw_Box

Create_draw_box(Integer width,Integer height,Integer border)

Name

Draw_Box Create_draw_box(Integer width,Integer height,Integer border)

Description

Create an input Widget of type **Draw_Box** with the **width**, **height** and **border**.
The function return value is the created **Draw_Box**.

Get_size(Draw_Box,Integer &width,Integer &height)

Name

Integer Get_size(Draw_Box,Integer &width,Integer &height)

Description

Get the **width** and **height** of the draw box.
A function return value of zero indicates the width and height were successfully returned.

Set_text_font(Draw_Box box,Text font)

Name

Integer Set_text_font(Draw_Box box,Text font)

Description

Set the text **font** font for the Draw_Box **box**.

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

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.

Set_text_align(Draw_Box box,Integer mode)

Name

Integer Set_text_align(Draw_Box box,Integer mode)

Description

Set the text alignment for Draw_Box **box** depending on the **mode** value.

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

For information on the other Input Widgets, go to Input Widgets

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**.

The File_Box is created with the title **title_text**.

The Message_Box **message** is used to display the file information.

The value of mode is listed in the Appendix A - File **mode**.

If the RB is pressed in the box area, 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 box area.

The function return value is the created File_Box.

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 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 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 4DML function return values

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.

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.

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.

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.

Get_directory(File_Box box,Text &data)

Name

Integer Get_directory(File_Box box,Text &data)

Description

Get directory from the File_Box **box**.

The type of data must be **Text**.

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

Set_directory(File_Box box,Text text_data)

Name

Integer Set_directory(File_Box box,Text text_data)

Description

Set the directory to the File_Box **box**.

The type of data must be **Text**.

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

For information on the other Input Widgets, go to Input Widgets

Function_Box

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.

The Function_Box is created with the title **title_text**.

The Message_Box **message** is selected to display information during the operation.

The value of **mode** is listed in the Appendix A - Function mode. See Function Mode

LJG? What is type

The function return value is the created **Function_Box**.

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 4DML function return values

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.

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.

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.

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

HyperLink_Box

HyperLink_Box Create_hyperlink_box(Text title_text,Message_Box message)

Name

HyperLink_Box Create_hyperlink_box(Text title_text,Message_Box message)

Description

Create an input Widget of type **Hyperlink_Box**.

The Hyperlink_Box is created with the title **title_text**.

The Message_Box message is used to display information.

The function return value is the created Hyperlink_Box.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Input_Box

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**.

The Input_Box is created with the title **title_text**.

The Message_Box **message** is used to display the input information.

The function return value is the created Input_Box.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Integer_Box

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**.

The Integer_Box is created with the title **title_text**.

The Message_Box **message** is used to display the integer information.

The function return value is the created Integer_Box.

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 4DML function return values

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Justify_Box

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**.

The Justify_Box is created with the title **title_text**.

The Message_Box **message** is used to display the justify information.

The function return value is the created Justify_Box.

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 4DML function return values

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Linestyle_Box

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**.

The Linestyle_Box is created with the title **title_text**.
The Message_Box **message** is used to display the linestyle information.
The value of **mode** is listed in the Appendix A - Linestyle mode.
The function return value is the created Linestyle_Box.

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 4DML function return values

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.

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.

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**.

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 used to display the select information.

The function return value is the created List_Box.

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.

Set_sort(List_Box box,Integer mode)

Name

Integer Set_sort(List_Box box,Integer mode)

Description

Set the sort model 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.

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.

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**.

The Map_File_Box is created with the title **title_text**.

The Message_Box **message** is used to display the map file information.

The value of **mode** is listed in the Appendix A - File mode.

The function return value is the created Map_File_Box.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Message_Box

Text messages can be sent and displayed in a Message_Box.

Create_message_box(Text title_text)

Name

Message_Box Create_message_box(Text title_text)

Description

Create a box of type **Message_Box** for writing out messages.

The Message_Box is created with the title **title_text**.

The function return value is the created Message_Box.

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.

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**.

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

For information on the other Input Widgets, go to Input Widgets

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**.

The Message_Box **box** is used to display information.

The value of the mode is listed in the Appendix A - Model mode.

The function return value is the created **Model_Box**.

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 the **mode** is listed in the Appendix A - Model mode. *See* Model Mode

The function returns the value of:

NO_NAME if the Widget Model_Box is optional and the box is left empty

NO_MODEL, MODEL_EXISTS, DISK_MODEL_EXISTS or NEW_MODEL

TRUE (1) if no other return code is needed and *result* is valid.

FALSE (zero) if there is an error.

A function return value of zero indicates that there is an error.

Warning this is the opposite of most 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Name_Box

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**.

The Name_Box is created with the title **title_text**.

The Message_Box **message** is used to display the name information.

The function return value is the created Name_Box.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Name_Tick_Box

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**.

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.

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.**

The function returns the value of

TRUE (1) if the Named_Tick_Box is ticked

FALSE (0) if the Named_Tick_Box is not ticked.

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

New_Select_Box

New_Select_Box 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**.

The New_Select_Box is created with the title **title_text**.

The value of mode is listed in the Appendix A - Select mode.

The Message_Box **message** is used to display information.

The function return value is the created New_Select_Box.

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 4DML function return values

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 4DML function return values

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

New_XYZ_Box

New_XYZ_Box 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**.

The New_XYZ_Box is created with the title **title_text**.

The Message_Box message is used to display information.

The function return value is the created New_XYZ_Box.

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 4DML function return values

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.

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.

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.

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**.
The Plotter_Box is created with the title **title_text**.

The Message_Box **message** is used to display the plotter information.
The function return value is the created Plotter_Box.

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 4DML function return values

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.

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.

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>

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>

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>

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**.

The Polygon_Box is created with the title **title_text**.

LJG? select_title

LJG ? mode

The Message_Box message is used to display information.

The function return value is the created Polygon_Box.

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 4DML function return values

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 4DML function return values

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Real_Box

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**.

The Real_Box is created with the title **title_text**.

The Message_Box message is used to display the real information.

The function return value is the created Real_Box.

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 4DML function return values

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Report_Box

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**.

The Report_Box is created with the title **title_text**.

The Message_Box **message** is used to display information about the report.

The value of **mode** is listed in the Appendix A - File mode.

The function return value is the created Report_Box.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Screen_Text

Create_screen_text(Text text)

Name

Screen_Text Create_screen_text(Text text)

Description

Create a **Screen_Text** with the Text **text**.

The function return value is the created Screen_Text.

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.

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.

For information on the other Input Widgets, go to Input Widgets

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 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_Box.

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 4DML function return values

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.

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.

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.

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.

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.

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.

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.

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.

Set_select_snap_mode(Select_Box select,Integer mode,Integer control,Text snap_text)

Name

Integer Set_select_snap_mode(Select_Box select,Integer mode,Integer control,Text snap_text)

Description

Set the snap mode **mode** and snap control **control** for the Select_Box **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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Select_Boxes

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.

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 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**.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Sheet_Size_Box

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**.

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.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Source_Box

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**.

The Source_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 Source_Box.

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.

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 4DML function return values

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.

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**.

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

For information on the other Input Widgets, go to Input Widgets

Symbol_Box

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**.

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.

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 4DML function return values

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.

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.

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**.

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.

Validate(Target_Box box)

Name

Integer Validate(Target_Box box)

Description

<no description>

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>

For information on the other Input Widgets, go to Input Widgets

Template_Box

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**.

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.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Text_Style_Box

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**.
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.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

Text_Units_Box

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**
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.

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 4DML function return values

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.

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.

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.

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**.

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.

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 4DML function return values

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.

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.

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.

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.

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**.

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.

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.

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Texture_Box

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**.

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.

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

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**.

The Message_Box message is used to display the tick information.

The function return value is the created Tick_Box.

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.**

LJG? The function returns the value of

TRUE (1) if the Named_Tick_Box is ticked

FALSE (0) if the Named_Tick_Box is not ticked.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Tin_Box

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**.

The Tin_Box is created with the title **title_text**.

The Message_Box message is used to display the tin information.

The value of **mode** is listed in the Appendix A Tin mode.

The function return value is the created Tin_Box.

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** is listed in the Appendix A Tin mode. *See* Tin Mode

The function returns the value of:

NO_NAME if the Widget Tin_Box is optional and the box is left empty

NO_TIN, TIN_EXISTS or DISK_TIN_EXISTS

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

View_Box

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**.

The View_Box is created with the title **title_text**.

The Message_Box message is used to display the view information.

The value of **mode** is listed in the Appendix A - View mode.

The function return value is the created View_Box.

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** is listed in the Appendix A - View mode. See View Mode

The function returns the value of:

NO_NAME if the Widget View_Box is optional and the box is left empty

NO_VIEW or VIEW_EXISTS

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 4DML function return values

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.

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.

For information on the other Input Widgets, go to Input Widgets

XYZ_Box

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**.

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.

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 4DML function return values

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.

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.

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.

For information on the other Input Widgets, go to Input Widgets

Buttons

See Button
See Finish Button
See Select_Button

Button

Create_button(Text title_text,Text reply)

Name

Button Create_button(Text title_text,Text reply)

Description

Create a button of type **Button**.

The button is created with a label text **title_text**.

The Text **reply** is the message that is sent to the widget.

The function return value is the created **Button**.

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.

Create_child_button(Text title_text)

Name

Button Create_child_button(Text title_text)

Description

Not implemented.

For information on the other Buttons, go to Buttons

Finish Button

Create_finish_button(Text title_text,Text reply)

Name

Button Create_finish_button(Text title_text,Text reply)

Description

<no description>

For information on the other Buttons, go to Buttons

Select_Button

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**.

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**.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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.

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.

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.

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.

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.

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.

For information on the other Buttons, go to Buttons

GridCtrl_Box

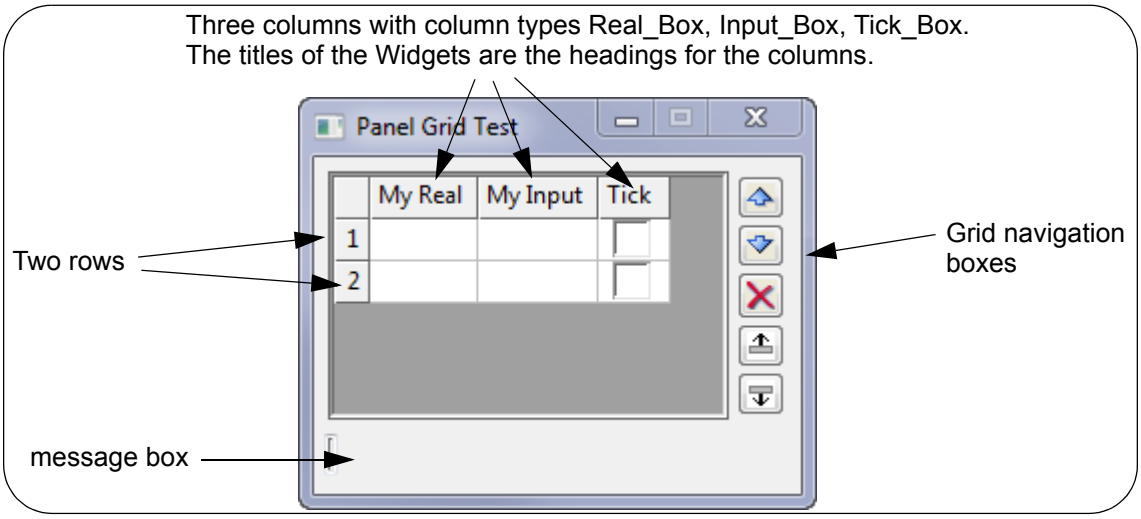
A GridCtrl_Box is made up of columns and rows of Widgets.

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.

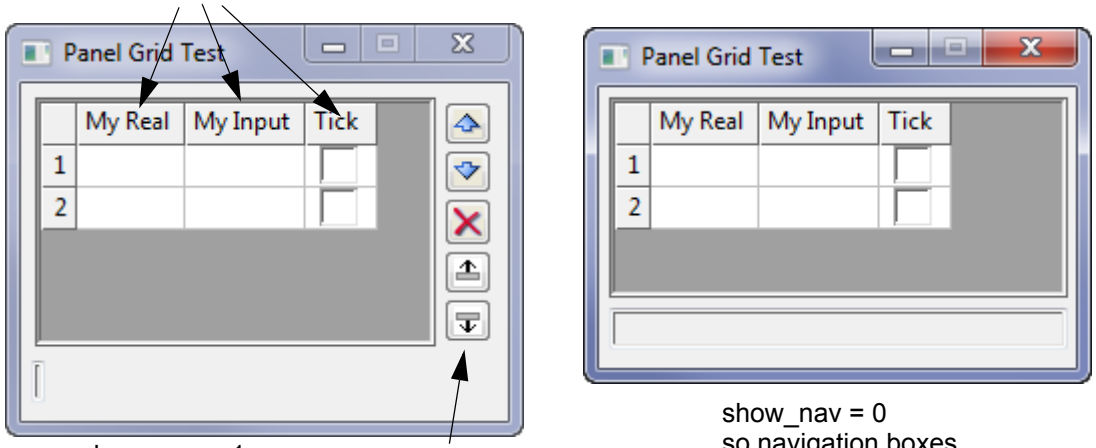
The width of the grid cell is **width** and the height of the grid cell is **height**, The units for width and height are screen units (pixels).

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

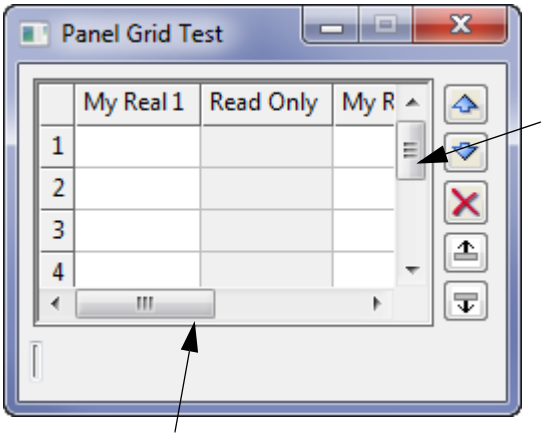


show_nav = 1
so navigation boxes

Grid navigation boxes

show_nav = 0
so navigation boxes

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.

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

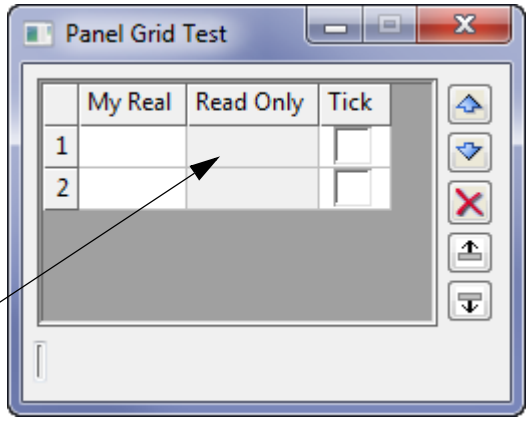
column_readonly[] is an Integer array of size **num_columns** where a value of 1 means that the cell is read only, and 0 means that the cell can be edited.

To set only the middle column to be read only -

Integer column_readonly[3];

column_readonly[1] = 0;
column_readonly[2] = 1;
column_readonly[3] = 0;

Second column is read only



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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Set_warn_on_modified(GridCtrl_Box grid,Integer warn_on_modified)

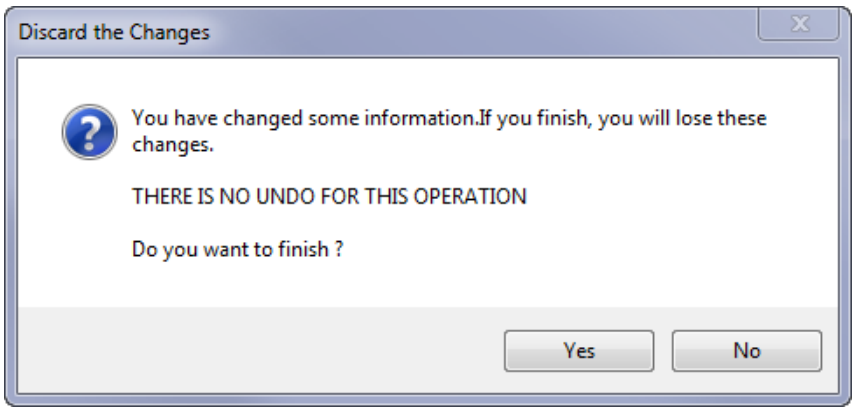
Name

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.

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

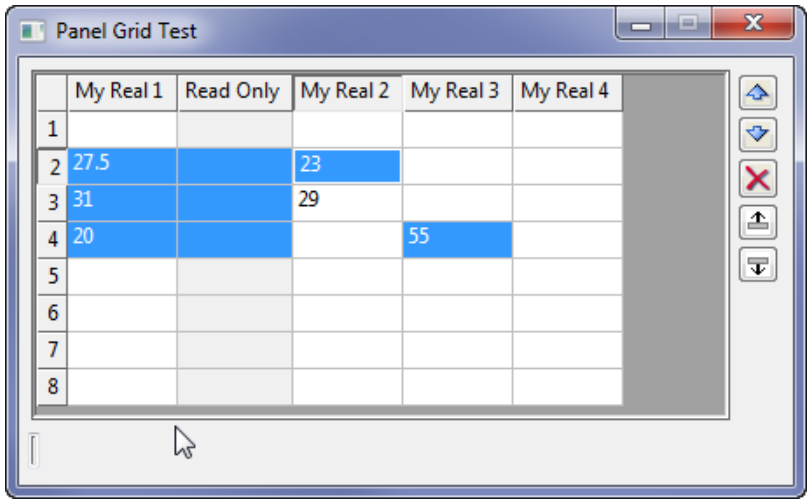
For the GridCtrl_Box **grid**, return the minimum and maximum row and column numbers for the current selected cells (the range of the selected cells).

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.

start_row = 2
start_col= 1

end_row = 4
end_col = 4



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.

Tree Box Calls

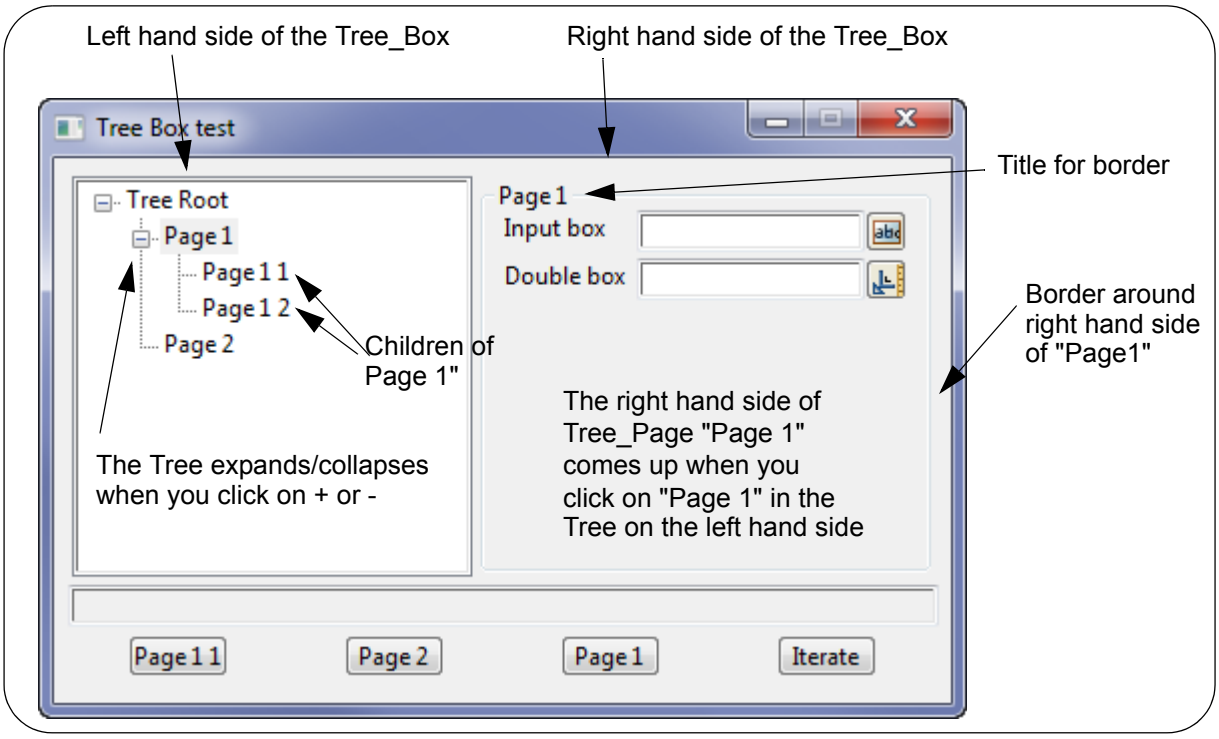
The tree box is a widget that consists of two parts - a left hand side (Tree) and a right hand side for displaying information for a particular part of the tree.

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)

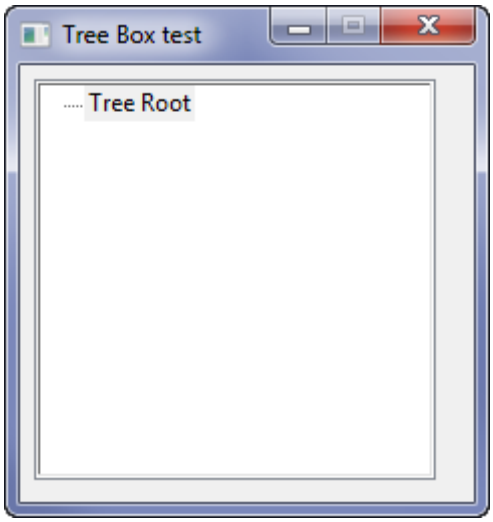
Description

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.

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.

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)

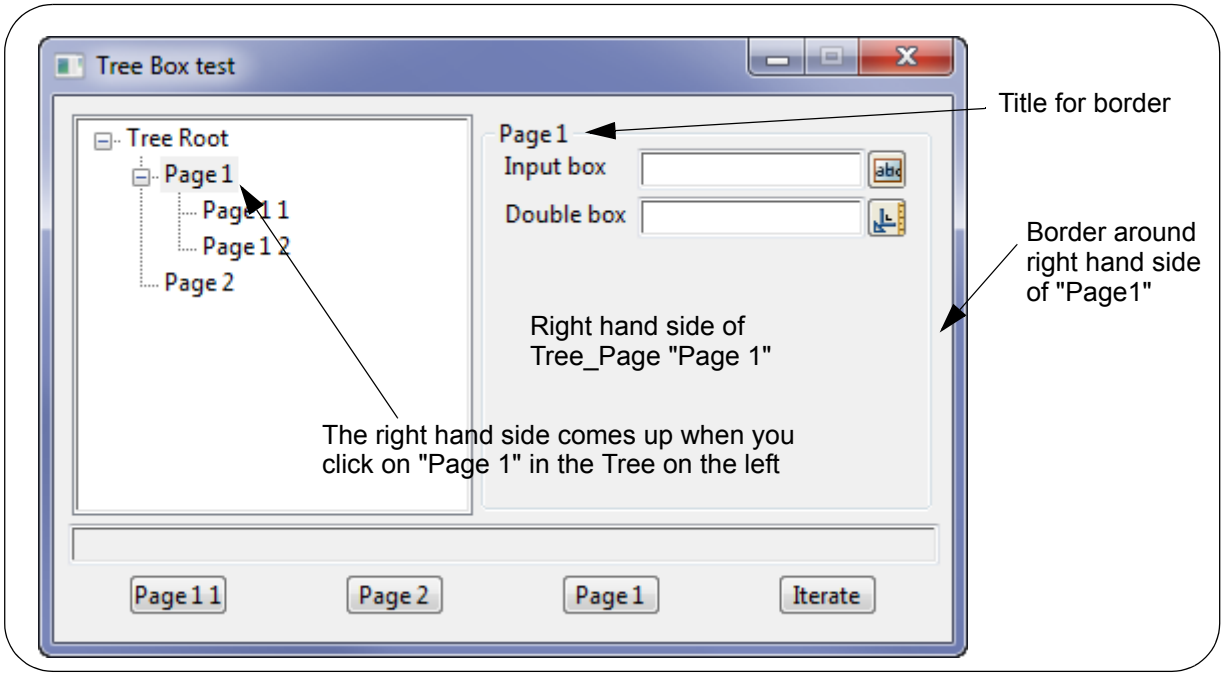
Description

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.

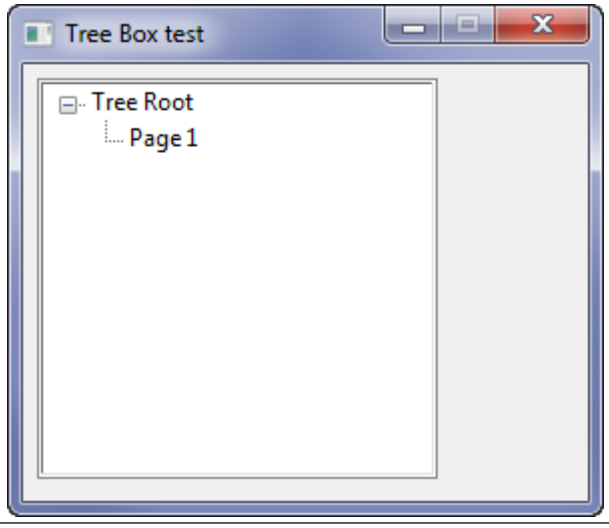


A parent page must exist before a child page can be created. The parent page may be the root page that is automatically created for a Tree_Box and the *Get_root_page* call is used to get the root page of a Tree_Box. *See* Get_root_page(Tree_Box tree_box)

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.

Append(Widget widget,Tree_Page page)

Name

Integer Append(Widget widget,Tree_Page page)

Description

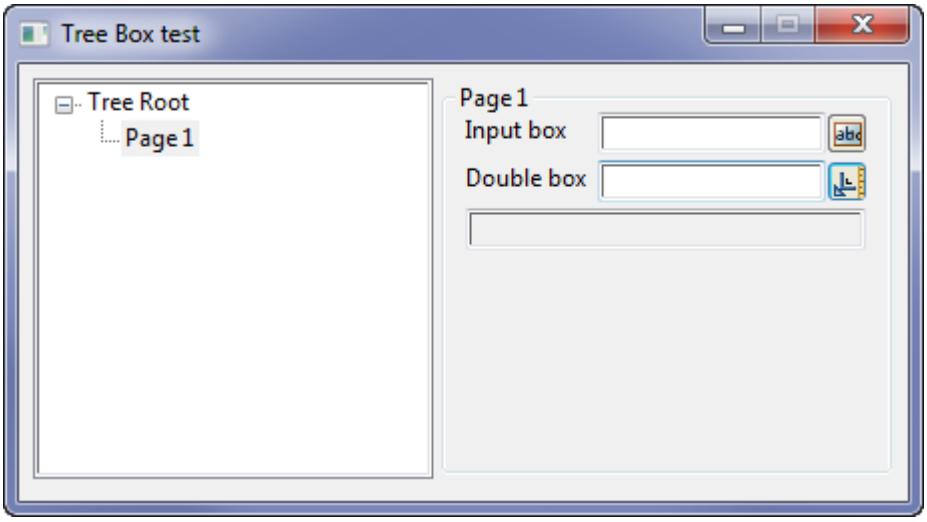
Append the Widget **widget** to the Tree_Page **page**.

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);
```



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.

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.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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.

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.

General

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 zero function return value indicates there is no match.

Warning - this is the opposite of most 4DML function return values

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.

Project Functions

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.

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.

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 4DML function return values

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.

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.

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.

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 4DML function return values

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

Rectangle_clip(Real x1,Real y1,Real x2,Real y2,Integer npts_in,Real zarray_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

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.

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.

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.

12d Model 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 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.

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.

Function_exists(Text function_name)

Name

Integer Function_exists(Text function_name)

Description

Checks to see if a 12d or user 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 4DML function return values.

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.

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.

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**.

LJG? Units of time?

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

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**.

LJG? Units of time?

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

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**.

LJG? Units of time?

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

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.

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.

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).

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.

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 **fun**c, 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.

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.

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 **fun**c, 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.

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 **fun**c, 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.

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 **fun**c, 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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
Real
Text

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.

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.

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

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.

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

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.

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

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.

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

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.

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

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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 their is no Function/Macro_Function with the Id **function_id,** then a null Function/ Macro_Function is returned as the function return value.

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 4DML function return values

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 their is no Function/Macro_Function with the Uid **function_id,** then a null Function/ Macro_Function is returned as the function return value.

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 4DML function return values

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 4DML function return values

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.

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.

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.

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.

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.

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

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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.

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.

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.

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.

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**.

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**.

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**.

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.**

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.**

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.**

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.

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.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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**.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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*.

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.

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.

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.

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.

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.

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 4DML function return values.

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.

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

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.

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.

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, 4DML 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 4DML 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 4DML 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*.

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*.

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*.

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*.

Add_undo_range(Text name,Integer id1,Integer id2)

Name

Undo Add_undo_range(Text name,Integer id1,Integer id2)

Description

Important note - Id's are no longer used is 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).*

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*.

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*.

For information on adding/removing Undo's to an internal 4DML list and how it interacts with the 12d Model Undo system, go to the next section Functions for a 4DML Undo_List

Functions for a 4DML 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.**

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**.

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.

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.

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.

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.

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.

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.

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.

Begin_transaction(Transaction transaction)

Name

Integer Begin_transaction(Transaction transaction)

Description

This call begins a new transaction. It will return 0 if successful.

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.

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.

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.

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.

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.

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.

Macro Console

Before Panels where introduced into the 12d Model macro Language, a macro console panel was the only method for writing information to the user, and soliciting answers from the user.

The **Macro Console** is no longer used in newer macros.

When a macro is invoked, a macro console panel is placed on the screen.

The macro console panel has three distinct areas
information/error message area
prompt message area
user reply area.

and optionally, three buttons, **restart**, **abort** and **finish**.

Using functions in this section, 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 panel.

Some of the functions have pop-ups defined (of models, tins etc.) so that information can be selected from pop-ups rather than being typed in by the user.

Also the **information/error message area** is used to display progress information. This information can be standard 4DML messages or user defined messages.

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 4DML 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.

The user can have the standard 4DML 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 4DML message
is displayed in the information/error message area.

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

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.

Prompt(Text msg)

Name

void Prompt(Text msg)

Description

Print the message **msg** to the **prompt message area** of the macro console

A function return value of zero indicates success.

Prompt(Text msg,Text &ret)

Name

Integer 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 console panel.

That is, write out the message **msg** and get a Text reply from the console panel. The reply is terminated by a <CR> or <enter>.

The reply is returned in Text **ret**.

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

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 panel.

That is, write out the message **msg** and get an integer reply from the console panel. The reply is terminated by a <CR> or <enter>.

The reply is returned in Integer **ret**.

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

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 panel. The reply is terminated by a <CR> or <enter>.

The reply is returned in Real **ret**.

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

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 panel.

If RB is pressed in the **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 a <CR> or <enter> for the macro to continue.

Hence the Choice_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a choice pop-up.

The reply is returned in Text **ret**.

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

Colour_prompt(Text msg,Text &ret)

Name

Integer Colour_prompt(Text msg,Text &ret)

Description

Print the message **msg** to the macro console and then read back a Text from the console panel.

If RB is pressed in 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 placed in the **user reply area**.

The reply, either typed or selected from the colour pop-up, must be terminated by a <CR> or <enter> for the macro to continue.

Hence the Colour_prompt writes out the message msg and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a colour pop-up.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text **ret** is returned successfully.

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 continue.

A function return value of zero indicates the function terminated successfully.

File_prompt(Text msg,Text key,Text &ret)

Name

Integer File_prompt(Text msg,Text 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 panel.

If RB is pressed in 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**.

The reply, either typed or selected from the file pop-up, must be terminated by a <CR> or <enter> for the macro to continue.

Hence the File_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a file pop-up.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in 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**.

The reply, either typed or selected from the model pop-up, must be terminated by a <CR> or <enter> for the macro to continue.

Hence the Model_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a model pop

-up.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in 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 a <CR> or <enter> for the macro to continue.

Hence the Template_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a template popup.

The reply is returned in Text **ret**.

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

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 panel.

If RB is pressed in the **user reply area**, a list of all existing templates 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 a <CR> or <enter> for the macro to continue.

Hence the Tin_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a tin popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing templates 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 the SuperTin is 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 a <CR> or <enter> for the macro to continue.

Hence the Tin_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a tin popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in 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 a <CR> or <enter> for the macro to continue.

Hence the View_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a view popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in 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 a <CR> or <enter> for the macro to continue.

Hence the Yes_no_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a yes-no popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing plotter 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 a <CR> or <enter> for the macro to continue.

Hence the plotter_prompt writes out the message msg and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a plotter popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing sheet_size 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 a <CR> or <enter> for the macro to continue.

Hence the sheet_size_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a sheet_size popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing linestyle 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 a <CR> or <enter> for the macro to continue.

Hence the linestyle_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a linestyle popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing textstyle 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 a <CR> or <enter> for the macro to continue.

Hence the textstyle_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a textstyle popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing Justify 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 a <CR> or <enter> for the macro to continue.

Hence the Justify_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a Justify popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in 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 a <CR> or <enter> for the macro to continue.

Hence the Angle_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a Angle pop-

up.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing Function 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 a <CR> or <enter> for the macro to continue.

Hence the Function_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a Function popup.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing Project 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 a <CR> or <enter> for the macro to continue.

Hence the Project_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a Project pop-up.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in the **user reply area**, a list of all existing Directory is placed in a pop-up. If a Directory is selected from the pop-up (using LB), the Directory name is placed in the **user reply area**.

The reply, either typed or selected from the Directory pop-up, must be terminated by a <CR> or <enter> for the macro to continue.

Hence the Directory_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a Directory pop-up.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 panel.

If RB is pressed in 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 a <CR> or <enter> for the macro to continue.

Hence the Text_units_prompt writes out the message **msg** and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a Text_units pop-up.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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 three values separated by one or more spaces.

The values are returned in **x, y** and **z**.

A function return value of zero indicates values x, y and z are successfully returned.

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 panel.

If RB is pressed in the user reply area, a list of all existing Name 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 a <CR> or <enter> for the macro to continue.

Hence the Name_prompt writes out the message msg and gets a Text reply from the console panel. The reply is terminated by a <CR> or <enter>. The reply may be selected from a Name pop-up.

The reply is returned in Text **ret**.

A function return value of zero indicates the Text ret is returned successfully.

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

Example of defining and using Panel_prompt.

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);
```

6 Examples

When using these code examples check the ends of lines for wordwrapping.

Example 1

A macro to select a string and write outs out to the console how many points there are in the string.

See Example 1 **OR** open 4dm file

Example 2

Macro to select a string and ask if its ok to delete it.

See Example 2 **OR** open 4dm file

Example 3

Write four lines of data out to a file and then read it back in again.

See Example 3 **OR** open 4dm file

Example 4

Read a file in and calculate the number of lines and words.

See Example 4 **OR** open 4dm file

Example 5

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.

See Example 5 **OR** open 4dm file

Example 6

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.

See Example 6 **OR** open 4dm file

Example 7

A macro that exercises many of the Text functions

See Example 7 **OR** open 4dm file

Example 8

A macro to label the spiral and curve lengths of an Alignment string

See Example 8 **OR** open 4dm file

Example 9

Macro to write out a line style or titleblock file from a 12d Model.

See Example 9 **OR** open 4dm file

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

See Example 10 **OR** open 4dm file

Example 11

Macro to delete a selected empty model or all empty models in a project.

See Example 11 **OR** open 4dm file

Example 12

Macro to change names of selected strings

See Example 12 **OR** open 4dm file

Example 13

Macro to use the x,y,z of a text string and create a new 3d point string at the same point.

See Example 13 **OR** open 4dm file

Example 14

This is an example of the 4DML functions for a dialogue that contains most of the common wigit controls. The text for the wigits and the on/off switch are contained in the function call go_panel.

Set_ups.h

See Set_ups.h

Set Ups.h

```
#ifndef set_ups_included
#define set_ups_included

#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                      22
#define CHECK_FILE_MUST_EXIST           1
#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 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 GET_TIN                             10

#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

#define TRUE                        1
#define FALSE                       0

#define OK                          1

// 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 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_World_Annotate              30
#define Att_Annotate_Type               31

#define concat(a,b) a##b
#define String_Super_Bit(n) (1 << concat(Att_,n))

#define All_String_Super_Bits 65535

#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
```

```
#endif
```

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 points there are in the string.
// This is then repeated.
// The macro terminates if cancel is selected from pick ops menu
//---------------------------------------------------------------
void main (){
  Element string;
  Integer ret,no_pts;
  Text text;
ask:
  ret = Select_string("Select a string",string);
  if(ret == -1) {
    Prompt("Macro finished - cancel selected");
    return;
  } else if (ret == 1) {
    if(Get_points(string,no_pts) !=0) goto ask;
    text = To_text(no_pts);
    text = "There are " + text + " points in the string";
    Prompt(text);
    goto ask;
  } else {
    goto ask;
  }
}
```

Example 2

```
// ------------------------------------------------------------
// Programmer   Lee Gregory
// Date         26/5/94
// Description of Macro
// Macro to select a string and ask if its ok to delete it.
// The macro loops round until cancel is selected from
// the pick ops menu.
// ------------------------------------------------------------
void main (){
  Element string;
  Integer ret,no_pts;
  Text text;
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);
  }
  goto ask;
}
```

Example 3

```
//-------------------------------------------------------------
// Programmer   Alan Gray
// Date         27/5/94
// Description of Macro
// Write four lines of data out to a file
// and then read it back in again.
// Report the number of lines read in.
//-------------------------------------------------------------
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);
  File_rewind(file);
  Integer count = 0;
  while(1) {
    Text line;
    if(File_read_line(file,line) == -1) break;
    ++count;
  }
  File_close(file);
// display # lines read
  Prompt(To_text(count) + " lines read");
}
```

Example 4

```
//-------------------------------------------------------------
// Programmer   Alan Gray
// Date         27/5/94
// 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.
//-------------------------------------------------------------
void main()
{
  File file;
  File_open("report.rpt","r",file);
  Integer eof,count = 0 , wordc = 0;
  while(1) {
    Text line;
    if(File_read_line(file,line) == -1) break;
    ++count;
// break line into words
    Dynamic_Text words;
    Integer      no_words = From_text(line,words);
    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);
      Prompt(t);
    }
  }
  File_close(file);
// display data read
  Prompt(To_text(count) + " lines & " + To_text(wordc) + "words read");
}
```

Example 5

```
//-------------------------------------------------------------
// 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 a called function as well as main.
//-------------------------------------------------------------

// Function to add new_model to all the non-section views that
// old_model is on
void add_to_view(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);
  }
}
// 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;
  Model   ljg_model,pad_model;
  Integer ret,side,error,closed;
  Text    text,tside,ok;
  Real    cut,fill,sep;
  Tin     tin;
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 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");
      goto ask;
    }
// get cut and fill slopes, side to interface
// and separation between sections
    Prompt("Cut slope",cut);
    Prompt("Fill slope",fill);
    Prompt("Separation",sep);
    Prompt("Left or Right (l or r)",tside);
    side = (tside == "l") ? -1 : 1;
tin:
    Prompt("Tin name",text);
    if(text == "") return;
```

```
    if(!Tin_exists(text)) goto tin;
    tin = Get_tin(text);
// calculate the interface
    Interface(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
    Model_delete(Get_model("LJG"));  // delete model LJG
    ljg_model = Get_model_create("LJG");
    Set_model(int_string,ljg_model);
    Get_model(pad,pad_model);
    add_to_view(ljg_model,pad_model);   // user defined function
    Prompt("OK to continue (y or n)",ok);
    if(ok == "n") {
      Element_delete(int_string);
      goto ask;
    }
// 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:
    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 views as the model containing the string was on
```

```
cleaned:
    Element_delete(int_string);
    Set_model(clean_string,ljg_model);
// add the cleaned string onto it
    add_to_view(ljg_model,pad_model);
  }
// triangulate the pad and interface
  Dynamic_Element detin;
  Append(clean_string,detin);
  Append(pad,detin);
  Integer   no_pts;
  Get_points(clean_string,no_pts);
  Tin pad_tin;
  Integer no_items;
  Tin_delete(Get_tin("pad"));    // delete the tin pad
  Triangulate(detin,"pad",1,1,1,pad_tin);
// do volumes between the ground and pad
  Real  cut_vol,fill_vol,bal_vol;
  Volume_exact(tin,pad_tin,clean_string,cut_vol,fill_vol,bal_vol);
// display the volumes
  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;
  Prompt(out_text);
  return;
}
```

Example 6

```
//-------------------------------------------------------------
// 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.
//-------------------------------------------------------------
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 7

```
//------------------------------------------------------------
// 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 l = 1234567;
  Real    r = 987654.321;
  Text    b = To_text(l,"l = %8ld") + " "+ 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 8

```
//-------------------------------------------------------------
// Programmer   Lee Gregory
// Date         30/9/94
// Description of Macro
// A macro to label the spiral and curve lengths of
// an Alignment string
//-------------------------------------------------------------
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 9

```
//-------------------------------------------------------------
// Programmer     Lee Gregory
// Date           6/9/94
// Description of Macro
// Macro to write out a line style or titleblock file from
//  a 12d Model.
// Only 2d, 3d line strings, circles, arcs and text will
// be used for the style.
//-------------------------------------------------------------
//  Routine to take the plot_type and create the appropriate
//  linestyle name
Text get_plot_type (Integer plot_type)

{
  switch (plot_type) {
    case 0 : {
      return("plot_frame_title_box");
    }
    case 1 : {
      return("long_section_title_box");
    }
    case 2 : {
      return("x_section_title_box");
    }
  }
  return("plot_frame_title_box");
}

//  Routine to write out a
//    move  (mode = 0) or draw  (mode != 0)
//  to the co-ordinates (x,y) with "prec" being the
//  number of decimal places tp write x and y out to.
void move_draw(File file,Integer mode,Real x,Real y,

               Integer prec)
{
  Text output,type;
  if(mode == 0) {
    type = "    move ";
```

```
  } else {
    type = "    draw ";
  }
  output = type + To_text(x,prec) + " " + To_text(y,prec);
  File_write_line(file,output);
}
//  Routine to take a text string justification and
//  create the equivalent linestyle text justification
Text text_justification (Integer justify)
{
  switch (justify) {
    case 1 : {
      return("\"bottom-left\"");

    }
    case 2 : {
      return("\"middle-left\"");
    }
    case 3 : {
      return("\"top-left\"");
    }
    case 4 : {
      return("\"bottom-centre\"");
    }
    case 5 : {
      return("\"middle-centre\"");
    }
    case 6 : {
      return("\"top-centre\"");
    }
    case 7 : {
      return("\"bottom-right\"");
    }
    case 8 : {
      return("\"middle-right\"");
    }
    case 9 : {

      return("\"top-right\"");
    }
  }
```

```
  return("\"bottom-left\"");
}
//  Main program
void main()
{
// Get the model, scale, style type, file to write
// the style to.
  Text    model_name,report_name,style_name,output;
  Real    scale,factor;
  File    file;
  Integer err,plot;
//  get the model to use
model:
  Model_prompt("Model to create a style from :",model_name);
  if(!Model_exists(model_name)) goto model;
//  get the scale

scale:
  err = Prompt("Scale for model 1: x (def 1000)",scale);
  if(err != 0) scale = 1000.0;
  if (Absolute(scale) < 1.0e-9) scale = 1000.0;
  factor = 1000.0/Absolute(scale);
// get Style name for titleblock or linestyle
plot_type:
  err = Prompt("Plot type :  frame = 0  long = 1  x = 2"+
               "  style = 3",plot);
  if(err != 0) goto plot_type;
  if((plot < 0) || (plot > 3)) goto plot_type;
  if(plot < 3) {
    style_name = "linestyle " + get_plot_type(plot) + " {";

  } else {
    if(Prompt("Style name",style_name) != 0) goto plot_type;
    style_name = "linestyle " + "\"" + style_name + "\" {";
  }
// get the file to write the style out to
report:
  Prompt("File for titleblock",report_name);
  if(File_exists(report_name)) goto report;
  if(File_open(report_name,"w",file) != 0) goto report;
  File_write_line(file,style_name);
```

```
  output = "  factor " + To_text(factor,3);
  File_write_line(file,output);

// Do the processing :
// Get handles to all the strings (Elements) in the model
  Model model;
  model = Get_model(model_name);
// Check in case the model has been deleted since
// first selected it.
  if(!Model_exists(model)) goto model;
  Dynamic_Element model_elts;
  Integer no_elts;
// Go through all the elements in the model
// and whenever possible, convert strings to
// line style commands.
  Get_elements(model,model_elts,no_elts)
;
  for (Integer i=1;i<=no_elts;i++) {

    Element elt;
    Text    type,colour,text;
    Integer break_type,num_pts,colour_num,justif,size_type;
    Real
x,y,z,radius,xs,ys,zs,xe,ye,ze,start,end,size,angle,degrees,offset,rise,cosang,sinang,xpos,ypos;
    Get_item(model_elts,i,elt);
    Get_type(elt,type);
    Get_points(elt,num_pts);
    Get_breakline(elt,break_type);
    Get_colour(elt,colour_num);
    Convert_colour(colour_num,colour);
    if(type == "2d") {

      if(break_type != 1) continue;
      if(num_pts <2) continue;
      output = "  colour " + colour;
      File_write_line(file,output);
      Get_2d_data(elt,1,x,y);
      move_draw(file,0,x,y,3);
      for(Integer j=2;j<=num_pts;j++) {
        Get_2d_data(elt,j,x,y);
        move_draw(file,1,x,y,3);
```

```
    }
    File_write_line(file,"  ");
  } else if(type == "3d") {
    if(break_type != 1) continue;
    if(num_pts <2) continue;
    File_write_line(file,"  colour " + colour);

    Get_3d_data(elt,1,x,y,z);
    move_draw(file,0,x,y,3);
    for(Integer j=2;j<=num_pts;j++) {
      Get_3d_data(elt,j,x,y,z);
      move_draw(file,1,x,y,3);
    }
    File_write_line(file," ");
  } else if(type == "Circle") {
    File_write_line(file,"  colour " + colour);
    Get_circle_data(elt,x,y,z,radius);
    move_draw(file,0,x,y,3);
    File_write_line(file,"    circle " +
                    To_text(radius,3));

  } else if(type == "Arc") {
    File_write_line(file,"  colour " + colour);
    err = Get_arc_data(elt,x,y,z,radius,xs,ys,zs,xe,ye,ze);
    angle = Atan2(ys-y,xs-x);
    Radians_to_degrees(angle,start);
    angle   = Atan2(ye-y,xe-x);
    Radians_to_degrees(angle,end);
    move_draw(file,0,x,y,3);
    output = "    arc " + To_text(radius,3) +" " +
          To_text(start,3) + " " + To_text(end,3);
    File_write_line(file,output);

  } else if(type == "Text") {
    File_write_line(file,"  colour " + colour);
    Get_text_data(elt,text,x,y,size,colour_num,angle,justif,size_type,offset,rise);
//  work out start of text with rise and offset
    cosang = Cos(angle);
    sinang = Sin(angle);
    xpos = x + offset*cosang + rise*sinang;
    ypos = y + offset*sinang - rise*cosang;
```

```
      move_draw(file,0,xpos,ypos,3);
      Radians_to_degrees(angle,degrees);
      output = "    text " + "\"" +text + "\"  " +To_text(degrees,3) +" " + To_text(size,3) + " "
+text_justification(justif);


      File_write_line(file,output);
    }
    File_write_line(file,"  ");
  }
  File_write_line(file,"}");
  File_close(file);
  Prompt("Finished writing titleblock");
}
```

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
//-------------------------------------------------------
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.
//--------------------------------------------------------------------
// 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"
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);
}
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);
}
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;
  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("Set new string name(s)");
```

```
  Message_Box    message = Create_message_box("  ");
  Choice_Box model_list = Create_choice_box("Empty model",message);
  update_list(model_list);
// buttons along the bottom
  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);
  Append(model_list,panel);
  Append(message,panel);
  Append(bgroup,panel);
  Show_widget(panel);
  Integer doit = 1;
  Integer no_deleted = 0;
  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(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
//--------------------------------------------------------------------
// 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"
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.
//--------------------------------------------------------------------
// 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"

#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 wgits 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,"4d.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,sel
ect1_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
}
```

A Appendix - Set_ups.h File

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 Mode
See Select Mode

Model Mode

MODE	MODE NUMBER
NO_MODEL	1
MODEL_EXISTS	2
DISK_MODEL_EXISTS	19
NEW_MODEL	3
CHECK_MODEL_MUST_EXIST	7
CHECK_MODEL_EXISTS	3
CHECK_MODEL_CREATE	4
CHECK_DISK_MODEL_MUST_EXIST	33
CHECK_EITHER_MODEL_EXISTS	38
GET_MODEL	10
GET_MODEL_CREATE	5
GET_MODEL_ERROR	13
GET_DISK_MODEL_ERROR	34
CHECK_MODEL_MUST_NOT_EXIST	60
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

File Mode

MODE	MODE NUMBER
NO_FILE	4
FILE_EXISTS	5
CHECK_FILE_MUST_EXIST	1
CHECK_FILE_CREATE	14
GET_FILE_CREATE	15
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

View Mode

MODE	MODE NUMBER
NO_VIEW	6
VIEW_EXISTS	7
CHECK_VIEW_MUST_EXIST	2
CHECK_VIEW_MUST_NOT_EXIST	25
GET_VIEW	11
GET_VIEW_ERROR	6
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

Tin Mode

MODE	MODE NUMBER
NO_TIN	9
TIN_EXISTS	11
DISK_TIN_EXISTS	12
CHECK_TIN_MUST_EXIST	8
CHECK_TIN_EXISTS	61
CHECK_EITHER_TIN_EXISTS	39
CHECK_TIN_NEW	12
GET_TIN_ERROR	9
CHECK_DISK_TIN_MUST_EXIST	16
GET_TIN_CREATE	24
GET_DISK_TIN_ERROR	35
CHECK_TIN_MUST_NOT_EXIST	91
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

Template Mode

MODE	MODE NUMBER
NO_TEMPLATE	13
TEMPLATE_EXISTS	14
DISK_TEMPLATE_EXISTS	20
NEW_TEMPLATE	15
CHECK_TEMPLATE_EXISTS1	7
CHECK_TEMPLATE_CREATE	18
CHECK_TEMPLATE_NEW	19
CHECK_TEMPLATE_MUST_EXIST	20
CHECK_TEMPLATE_MUST_NOT_EXIST	59
GET_TEMPLATE	21
GET_TEMPLATE_CREATE	22
GET_TEMPLATE_ERROR	23
GET_DISK_TEMPLATE_ERROR	40
CHECK_DISK_TEMPLATE_MUST_EXIST	48
CHECK_EITHER_TEMPLATE_EXISTS	49
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

Project Mode

MODE	MODE NUMBER
NO_PROJECT	16
PROJECT_EXISTS	17
NEW_PROJECT	18
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
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

Directory Mode

MODE	MODE NUMBER
NO_DIRECTORY	21
DIRECTORY_EXISTS	22
NEW_DIRECTORY	23
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
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

Function Mode

MODE	MODE NUMBER
NO_FUNCTION	24
FUNCTION_EXISTS	25
DISK_FUNCTION_EXISTS	26
NEW_FUNCTION	27
CHECK_FUNCTION_MUST_EXIST	50
CHECK_FUNCTION_EXISTS	51
CHECK_FUNCTION_CREATE	52
CHECK_DISK_FUNCTION_MUST_EXIST	53
CHECK_EITHER_FUNCTION_EXISTS	54
GET_FUNCTION	55
GET_FUNCTION_CREATE	56
GET_FUNCTION_ERROR	57
GET_DISK_FUNCTION_ERROR	58
CHECK_FUNCTION_MUST_NOT_EXIST	90
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

Linestyle Mode

MODE	MODE NUMBER
LINESTYLE_EXISTS	80
NO_LINESTYLE	81
CHECK_LINESTYLE_MUST_EXIST	82
CHECK_LINESTYLE_MUST_NOT_EXIST	83
GET_LINESTYLE	84
GET_LINESTYLE_ERROR	85
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

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 Mode

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	

Select Mode

MODE	MODE NUMBER
SELECT_STRING	5509
SELECT_STRINGS	5510
NO_NAME	10
NO_CASE	8
TRUE	1
FALSE	0
OK	1

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Y

12D Solutions Pty Ltd

Civil and Surveying Software

Course Notes

12dModel

Programming Language

12D Solutions Pty Limited

ACN 101 351 991

Phone: +61 (2) 9970 7117Fax: +61 (2) 9970 7118Email training@12d.com Web www.12d.com

COURSE NOTES

Macro Language

12d Model 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

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.

Copyright © 12D Solutions Pty Limited 2010

Macro Language Course

1.0 Course Introduction

The 12D Solutions Macro Language (4DML) is a powerful programming language designed to run from within 12D Solutions software12d Model.

Its main purpose is to allowusers to enhance the existing 12D Solutions package by writing their own programs (macros).

4DML 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.

4DML 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 4DML.

This course intends to teach you how to

1. · Learn the basic 4DML variable types and "handles" to 12d elements (strings etc.).
2. · How to use the 4DML manual as a "live" programming reference.
3. · How to create/compile and run 4DML code.
4. · How to retrieve and change basic element properties.
5. · File input/output (creating reports).
6. · An introduction to 4DML screen input/output through panels.
7. · How to include your 4DML programs in the 12d menu system.

2.0 Getting Started

2.1 Comments

Comments are extremely important for writing any program. The following is an example of 4DML code with single and multiple line comments. **More**

```
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 */
}
```

2.2 Variables and Operators

2.2.1 Reserved Words

Reserved words (or Keywords) are the words that you are **not** allowed to use as variable names in your

4DML code**. More.**

2.2.2 Integers, Real and Text

All variables must be declared before they are used. **More**

for example

```
Integer i;
```

or

```
Integer i=2;
```

2.2.3 Arrays

Arrays may be allocated statically or dynamically. **More**

WARNING: subscripts start at 1!

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];
```

2.2.4 Operators

The most common operators are

assignment

=	assignment	e.g. x = y

More

binary arithmetic operators

+	addition
-	subtraction
*	multiplication
/	division - note that integer division truncates any fractional part

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

increment and decrement operators

++	post and pre-increment
--	post and pre-decrement

3.0 Functions and Your First Macro

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 first line of a function would look like the following

```
Real calc_distance(Real x1, Real y1, Real x2, Real y2);
```

This function has the real values of x1,y1,x2,y2 passes to it. The function body (not shown) would calculate the distance and return the distance as a real number. When the function is called inside the 4DML the code would look like the following.

```
distance = calc_distance(x1,y1,x2,y2);
```

The arguments (constants or variables) of the function can be **passed by value** (a one way transfer) as above 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 below are passed by reference.

```
Real calc_distance(Real &x1, Real &y1, Real &x2, Real& y2);
```

With the **passed by reference** the argument variable in the calling routine can be changed by the function.

WARNING! Function named are case sensitive!

3.1 Prompt().... your first 4DML function

This is the first function from 4DML that we will examine. If we search for print in the help system we will find the following function.

void Prompt(Text msg)

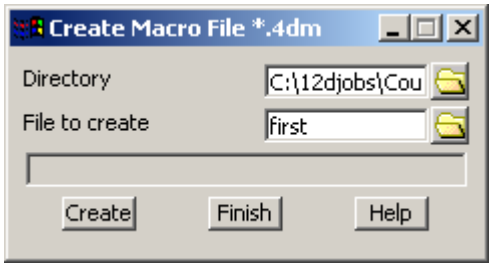
This may be read as, “The function **prompt** has no return value (void) and has a text argument (msg for example)”. The argument is passed by value (there is not ampersand &).

3.2 Creating Your First Macro

From the Main menu select

Utilities=>Macros=>Create

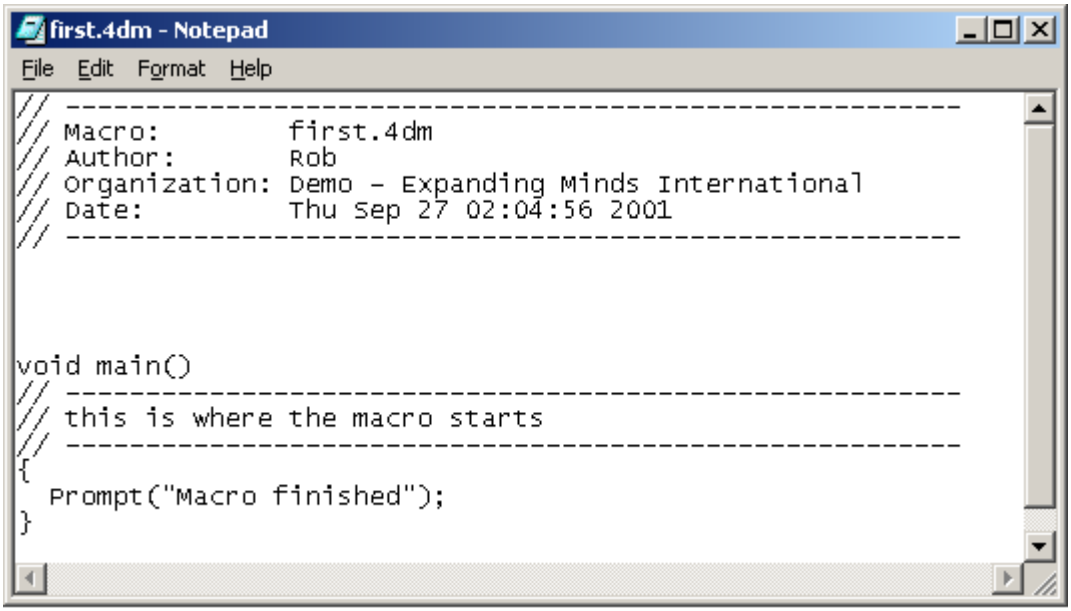
and the following panel will appear.



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.



The first few lines are comments (beginning with the //). Following is the function main().

All macros 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).

You will note that the main function has one line of executable code and that includes the **Prompt()** function. The **Prompt()** function can have a constant or text variable as its argument. In this case it is a constant.

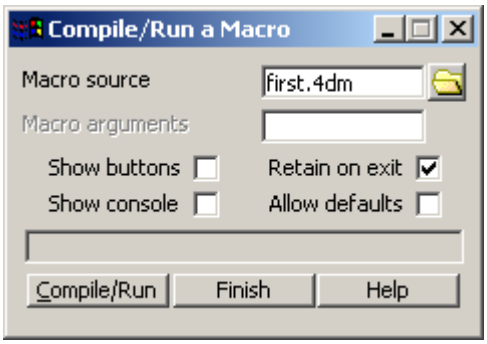
When run, this macro will place the words **Macro finished** in the prompt box and then stop.

3.3 Compiling the Macro

From the Main menu select

Utilities=>Macros=>Compile/run

and the following panel will appear.

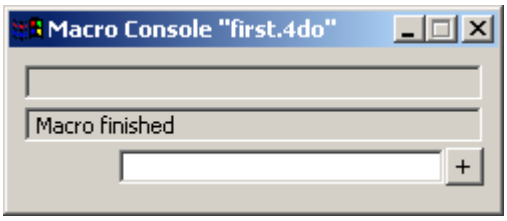


Select the **browse** icon and then select the macro code text file.

Select **Retain on Exit** so that the prompt box will remain after the macro finishes.

Select **Compile/Run** and the following prompt dialogue will appear.

Note that the dialogue has the macro name on the top and in the message area the words **Macro finished** appear.



You have just created and run your first macro!

4.0 Common Compile 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 **Prompt()** function and then recompile the macro. What do you notice about the line number that the compiler reports?

Next put the semi colon back in and remove one of the " marks in the command. Now recompile this and check the messages.

5.0 Using Input and Output Functions

You have seen one method of output from the 4DML. You may also create output by writing to the output window, placing text on the clipboard or writing to files. Input to the 4DML may be via the Macro console or via custom dialogue boxes with advanced error checking.

5.1 Output to the Output Window

The **Print()** function is used to print to the output window. Unlike the **Prompt()** function which can only take a text message as an argument, there are 3 functions with the same name **Print()** but each **Print()** function has different argument types. This is called **Overloading of Function Names**. If you use the Find feature of the help file, you will find the 3 **Print()** functions.

void Print(Integer value)

void Print(Real value)

and

void Print(Text msg)

Inside the brackets are the arguments that are passed to the function. Since there is a Print() function for all three variable types (integer, real and text), the Print() function will print an integer, real or text.

Prior to using the **Print()** function, consider using the **Clear_console()** function to clear the window. This function does not have any arguments.

Edit your macro so that it now contains the following lines of code.

```
void main()
{
  Clear_console();

  Print("this is text\n");

  Print(1);
  Print("\n");

Prompt("did you see this?");

  Print(2.2);
  Print("\n");

  Prompt("Macro finished");
}
```

Note the special line feed character "\n" has been printed to move the printing to the next line. If there is no line feed character then the line of text will not be printed.

You will also note that the message "did you see this?" flashed by the prompt window so fast that you never saw it. If you want the macro to stop execution 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.

Try changing

```
Prompt("did you see this?");
```
to

```
Error_prompt("did you see this?");
```

5.1.1 Input via the Macro Console (quick and easy)

A simple method to input data is via the Macro Console. The **Prompt()** function can again be used but now with 2 arguments. Note that in the help file the variable name of the second argument is preceded with a **&**. This indicates that the variable is **passed by reference.**

Integer Prompt(Text msg,Text &ret)

Integer Prompt(Text msg,Integer &ret)

Integer Prompt(Text msg,Real &ret)

Lets change our macro so that it now asks for the values before they are printed.

```
void main()
{
  Clear_console();

  Text input_text;
  Prompt("Enter some text",input_text);
  Print(input_text+"\n");


  Integer input_integer;
  Prompt("Enter an integer",input_integer);
  Print(input_integer);
  Print("\n");

  Real input_real;
  Prompt("Enter a real",input_real);
  Print(input_real);
  Print("\n");

  Prompt("Macro finished");
}
```

5.1.2 Dialogue Boxes (covered later)

4DML can create advanced dialogue boxes complete with error checking. We will be discussing these in more detail later.**More**

5.1.3 Files

ASCII text files can be created and read via the 4DML functions. **More**

5.1.4 Clipboard

ASCII data may be written to and read from the windows clipboard with the following 4DML functions.

Integer Set_clipboard_text(Text string);

Integer Get_clipboard_text(Text &string);

6.0 Using Flow Control

The 4DML has a subset of the C++ flow control statements. **More** We will work with only three in this course.

4DML statements are grouped together as **blocks**.A block begins with a { and ends with a }.

IMPORTANT!!!

Note that any variables declared inside the block will “go out of scope” (evaporate) as soon as execution leaves the block.

6.1 “if” statements

If statements are used frequently to execute a block of statements only if a condition is true or false.

```
if (conditional) {
     // these statements are executed if the conditional is true
} else {
     // these statements are executed if the conditional is false
}
```

Now change your macro so that it has the following conditional statements.

```
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;
  Prompt("Enter an integer",input_integer);
  if(input_integer > 10) Print(input_integer);
  else                   Print("The number is less than 10");
  Print("\n");

  Real input_real;
  Prompt("Enter a real",input_real);
  Print(input_real);
  Print("\n");

  if(input_real > 0) Print(20./input_real);
  Print("\n");

  Prompt("Macro finished");
}
```

6.2 “for” loops

A **for** loop is appropriate when a block has to be executed a fixed number of times. **More**

Here is an example of the **for loop.**

```
void main()
{
  Clear_console();

  Integer loop;
  Prompt("Enter number of loops",loop);

  for(Integer counter = 1;counter<=loop;counter++) {

    if(counter < (loop / 2)) {
      Print("first half ");
      Print(counter);
      Print("\n");
      continue;
    }

    Print("Last half ");
    Print(counter);
    Print("\n");

  }

}
```

Try entering a value of 5 when you run the macro. Can you explain the results?

6.3 “while” loops

while loops are convienent for executing a block of statements until a condition is reached. Below is an example of a **while** loop.

```
void main()
{
  Clear_console();

  Text data;

  while (data != "stop") {

    Prompt("Enter some text",data);
    Print(data+"\n");

  }
}
```

7.0 Unleashing the Power - 12d Database Handles

The real power of the 4DML is accessed via the 12d database. This database holds all of the elements inside the project. Every entity in the database has an handle. Once this handle has been retrieved the properties of the entity may be obtained, printed in a report or changed.

New entities can also be created. Data can be read from reports and then strings can be created and formatted to the users specifications. **More**

7.1 Locks

Whenever an handle for an entity (string, model, tin etc.) is retrieved from the database and assigned to a variable, the entity becomes locked to other processes. In order to remove the lock, the variable holding the handle most 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 macro is in a user input mode. **More**

7.2 Models

Macros often operate on all of the elements in a model. When a model is requested by the user the first step is to retrieve the model handle.

Sample code for this follows,

```
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);
  }

Integer model_id;
  Get_id(my_model,model_id);
  Print("Model id ");
  Print(model_id);
  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");

}
```

7.3 Elements, Dynamic_Elements, Points and Properties

Note that in the example above, we have declared a variable as a **Dynamic_Element.** This variable will hold as many **elements** as the model has. This is convienent since we do not initially know how may elements are in the model. The

Integer **Get_elements**(Model model,Dynamic_Element &de, Integer &total_no)

function gets all of the element handles and the number of elements retrieved. While this Dynamic_Element exists, all of the elements will be locked.

Now we will add to this macro to retrieve and print the element names, the type and the number of points on each element.

```
void main() {

  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);
  }

  Integer model_id;
  Get_id(my_model,model_id);
  Print("Model id ");
  Print(model_id);
  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(element_name+"\n");


    Integer element_id;
    Get_id(element,element_id);
    Print(element_id);
    Print("\n");

    Text element_type;
    Get_type(element,element_type);
    Print(element_type+"\n");

    Integer num_points;
    Get_points(element,num_points);
    Print(num_points);
    Print("\n\n");


  }
}
```

8.0 Writing Reports

The previous example can be modified to write the data to a file rather than to the output window.

To write a report three 4DML functions are required.

Integer File_open(Text file_name, “w”,File &file) to write a new file

or

Integer File_open(Text file_name, “a”,File &file) to append

Integer File_write_line(File file,Text text_out)

and finally a close command

Integer File_close(File file)

More

A routine with the file commands follows;

```
void main() {

  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,"a",my_file);

  Integer model_id;
  Get_id(my_model,model_id);
  File_write_line(my_file,"Model id "+To_text(model_id));

  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";


    Integer element_id;
    Get_id(element,element_id);
    line_out += To_text(element_id)+"\t";

    Text element_type;
    Get_type(element,element_type);
    line_out += element_type+"\t";

    Integer num_points;
    Get_points(element,num_points);
    line_out += To_text(num_points);
    File_write_line(my_file,line_out);

  }

  File_close(my_file);
}
```

9.0 12d Menu System (Usermenu.4d)

The macros that you create should be stored in the user library. If you want to access these macros via the 12d menu system you will need to create the usermenu.4d file and keep it in the user area (not the user_lib). An example of the entries in the usermenu.4d follow.

```
Menu "User String Create" {
     Button "Create 4d strings" {
          Command "macro  -close_on_exit  $USER_LIB/ref_points.4do"
     }
     Button "Create point strings" {
          Command "macro  -close_on_exit  $USER_LIB/x_sects.4do"
     }
}
```

The menu item (`"User String Create" for example)` must correspond to the name on the top of the 12d user menu that you wish to attach your macro to. Buttons and sub menus may be created as desired.

10.0 Dialogue Basics

The basic structure of 12d dialogue code is as follows.

```
Create the panel
Create the vertical group
Create the wigits and add them to the vertical group

create a while loop that returns processing to the top of the loop
until a process or finish buttons are selected

perform final validation of retrieve database handles

execute the desired task

return to the top of the loop
```

Sample code for a model selection of an existing model follows. This code requires the set_ups.h file that should be found in the 12d library.

COURSE NOTES

Macro Language

```
// ------------------------------------------------------
// Macro:        final.4dm
// Author:       Rob
// Organization: Demo - Expanding Minds International
// Date:         Thu Sep 27 05:41:44 2001
// ------------------------------------------------------


#include "set_ups.h"

//--------------------------------------------------------------
//                         MODEL CHECK
//--------------------------------------------------------------

Integer list_model(Model model1_model)
{
    Text my_model_name;
    Get_name(model1_model,my_model_name);

    Integer model_id;
    Get_id(model1_model,model_id);
    Print("Model id ");
    Print(model_id);
    Print("\n");

    Dynamic_Element model_elts;
    Integer num_elts;

    Get_elements(model1_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(element_name+"\n");


      Integer element_id;
      Get_id(element,element_id);
      Print(element_id);
      Print("\n");

      Text element_type;
      Get_type(element,element_type);
      Print(element_type+"\n");

      Integer num_points;
      Get_points(element,num_points);
```

```
        Print(num_points);
        Print("\n\n");


      }

      return 0;
  }


  Integer go_panel()
  {
    // ================================================================
    // get defaults at the start of a routine and set up the panel


    Integer ok=0;

    //----------------------------------------------------------------
  -
    //                          CREATE THE PANEL
    //----------------------------------------------------------------
  -

    Panel panel = Create_panel("Model Select");
    Vertical_Group  vgroup = Create_vertical_group(0);
    Message_Box message_box = Create_message_box("");



    // ----------------- model1_ name --------------------------------

    // model1_name

    Model_Box model1_box;
    model1_box = Create_model_box("Select
```

```
model",message_box,CHECK_MODEL_MUST_EXIST);
  Append(model1_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","process");
  Append(process_button,button_group);

  Button finish_button = Create_button("Finish","finish");
  Append(finish_button,button_group);

  Append(button_group,vgroup);

  Append(vgroup,panel);

  // ---------------- display the panel -------------------------

  Integer wx = 100,wy = 100;
  Show_widget(panel,wx,wy);

  // ------------------------------------------------------------
  //                    GET AND VALIDATE DATA
  // ------------------------------------------------------------

  Integer done = 0;
  while (1) {

    Integer id,ierr;
    Text cmd,msg;
    Wait_on_widgets(id,cmd,msg);

    Print(" id  <"+To_text(id));
    Print("> cmd <"+cmd);
    Print("> msg <"+msg+">\n");


//-------------------------------------------------------------------
-------------------------
//  first process the command that are common to all wigits or are
rarely processed by the wigit ID
//-------------------------------------------------------------------
```

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

    switch(cmd) {
      case "keystroke" :
      case "set_focus"  :
      case "kill_focus" : {

        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;
      } break;

      case Get_id(finish_button) : {
        Print("Normal Exit\n");
        return(0);
      } break;

      case Get_id(process_button) : {

        Model model1_model;
        if(Validate(model1_box,GET_MODEL_ERROR,model1_model) !=
MODEL_EXISTS) continue;

       if(list_model(model1_model)) Set_data(message_box,"Processing
```

```
  encountered an error");
          else Set_data(message_box,"Processing complete");

        } break; // process

        default : {
          continue;
        }

      }  // switch id

  }  // while !done

    return ok;
  }

  void main() {
    Clear_console();
    go_panel();
  }
```

More controls

Quick start panel example