12d® Model™

12d XML File Format

Version 15

May 2022

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 www.12d.com

12d XML File Format

This document is the 12d XML File Fromat taken from the Reference Manual for the software product 12d Model.

Disclaimer

Copyright

Table of Contents

1 12d XML File Format

Extensible Markup Language (***XML***) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable. It is defined by the ***World Wide Web Consortium's*** (***W3C***) XML Specifications which are free open standards.[

The 1***2d XML*** file format is a text file definition from ***12d Solutions*** which is used for reading and writing out string data from ***12d Model***. 12d XML files normally end in **.12dxml**.

The ***12d XML*** file is a **Unicode** file.

This document is for the ***12d XML*** file format used in ***12d Model 15***.

For general comments see:

1.1 General Information about XML
1.2 General Information about a 12d XML File

For the **12d XML** definitions see:

1.4 Attributes
1.5 Model
1.6 Elements Contained in Models which includes
1.6.1 Tin
1.6.2 Super Tin
1.6.5 Arc String
1.6.6 Circle String
1.6.7 Drainage String
1.6.8 Feature String
1.6.9 Plot Frame String
1.6.10 Super String
1.6.11 Super Alignment String
1.6.12 Text String
1.6.13 Trimesh

For documentation on the ***12d Archive*** (***12da***) file format, see 38 12d Archive File Format.

1.1 General Information about XML

(Unicode) Character

By definition, an XML document is a string of characters. Almost every legal Unicode character may appear in an XML document.

Markup and Content

The characters making up an XML document are divided into ***markup*** and ***content***, which may be distinguished by the application of simple syntactic rules.

Generally, strings that constitute markup either begin with the character **<** and end with a **>**, or they begin with the character **&** and end with a **;**.

Strings of characters that are not markup are content.

However, in a CDATA section, the delimiters <![CDATA[and]]> are classified as markup, while the text between them is classified as content. In addition, whitespace before and after the outermost element is classified as markup.

Characters "<", ">" and "&"

The characters "<", ">" and "&" are key syntax markers and may never appear in content outside a CDATA section. They need to be represented by special escape sequences:

&lt represents "<"

&gt represents ">"

&amp represents "&"

Tag

An XML tag is a markup construct that begins with **<** and ends with **>**.

Tags come in three flavours:

(a) start-tags - for example: <section>

(b) end-tags - for example: </section>

(c) empty-element tags - for example: <line-break />

XML Element

A logical document component which either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag.

The characters between the start- and end-tags, if any, are the element's content, and may contain markup, including other elements, which are called child elements.

An example of an element is **<Greeting>**Hello, world.**</Greeting>**.

Another is **<**line-break **/>**.

Note: Because elements are ***12d Model*** items that are in a model, in the documentation of 12d XML we will refrain from using element for the element in XML. Instead we will use the words ***keyword block*** to refer to special XML Elements in 12d XML.

Empty XML Elements <keyword/>

When an XML element has no content it is called an **empty** element.

For example <name> </name>

There is special shorthand for empty elements:

<keyword/> is shorthand for **<keyword></keyword>**

XML Attribute

A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag. In the example (below) the element **img** has two attributes, **src** and **alt**:

<img src="madonna.jpg" alt='Foligno Madonna, by Raphael' **/>**

Another example would be

<step number="3">Connect A to B.**</step>**

where the name of the attribute is "number" and the value is "3".

An XML attribute can only have a single value and each attribute can appear at most once on each element.

Note: Because attributes are fundamental ***12d Model*** items, in the documentation of 12d XML the word attribute will refer to ***12d Model*** attributes.

The words **XML attribute** will always be used when there is need to refer to an XML attribute.

XML declaration

XML documents may begin by declaring some information about themselves, as in the following example:

<?xml version="1.0" encoding="UTF-8"?>

Escaping

XML provides escape facilities for including characters which are problematic to include directly. For example:

There are five predefined entities:

&lt represents "<"

&gt represents ">"

&amp represents "&"

&apos represents '

&quot represents "

**
** represents a new line.

XML Comments

Comments may appear anywhere in a document outside other markup. Comments cannot appear before the XML declaration.

Comments start with "**<!--**" and end with "**-->**".

For compatibility with SGML, the string "--" (double-hyphen) is not allowed inside comments; this means comments cannot be nested.

The ampersand has no special significance within comments, so entity and character references are not recognized as such, and there is no way to represent characters outside the character set of the document encoding.

An example of a valid comment: "<!--no need to escape <code> & such in comments-->"

Continue to 1.2 General Information about a 12d XML File or return to 1 12d XML File Format.

1.2 General Information about a 12d XML File

Unicode

12d XML file is a Unicode file.

Blank Lines

Unless they are part of a string of characters making up text, blank lines are ignored.

Names of Models, Tins and Super Tins

Models, tins, styles (linestyles), textstyles and colours can include the characters **a** to **z**, **A** to **Z**, **0** to **9** (alphanumeric characters) and **space**. Leading and trailing spaces are ignored. The names can be up to 255 characters in length.

The names for models, tins and super tins **can not** be blank.

The names for models, tins and super tins can contain upper and lower alpha characters which are stored, but for comparisons, the model names, tin names and super tin names are case insensitive. For example the model name "Fred" will be stored as "Fred" but "FRED" is considered to be the *same* model name as "Fred".

Within a project, each model name must be unique amongst all the model names in the project.

For tins and super tins, the name of a tin or a super tin must be unique amongst the combined list of tin names and super tin names.

Object Tree Names for Models and Tins

12d Model supports hierarchical (tree) names for models, tins and super tins, and the forward slash (**/**) is used to separate the different levels of the tree.

As for model and tins names themselves, each level name can include the characters **a** to **z**, **A** to **Z**, **0** to **9** (alphanumeric characters) and **space**. Leading and trailing spaces are ignored. The level names can be up to 255 characters in length.

For example, an object tree model name can be:

Stage 1/Water/Drainage

String Names

String names can include the characters **a** to **z**, **A** to **Z**, **0** to **9** (alphanumeric characters), **space**, **decimal point** (.), **plus** (+), **minus** (-), **comma** (,), **open** and **closed round brackets** and **equals** (=).

Leading and trailing spaces are ignored.

String names do not have to be unique and **can** be blank.

String names can contain upper and lower alpha characters which are retained but case is ignored when selecting by string name. That is, the string name **Fred** will be stored as **Fred** but **FRED** is not considered to be a different string name.

Attribute Names

Attribute names can include the characters a to z, A to Z, 0 to 9 (alphanumeric characters) and space. Leading and trailing spaces are ignored. The names can be up to 255 characters in length. Attributes names **can not** be blank.

Attribute names are case sensitive. That is, the attribute name "Fred" is different to "FRED".

Names of Linestyles (Styles), Textstyles and Colours

Linestyles (styles)), textstyles and colours can include the characters **a** to **z**, **A** to **Z**, **0** to **9** (alphanumeric characters) and **space**. Leading and trailing spaces are ignored. The names can be up to 255 characters in length.

The names for linestyles, textstyles or colours **can not** be blank.

The names for linestyles, textstyles and colours can contain upper and lower alpha characters which are stored, but for comparisons, the linestyle names, textstyle names and colour names are case insensitive. For example the linestyle name "Bypass" will be stored as "Bypass" but "BYPASS" is considered to be the *same* linestyle name as "Bypass".

Within a project, each colour name must be unique amongst all the colour names in the project, each linestyle name must be unique amongst all the linestyle names in the project, and each textstyle name must be unique amongst all the textstyle names.

Keywords Blocks

There are many regularly used blocks of information in 12d XML that are identified and documented by keywords.

The keyword and its block consist of a starting **<keyword>**, followed by the information in the keyword block, and ending in **</keyword>**

That is

<keyword> information in the keyword block **</keyword>**

Archive Version and Export Comments

When writing a 12d Archive file (12da/12daz/12dxml/12xmlz), comments about the project, settings used to control the contents of the file are written at the header of the file.

Beginning with V14 C2k, one critical comment is

archive_version "aa.bb.cc.ddd"

The first 3 parts (aa.bb.cc) describe the version of the database written.

When any change is made to the underlying database of 12d is made, the (aa.bb.cc) is updated to reflect a change.

The final part (d) is used when there is a change in the archive definition, that is not related to a database change.

Typically, this number will almost always be 0.

The archive_version is related to what you see in the title area of the program. Using V14 C2k as the example:

aa the release version of 12d (being 14)

bb the major number of 12d (2 for C2)

cc the minor number of 12d (11 for K)

As a general rule

aa the release version of 12d (being 14, 15 etc)

bb the major number of 12d (being 0 for Alpha/Beta, 1 for C1, 2 for C2, etc)

cc the minor number of 12d (being 1 for A, 2 for B, 11 for K, etc)

12dxml example:

```
<!--
<archive_version>15.00.00.473</archive_version>
```

```
<decimal_places>8</decimal_places>
<null>-999</null>
<do_null>true</do_null>
<output_times>true</output_times>
<output_ids>false</output_ids>
<output_point_ids>true</output_point_ids>
<output_attribute_ids>true</output_attribute_ids>
<output_super_string_uids>true</output_super_string_uids>
<output_sa_parts>true</output_sa_parts>
<output_drawables>true</output_drawables>
<output_new_pipes>true</output_new_pipes>
<dereference>false</dereference>
<output_project_description>false</output_project_description>
<output_compact_clouds>true</output_compact_clouds>
<output_full_tin>true</output_full_tin>
<output_model_paths>false</output_model_paths>
<output_hex_floats>false</output_hex_floats>
<output_tin_hex_floats>true</output_tin_hex_floats>
<model_attributes_mode>0</model_attributes_mode>
  -->
```

Continue to 1.3 Regularly Used Keyword Blocks or return to 1 12d XML File Format.

1.3 Regularly Used Keyword Blocks

In the documentation of ***12d XML*** the term ***keyword* block** refers to a **<*keyword*>** followed by various information then a **</*keyword*>***.*

For the definition of some of the regularly used keyword blocks used in the 12d XML *see*:

Or return to 1 12d XML File Format.

1.3.1 Name

The format of the **name** keyword block is:

<name>*name_text*</name>

where ***name_text*** is a string of characters.

What characters can be in the name depends on where the name is used. See Names of Models, Tins and Super Tins and String Names.

Continue to 1.3.2 Colour or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.2 Colour

The format of the **colour** keyword block is:

<colour>*colour_name*</colour>

where ***colour_name*** is a string of characters that is to be the name of a colour or the colour number.

When reading a 12d XML file, there is a ***current colour,*** which has the default value of ***red***, and when a **colour** command is read, the ***current colour*** is set to *colour_name.*

When strings are read in a 12d XML file, they are given the *current colour*.

This can be overridden for a string by a *string colour command* inside the string command defining that string. For the definition of the string commands, see 1.6.3 String Header Block.

Continue to 1.3.3 Line Style or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.3 Line Style

The format of the **line style** keyword block is:

<style>*line_style_name*</style>

where *line_style_name* is the name of a line style. It is a string of characters.

When reading a 12d XML file, there is a ***current linestyle,*** which has the default value of ***1***, and when a **style** command is read, the ***current linestyle*** is set to *linestyle_name.*

When strings are read in a 12d XML file, they are given the *current linestyle*.

This can be overridden for a string by a *string style command* inside the string command defining that string. For the definition of the string command, see 1.6.3 String Header Block.

Continue to 1.3.4 Chainage or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.4 Chainage

The format of the **chainage** keyword block is:

<chainage> *chainage_real* </chainage>

where *chainage_real* is a real value.

Continue to 1.3.5 Weight or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.5 Weight

The format of the **weight** keyword block is:

<weight> *weight_real* </weight>

where *weight_real* is a real value.

Continue to 1.3.6 Interval or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.6 Interval

For all elements other than the super string, the format of the **interval** keyword block is:

<interval> *interval_real* </interval>

where *interval_real* is a real value.

For a super string, the format of the **interval** keyword block is:

<interval>

<chord_arc> chord_arc_real</chord_arc>

<distance> distance_real</chord_arc>

</interval>

where *chord_arc_real and distance_real* are real values.

Continue to 1.3.7 Time Created or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.7 Time Created

The format of the **time_created** keyword block is:

<time_created>*time_text*</time_created>

where ***time_text*** is a string of characters in the W3C time format.

*YYYY***-***MM***-***DD***T***hh***:***mm***:***ss***Z** *see* 1.3.8 W3C Time Format.

For example, 2015-09-28T06:42:45Z

Note that the time format for 12da is different from the one of 12dxml.

Continue to 1.3.8 W3C Time Format or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.8 W3C Time Format

The W3C time format is the string of characters:

*YYYY***-***MM***-***DD***T***hh***:***mm***:***ss***Z**

and

DD in the day of the month

MM two-digit month (01=January, etc.)

YYYY in the year

hh in the hour in the 24-hour clock

mm in the number of minutes

ss in the number of seconds

For example, 2015-09-28T06:42:45Z

Continue to 1.3.9 Time Updated or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.9 Time Updated

The format of the **time_updated** keyword block is:

<time_updated>*time_text*</time_updated>

where ***time_text*** is a string of characters in the W3C time format.

*YYYY***-***MM***-***DD***T***hh***:***mm***:***ss***Z** *see* 1.3.8 W3C Time Format.

For example, 2015-09-28T06:42:45Z

Note that the time format for 12da is different from the one of 12dxml.

Continue to 1.3.10 Breakline or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.10 Breakline

The format of the **breakline** keyword block is:

<breakline> *breakline_type_text* </breakline>

where ***breakline_type_text*** is text and can only have the values ***point*** or ***line***.

When reading a 12d XML file, there is a ***current breakline type,*** which has the default value of ***point***, and when a **breakline** command is read, the ***current breakline type*** is set to *breakline_type_text.*

When strings are read in a 12d XML file, they are given the *current breakline type*.

This can be overridden for a string by a *string breakline command* inside the string command defining that string. For the definition of the string command, see 1.6.3 String Header Block.

Continue to 1.3.11 Null or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.11 Null

The format of the **null** command is:

null *null_value*

When reading a 12d XML file, there is a ***current null value,*** which has the default value of ***-999***, and when a **null** command is read, the ***current null value*** is set to *null_value.*

When strings are read in a 12d XML file and the string has z-values equal to *null_value*, then the z-value is replaced by the ***12d Model*** null value.

This can be overridden for a string by a *null_value command* inside the string command defining that string. For the definition of the string command, see 1.6.3 String Header Block.

Continue to 1.3.12 Radius or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

Continue to 1.4 Attributes or return to 1 12d XML File Format.

1.3.12 Radius

The format of the **radius** keyword block is:

<radius> *radius_real* </radius>

where *radius_real* is a real value.

Continue to 1.3.13 data_2d or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.13 data_2d

For some strings, there is a constant z for the entire string, or even no z value at all. For such strings only the (x,y) coordinates are required for each vertex and no space is taken up by redundant z values. Vertex data with no z-values is written out in a ***data_2d*** block.

The definition of a ***data_2d*** block is:

<data_2d>

<p>*x_value_1 y_value_1***</p>**

<p>*x_value_2 y_value_2***</p>**

...

<p>*x_value_n y_value_n***</p>**

</data_2d>

where (x_value_i, y_value_i) are the 2D coordinates of the i'th vertex.

Continue to 1.3.14 data_3d or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.3.14 data_3d

For most string, the z value can vary for each vertex along the string and so the (x,y,z) values are required for each vertex. This vertex data is written out as a ***data_3d*** block.

The definition of a ***data_3d*** block is:

<data_3d>

<p>*x_value_1 y_value_1 z_value_1***</p>**

<p>*x_value_2 y_value_2 z_value_2***</p>**

...

<p>*x_value_n y_value_n z_value_n***</p>**

</data_3d>

where (x_value_i, y_value_i, z_value_i) are the 3D coordinates of the i'th vertex.

For example, for a string of 5 vertices

```
<data_3d>
  <p>42578.27649249 37366.79821468 null</p>
  <p>42523.36402317 37252.26649295 null</p>
  <p>42575.1386371 37043.59910954 null</p>
  <p>42826.16706828 37026.34090489 null</p>
  <p>42766.49603263 37412.54781911 61.53707464</p>
</data_3d>
```

1.3.15 radius_data and major_data

If there are only straight and arc segments for the string, then for either **data_2d** or **data_3d**, it is possible to add a radius and major/minor arc flag for each segment of the string using the ***radius_data*** and **major_data** blocks respectively.

The order of the entries in the *radius_data* and *major_data* blocks must match the order of the segments in the string (which is also the order in the data_2d or data_3d block).

So there is exactly one entry for each segment.

Note: If there are n vertices in the super string, then there are (n-1) segments for a open string (not closed) and n segments for a closed string.

For each segment there are five possibilities for an arc going between the vertices and these are specified by using **positive**, **zero** or **negative** values for the **radius**, and **1** or **0** for the **major flag**.

1. Straight segment - radius = 0. Major flag can be 1 or 0.
2. Positive radius and major flag 0

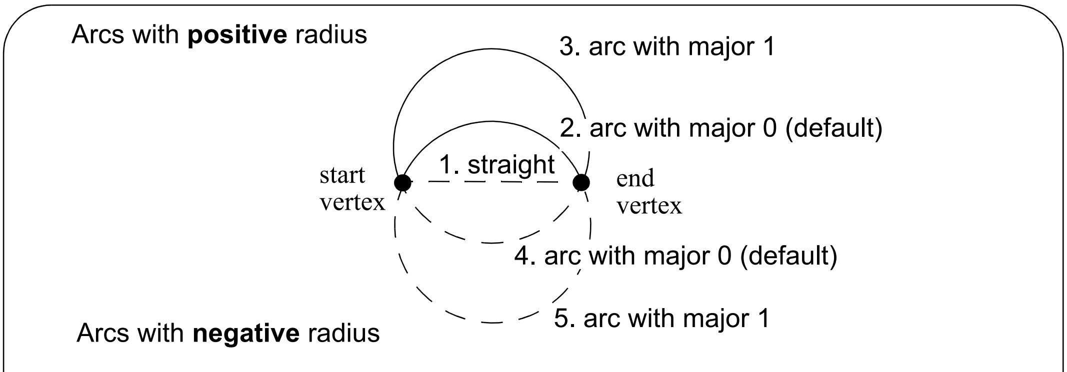
   The arc is above the straight line joining the two vertices but the arc is the smaller of the two possibilities (minor arc).
3. Positive radius and major flag1

   The arc is above the straight line joining the two vertices but the arc is the larger of the two possibilities (major arc).
4. Negative radius and major flag 0

   The arc is below the straight line joining the two vertices but the arc is the smaller of the two possibilities (minor arc).
5. Negative radius and major flag1

   The arc is below the straight line joining the two vertices but the arc is the larger of the two possibilities (major arc).

,



Arcs with same absolute radius but with major 1 or 0

The ***radius_data*** block is

<radius_data>

radius_for_segment_1

radius_for_segment_2

...

radius_for_segment_m

</radius_data>

where

radius_for_segment_i is the radius for the i'th segment and can be positive, zero or negative, and

m = n-1 for an open string or m = n for a closed string.

If the ***radius_block*** is missing then the radius is taken to be 0 and all the segments are straight lines.

The ***major_data*** block is

<major_data>

major_flag_for_segment_1

major_flag_for_segment_2

...

major_flag_for_segment_m

</major_data>

where

major_flag_for_segment_i for the i'th segment is 1 or t if the arc is a major arc, and 0 or f if it is a minor arc, and

m = n-1 for an open string or m = n for a closed string.

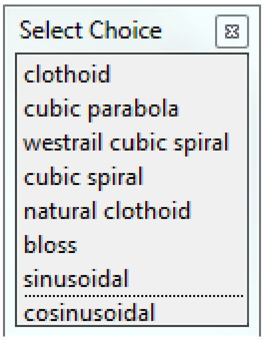
If the ***major_block*** is missing then the major flag is taken to be 0 and any segments with arcs are always the minor arcs.

For example, for a closed string of five vertices

```
<radius_data>
    100 -300 0 0 0
</radius_data>
<major_data>
    f f f f f
</major_data>
```

1.3.16 Available Transition Types

The transition that are available inside ***12d Model*** are called:



and these are defined as:

clothoid is the **clothoid spiral approximation** used by Australian road authorities and Queensland Rail

cubic parabola (or **NSW cubic parabola**) is a special transition curve used by NSW Railways. It is not a spiral.

westrail cubic spiral (or **westrail-cubic)** is a clothoid spiral approximation used by Westrail (WA railways).

cubic spiral (or **spiral**) is a low level spiral approximation. Mainly only used in surveying textbooks.

natural clothoid (or LandXML clothoid) is the full Euler clothoid spiral. This is not currently used by any Authority in Australia or New Zealand.

bloss is a Bloss curve. Not a spiral.

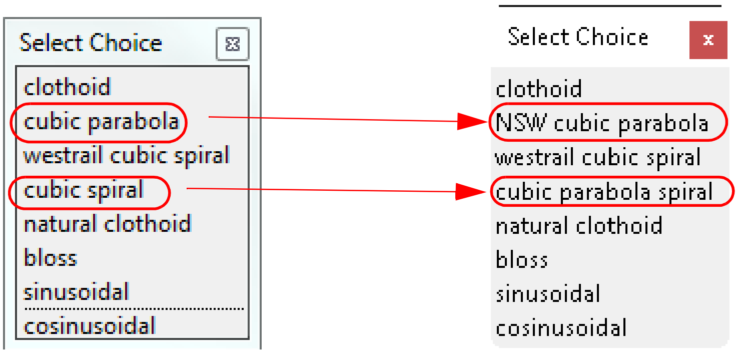
sinusoidal is a sinusoidal curve. Not a spiral.

cosinusoidal is a cosinusoidal curve. Not a spiral.

Although these are the names stored internally inside ***12d Model*** and match the standard ones used in Australia, unfortunately there is no universal definition of what names match which transitions.

So to make it clearer, especially because of the confusion about the term "cubic parabola", in some ***12d Model*** options the pop-up displays different names. This is especially true for options using a transition mapping file (trans_map file) to map the transition names used inside ***12d Model*** to those used in another software package.

In the alternate transition pop-up, "**cubic parabola"** is displayed as "**NSW cubic parabola**" and "**cubic spiral**" is displayed as "**cubic parabola spiral**" to help users realise that the word "cubic parabola" is confusing and could refer to the NSW Rail cubic transition and what is sometimes called the "cubic parabola" approximation to the clothoid spiral.



See 10.2.16 Transitions and Spirals and 10.2.16.4 Transition Mapping File.

1.4 Attributes

Many ***12d Model*** objects (models and elements such as individual strings and tins) can have an unlimited number of named **attributes** of type integer (numbers), real and text. Super strings and drainage strings can also have attributes on each vertex and segment.

The attributes for an object are given in an ***attributes block*** which consists of the keyword **attributes** followed by the definitions of the *individual attributes* enclosed in start and end curly braces { and }. That is, an ***attributes_block*** is

<attributes>

attribute_1

attribute_2

...

attribute_n

</attributes>

where the attribute definitions for the individual attributes *attribute_i* consists of

***<attribute_type*>**

<name> *attribute_name* ***</name>*** ***<value> attribute_value </value>***

***</attribute_type*>**

where

attribute_type is **integer**, **real** or **text**

attribute_name is the unique attribute name for the object.

and

attribute_value is the appropriate value of the integer, real or a text.

OR

where *attribute_type* is **group**

***<group*>**

<name> *group_name* ***</name>*** ***attributes_block***

***</group*>**

where

group_name is the unique name of the group at this level

and

attributes_block is another attributes_block.

Note that the definition of **<group>** includes an attribute_block which can contain another **<group>** so the definition is recursive.

Hence you can have a hierarchy or tree of attributes going down to any level.

Within an object, the attribute names are case sensitive and must be unique. That is, for attribute names, upper and lower case alphabet characters are considered to be different characters.

An example of and ***attribute block*** defining four attributes named "pole id", "street", "pole height" and "pole wires" is:

```
<attributes>
  <text> <name>pole id</name> <value>QMR-37</value> </text>
```

```
    <text> <name>street</name> <value>477 Boundary St</value> </text>
    <real> <name>pole wires</name> <value>3</value> </text>
  </attributes>
```

Continue to 1.5 Model or return to 1 12d XML File Format.

1.5 Model

Within a ***12d Model*** project, information is collected in units called **MODELS**. The items that can be stored in a model are called elements and elements include strings, tins, super tins, grid tins, trimeshes and plot frames.

Each model has a unique user-defined text name, ***model_name***, of up to two hundred alphanumeric characters and spaces.

The format for the **model** keyword block is:

<model>

<name>*model_name*</name>

attribute_block

time_created_block

time_updated_block

<children>

element_data_1

...

element_data_n

</children>

</model>

where:

model_name is a string of characters for the model name. For the characters allowed, see

attribute_block is option. For *attributes_block* see 1.4 Attributes.

time_created_block is optional. See 1.3.7 Time Created.

time_updated_block is optional. See 1.3.9 Time Updated.

element_data_i is an element stored in the model. See1.6 Elements Contained in Models.

The ***children*** block is ***optional*** and is mainly there so that in an xml editor, the *element_data_i* items can be collapsed into the ***children*** section.

An example of a model with no elements and no *children* block:

```
<model>
  <name>telegraph poles,/name>
  <attributes>
    <text> <name>pole id</name> <value>QMR-37</value> </text>
    <text> <name>street</name><value>477 Boundary St</value></text>
    <real> <name>pole wires</name> <value>3</value> </text>
  </attributes>
</model>
```

Continue to 1.6 Elements Contained in Models or return to 1 12d XML File Format.

1.6 Elements Contained in Models

See

1.6.1 Tin
1.6.2 Super Tin
1.6.5 Arc String
1.6.6 Circle String
1.6.7 Drainage String
1.6.8 Feature String
1.6.9 Plot Frame String
1.6.10 Super String
1.6.11 Super Alignment String
1.6.12 Text String

1.6.1 Tin

A *tin* (***t***riangulated ***i***rregular ***n***etworks) is an element that may, or may not, be in a model.

Each tin has text name, ***tin_name***, of up to two hundred alphanumeric characters and spaces and although the tin names are stored with upper or lower case alphabet characters, for comparisons of the tin names, the names are considered to be case insensitive.

Within a project, the name of a tin or a super tin must be unique amongst the combined list of tin names and super tin names.

There are two formats for a tin - one that lists **all the triangles**, including the nulled (invisible) triangles in the tin, and the other that only lists the **visible triangles** that make up the tin.

See

1.6.1.0.1 All Triangles in the Tin - Visible and Invisible
1.6.1.0.2 Visible Triangles Only

1.6.1.0.1 All Triangles in the Tin - Visible and Invisible

This format writes out all the triangles in the tin, including the invisible triangles and construction triangles.

This format take more disk space but cannot be misinterpreted because it includes all the points, triangles and all the neighbouring triangles for each edge of a triangle.

It is also the best method for writing out **large tins** as it is much faster to read in and create a tin.

The keyword for the full format for a **tin** element is **full_tin** and it is defined by:

<full_tin>

<name>*tin_name***</name>**

attribute_block

time_created_block

time_updated_block

colour_block

points_block

triangles_block

neighbours_block

nulling_block

colours_block

input_block

</full_tin>

where

tin_name

is a string of characters for the tin name and can't be blank. This must be unique in a project.

For the characters that can make up a tin_name, see Names of Models, Tins and Super Tins.

time_created_block

is the time the tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

colour_block

this colour number is the primary (base) colour for all the triangles in the tin. A triangle in the tin will have this colour unless it is overridden by a *colours_block*. For the syntax of colour_block, see 1.3.2 Colour.

attribute_block is optional: For the syntax of an *attributes_block* see 1.4 Attributes.

The attributes in this block and the attributes_block itself are optional.

The attributes **Style**, **Weed, Faces**, **Boundary_String**, **null_length**, **null_angle**, **null_combined_length** and **null_combined_angle** are special attributes that have extra information used by ***12d Model*** to create the tin. These special attributes should not be deleted.

The format of the special attributes inside the **<attributes> ... </attributes>** is:

```
<text> <name>Style</name> <value>style_name</value> </text>
<integer> <name>Weed</name> <value>weed_value</value> </integer>
<integer> <name>Faces</name> <value>faces_value</value> </integer>
<text> <name>Boundary_String</name><value>full_string_name</value></text>
<real> <name>null_length</name> <value>null_len_val</value> </real>
<real> <name>null_angle</name> <value>null_angle_rad</value> </real>
<real> <name>null_combined_length</name> <value>null_com_ln/value> <real>
<real> <name>null_combined_angle</name><value>null_com_rad</value></real>
```

where

style_name is the style for the tin

weed_value is 0 or 1

faces_value is 0 if the data is not from triangles, 1 if the data is from triangles

full_string_name is the name of a polygon for nulling outside. This is optional.

null_len_val is value for nulling by angle

null_angle_rad is in radians value for nulling by angle

null_com_ln is for nulling by combined angle and length

null_com_rad is in radians for nulling by combined angle and length

points_block

This gives the coordinates of the points that will be vertices of the triangles in the tin, including the **first four points that are construction points**. The construction points are on the four corners of a rectangle that totally surrounds the actual data.

The points are implicitly numbered by the order in the list (starting at point 1).

The Points Block is **MANDATORY**.

<points>

<p>x_value_1 y_value_1 z_value_1</p>

<p>x_value_2 y_value_2 z_value_2</p>

...

<p>x_value_m y_value_m z_value_m</p>

</points>

where (x_value_j, y_value_j, z_value_j) are the coordinates of the j'th point.

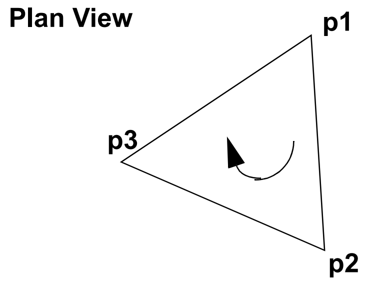
Points 1, 2,3 and 4 are not valid data points but are **construction points**. These are usually not displayed.

triangles_block

This gives the triangles that make up the tin.

Each triangle in the tin is given as a triplet of the point numbers in the Points block that are the triangle vertices. The order of the triangles is unimportant but the **order of the points in the triangle** is **important**.

The vertices of each triangle must be listed in a **clockwise** order when looking at the tin **from above**.



The Triangles Block is **MANDATORY**

<triangles>

<t>t1_pt_1 t1_pt_2 t1_pt_3</t>

<t>t2_pt_1 t2_pt_2 t2_pt_3</t>

...

<t>tn_pt_1 tn_pt_2 tn_pt_3</t>

</triangles>

where *tk_pt_1 tk_pt_2 tk_pt_3* are point numbers from the points_block of the three vertices of the k'th triangle.

The first edge of triangle k is from Point *tk_pt_1* to Point *tk_pt_2*.
The second edge of triangle k is from Point *tk_pt_2* to Point *tk_pt_3*.
The third edge of triangle k is from Point *tk_pt_3* to Point *tk_pt_1*.

Note: Construction Triangles

Any triangle that contains any of the first four points (construction points) is a **construction triangle** and is usually not displayed.

neighbours_block

For each triangle, this gives for each edge the number of the triangle that is the neighbour of that edge of the triangle.

The order of the entries in the neighbours block must match the order of the triangles in the Triangles Block. So there is exactly one entry for each triangle.

The Neighbours Block is MANDATORY

<neighbours>

<t>t1_e1_nb_tr t1_e2_nb_tr t1_e3_nb_tr</t>

<t>t2_e1_nb_tr t2_e2_nb_tr t2_e3_nb_tr</t>

...

<t>tn_e1_nb_tr tn_e2_nb_tr tn_e3_nb_tr</t>

</neighbours>

where *tk_e1_nb_tr tk_e2_nb_tr tk_e3_nb_tri* are the triangle numbers from the triangles_block of the neighbouring triangle for each edge of the k'th triangle.

For each triangle, the order of the neighbouring triangles must match the order that the edges are defined for the triangle in the triangles block.

Note: the neighbour value of **0** is used for the outside triangles that contain exactly two of the points 1, 2, 3 or 4 and so have edges that have no neighbouring triangle.

nulling_block

Triangles can be visible or nulled (invisible).

Any triangle including points 1, 2 3 or 4 are construction triangles and must be null.

All other triangles can be visible or null (invisible).

Whether a triangle is null or visible is individually given where:

1 means the triangle is **null**, and

2 means the triangle is **visible**.

The order of the entries in the nulling block must match the order of the triangles in the Triangles Block. So there is exactly one entry for each triangle

The Nulling Block is MANDATORY

<nulling>

v1 v2 **...** *v15 v16*

v17 v18 **...** *v31 v32*

...

vn-2 vn-1 vn

</nulling>

where **vk** is the nulling value of the k'th triangle in the *triangles_block*.

colours_block

Triangles can be given colours other than the base colour by including a Colours Block. The colour for each triangle in then individually given where -1 means use the base colour. The order of the entries in the colours block must match the order of the triangles in the Triangles Block. So there is exactly one entry for each triangle

If all the triangles are the base colour, then the *Colours Block* is omitted.

<colours>

c1 c2 **...** *c15 c16*

c17 c18 **...** *c31 c32*

...

cn-2 cn-1 cn

</colours>

where **ck** is the colour number of the k'th triangle in the *triangles_block*.

ck equals -1 when there is no special colour set and the triangle is drawn in the base colour.

input_block

The *input_block* gives more information about how the tin was created by ***12d Model***.

None of this information is needed when reading a tin into ***12d Model*** and the *input_ block* can be omitted.

<input>

<**preserve_strings**> *pres_str_text_logical* **</preserve_strings>**

<**remove_bubbles**> *rem_bub_text_logical* **</remove_bubbles>**

<**weed_tin**> *weed_tin_text_logical* **</weed_tin>**

<**triangle_data**> *triangle_data_text_logical* **</triangle_data>**

<**sort_tin>** *sort_tin_text_logical* **</sort_tin>**

<**cell_method>** *cell_method_text_logical* **</cell_method>**

<models>

model_name_1

model_name_2

...

model_name_p

</models>

</input>

where

pres_str_text_logical, rem_bub_text_logical, weed_tin_text_logical, triangle_data_text_logical, sort_tin_text_logical and *cell_method_text_logical* are text and can only have the values **true** or **false**.

<models> ... </models> is the list of models in the tin where

model_name_i is the name of the i'th model making up the tin.

1.6.1.0.2 Visible Triangles Only

The format to write out only the visible triangles in a tin is a simple format for most software packages to write. However because the null regions are not explicitly given, more processing time is required to read the tin back in and construct all the null regions.

The keyword denoting the format where just the visible triangles of a **tin** element are written out is **tin** and its definition is:

<tin>

<name>*tin_name***</name>**

attribute_block

time_created_block

time_updated_block

colour_block

points_block

triangles_block

colours_block

input_block

</tin>

where

tin_name

is a string of characters for the tin name and can't be blank. This must be unique in a project.

For the characters that can make up a tin_name, see Names of Models, Tins and Super Tins.

time_created_block

is the time the tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

colour_block

this colour number is the primary (base) colour for all the triangles in the tin. A triangle in the tin will have this colour unless it is overridden by a *colours_block*. For the syntax of colour_block, see 1.3.2 Colour.

attribute_block is optional: For the syntax of an *attributes_block* see 1.4 Attributes.

The attributes in this block and the attributes_block itself are optional.

The attributes **Style**, **Weed, Faces**, **Boundary_String**, **null_length**, **null_angle**, **null_combined_length** and **null_combined_angle** are special attributes that have extra information used by ***12d Model*** to create the tin. These special attributes should not be deleted.

The format of the special attributes inside the **<attributes> ... </attributes>** is:

```
<text> <name>Style</name> <value>style_name</value> </text>
<integer> <name>Weed</name> <value>weed_value</value> </integer>
<integer> <name>Faces</name> <value>faces_value</value> </integer>
<text> <name>Boundary_String</name><value>full_string_name</value></text>
<real> <name>null_length</name> <value>null_len_val</value> </real>
<real> <name>null_angle</name> <value>null_angle_rad</value> </real>
<real> <name>null_combined_length</name> <value>null_com_ln/value> <real>
<real> <name>null_combined_angle</name><value>null_com_rad</value></real>
```

where

style_name is the style for the tin

weed_value is 0 or 1

faces_value is 0 if the data is not from triangles, 1 if the data is from triangles

full_string_name is the name of a polygon for nulling outside. This is optional.

null_len_val is value for nulling by angle

null_angle_rad is in radians value for nulling by angle

null_com_ln is for nulling by combined angle and length

null_com_rad is in radians for nulling by combined angle and length

points_block

This gives the coordinates of the points that will be vertices of the triangles in the tin. The points are implicitly numbered by the order in the list (starting at point 1). The Points Block is MANDATORY.

<points>

<p>x_value_1 y_value_1 z_value_1</p>

<p>x_value_2 y_value_2 z_value_2</p>

...

<p>x_value_m y_value_m z_value_m</p>

</points>

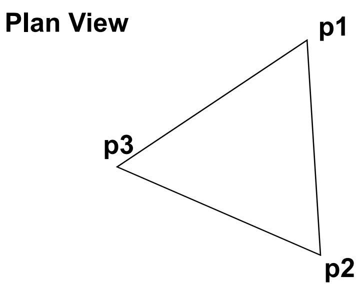
where (x_value_j, y_value_j, z_value_j) are the coordinates of the j'th point.

triangles_block

This gives the triangles that make up the tin.

Each triangle is given as a triplet of the point numbers in the Points block that are the triangle vertices. The order of the triangles is unimportant but the **order of the points in the triangle** is **important**.

The vertices of each triangle must be listed in a **clockwise** order when looking at the tin from above.



.

The Triangles Block is MANDATORY

<triangles>

<t>t1_pt_1 t1_pt_2 t1_pt_3</t>

<t>t2_pt_1 t2_pt_2 t2_pt_3</t>

...

<t>tn_pt_1 tn_pt_2 tn_pt_3</t>

</triangles>

where *tk_pt_1 tk_pt_2 tk_pt_3* are point numbers from the points_block of the three vertices of the k'th triangle.

colours_block

Triangles can be given colours other than the base colour by including a Colours Block. The colour for each triangle in then individually given where -1 means use the base colour. The order of the entries in the colours block must match the order of the triangles in the Triangles Block. So there is exactly one entry for each triangle

If all the triangles are the base colour, then the *Colours Block* is omitted.

<colours>

c1 c2 **...** *c15 c16*

c17 c18 **...** *c31 c32*

...

cn-2 cn-1 cn

</colours>

where **ck** is the colour number of the k'th triangle in the *triangles_block*.

ck equals -1 when there is no special colour set and the triangle is drawn in the base colour.

input_block

The *input_block* gives more information about how the tin was created by ***12d Model***.

None of this information is needed when reading a tin into ***12d Model*** and the *input_ block* can be omitted.

<input>

<**preserve_strings**> *pres_str_text_logical* **</preserve_strings>**

<**remove_bubbles**> *rem_bub_text_logical* **</remove_bubbles>**

<**weed_tin**> *weed_tin_text_logical* **</weed_tin>**

<**triangle_data**> *triangle_data_text_logical* **</triangle_data>**

<**sort_tin>** *sort_tin_text_logical* **</sort_tin>**

<**cell_method>** *cell_method_text_logical* **</cell_method>**

<models>

model_name_1

model_name_2

...

model_name_p

</models>

</input>

where

pres_str_text_logical, rem_bub_text_logical, weed_tin_text_logical, triangle_data_text_logical, sort_tin_text_logical and *cell_method_text_logical* are text and can only have the values **true** or **false**.

<models> ... </models> is the list of models in the tin where

model_name_i is the name of the i'th model making up the tin.

Continue to 1.6.2 Super Tin or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.6.2 Super Tin

A *Super Tins* consists of a number of **tins** (***t***riangulated ***i***rregular ***n***etworks).

Each super tin has text name, ***tin_name***, of up to two hundred alphanumeric characters and spaces and although the tin names are stored with upper or lower case alphabet characters, for comparisons of the tin names, the names are considered to be case insensitive.

Within a project, the name of a tin or a super tin must be **unique** amongst the **combined** list of tin names and super tin names.

The format for the **super_tin** element is:

<super_tin>

<name>tin_name</name>

attribute_block

time_created_block

time_updated_block

colour_block

exact_block

tins_block

</super_tin>

where

tin_name

is a string of characters for the super tin name and can't be blank. This must be unique in a project.

For the characters that can make up a tin_name, see Names of Models, Tins and Super Tins.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

colour_block

this colour number is the primary (base) colour for the super tin. For the syntax of colour_block, see 1.3.2 Colour.

attribute_block is optional: For the syntax of an *attributes_block* see 1.4 Attributes.

The attributes in this block and the attributes_block itself are optional.

The attribute **Style** is a special attribute that is used by ***12d Model*** to create the super tin. This special attribute should not be deleted.

The format of the **Style** attribute inside the <attributes> **...** </attributes> is:

<text> <name>**Style**</name> <value>*style_name*</value> </text>

where

style_name is the style for the super tin

exact_block

<exact> *exact_text_logical* **</exact>**

where

exact_text_logical is text and can only have the value **true** or **false**.

tins_block

This gives the tins that make up the super tin within the keyword block **tins**.

<**tins**>

tin_info_1

tin_info_2

...

tin_info_p

</**tins**>

where

there are p tins in the super tin and *tin_info_i* is information about the i'th tin. The information about a tin is contained in a **tin** block.

<**tin**>

<name> tin_name_i</name>

<active> active_text_logical</active>

<mode> mode_text_logical</mode>

</**tin**>

where

tin_name_i is the name of the i'th tin making up the super tin.

active_text_logical and *mode_text_logical* are text and can only have the value **true** or **false**.

For example

```
<super_tin>
  <name>super tin</name>
  <colour>green</colour>
  <attributes>
    <text> <name>Style</name> <value>1</value> </text>
  </attributes>
  <time_created>28-Apr-2015 06:42:45</time_created>
  <time_updated>28-Apr-2015 06:42:45</time_updated>
  <exact>true</exact>
  <tins>
    <tin>
      <name>DESIGN ALL</name>
      <active>true</active>
      <mode>replace</mode>
    </tin>
    <tin>
      <name>HILL</name>
      <active>true</active>
      <mode>replace</mode>
    </tin>
  </tins>
</super_tin>
```

Note that the tins that make up the super tin must exist in the ***12d Model project*** for the super tin to be fully defined.

Continue to 1.6.3 String Header Block or return to 1.3 Regularly Used Keyword Blocks or 1 12d XML File Format.

1.6.3 String Header Block

Strings are special types of elements that reside in a model.

Strings have common header information and this will be documented in this one spot as a ***string_header_block***.

The format for the string_header_block is:

string_name_block

chainage_block

colour_block

style_block

weight_block

interval_block

time_created_block

time_updated_block

attribute_block

where

string_name_block

The format of the **string_name_block** is:

<name> *string_name_text* </name>

where

string_name_text is a string of allowable characters that is the name of the string.

For the characters that can make up a string_name, see String Names.

Any leading and trailing spaces will be removed in the string name.

string_name can be blank.

An example of a string name is:

```
<name> design 100.0 </name>
```

chainage_block

is the start chainage of the string. This is optional. For the syntax see 1.3.4 Chainage.

colour_block

the colour name is the primary colour for the string. For the syntax of colour_block, see 1.3.2 Colour.

style_block

is the line style of the string. This is optional. For the syntax of style_block see 1.3.3 Line Style.

weight_block

is the weight (thickness) of the string. This is optional. For the syntax of weight_block see 1.3.5 Weight.

interval_block

the chainage interval to temporarily introduce extra vertices into the string when the string is in a triangulation to form a tin. For the syntax of interval_block, see 1.3.6 Interval.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax of time_created_block see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax of time_updated_block see 1.3.9 Time Updated.

attribute_block

The string attributes are in this block. For the syntax of an *attributes_block* see 1.4 Attributes

The attributes_block is optional.

For example

string_header_block

```
<string_arc>
  <name>arc</name>
  <chainage>0</chainage>
  <breakline>line</breakline>
  <colour>yellow</colour>
  <style>1</style>
  <weight>2</weight>
  <time_created>28-Apr-2015 07:46:57</time_created>
  <time_updated>28-Apr-2015 07:46:57</time_updated>
  <interval>10</interval>
  <centre>1067.40263766 530.14953857 0</centre>
  <radius>226.6814323</radius>
  <chord_arc>0.1</chord_arc>
  <start>867.42825529 423.40349345 0</start>
  <end>1118.02452861 751.10631241 0</end>
</string_arc>
```

Continue to 1.6.4 Text Information or return to 1.6 Elements Contained in Models or 1 12d XML File Format.

1.6.4 Text Information

See

1.6.4.1 Vertex Annotation Information
1.6.4.2 Segment Annotation Information

1.6.4.1 Vertex Annotation Information

The *vertex_annotation_information* is

<worldsize> *world_size_real* **</worldsize>**

<textstyle> *textstyle_name* **</textstyle>**

<angle> *angle_dec_deg_real* **</angle>**

<x_factor> *x_factor_real* **</x_factor>**

<slant> *slant_dec_deg_real* **</slant>**

<offset> *offset_real* **</offset>**

<raise> *raise_real* **</raise>**

<text_colour> text_colour_name **</text_colour>**

<justify> *text_justification_text* **</justify>**

where

world_size_real is the size of the text in world units.

textstyle_name is the name of the textstyle for the text.

angle_dec_deg_real is the angle of the text. The value is in decimal degrees and is measured in a counter clockwise direction from the positive x-axis.

x_factor_real is the factor to apply to the width of the text.

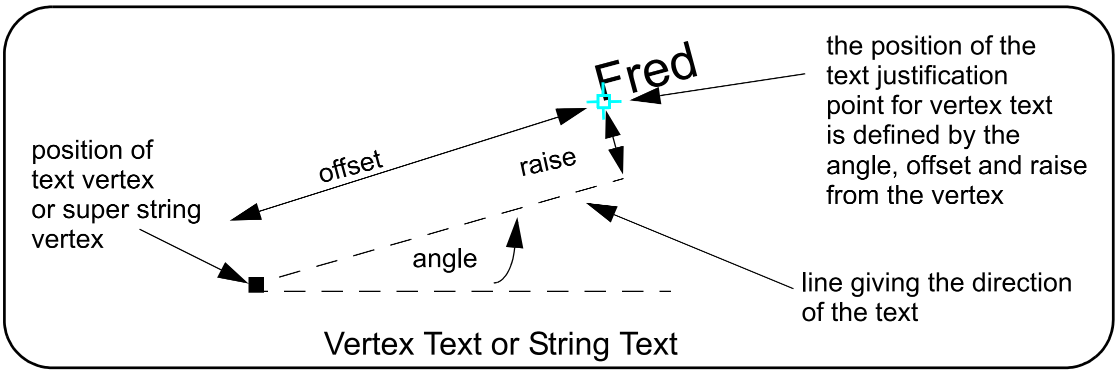
slant_dec_deg_real is the angle the text is slanted from the vertical. The value is in decimal degrees and is measured in a clockwise direction from the positive y-axis.

offset_real is distance to offset the text from the text vertex.

raise_real is the perpendicular distance the text is off the direction line of the text.

text_colour_name is the colour of the text. This should be the same as the colour in the *string_header_block*.For the syntax of colour_block, see 1.3.2 Colour.

text_justification_text is the text giving the justification point of the text.



1.6.4.2 Segment Annotation Information

The *segment_annotation_information* is

<worldsize> *world_size_real* **</worldsize>**

<textstyle> *textstyle_name* **</textstyle>**

<angle> *angle_dec_deg_real* **</angle>**

<x_factor> *x_factor_real* **</x_factor>**

<slant> *slant_dec_deg_real* **</slant>**

<offset> *offset_real* **</offset>**

<raise> *raise_real* **</raise>**

<text_colour> text_colour_name **</text_colour>**

<justify> *text_justification_text* **</justify>**

where

world_size_real is the size of the text in world units.

textstyle_name is the name of the textstyle for the text.

angle_dec_deg_real is the angle of the text. The value is in decimal degrees and is measured in a counter clockwise direction from the segment.

x_factor_real is the factor to apply to the width of the text.

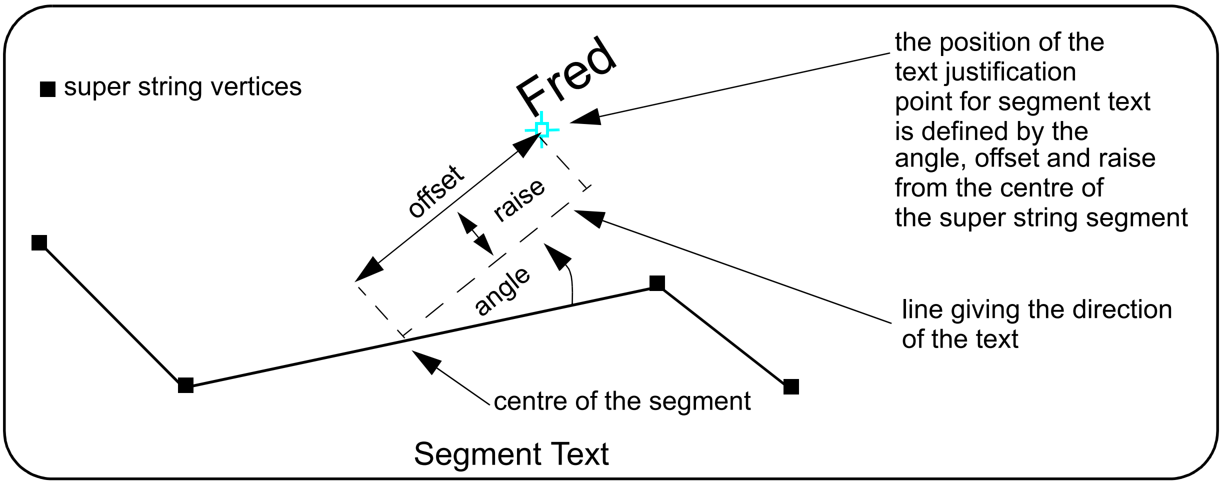
slant_dec_deg_real is the angle the text is slanted from the vertical. The value is in decimal degrees and is measured in a clockwise direction from the positive y-axis.

offset_real is distance to offset the text from the centre of the segment.

raise_real is the perpendicular distance the text is off the direction line of the text.

text_colour_name is the colour of the text. This should be the same as the colour in the *string_header_block*.For the syntax of colour_block, see 1.3.2 Colour.

text_justification_text is the text giving the justification point of the text.



1.6.5 Arc String

The format for the **string_arc** element is:

<string_arc>

string_header_block

centre_block

radius_block

chord_arc_block

start_block

end_block

</string_arc>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block.

centre_block

The format of the **centre_block** is:

<centre> *x_centre_real y_centre_real z_centre_real* </centre>

where

(*x_centre_real,y_centre_real,z_centre_real*) is the centre of the arc.

radius_block

the radius of the arc. For the syntax of radius_block, see 1.3.10 Breakline.

A positive radius means that the arc goes from the start point in a clockwise direction (goes to the right) and a negative radius means that the arc goes is in a counter clockwise direction (goes to the left).

chord_arc_block

The format of the **chord_arc_block** is:

<chord_arc>*chord_arc_real* </chord_arc>

where

chord_arc_real is a real number and is the chord to arc tolerance to use to temporarily insert vertices into the arc when the arc is included in a triangulation to form a tin.

start_block

The format of the **start_block** is:

<start> *x_start_real y_start_real z_start_real* </start>

where

(*x_start_real,y_start_real,z_start_real*) is the start coordinate of the arc.

end_block

The format of the **end_block** is:

<end> *x_end_real y_end_real z_end_real* </end>

where

(*x_end_real,y_end_real,z_end_real*) is the end coordinate of the arc.

For example

```
<string_arc>
  <name>arc</name>
  <chainage>0</chainage>
  <breakline>line</breakline>
  <colour>yellow</colour>
  <style>1</style>
  <weight>2</weight>
  <time_created>28-Apr-2015 07:46:57</time_created>
  <time_updated>28-Apr-2015 07:46:57</time_updated>
  <interval>10</interval>
  <centre>1067.40263766 530.14953857 0</centre>
  <radius>226.6814323</radius>
  <chord_arc>0.1</chord_arc>
  <start>867.42825529 423.40349345 0</start>
  <end>1118.02452861 751.10631241 0</end>
</string_arc>
```

Continue to 1.6.6 Circle String or return to 1.6.3 String Header Block or 1 12d XML File Format.

1.6.6 Circle String

The format for the **string_circle** element is:

<string_circle>

string_header_block

centre_block

radius_block

chord_arc_block

</string_arc>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block.

centre_block

The format of the **centre_block** is:

<centre> *x_centre_real* *y_centre_real* *z_centre_real* </centre>

where

(*x_centre_real,y_centre_real,z_centre_real*) is the centre of the circle.

radius_block

the radius of the circle. For the syntax of radius_block, see 1.3.10 Breakline.

A positive radius means that the circle goes in a clockwise direction (goes to the right) and a negative radius means that the circle goes is in a counter clockwise direction (goes to the left).

chord_arc_block

The format of the **chord_arc_block** is:

<chord_arc>*chord_arc_real* </chord_arc>

where

chord_arc_real is a real number and is the chord to arc tolerance to use to temporarily insert vertices into the circle when the circle is included in a triangulation to form a tin.

For example

```
<string_circle>
  <name>circle</name>
  <chainage>0</chainage>
  <breakline>line</breakline>
  <colour>yellow</colour>
  <style>1</style>
  <weight>5</weight>
  <interval>10</interval>
  <time_created>28-Apr-2015 07:45:53</time_created>
  <time_updated>28-Apr-2015 07:46:23</time_updated>
  <centre>409.93551 548.76354 null</centre>
  <radius>100</radius>
  <chord_arc>0.1</chord_arc>
</string_circle>string circle
```

Continue to 1.6.7 Drainage String or return to 1.6.3 String Header Block or 1 12d XML File Format.

1.6.7 Drainage String

The full 12dXML definition of the drainage string is:

<string_drainage>

string_header_block

outfall_block

flow_direction_block

use_pit_con_points_block

drainage_sewer_block

data_3d_block

radius_data_block

major_data_block

pit_records

pipe_records

</string_drainage>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block.

There are also some special attributes in the string attributes in the String Header Block that provide extra information for the drainage string.

outfall_block

<outfall> *outfall_real* **</outfall>**

where *outfall_real* is the z-value of the outfall (the low end of the string.

flow_direction_block

<flow_direction> *flow_direction_flag* **</flow_direction>**

where *flow_direction_flag* is **1** if the flow is the same as the string direction, or **0** if the flow is opposite to the string direction.

use_pit_con_points_block

<user_pit_con_points> *use_pit_connection_points_logical_text* **</user_pit_con_points>**

where *use_pit_connection_points_logical_text* is **true** if pit connection points are used, or **false** if pit connection points are not being used and hence the pipes go to the centre of he pits.

drainage_sewer_block

<drainage_sewer> *drainage_sewer_choice_text* **</drainage_sewer>**

where *drainage_sewer_choice_text* is **drainage** (storm water) if it is for drainage and **sewer** if it is for sewer (foul water).

data_3d_block, radius_data_block and major_data_block

the drainage string has an underlying string that is used to define locations of the pits and the geometry for the pipes. The underlying string can have straight and arc segments.

The vertex data for the underlying string is given in a **data_3d** block, and if there any arcs, then these are specified in **radius_data** and **major_data** blocks. See 1.3.14 data_3d and 1.3.15 radius_data and major_data.

pit_records

In plan the pits sit on the underlying string and there is one pit record for each pit. The pits do not have to be on a vertex of the underlying string.

There is one **pit** or **pit_v2** block for each pit in the string and they are in the order that they occur along the string.

The information for each **pit** is:

<pit>

<name> *pit_name_text* **</name>**

<type> *pit_type_text* **</type>**

<chainage> *pit_chainage_real* **</chainage>**

<ip> *pit_ip_text* **</ip>**

<ratio> *pit_ratio_real* **</ratio>**

<x> *pit_x_real* **</x>**

<y> *pit_y_real* **</y>**

<z> *pit_z_real* **</z>**

<road_chainage> *pit_road_chainage_real* **</road_chainage>**

<diameter> *pit_diameter_real* **</diameter>**

<width> *pit_width_real* **</width>**

<sump_level> *pit_sump_level_real* **</sump_level>**

<floating_sump> *pit_floating_sump_flag* **</floating_sump>**

<thickness> *pit_thickness_real* **</thickness>**

<thickness_bottom> *pit_thickness_bottom_real* **</thickness_bottom>**

<thickness_back> *pit_thickness_back_real* **</thickness_back>**

<thickness_left> *pit_thickness_left_real* **</thickness_left>**

<thickness_right> *pit_thickness_right_real* **</thickness_right>**

<con_point_mode> *pit_con_points_mode_text* **</con_point_mode>**

<floating> *pit_floating_logical_text* **</floating>**

<hgl> *pit_hgl_real* **</hgl>**

pit_attributes_block

</pit>

The information for each **pit_v2** is:

<pit_v2>

<name> *pit_name_text* **</name>**

<type> *pit_type_text* **</type>**

<chainage> *pit_chainage_real* **</chainage>**

<ip> *pit_ip_text* **</ip>**

<ratio> *pit_ratio_real* **</ratio>**

<x> *pit_x_real* **</x>**

<y> *pit_y_real* **</y>**

<z> *pit_z_real* **</z>**

<road_chainage> *pit_road_chainage_real* **</road_chainage>**

<diameter> *pit_diameter_real* **</diameter>**

<width> *pit_width_real* **</width>**

<sump_level> *pit_sump_level_real* **</sump_level>**

<floating_sump> *pit_floating_sump_flag* **</floating_sump>**

<thickness> *pit_thickness_real* **</thickness>**

<thickness_bottom> *pit_thickness_bottom_real* **</thickness_bottom>**

<thickness_back> *pit_thickness_back_real* **</thickness_back>**

<thickness_left> *pit_thickness_left_real* **</thickness_left>**

<thickness_right> *pit_thickness_right_real* **</thickness_right>**

<con_point_mode> *pit_con_points_mode_text* **</con_point_mode>**

<floating> *pit_floating_logical_text* **</floating>**

<hgl> *pit_hgl_real* **</hgl>**

pit_attributes_block

<base_extended>none/width/length**</base_extended>**

<base_angle_mode>choices_text**</base_angle_mode>**

<base_angle>angle_real**</base_angle>**

<base_height>value_real**</base_height>**

<riser_enabled>logical_text**</riser_enabled>**

<riser_diameter>value_real**</riser_diameter>**

<riser_width>value_real**</riser_width>**

<riser_extended>none/width/length**</riser_extended>**

<riser_offset_x>value_real**</riser_offset_x>**

<riser_offset_y>value_real**</riser_offset_y>**

<riser_colour>colour**</riser_colour>**

<base_thickness_top>value_real**</base_thickness_top>**

<riser_thickness_front>value_real**</riser_thickness_front>**

<riser_thickness_back>value_real**</riser_thickness_back>**

<riser_thickness_left>value_real**</riser_thickness_left>**

<riser_thickness_right>value_real**</riser_thickness_right>**

</pit_v2>

where

pipe_records

In plan the pipes sit on the underlying string and the plan geometry is based on the underlying string. Each pipe goes between two adjacent pits.

There is one **pipe** block for each pipe in the string and they are in the order that they occur along the string.

<pipe>

<name> *pipe_name_text* **</name>**

<type> *pipe_type_text* **</type>**

<colour> *pipe_colour_text* **</colour>**

<diameter> *pipe_diameter_real* **</diameter>**

<nominal_diameter> *pipe_nominal_diameter_real* **</nominal_diameter>**

<width> *pipe_width_real* **</width>**

<top_width> *pipe_top_width_real* **</top_width>**

<thickness> *pipe_thickness_real* **</thickness>**

<thickness_bottom> *pipe_thickness_bottom_real* **</thickness_bottom>**

<thickness_back> *pipe_thickness_back_real* **</thickness_back>**

<thickness_left> *pit_thickness_left_real* **</thickness_left>**

<thickness_right> *pipe_thickness_right_real* **</thickness_right>**

<separation> *pipe_separation_real* **</separation>**

<number_of_pipes> *pipe_number_of_pipes_integer* **</number_of_pipes>**

<us_level> *pipe_us_level_real* **</us_level>**

<ds_level> *pipe_ds_level_real* **</ds_level>**

<us_hgl> *pipe_us_hgl_real* **</us_hgl>**

<ds_hgl> *pipe_ds_hgl_real* **</ds_hgl>**

<flow_velocity> *pipe_flow_velocity_real* **</flow_velocity>**

<flow_volume> *pipe_flow_volume_real* **</flow_volume>**

pipe_attributes_block

</pipe>

```
string drainage {
  chainage    start_chainage
  model    model_name    name  string_name
  colour    colour_name    style  style_name
  breakline  point or line
  attributes {
    text   Tin    finished_surface_tin
    text   NSTin    natural_surface_tin
    integer    "_floating"    1|0              // 1 for floating, 0 not floating
  }
  outfall      outfall_value                    // z-value at the outfall
  flow_direction      0|1                       // 0 drainage line is defined from downstream
                                                // to upstream

  data {                                        // key word - geometry of the drainage string
    x-value      y-value      z-value      radius   bulge
       "            "            "
       "            "            "
  }
  pit {                              // pit/manhole - one pit record for each pit/manhole
                                     // in the order along the string
      name             text                     // pit name
      type             text                     // pit type
      road_name        text                     // road name
      road_chainage        chainage             // road chainage
      diameter             value                // pit diameter
      floating         yes|no                   // is pit floating or not
```

```
    chainage      pit_chainage          // internal use only
   ip             value                 // internal use only
    ratio         value                 // internal use only
    x             x-value               // x-value of top of pit
    y             y-value               // y-value of top of pit
    z             z-value               // z-value of top of pit
}
pipe {                        // one pipe record for each pipe connecting pits/manholes
                              // in the order they occur along the string
    name          text                  // pipe name
    type          text                  // pipe type
    diameter      value                 // pit diameter
    us_level      value                 //
    ds_level      value                 //
    us_hgl        value                 //
    ds_hgl        value                 //
    flow_velocity      value            //
    flow_volume        value            //
}
property_control {
  name            text                  // lot name
  colour          colour_name
  grade           value                 // grade of pipe in units of "1v in"
  cover           value                 // cover of the of pipe
  diameter        value                 // diameter of the of pipe
  boundary        value                 // boundary trap value
  chainage        chainage              // internal use only
  ip              value                 // internal use only
  ratio           value                 // internal use only
  x               x-value               // x value of where pipe connects to sewer
  y               y-value               // y value of where pipe connects to sewer
  z               z-value               // internal use only

  data {                      // key word - geometry of the property control
  x-value     y-value     z-value     radius   bulge
     "           "           "
     "           "           "
}
house_connection {   // warning - house connections may change in future versions
  name            text                  // house connection name
  hcb             integer               // user given integer
  colour          colour_name
  grade           value                 // grade of connection in units of "1v in"
  depth           value
  diameter        value
  side            left or right
  length          value
  type            text                  // connection type
  material        text                  // material type
  bush            text                  // bush type
  level           value
  adopted_level   value
  chainage        chainage              // internal use only
  ip              value                 // internal use only
  ratio           value                 // internal use only
  x               x-value               // x value of where pipe connects to sewer
  y               y-value               // y value of where pipe connects to sewer
  z               z-value               // internal use only
```

```
    }

  }                    // end of drainage-sewer data
```

Continue to 1.6.8 Feature String or return to 1.6.3 String Header Block or 1 12d XML File Format.

1.6.8 Feature String

The full 12dXML definition of the drainage string is:

<string_feature>

string_header_block

<radius> *feature_radius_real* **</radius>**

<centre> *x_centre_real y_centre_real z_centre_real* **</centre>**

</string_feature>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block.

There are also some special attributes in the string attributes in the String Header Block that provide extra information for the drainage string.

feature_radius_real is the radius of the feature string.

(xcentre_real, y_centre_real, z_centre_real) is the centre of the feature string.

For example

```
<string_feature>
  <name>Line 1</name>
  <chainage>0</chainage>
  <breakline>line</breakline>
  <colour>cyan</colour>
  <style>1</style>
  <time_created>2015-05-19T08:06:01Z</time_created>
  <time_updated>2015-05-19T08:06:01Z</time_updated>
  <centre>42200.06055 37384.05873 null</centre>
  <radius>20</radius>
</string_feature>
```

Continue to 1.6.9 Plot Frame String or return to 1.6.3 String Header Block or 1 12d XML File Format.

1.6.9 Plot Frame String

The format for the **string_plot_frame** element is:

<string_plot_frame>

info_block

time_created_block

time_updated_block

sheet_details_block

title_block_block

origin_block

scale_block

rotation_block

plotter_details_block

</string_plot_frame>

where

info_block

The format of the **info_block** is:

<info>

<name> *plot_frame_name_text*</name>

<colour> *plot_frame_name_colour_text*</colour>

<plot_file> *plot_file_name_text*</plot_file>

</info>

where

plot_frame_name_text is a string of allowable characters that is the name of the plot file string. For the characters that can make up a string_name, see String Names.

plot_frame_colour_text is the colour of the plot frame. For the syntax of colour_block, see 1.3.2 Colour.

plot_file_name_text is the name of file that the plot frame will plot to.

time_created_block

is the time the plot frame was originally created, This is optional. For the syntax of the time_created_block see 1.3.7 Time Created.

time_updated_block

is the last time the plot frame was last modified, This is optional. For the syntax of the time_updated_block see 1.3.9 Time Updated.

sheet_details_block

The format of the **sheet_details_block** is:

<sheet_details>

<sheet_code> *sheet_code_text*</sheet_code>

<width> *sheet_width_real*</width>

<height> *sheet_height_real*</height>

<left_margin> *sheet_left_margin_real*</left_margin>

<right_margin> *sheet_right_margin_real*</right_margin>

<top_margin> *sheet_top_margin_real*</top_margin>
<bottom_margin> *sheet_bottom_margin_real*</bottom_margin>
<border> *sheet_border_text_logical*</border>
<viewport> *sheet_viewport_text_logical*</viewport>

</sheet_details>

where

sheet_code_text is the name of the sheet. This can be blank.

sheet_width_real, sheet_height_real, sheet_left_margin_real, sheet_right_margin_real, sheet_top_margin_real, sheet_bottom_margin_real are all real values and give the size and margins for he sheet that the plot frame will plot. The units for all of them is millimetres.

plot_frame_border_text_logical and *plot_frame_viewport_text_logical* are text and can only have the value **true** or **false**.

origin_block

The format of the **origin_block** is:

<origin> *x_real y_real z_real*</origin>

where

(*x_real,y_real,z_real*) is the coordinates of the origin of the plot frame.

scale_block

The format of the **scale_block** is:

<scale> *scale_real*</scale>

where

scale_real is the 1: scale for the plots created by the plot frame.

rotation_block

The format of the **rotation_block** is:

<rotation> *rotation_dec_deg_real*</rotation>

where

rotation_dec_deg_real is rotation of the plot frame. The value is in decimal degrees and is measured in a counter clockwise direction from the positive x-axis.

plotter_details_block

The format of the **plotter_details_block** is:

<plotter_details>

<title_1>*title_1_text*</title_1>
<title_2>*title_2_text*</title_2>
<use_title_file> *title_file_text_logical*</border>
<title_file> *title_file_name_text*</title_file>
<text_size> *title_text_size_real*_mm</text_size>
<textstyle> *title_text_style*</textstyle>

</plotter_details>

where

title_1_text and *title_2_text* are two lines of text for the title block. They can be blank.

use_title_file_text_logical is text and can only have the value **true** or **false**.

title_file is the path name of the file to use as a title block file. This can be blank.

title_text_size_real_mm is the size of the text in the title block. The units are millimetres.

For example

```
<string_plot_frame>
  <info>
    <name>Plot frame</name>
    <colour>green</colour>
    <plot_file>plot</plot_file>
  </info>
  <sheet_details>
    <sheet_code>A0</sheet_code>
    <width>1189</width>
    <height>841</height>
    <left_margin>5</left_margin>
    <right_margin>10</right_margin>
    <bottom_margin>5</bottom_margin>
    <top_margin>10</top_margin>
    <border>true</border>
    <viewport>true</viewport>
  </sheet_details>
  <title_block>
    <title_1>Title 1</title_1>
    <title_2>Title 2</title_2>
    <use_title_file>true</use_title_file>
    <title_file>A0 title.tbf</title_file>
    <text_size>5</text_size>
    <textstyle>1</textstyle>
  </title_block>
  <origin>695.2353 1464.6248</origin>
  <scale>100</scale>
  <rotation>45</rotation>
  <plotter_details>
    <id>9</id>
    <type>model</type>
    <mode>""</mode>
    <names>""</names>
  </plotter_details>
  <time_created>29-Apr-2015 01:11:52</time_created>
  <time_updated>29-Apr-2015 01:11:52</time_updated>
</string_plot_frame>
```

Continue to 1.6.10 Super String or return to 1.6.3 String Header Block or 1 12d XML File Format.

1.6.10 Super String

Because the super string is so versatile, its **12d XML** format looks complicated but it is very logical and actually quite simple.

In its most primitive form, the super string is simply a set of (x,y) values as in a 2d string, or (x,y,z) values as in a 3d string.

Additional blocks of information can extend the definition of the super string and ***only need to be included if they exist***. For example, segment arcs or transitions, vertex ids, vertex and segment text, round pipe diameters or box pipes widths and heights and tinability.

Some of the properties of the super string can be constant for the entire string or can vary for each vertex and/or segment. For example, there can be one colour for the entire string or individual colours for each segment.

For user attributes, the super string not only has the standard user attributes defined for the entire string (string attributes), but also can have user attributes for each vertex (vertex attributes) and for each segment (segment attributes).

Being closed or not is another property of the super string and if the super string is closed then the super string knows there is an additional segment going from the last vertex back to the first vertex. This means that no duplication of the first and last vertex is needed.

Thus if a super string has *n* vertices, then an open super string has *n-1* segments joining the vertices and a closed super string has *n* segments since there is an additional segment from the last to the first vertex.

With the additional data for vertices and/or segments in the super string, the data is in vertex or segment order.

So for a string with *n* vertices, there must be *n* bits of vertex data.

For segments, if the string is open then there only needs to be *n-1* bits of segment data but for closed strings, there must be *n* bits of data.

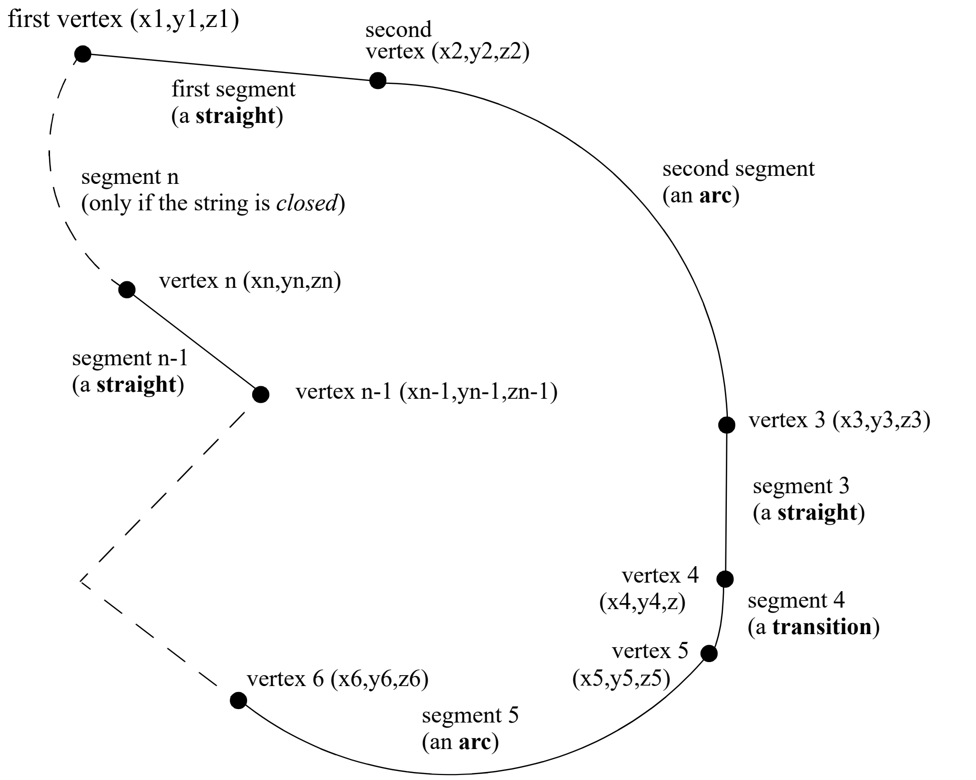
For an open string, *n* bits of segment data can be specified and the *nth* bit will be read in and stored. If the string is then closed, the *nth* bit of data will be used for the extra segment.

Important Note

For a super string, the arcs, transitions and offset transitions are that shape in plan.

Hence an arc with a z-value at each end is actually a **helix** and **NOT** part of a three dimensional circle.

Transitions and offset transitions are helixes with a constantly changing radius.



Vertices and Segments Forming the Super String

The 12dXML definition of the super string is:

<string_super>

string_header_block

closed_block

interval_block

blocks_of_info_1

blocks_of_info_2

...

blocks_of_info_n

</string_super>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block.

closed_block

<closed> closed_text_logical **</closed>**

where *closed_text_logical is* **true** if the super string is closed and **false** if the super string is open.

interval_block

The interval_block for a super string has a *distance* (a chainage interval) and a *chord_to_arc*_real

where

the ***distance*** to temporarily introduce extra vertices into the string at the given chainage distance when the string is in a triangulation to form a tin.

chord_arc_real is a real number and is the chord to arc tolerance to use on any arcs in the super string to temporarily insert vertices into the arc when the arc is included in a triangulation to form a tin.

For the syntax of interval_block, see 1.3.6 Interval.

blocks_of_info

The blocks of info can be broken up into four types.

(a) blocks defining the position of the vertices in z, y and z

Each vertex must have at least an (x,y) value but there may be one z-value for the entire string and (x,y) at each vertex (data_2d), **or** an (x,y,z) for each vertex (data_3d).

See 1.6.10.1 Defining the Coordinates of the Vertices

(b) blocks defining the geometry of the segments

Segments can be straights, arcs, transitions or offset transitions.

radius_data and *major_data* **or** *geometry_data.*

See 1.6.10.2 Geometry of the Horizontal Segments

(c) extra information for the vertices and/or segments such as colour, attributes, vertex ids, symbols tinability *etc*.

The definition for the blocks of each type now follows.

1.6.10.1 Defining the Coordinates of the Vertices

1.6.10.2 Geometry of the Horizontal Segments

1.6.10.3 Colour

1.6.10.4 String, Vertex and Segment Attributes

1.6.10.5 Vertex Id's (Point Id's)

1.6.10.6 Symbols at Vertices

1.6.10.7 Tinability

1.6.10.8 Round or Box (Culvert) Pipes

1.6.10.9 Vertex Text and Vertex Annotation

1.6.10.10 Segment Text and Segment Annotation

1.6.10.1 Defining the Coordinates of the Vertices

See

1.6.10.1.1 One Z or No Z for the String

1.6.10.1.2 Varying Z Values along the String

1.6.10.1.1 One Z or No Z for the String

If there is a non-null constant z value for the entire string then the z value is given by a **z** block:

<z> *z_value* **</z>**

where *z_value* is the constant z value for the entire string.

And when there is a constant z, or no z, for the string, then only the (x,y) coordinates are required for each vertex and these are given in a ***data_2d*** block. See 1.3.13 data_2d

1.6.10.1.2 Varying Z Values along the String

If the z value can vary for different vertices along the string then the (x,y,z) values must be given for each vertex and the data is then written out as a ***data_3d*** block. See 1.3.14 data_3d.

1.6.10.2 Geometry of the Horizontal Segments

There are three different methods of specifying the geometry of the horizontal segments and which one is used depends on the types of segments in the string.

The different methods are:

1. If the **segments** are **straight lines** only then that is the default and no further information is required.
2. If the **segments** are **only straight lines and arcs**, then the ***radius_data*** and **major_data** blocks are used to define a ***radius*** and ***bulge_flag*** data for each segment of the super string. See 1.6.10.2.1 Only Straights and Arcs for Segments.
3. If **any of the segments** are **transitions or offset transitions** then ***geometry_data*** must be used for each segment. ***geometry_data*** can represent a straight, arc, transition or offset transition. See 1.6.10.2.2 Straights, Arcs and Transitions for Segments.

1.6.10.2.1 Only Straights and Arcs for Segments

If there are only straight and arc segments for the string, then for either **data_2d** or **data_3d**, it is possible to add a radius and major/minor arc flag for each segment of the super string using the ***radius_data*** and **major_data** blocks respectively. See 1.3.15 radius_data and major_data.

1.6.10.2.2 Straights, Arcs and Transitions for Segments

When some of the segments are transitions or offset transitions, then the ***geometry_data*** block **must** be used the give the geometry for each segments.

Either a **data_2d** or **data_3d** block defines the coordinates for the vertices and the ***geometry_data*** block defines for each segment whether the segment is a straight, an arc or a transition or offset transition.

The definition of the ***geometry_data*** block is

<geometry_data>

info_for_segment_1_block

info_for_segment_2_block

...

info_for_segment_m_block

</geometry_data>

where

info_for_segment_i_block is the information defining the i'th segment as either a straight, an arc or an offset transition (offset transition or transition), and

m = n-1 for an open string or m = n for a closed string.

For the definition of *info_for_segment_i_block* see:

1.6.10.2.2.1 Straight

1.6.10.2.2.2 Arc

1.6.10.2.2.3 Transition and Offset Transitions

1.6.10.2.2.1 Straight

No parameters are needed for defining a straight segment. The *straight* block is simply:

<straight> </straight>

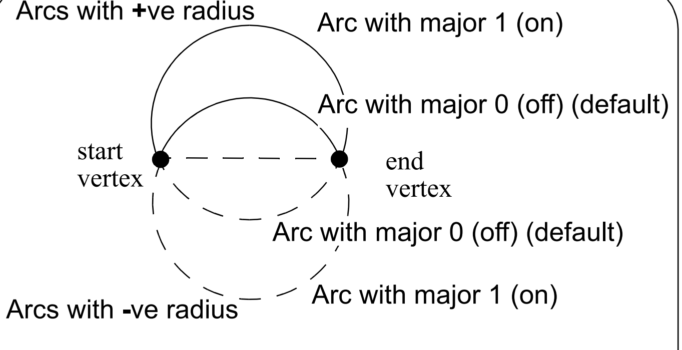
or simply

<straight/>

1.6.10.2.2.2 Arc

There are four possibilities for an arc of a given radius placed between two vertices.

We use *positive* and *negative* radius, and a flag *major* which can be set to 1 (on) or off (0) to differentiate between the four possibilities.



Arcs with same radius but with major on or off

The *arc* block is:

<arc>

<radius> *radius_for_segment*</radius>

<major> *major_flag_for_segment*</major>

</arc>

where

radius_for_segment is the radius for the segment and

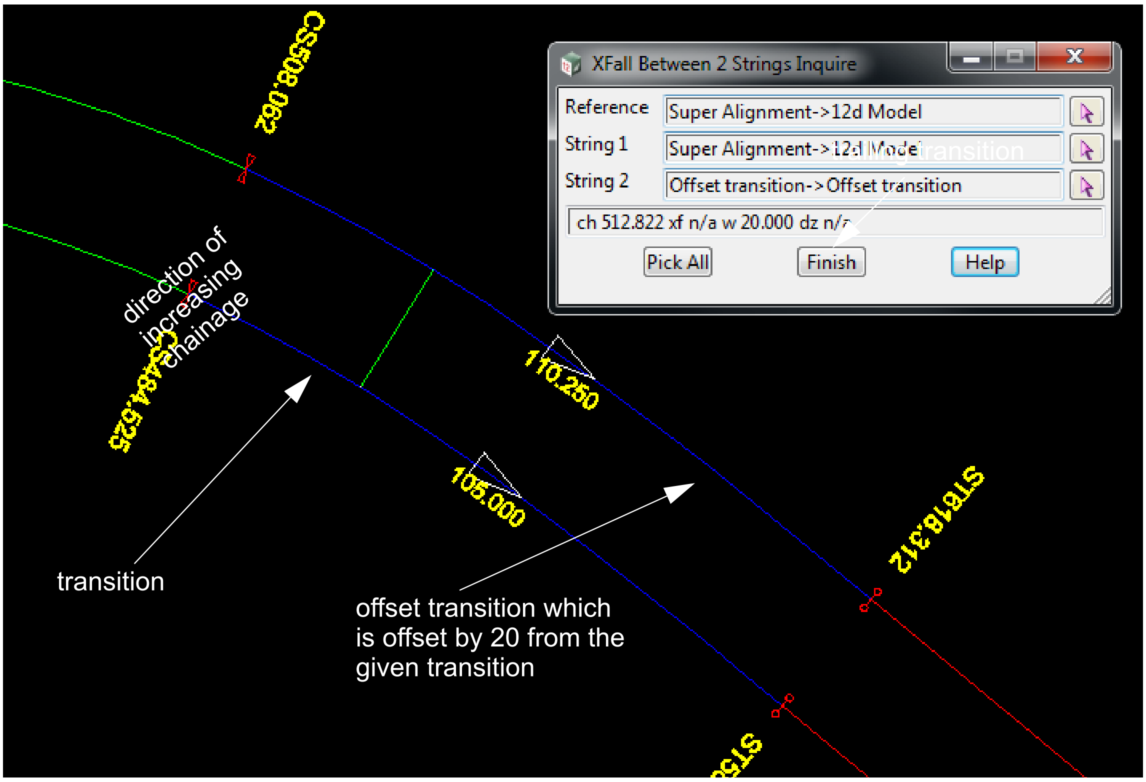
major_flag_for_segment is 1 if the arc is a major arc and 0 if it is a minor arc.

1.6.10.2.2.3 Transition and Offset Transitions

When a straight line is perpendicularly offset by a constant distance, you get another parallel straight line. Similarly when an arc is perpendicularly offset by a constant distance, you get a parallel arc with a radius of the existing radius plus the offset distance.

However when a transition curve is perpendicularly offset by a constant distance, you do not get another transition curve of the same type. Instead you get what we will call an **offset transition**.

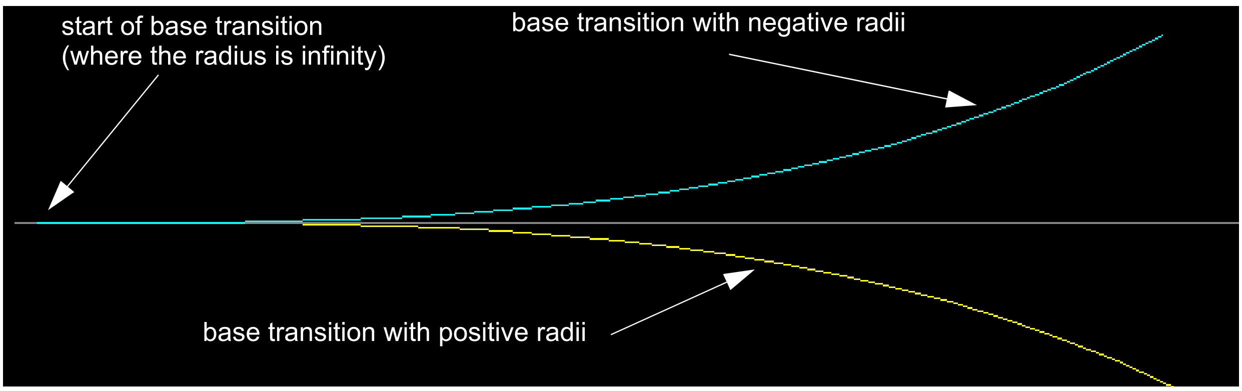
An ***offset transition*** is the curve that is a fixed perpendicular offset (***offset_real)*** from a transition where the transition is a Euler spiral (or a certain approximation to it) or some other specially defined transition curve. An offset of zero is the standard **transition**.



To fully describe an offset transition, we will first define a base transition.

A **base transition** is a **full transition curve** which has a **start point** where the absolute radius of the curve is infinity and then has a monotonically decreasing absolute radius as you continue along the base transition. The **base transition** is fully determined by specifying other parameters such as the radius at a given length along the base transition.

As you go along a **base transition** in decreasing absolute radius, the curve curls to the **right** if the **radius** is **positive**, and curls to the **left** if the **radius** is **negative**.



A general ***base transition*** is defined by giving

(a) its starting point **(xorigin, yorigin)** where the radius is infinity

(b) the angle of the tangential line at the start point ***angle_decimal_degrees_real,***

(c) the radius ***radius_real*** that occurs at a given curve length ***length_real*** along the base transition.The radius ***radius_real*** can be ***positive*** or ***negative***.

An ***offset transition*** is a fixed offset (***offset_real***) from a base transition and goes from a start point that is specified by giving the length on the base transition where the start point drops perpendicularly onto the base transition (***start_length_real***) and to the end point that is specified by the length on the base transition where the end point drops perpendicularly onto the base transition (***end_length_real***). The **offset_real** can be **positive** or **negative**.

The direction of the offset transition (increasing chainage) does not have to be the same as the direction of the base transition. That is, the absolute radius at the ***start_length_real*** may be greater than the absolute radius at the ***end_length_real***.

Hence if you are travelling along the offset transition in a forward direction (increasing chainage) then the offset transition is said to be a

(a) **leading offset transition** if the absolute radius of the points dropped onto the base transition decreases as you go along the offset transition.

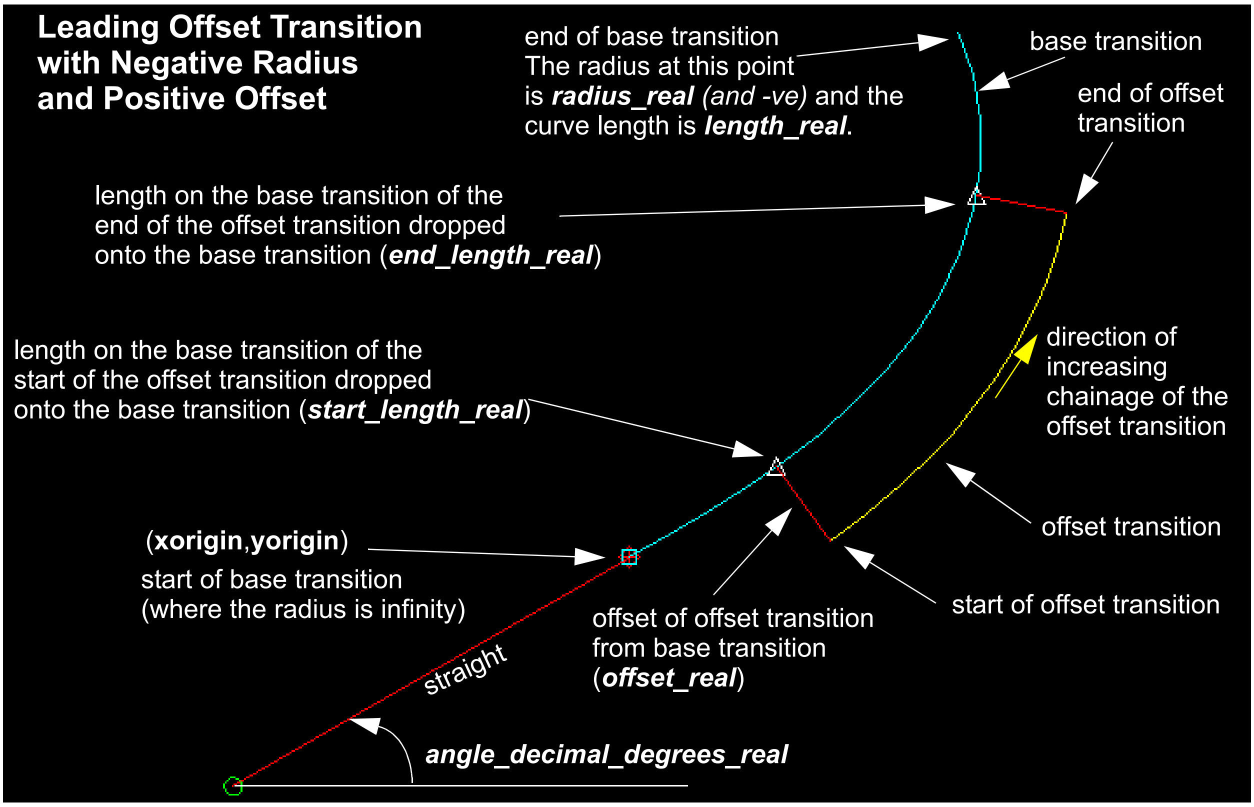
For a **leading offset transition**, if the **end radius** is **positive** then the curve curls to the **right**, and for a **negative end radius**, the curve curls to the **left**.

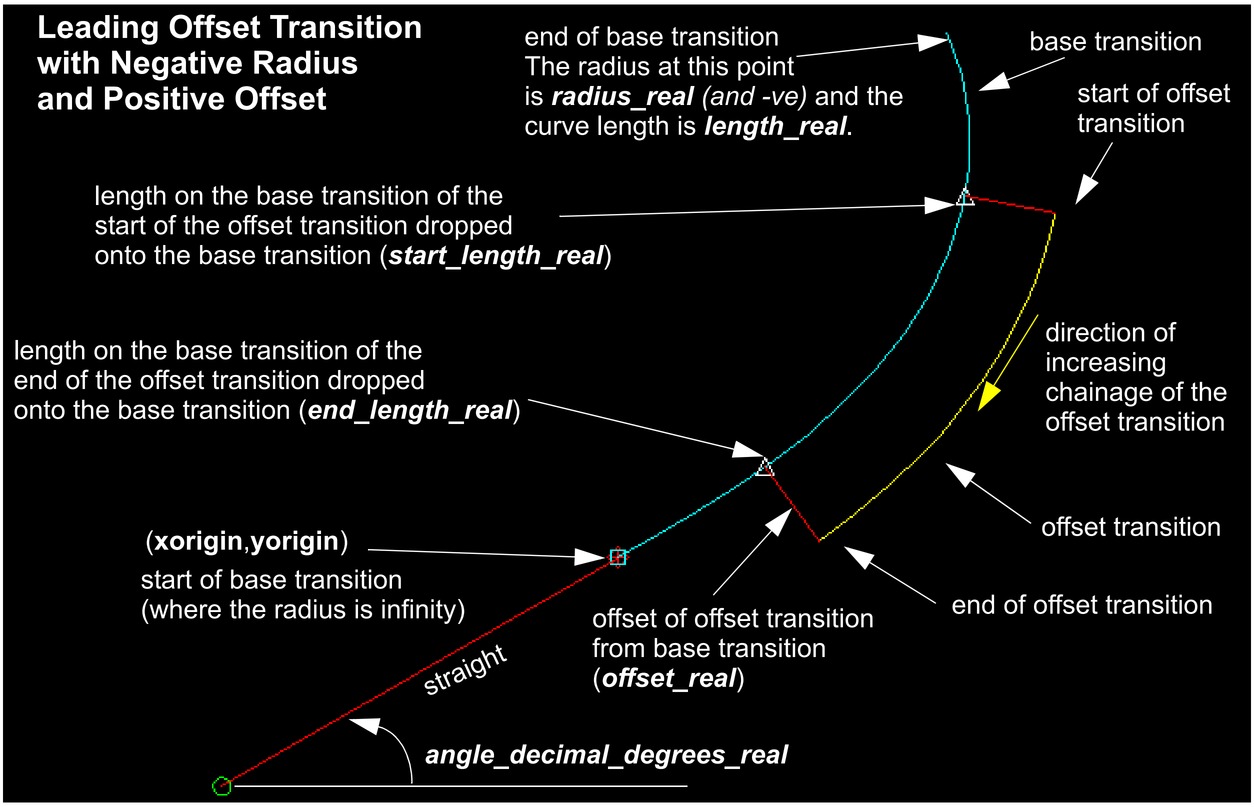
And

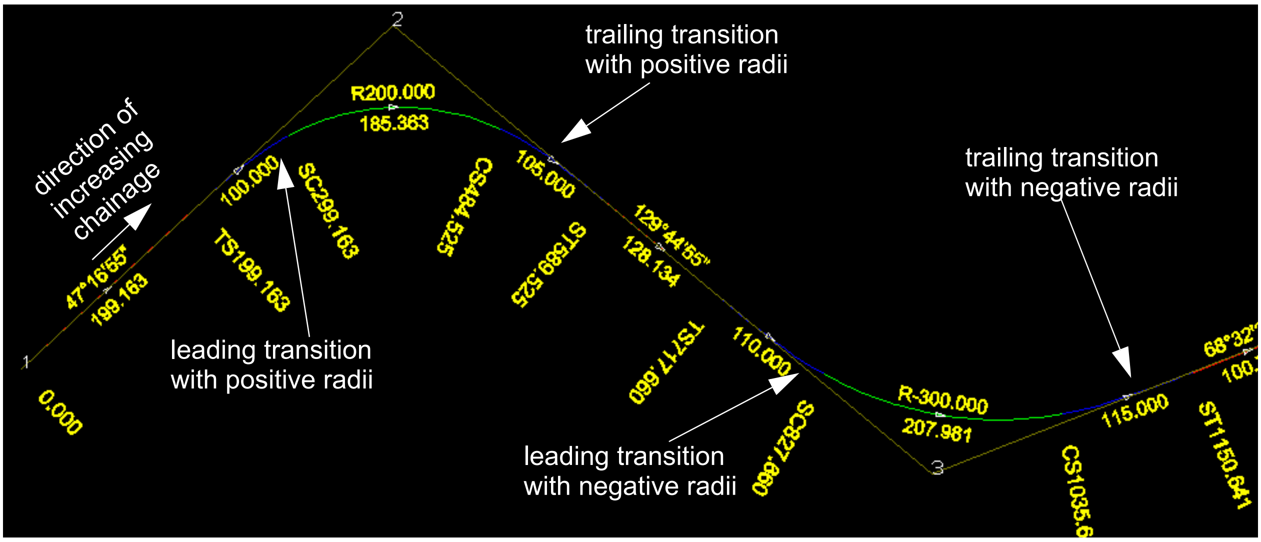
(b) a **trailing offset transition** if the absolute radius of the points dropped onto the base transition increases as you go along the offset transition.

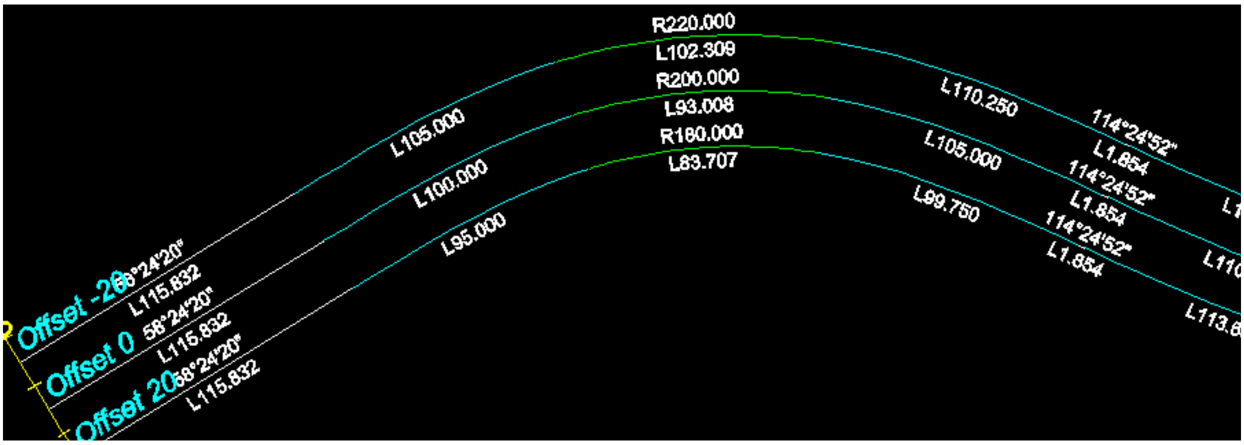
For a **trailing offset transition**, if the **end radius** is **positive** then the curve curls to the **right**, and for a **negative end radius**, the curve curls to the **left**.

The **offset transition** can be a **partial transition**. That is, none of the points dropped onto the base transition have an infinite radius.




Leading Offset Transition
with Negative Radius
and Positive Offset
end of base transition
The radius at this point
is radius_real (and -ve) and the
curve length is length_real.
base transition
start of offset
transition
length on the base transition of the
start of the offset transition dropped
onto the base transition (start_length_real)
direction of
increasing
chainage of the
offset transition
length on the base transition of the
end of the offset transition dropped
onto the base transition (end_length_real)
offset transition
(xorigin,yorigin)
start of base transition
(where the radius is infinity)
end of offset transition
offset of offset transition
from base transition
(offset_real)
straight
angle_decimal_degrees_real


trailing transition
with positive radii
R200.000
185.363
direction of
increasing
chainage
100.000
SC299.163
CS484.525
105.000
trailing transition
with negative radii
ST589.525
129°44'55"
128.134
47°16'55"
199.163
TS199.163
TS717.660
110.000
leading transition
with positive radii
R-300.000
207.981
115.000
0.000
leading transition
with negative radii
SC827.660
CS1035.6
ST1150.641



The ***curve*** block covers both spiral and non-spiral transitions with a zero or non zero offsets.

The *curve* block is:

<curve>

<curve_type> *curve_type_text*</curve_type>

<leading> *leading_logical_text*</leading>

<xorigin> *xorigin_real*</xorigin>

<yorigin> *yorigin_real*</xorigin>

<radius> *radius_real*</radius>

<length> *length_real*</length>

<start> *start_length_real*</start>

<end> *end_length_real*</end>

<angle> *angle_decimal_degrees_real*</angle>

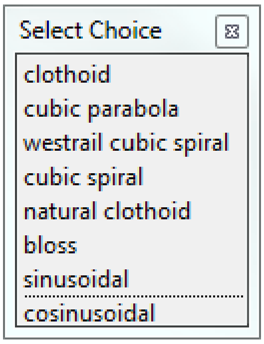
<offset> *offset_real*</offset>

<mvalue> *mvalue_real*</mvalue>

</curve

where

curve_type_text is the type of base transition.



For more information on the choices, see 1.3.16 Available Transition Types.

leading_logical_text is ***true*** if it is a **leading** base **transition** or *false* if is a trailing base transition.

*(**xorigin**, **yorigin**)* is the origin of the base transition. That is, where the radius is infinity.

radius_real is the radius at the end of the base transition. The radius is positive if the curve goes to the right when travelling **in decreasing absolute radius**. This direction may be the opposite to the string direction

length_real is the curve length to the end of the base transition and the radius is **radius_real**.

start_length_real is the curve length on the base transition where the start of the offset transition drops perpendicularly onto the base transition.

end_length_real is the curve length on the base transition where the end of the offset transition drops perpendicularly onto the base transition.

angle_decimal_degrees_real is the angle of the tangent of the base transition at the origin of the base transition. It is measured in decimal degrees in a counter clockwise direction from the positive x-axis.

offset_real is the perpendicular offset distance of the offset transition from the base transition. For a leading transition, a positive value offsets from the base transition to the right and a negative value offsets it to the left, as you travel in a forward direction.

mvalue_real - *if the transition is a cubic parabola then mvalue_real* is the mvalue for the cubic parabola. Otherwise, *mvalue_real* is zero.

For example, for a string with four segments

```
<geometry_data>
  <arc>
    <radius>-222.77841769</radius>
    <major>0</major>
  </arc>
  <curve>
    <type>clothoid</type>
    <leading>false</leading>
    <xorigin>114.78632204</xorigin>
    <yorigin>22.22840069</yorigin>
    <radius>222.77841769</radius>
    <length>194.18990415</length>
    <start>50.95749554</start>
    <end>194.18990415</end>
     <angle>174.01773651</angle>
    <offset>0</offset>
    <mvalue>0</mvalue>
  </curve>
  <arc>
    <radius>-848.96871636</radius>
    <major>0</major>
  </arc>
  <straight/>
</geometry_data>
```

1.6.10.3 Colour

There can be one colour for the entire super string which is given by the **<colour>** keyword block in the ***string_header_block***, or the colour varies for each segment of the super string and is then specified in a ***<colour_data>*** block.

The order of the entries in the **<colour_data>** block must match the order of the segments in the super string. So there is exactly one entry for each segment.

If all the segment are the string colour, then simply omit the **<colour_data>** block.

For a super string with **n** vertices

<colour_data>

colour_text_for_segment_1

colour_text_for_segment_2

...

colour_text_segment_m

</colour_data>

where

colour_text_segment_i is the colour name or colour number for the i'th segment OR is **no_colour** when no special colour has been set for the segment and the string colour is then used for that segment. If the name includes spaces then it must be enclosed in **"**, and

m = n-1 for an open string or m = n for a closed string.

For example for a string with four segments

```
<colour_data>
    "off yellow" magenta no_colour no_colour
</colour_data>    <leading>false</leading>
```

1.6.10.4 String, Vertex and Segment Attributes

The super string can have attributes for the entire string (string attributes) but can also have attributes for each vertex (vertex attributes) and attributes for each segment (segment attributes).

See

1.6.10.4.1 String Attributes
1.6.10.4.2 Vertex Attributes
1.6.10.4.3 Segment Attributes

1.6.10.4.1 String Attributes

There can be attributes for the entire string. They are part of the String Header Block and are described in 1.6.3 String Header Block.

For example

```
<string_super>
  <name>Line 1</name>
  <chainage>0</chainage>
  <breakline>line</breakline>
  <colour>cyan</colour>
  <style>1</style>
  <attributes>
    <text>
    <name>Street</name>
    <value>Weemala Road</value>
    </text>
  </attributes>
  <time_created>2015-05-11T09:08:06Z</time_created>
  <time_updated>2015-05-11T11:59:29Z</time_updated>
           ...
```

1.6.10.4.2 Vertex Attributes

Each vertex can have one or more user defined attributes.

For a super string with **n** vertices

<vertex_attribute_data>

vertex_attributes_for_vertex_1_block

vertex_attributes_for_vertex_2_block

...

vertex_attributes_for_vertex_n_block

</vertex_attribute_data>

where

vertex_attributes_for_vertex_j_block is the *attribute_block* for vertex j. The attribute_block is defined in 1.4 Attributes.

For example, for a string with four vertices

```
<vertex_attribute_data>
  <attributes>
    <real>
      <name>Flow</name>
      <value>27.4</value>
    </real>
  </attributes>
  <attributes/>
  <attributes/>
  <attributes/>
</vertex_attribute_data>
```

1.6.10.4.3 Segment Attributes

Each segment can have one or more user defined attributes.

For a super string with **n** vertices

<segment_attribute_data>

segment_attributes_for_segment_1_block

segment_attributes_for_segment_2_block

...

segment_attributes_for_segment_m_block

</segment_attribute_data>

where

segment_attributes_for_segment_j_block is an *attribute_block* for segment j. The attribute_block is defined in 1.4 Attributes, and

m = n-1 for an open string or m = n for a closed string.

For example, for an open string with four vertices

```
<segment_attribute_data>
  <attributes>
    <real>
      <name>Material</name>
      <value>clay</value>
    </real>
  </attributes>
  <attributes/>
  <attributes/>
  <attributes/>
</segment_attribute_data>
```

1.6.10.5 Vertex Id's (Point Id's)

Each vertex can have a vertex id (point id).

This is not the number position of the vertex in the string but is a separate id which is usually different for every vertex in every string.

The ***vertex id*** can be alphanumeric and include spaces.

The definition is:

For a super string with **n** vertices

<point_data>

point_id_text_for_vertex_1

point_id_text_for_vertex_2

...

point_id_text_for_vertex_n

</point_data>

where

point_id_text_for_vertex_i is the point id of the i'th vertex.

m = n-1 for an open string or m = n for a closed string.

point_id_text_for_vertex_i is the actual text enclosed in **"**, even if the text does not include spaces. If the point id has not been set for a vertex, the value should be included as "".

For example "Point 1" or "Point2" or "".

If the ***point_data*** block does not exist then there are no Vertex Ids.

For example, for a string with 4 vertices

```
<point_data>
  "Point 1" "Point2" "" ""
</point_data>
```

1.6.10.6 Symbols at Vertices

There can be no symbols at all, or the same symbol for every vertex in the using the ***symbol_value*** block or the symbol varies for each vertex of the super string using the ***symbol_data*** block.

If a symbol does not have a colour, or there is no colour in the symbol definition, then it uses the string colour.

The definitions are:

<symbol_value>

symbol_properties_block

</symbol_value>

where

symbol_properties_block is the description for the symbol to be used at every vertex of the super string, and

OR

For a super string with **n** vertices

<symbol_data>

symbol_properties_for_vertex_1_block

symbol_properties_for_vertex_2_block

...

symbol_properties_for_vertex_n_block

</symbol_data>

where

symbol_properties_for_vertex_i_block is the description for the symbol at vertex i.

The format of ***symbol_properties_block*** and ***symbol_properties_for_vertex_i_block*** is:

<properties>

<style> *symbol_name_text* </style>

<colour> *symbol_colour_name_text* </colour>

<size> symbol_size_*real* </size>

<rotation> *angle_dec_deg_real*</rotation>

<offset_x> *symbol_offset_x_real* </offset_y>

<offset_y> *symbol_offset_y_real* </offset_y>

</properties>

where

symbol_name_text is the name of the symbol.

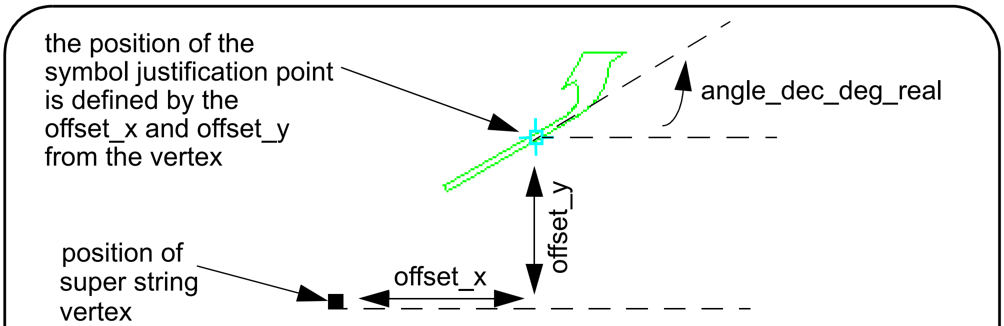
symbol_colour_name is the colour of the symbol is there is no colours in the symbol definition. If the ***colour*** block is missing and there is no colours in the symbol definition then the string colour is used. For the syntax of the ***colour*** block, see 1.3.2 Colour.

symbol_size_real is the size of the symbol in the units of the symbol.

angle_dec_deg_real is the angle of the symbol. The value is in decimal degrees and is measured in a counter clockwise direction from the positive x-axis.

offset_x_real is x distance to offset the symbol from the super string vertex.

offset_y_real is the y distance to offset the symbol from the super string.



Symbol at a Vertex

1.6.10.7 Tinability

For a *super string*, the concept of breakline has been extended to a property called **tinable** which can be set independently for each vertex and each segment of the super string.

If a vertex is tinable, then the vertex is used in triangulations. If the vertex is not tinable, then the vertex is ignored when triangulating.

If a segment is tinable, then the segment is used as a side of a triangle during triangulation. This may not be possible if there are *crossing* tinable segments.

Vertex tinability is given by the ***vertex_tinable_data*** block where for a string of n vertices,

<vertex_tinable_data>

tinable_flag_for_vertex_1

tinable_flag_for_vertex_2

...

tinable_flag_for_vertex_n

</vertex_tinable_data>

where

tinable_flag_for_vertex_i for the i'th vertex is 1 or t if the vertex is tinable, or 0 or f if the vertex is not tinable.

Segment tinability is given by the ***segment_tinable_data*** block where

<*segment_tinable_data*>

tinable_flag_for_segment_1

tinable_flag_for_segment_2

...

tinable_flag_for_segment_m

</*segment_tinable_data*>

where

tinable_flag_for_segment_i for the i'th segment is 1 or t if the segment is tinable, or 0 or f if the segment is not tinable, and

m = n-1 for an open string or m = n for a closed string.

For example, for a open string with four vertices

```
<vertex_tinable_data>
  t t f t
</vertex_tinable_data>
<segment_tinable_data>
  f t t
</segment_tinable_data>
```

Note that even if a segment is set to tinable, it can only be used in a triangulation if both its end vertices are also tinable.

1.6.10.8 Round or Box (Culvert) Pipes

All segments of a super string can be: round pipes; box pipes (culvert); or no pipe. This is the property of the whole string, that is, some segments can't be round while others be box. In another word, one super string cannot have both **pipe diameters** and **culvert dimensions**.

There is also one justification that applies to all (segments) pipes of a super string.

See

1.6.10.8.1 Pipe Diameters
1.6.10.8.2 Culvert Dimensions
1.6.10.8.3 Justification for Round or Culvert Pipes

1.6.10.8.1 Pipe Diameters

There can be one pipe diameter value for the entire super string using the ***pipe_value*** block **or** the pipe diameter varies for each segment of the super string using the ***pipe_data*** block.

The definitions are:

<pipe_value> *pipe_diameter_real* **</pipe_value>**

where *pipe_diameter_real* is the diameter for ***every*** segment of the string.

OR

For a super string with **n** vertices

<pipe_data>

<properties>

<diameter> pipe_diameter_for_segment_1 </diameter>

</properties>

<properties>

<diameter> pipe_diameter_for_segment_2 </diameter>

</properties>

...

<properties>

<diameter> pipe_diameter_for_segment_m </diameter>

</properties>

</pipe_data>

where

pipe_diameter_for_segment_i is the pipe diameter for the i'th segment, and

m = n-1 for an open string or m = n for a closed string.

1.6.10.8.2 Culvert Dimensions

There can be one culvert width and height for the entire super string using the ***culvert_value*** block **or** the culvert width and height vary for each segment of the super string using the ***culvert_data*** block.

The definitions are:

<culvert_value>

<width> *pipe_width_real* **</width>**

<height> *pipe_height_real* **</height>**

</culvert_value>

where *pipe_width_real* is the width and *pipe_height_real* is the height for every segment of the string.

OR

For a super string with **n** vertices

<culvert_data>

<properties>

<width> pipe_width_for_segment_1 </width>

<height> pipe_height_for_segment_1 </height>

</properties>

<properties>

<width> pipe_width_for_segment_2 </width>

<height> pipe_height_for_segment_2 </height>

</properties>

...

<properties>

<width> pipe_width_for_segment_m </width>

<height> pipe_height_for_segment_m </height>

</properties>

</culvert_data>

where

pipe_width_for_segment_i is the width and *pipe_height_for_segment_i* is the height for the i'th segment and

m = n-1 for an open string or m = n for a closed string.

1.6.10.8.3 Justification for Round or Culvert Pipes

There can be only one justification for all the round or culvert pipe segments in the super string.

The definition is:

<justify> *pipe_justification_text* **</justify>**

where

pipe_justification_text is the justification for the entire pipe and can have the values **centre**, **top**, **obvert**, **bottom** or **invert**.

If the ***justify*** block is missing then the round pipe or culvert pipe is **centre** justified.

1.6.10.9 Vertex Text and Vertex Annotation

See

1.6.10.9.1 Vertex Text
1.6.10.9.2 Vertex Annotation

1.6.10.9.1 Vertex Text

There can be not text at each vertex, the same piece of text for every vertex in the super string or a different text for each vertex of the super string.

Note: How the vertex text is drawn is specified by the vertex annotation. See 1.6.10.9.2 Vertex Annotation.

If there is a constant text value for each vertex in the string, then the text value is given by a **vertex_text_value** block:

<vertex_text_value> *text_value_text* **</vertex_text_value>**

where *text_value_text* is the constant text value for each vertex in the string.

For example, for a string of 5 vertices

```
<vertex_text_value>Constant text</vertex_text_value>
```

If there is a different text value for each vertex in the string, then the value of the text for each vertex is given in a ***vertex_text_data*** block.

<***vertex_text_data***>

<p> *text_value_for_vertex_1*</p>

<p> *text_value_for_vertex_2*</p>

...

<p> *text_value_for_vertex_n*</p>

</***vertex_text_data***>

where *text_value_for_vertex_i* is the vertex text for the i'th vertex.

For example, for a string of four vertices

```
<vertex_text_data>
  <p>First vertex</p>
  <p>Second vertex</p>
  <p/>
  <p/>
  </vertex_text_data>
```

1.6.10.9.2 Vertex Annotation

How the vertex text is drawn at each vertex is specified by the vertex annotation.

There can be no vertex annotations at all, or the same vertex annotation is used for every vertex in the string using the ***vertex_annotation_value*** block, or the vertex annotation varies for each vertex of the super string using the ***vertex_annotation_data*** block.

Note that in vertex annotations, the size of the text for all vertices must be either world size or all paper size or all screen size. That is, world size, paper size and screen size can not be mixed. The first one found is used for all vertices.

The definitions are:

<vertex_annotate_value>

vertex_annotation_information

</vertex_annotate_value>

where

vertex_annotation_information is the annotation to be used for drawing the text at every vertex of the super string. For the definition of *vertex_annotation_information* see 1.6.4.1 Vertex Annotation Information.

OR

For a super string with **n** vertices

<vertex_annotation_data>

annotation_for_vertex_1_block

annotation_for_vertex_2_block

...

annotation_for_vertex_n_block

</vertex_annotation_data>

where

annotation_for_vertex_i_block is the description for the vertex annotation for vertex i.

The format of the ***annotation_for_vertex_i_block*** is:

<properties>

vertex_*annotation_information*

<properties>

where

vertex_annotation_information is the annotation for drawing the text at the vertex. For the definition of *vertex_annotation_information* see 1.6.4.1 Vertex Annotation Information.

1.6.10.10 Segment Text and Segment Annotation

See

1.6.10.10.1 Segment Text
1.6.10.10.2 Segment Annotation

1.6.10.10.1 Segment Text

There can be no text on each segment, the same piece of text for every segment in the super string or a different text for each segment of the super string.

Note: How the segment text is drawn is specified by the segment annotation. See 1.6.10.10.2 Segment Annotation.

If there is a constant text value for each segment in the string, then the text value is given by a **segment_text_value** block:

<segment_text_value> *text_value_text* **</segment_text_value>**

where *text_value_text* is the constant text value for each segment in the string.

For example, for a string of 5 vertices

```
<segment_text_value>Constant text</segment_text_value>
```

If there is a different text value for each segment in the string, then the value of the text for each segment is given in a ***segment_text_data*** block.

<***segment_text_data***>

<p>*text_value_for_segment_1*</p>

<p>*text_value_for_segment_2*</p>

...

<p> *text_value_for_segment_m*</p>

</***segment_text_data***>

where

text_value_for_segment_i is the segment text for the i'th segment, and

m = n-1 for an open string or m = n for a closed string.

For example, for a string of four segments

```
<segment_text_data>
  <p>First segment</p>
  <p>Second Segment&#xa;Two lines</p>
  <p>seg3</p>
  <p/>
</segment_text_data>
```

1.6.10.10.2 Segment Annotation

How the segment text is drawn at each segment is specified by the segment annotation.

There can be no segment annotations at all, or the same segment annotation is used for every segment in the string using the ***segment_annotation_value*** block, or the segment annotation varies for each segment of the super string using the ***segment_annotation_data*** block.

Note that in segment annotations, the size of the text for all segments must be either world size or all paper size or all screen size. That is, world size, paper size and screen size can not be mixed. The first one found is used for all segments.

The definitions are:

<segment_annotate_value>

segment_annotation_information

</segment_annotate_value>

where

segment_annotation_information is the annotation to be used for drawing the text at every segment of the super string. For the definition of *segment_annotation_information* see 1.6.4.2 Segment Annotation Information.

OR

For a super string with **n** vertices

<segment_annotation_data>

annotation_for_segment_1_block

annotation_for_segment_2_block

...

annotation_for_segment_m_block

</vertex_annotation_data>

where

annotation_for_segment_i_block is the description for the annotation at segment i, and.

m = n-1 for an open string or m = n for a closed string.

The format of the ***annotation_for_segment_i_block*** is:

<properties>

segment_*annotation_information*

<properties>

where

segment_annotation_information is the annotation for drawing the text at the segment. For the definition of *segment_annotation_information* see 1.6.4.2 Segment Annotation Information.

Continue to 1.6.11 Super Alignment String or return to 1.6 Elements Contained in Models or 1 12d XML File Format.

1.6.11 Super Alignment String

Many software packages only allow *alignment* strings to be use the intersection point method (IP's) to construct the horizontal and vertical geometry. The IP definition is actually a *constructive* definition and the tangents points and segments between the tangent points (lines, arcs, transitions *etc*.) are calculated from the IP definition.

However for the ***12d Model super alignment***, the ***horizontal*** and ***vertical geometry*** are still defined separately and with construction definitions **but** the construction definition can be much more complex than just IP's. For example, an arc could be defined as being tangential to two offset elements, or constrained to go through a given point.

If the horizontal construction methods are consistent then the horizontal geometry can be solved, and the horizontal geometry expressed in terms of consecutive segments (lines, arcs, transitions and offset transitions) that are easily understood and drawn.

Similarly if the vertical construction methods are consistent then the vertical geometry can be solved, and the vertical geometry expressed in terms of consecutive segments (lines, arcs, parabolas) that are easily understood and drawn.

For the *super alignment* **both** the ***construction methods*** (the ***parts***) and the resulting ***vertices*** and ***segments*** (lines, arcs, transitions *etc*.) that make up the horizontal and vertical geometry (the ***data***) are written out to the 12d XML file.

For most applications such as uploading to survey data collectors or machine control devices, only the ***horizontal data*** and the ***vertical data*** are required, not the *construction* methods (*i.e.* not the ***horizontal*** and ***vertical parts***). So when reading the 12d XML of a *super alignment*, only the ***horizontal*** and ***vertical data*** needs to be read in and the constructive methods (the ***horizontal*** and ***vertical parts***) can be skipped over.

Consequently only the ***horizontal data*** and the ***vertical data*** are full documented for the super alignment.

However to allow ***12dXML*** to be easily written out by software packages that can only support HIP and VIP methods, there are special flags to denote these cases and the *horizontal_parts* and *vertical_parts* are fully defined for these special cases.

Special Cases for HIP Method Only and VIP Method Only

To allow 12dXML to be easily written out by software packages that can only support HIP and VIP methods, there are special flags to denote these cases and the *horizontal_parts* and *vertical_parts* are fully defined for these special cases. This also means that such a software package can easily read in from 12dXML any super alignments that use HIP and VIP methods only.

(a) If the **horizontal geometry** of the super alignment **only uses the HIP method** and hence only has Horizontal IP's with curves and transitions on them, then the HIP definition can be easily read in from the **horizontal_parts**. To alert any software reading 12dXML reader of this special case, there is a special flag **horizontal_ips_only** which is then set to **true**. Other wise it is false.

    This special case for horizontal_parts **is** fully documented in 1.6.11.2 Horizontal_Parts When Geometry is Defined by IP Method Only.

(b) If the **vertical geometry** of the super alignment **only uses the VIP method** and hence only has Vertical IP's with parabolic curves and arcs on them, then the VIP definition can be easily read in from the **vertical_parts**. To alert any software reading 12dXML reader of this special case, there is a special flag **vertical_ips_only** which is then set to **true**. Other wise it is false.

    This special case for horizontal_parts **is** fully documented in 1.6.11.5 Vertical_parts When VG is Defined by IP Method Only.

Notes

1. Just using the horizontal and vertical data is valid ***as long as the super alignment geometry is consistent*** and hence ***solves***, and the horizontal and vertical parts can then be created.

   There are the flags **valid_horizonal** and **valid_vertical** in the 12d XML of the super alignment and they are set to **true** if the horizontal and vertical geometry is consistent and solves.

2. Segments meeting at a common vertex do not have to be tangential although for most road and rail centre lines, they should be.

3. When ***12d Model*** reads in a 12d XML file and there is only ***horizontal_parts*** and no ***horizontal_data*** then if possible, ***12d Model*** generates the ***horizontal_data*** from the horizontal parts.

   This is very useful if you are creating a 12d XML file for a super alignment string that only uses HIP methods as it is fairly simple to create the ***horizontal_parts*** for such a string and that is fully documented in 1.6.11.2 Horizontal_Parts When Geometry is Defined by IP Method Only. For this case the flag **horizontal_ips_only** should be set to **true.**

4. When ***12d Model*** reads in a 12d XML file and there is only ***vertical_parts*** and no ***vertical_data*** then if possible, ***12d Model*** generates the ***vertical_data*** from the vertical parts.

   This is very useful if you are creating a 12d XML file for a super alignment string that only uses VIP methods as it is fairly simple to create the ***vertical_parts*** for such a string and that is fully documented in 1.6.11.5 Vertical_parts When VG is Defined by IP Method Only. For this case the flag **vertical_ips_only** should be set to **true.**

The 12*d XML* definition of the super alignment string is:

<string_super_alignment>

string_header_block

drawables_block

spiral_type_block

closed_block

valid_horizontal_block

valid_vertical_block

synch_vertical_block

label_style_block

horizontal_ips_only_block

vertical_ips_only_block

horizontal_parts_block

horizontal_data_block

vertical_parts_block

vertical_data_block

geometry_modifiers_block

</string_super_alignment>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block.

drawables_block

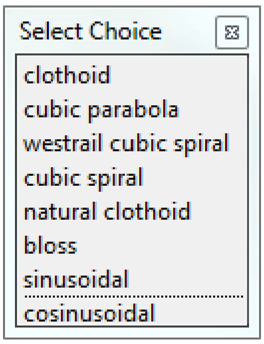
the drawables block contains information on how the super alignment is labelled.

This block is not documented.

spiral_type_block

<spiral_type> *transition_type_text* **</spiral_type>**

where *transition_type_text* is the default transition type use in the super alignment and is one of



For more information on the choices, see 1.3.16 Available Transition Types.

closed_block

<closed> closed_text_logical **</closed>**

where *closed_text_logical is* **true** if the super alignment string is closed and **false** if the super alignment string is open.

valid_horizontal_block

<*valid_horizontal*> *valid_horizontal_text_logical* **</*valid_horizontal*>**

where *valid_horizontal_text_logical is* **true** if the super alignment string horizontal geometry solves and **false** if the horizontal geometry does not solve.

If the horizontal geometry does not solve then the *horizontal_data* may be rubbish.

valid_vertical_block

<*valid_vertical*> *valid_vertical_text_logical* **</*valid_vertical*>**

where *valid_vertical_text_logical is* **true** if the super alignment string vertical geometry solves and **false** if the vertical geometry does not solve.

If the vertical geometry does not solve then the *vertical_data* may be rubbish.

synch_vertical_block

<*synch_vertical*> *synch_vertical_text_logical* **</*synch_vertical*>**

where *synch_vertical_text_logical is* **true** if the super alignment vertical geometry is to be synchronized to the horizontal geometry whenever the horizontal geometry is modified.

This is an internal ***12d Model*** flag.

label_style_block

<label_style> *label_style_text* **</label_style>**

where *label_style_text* is the name of the super alignment label style used for drawing the super alignment.

horizontal_ips_only_block

<*horizontal_ips_only*> *horizontal_ips_only_text_logical* **</*horizontal_ips_only*>**

where *horizontal_ips_only_text_logical is* **true** if the horizontal geometry of the super alignment consists of HIP methods only, and **false** if the horizontal geometry does not consist of HIP methods only.

vertical_ips_only_block

<*vertical_ips_only*> *horizontal_ips_only*_text_logical **</*vertical_ips_only*>**

where *vertical_ips_only_text_logical is* **true** if the vertical geometry of the super alignment consists of VIP methods only, and **false** if the vertical geometry does not consist of HIP methods only.

horizontal_parts_block

the *horizontal_parts* block contains the **methods** to construct the super alignment horizontal geometry. For example float (fillet) an arc of a certain radius between two given lines or create a transition (spiral or non-spiral transition) between a line and an arc.

The parts that make up the horizontal geometry are defined in chainage order from the start to the end of the super alignment.

If the horizontal construction methods are consistent, then they can be solved to form a plan string made up of lines, arcs and transitions and this is given in the ***horizontal_data*** block.

Because the construction methods can be very complex, the horizontal_parts block will only be documented for the case where all the horizontal parts are horizontal intersection points (HIPs) with an arc and leading and trailing transitions. See 1.6.11.2 Horizontal_Parts When Geometry is Defined by IP Method Only.

horizontal_data_block

the *horizontal_data* block contains the segments that define the horizontal geometry.

The *horizontal_data* block needs to be read in.

For the description of the *horizontal_data* block, see 1.6.11.1 Horizontal Data Block.

vertical_parts_block

the *vertical_parts* block contains the **methods** to construct the super alignment vertical geometry. For example float (fillet) an arc of a certain radius between two given lines.

The parts that make up the vertical geometry are defined in chainage order from the start to the end of the super alignment.

If the vertical construction methods are consistent, then they can be solved to form a string in (chainage, offset) space made up of lines, arcs and parabolas and this is given in the ***vertical_data*** block.

Because the construction methods can be very complex, the vertical_parts block will only be documented for the case where all the vertical parts are vertical intersection points (VIPs) with an arc or a parabola on the VIP. See 1.6.11.5 Vertical_parts When VG is Defined by IP Method Only.

vertical_data_block

the *vertical_data* block contains the segments that define the vertical geometry.

The *vertical_data* block needs to be read in.

For the description of the *vertical_data* block, see 1.6.11.3 Vertical Data Block.

geometry_modifiers_block

the *geometry_modifiers_parts* block contains extra construction information for the super alignment.

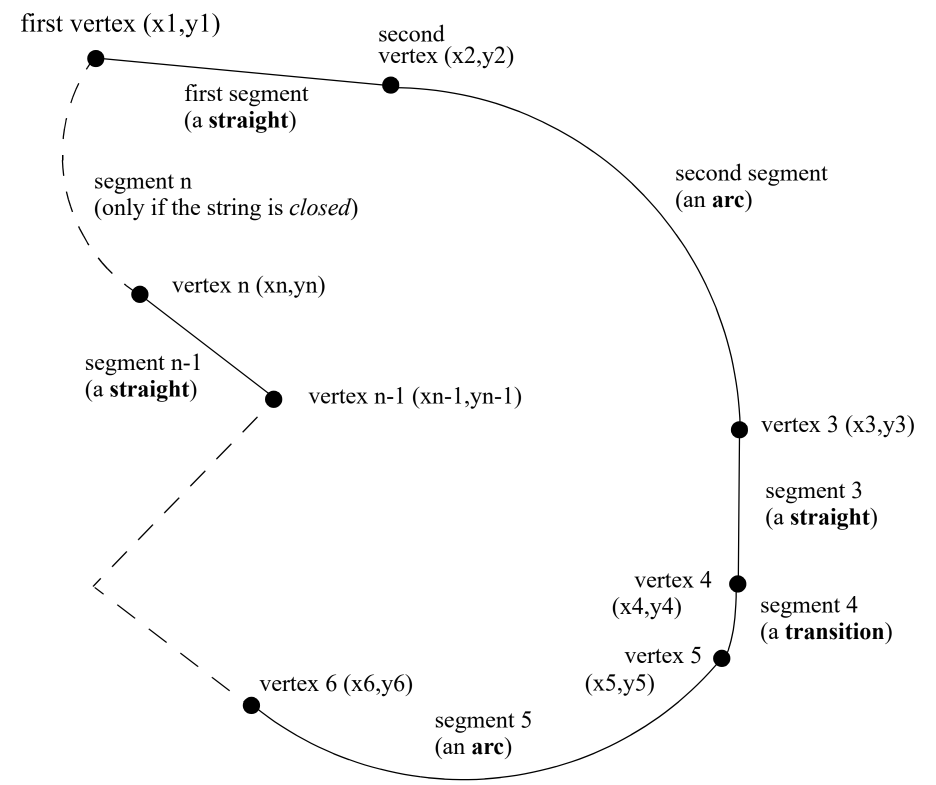
This block is not documented.

1.6.11.1 Horizontal Data Block

The *horizontal_data* block contains the ***solved*** horizontal geometry of the super alignment.

The *solved horizontal geometry* is made up of a series of (x,y) vertices given in a *data_2d* block followed by a *geometry_data* block specifying the geometry of the segments between adjacent vertices. Each segment can be a straight line, an arc, a transition or an offset transition.

If the solved horizontal geometry has **n** vertices, then there will be **n-1 segments** for an ***open*** super alignment or **n segments** if the super alignment is ***closed***.



Vertices and Segments Forming the Horizontal Data for the Super Alignment

The format of the *horizontal_data* block is the same as for the segments of a super string except that the data is only in 2D. Unlike a super string where there is just a z-value at each vertex, the third dimension of the super alignment is given by the *vertical_data* block (see 1.6.11.3 Vertical Data Block).

The definition of the *horizontal_data* block is:

<horizontal_data>

string_header_block

closed_block

interval_block

data_2d_block

geometry_data_block

blocks_of_info_1

blocks_of_info_2

...

blocks_of_info_n

</horizontal_data>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block. This provides information such as colour for the horizontal data.

interval_block

The interval_block for a super string has a *distance* (a chainage interval) and a *chord_to_arc*_real

where

the ***distance*** to temporarily introduce extra vertices into the string at the given chainage distance when the string is in a triangulation to form a tin.

chord_arc_real is a real number and is the chord to arc tolerance to use on any arcs in the horizontal data to temporarily insert vertices into the arc when the arc is included in a triangulation to form a tin.

For the syntax of interval_block, see 1.3.6 Interval.

data_2d_block

the *data_2d* block defines the (x,y) value of the vertices that makes up the horizontal data.

For the definition of the data_2d block, see 1.3.13 data_2d.

geometry_data_block

the segments of the horizontal data can be straights, arcs, transitions or offset transitions and they are identical to the definitions of the horizontal segments for super strings.

So for the definition of the *geometry_data* block*, see* the section for super strings1.6.10.2 Geometry of the Horizontal Segments

blocks_of_info

extra information for the vertices and/or segments such as colour, attributes, vertex text, vertex uids *etc* are defined in the same way as for super strings.

1.6.11.2 Horizontal_Parts When Geometry is Defined by IP Method Only

When the horizontal geometry is defined by IP methods only, then the ***horizontal_parts*** is fairly straight forward.

When ***12d Model*** reads in a 12d XML file and there is no ***horizontal_data*** section, then ***12d Model*** will calculate the ***horizontal_parts***. So you are writing a 12d XML with only IP methods for the horizontal geometry then simply leave out the ***horizontal_data*** section and ***12d Model*** will calculate it for you.

For a horizontal geometry is defined by IP methods only, the ***horizontal_parts*** definition is:

<horizontal_data>

info_for_HIP_1_block

info_for_HIP_2_block

...

info_for_HIP_n_block

</horizontal_data>

where ***info_for_HIP_i_block*** is the information about the successive HIPs in the super alignment and is one of:

(a) A horizontal intersection point (HIP) with no arc.

This is defined by:

<ip>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<x> *x_ip_coordinate_real* **</x>**

<y> *y_ip_coordinate_real* **</y>**

</ip>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

x_ip_coordinate_real is the x coordinates of the HIP.

y_ip_coordinate_real is the y coordinates of the HIP.

(b) A horizontal intersection point (HIP) with an arc of a given radius at the HIP.

This is defined by:

<arc>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<r> *arc_radius_real* **</r>**

<x> *x_ip_coordinate_real* **</x>**

<y> *y_ip_coordinate_real* **</y>**

</arc>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

arc_radius_real is the radius of the arc on the HIP.

x_ip_coordinate_real is the x coordinate of the HIP.

y_ip_coordinate_real is the y coordinate of the HIP.

(c) A horizontal intersection point (HIP) with an arc of a given length at the HIP

This is defined by:

<length>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<l> *arc_length_real* </l>

<x> *x_ip_coordinate_real* **</x>**

<y> *y_ip_coordinate_real* **</y>**

</length>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

arc_length_real is the length of the arc on the HIP.

x_ip_coordinate_real is the x coordinate of the HIP.

y_ip_coordinate_real is the y coordinate of the HIP.

(d) A horizontal intersection point (HIP) with an arc and transitions

This is defined by:

<spiral>

<id> *part_id_integer* </id>

time_created_block

time_updated_block

transition_type_block

<r> *arc_radius_real* **</r>**

<l1> *leading_transition_length_real* **</l1>**

<l2> *trailing_transition_length_real* **</l2>**

<x> *x_ip_coordinate_real* **</x>**

<y> *y_ip_coordinate_real* **</y>**

</spiral>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

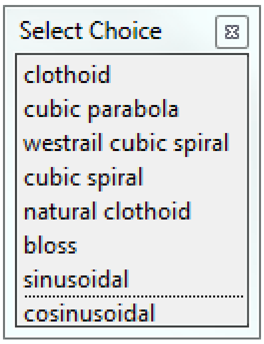
time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

transition_type_block

<transition_type> *transition_type_text* **</transition_type>**

where *transition_type_text* is the default transition type use in the super alignment and is one of



This block is optional and if it is missing then the ***default transition type*** for the super alignment is used.

For more information on the choices, see 1.3.16 Available Transition Types.

arc_radius_real is the radius of the arc on the HIP.

leading_transition_length_real is the length of the leading transition on the HIP.

trailing_transition_length_real is the length of the trailing transition on the HIP.

x_ip_coordinate_real is the x coordinate of the HIP.

y_ip_coordinate_real is the y coordinate of the HIP.

Notes

1. A <length> block with *arc_length_real* equal to zero, or a <spiral> block with the *arc_radius_real*, *leading_transition_length_real* and *trailing_transition_length_real* all zero, will also represent a HIP with no arcs or transitions on it.:

   <length>

   <id> *part_id_integer* **</id>**

   time_created_block

   time_updated_block

   <l> *0* </l>

   <x> *x_ip_coordinate_real* **</x>**

   <y> *y_ip_coordinate_real* **</y>**

   </length>

   OR

   <spiral>

   <id> *part_id_integer* </id>

   time_created_block

   time_updated_block

   transition_type_block

   <r> *0* **</r>**

   <l1> *0* **</l1>**

   <l2> *0* **</l2>**

   <x> *x_ip_coordinate_real* **</x>**

   <y> *y_ip_coordinate_real* **</y>**

   </spiral>

2. If the HIP is the first HIP or the last HIP then no arc or transitions will be drawn even if the relevant parameters are non zero.

As an example of horizontal_parts with only HIP methods:

```
<horizontal_parts>
  <ip>
   <id> 100 </id>
   <x> 42606.66161172 </x>
   <y> 37239.28824481 </y>
  </ip>

  <ip>
    <id> 200 </id>
    <x> 43134.36832349 </x>
    <y > 37330.26705997</y>
  </ip>

  <spiral>
    <id> 300 </id>
    <r> 50 </r>
    <l1> 30 </l1>
    <l2> 40 </l2>
    <x> 43336.6595 </x>
   <y> 37469.2563 </y>
  </spiral>

  <arc>
   <id> 400 </id>
   <r> 75 </r>
   <x> 43481.15324268 </x>
   <y> 37331.6431906 </y>
  </arc>

   <ip>
   <id> 500 </id>
   <x> 43627.02308964 </x>
    <y> 37544.94343852 </y>
   </ip>
</horizontal_parts>
```

1st HIP
HIP only

Unique Part id
incrementing by 100

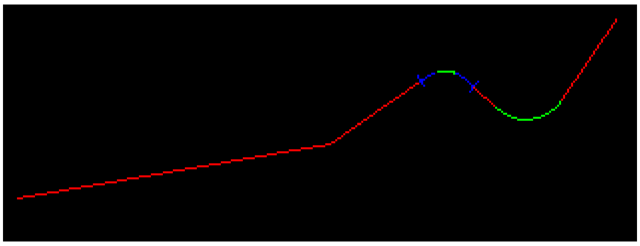
2nd HIP
HIP only

3rd HIP
HIP with arc and
leading and trailing
transitions

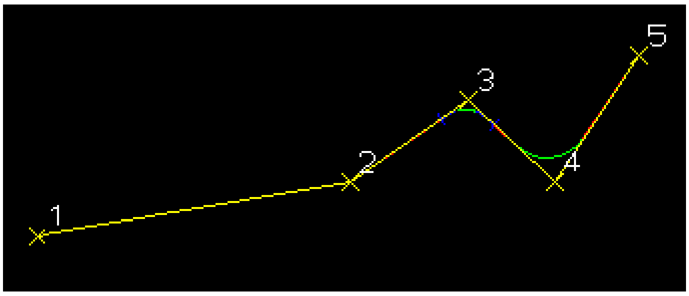
4th HIP
HIP with arc only

5th HIP
HIP only

Plan View of Super Alignment



Super Alignment Being Edited



Horizontal Parts with IP Methods Only

1.6.11.3 Vertical Data Block

The *vertical_data* block contains the ***solved*** vertical geometry of the super alignment.

The *solved vertical geometry* is made up of a series of (chainage,height) vertices given in a *data_2d* block followed by a *geometry_data* block specifying the geometry of the segments between adjacent vertices. The segment can be a straight line, a parabola or an arc.

Note that the ***chainage*** is the chainage of the horizontal geometry defined in the *horizontal_data* block (see 1.6.11.1 Horizontal Data Block).

If the vertical geometry has **n** vertices, then there will be **n-1 segments** for an ***open*** super alignment or **n segments** if the super alignment is ***closed***.

The format of the *vertical_data* block is the same as for the segments in a horizontal_data block except that the data is (chainage, height) rather than (x,y) and there is no transitions but a parabola instead.

The definition of the *vertical_data* block is:

<vertical_data>

string_header_block

closed_block

interval_block

data_2d_block

geometry_data_block

blocks_of_info_1

blocks_of_info_2

.…

blocks_of_info_n

</vertical_data>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block. This provides information such as colour for the vertical data.

interval_block

The interval_block for a super string has a *distance* (a chainage interval) and a *chord_to_arc*_real

where

the ***distance*** to temporarily introduce extra vertices into the string at the given chainage distance when the string is in a triangulation to form a tin.

chord_arc_real is a real number and is the chord to arc tolerance to use on any arcs in the vertical data to temporarily insert vertices into the arc when the arc is included in a triangulation to form a tin.

For the syntax of interval_block, see 1.3.6 Interval.

data_2d_block

the *data_2d* block defines the (chainage,height) value of the vertices that makes up the vertical data.

For the definition of the data_2d block, see 1.3.13 data_2d where x is chainage and y is height.

geometry_data_block

the segments of the vertical data can be straights, arcs or parabolas.

For the definition of the *geometry_data* block*, see* 1.6.11.4 Geometry of the Vertical Segments

blocks_of_info

extra information for the vertices and/or segments such as colour, attributes, vertex text, vertex uids *etc* are defined in the same way as for super strings.

1.6.11.4 Geometry of the Vertical Segments

If the segments are straight lines only then that is the default and no further information is required.

If the segments are only straight lines and arcs, then the ***radius_data*** and **major_data** blocks are used to define a ***radius*** and ***bulge_flag*** data for each segment of the super string. See 1.6.11.4.1 Only Straights and Arcs for Segments.

If any of the segments are parabolas then ***geometry_data*** must be used for each segment. ***geometry_data*** can represent a straight, arc, transition or offset transition. See 1.6.11.4.2 Straights, Arcs and Parabolas for Segments.

1.6.11.4.1 Only Straights and Arcs for Segments

If there are only straight and arc segments for the string, then for the **data_2d** it is possible to add a radius and major/minor arc flag for each segment of the super string using the ***radius_data*** and **major_data** blocks respectively. See 1.3.15 radius_data and major_data.

1.6.11.4.2 Straights, Arcs and Parabolas for Segments

When some of the segments are parabolas then the ***geometry_data*** block **must** be used the give the geometry for each segments.

When the vertical_data has **n** vertices, then the definition of the ***geometry_data*** block is

<geometry_data>

info_for_segment_1_block

info_for_segment_2_block

...

info_for_segment_m_block

</geometry_data>

where

info_for_segment_i_block is the information defining the i'th segment as either a straight, an arc or an parabola and m = n-1 for an open string or m = n for a closed string.

For the definition of *info_for_segment_i_block* see:

1.6.11.4.2.1 Straight

1.6.11.4.2.2 Arc

1.6.11.4.2.3 Parabola

1.6.11.4.2.1 Straight

No parameters are needed for defining a straight segment. The *straight* block is simply:

<straight> </straight>

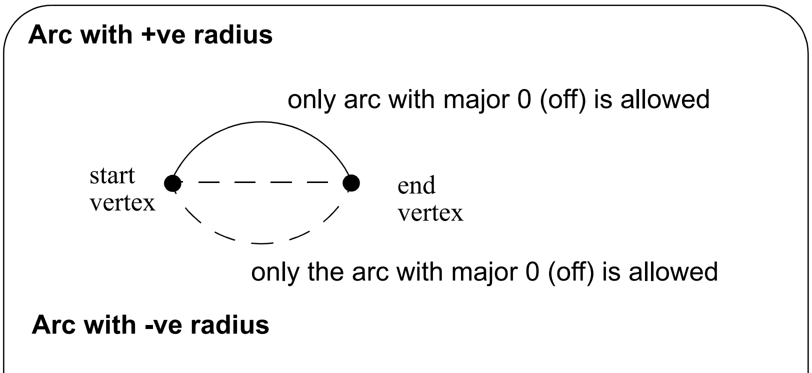
or simply

<straight/>

1.6.11.4.2.2 Arc

Since vertical geometry can’t go backwards in chainage value, the majors arcs can not be used and hence there are only possibilities for an arc of a given radius placed between two vertices.

We use *positive* and *negative* radius to differentiate between the four possibilities.



Arcs with same absolute radius

The *arc* block is:

<arc>

<radius> *radius_for_segment*</radius>

<major> *major_flag_for_segment*</major>

</arc>

where

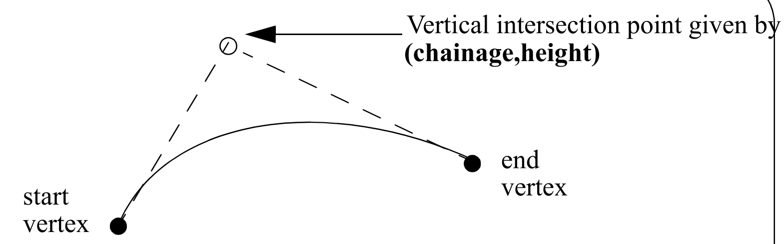
radius_for_segment is the radius for the segment where positive is above the line connecting the vertices.

major_flag_for_segment is ignored because only minor arcs are allowed.

1.6.11.4.2.3 Parabola

There can be a parabola between adjacent vertices. The parabola is defined by giving the coordinates of the vertical intersection point for the parabola

chainage chainage of the VIP of the parabola
height height of the VIP of the parabola



Example of a Parabola

The *parabola* block is:

<parabola>

<chainage> *vip_chainage_real* </chainage>

<height> *vip_height_real* </height>

</parabola

where

vip_chainage_real is the chainage of the VIP of the parabola

vip_height_real is the height of the VIP of the parabola

1.6.11.5 Vertical_parts When VG is Defined by IP Method Only

When the vertical geometry is defined by IP methods only, then the ***vertical_parts*** is fairly straight forward.

When ***12d Model*** reads in a 12d XML file and there is no ***vertical_data*** section, then ***12d Model*** will calculate the ***vertical_parts***. So if you are writing a 12d XML with only VIP methods for the vertical geometry then simply leave out the ***vertical_data*** section and ***12d Model*** will calculate it for you.

For a vertical geometry is defined by VIP methods only, the ***vertical_parts*** definition is:

<vertical_data>

info_for_VIP_1_block

info_for_VIP_2_block

...

info_for_VIP_n_block

</vertical_data>

where ***info_for_VIP_i_block*** is the information about the successive VIPs in the super alignment and is one of:

(a) A vertical intersection point (VIP) with no arc or parabola.

This is defined by:

<ip>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<x> *chainage_ip_coordinate_real* **</x>**

<y> *height_ip_coordinate_real* **</y>**

</ip>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

chainage_ip_coordinate_real is the chainage of the VIP.

height_ip_coordinate_real is the height of the VIP.

(b) A vertical intersection point (VIP) with an parabola of a given chainage length at the VIP

This is defined by:

<length>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<l> *parabola_chainage_length_real* </l>

<x> *chainage_ip_coordinate_real* **</x>**

<y> *height_ip_coordinate_real* **</y>**

</length>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

parabola_chainage_length_real is the chainage length of the parabola on the VIP.

chainage_ip_coordinate_real is the chainage of the VIP.

height_ip_coordinate_real is the height of the VIP.

(c) A vertical intersection point (VIP) with an parabola of a given k value at the VIP

This is defined by:

<kvalue>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<k> *parabola_k_value_real* </k>

<x> *chainage_ip_coordinate_real* **</x>**

<y> *height_ip_coordinate_real* **</y>**

</kvalue>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

parabola_k_value_real is the k value of the parabola on the VIP.

chainage_ip_coordinate_real is the chainage of the VIP.

height_ip_coordinate_real is the height of the VIP.

(d) A vertical intersection point (VIP) with an parabola of a given effective radius value at the VIP

This is defined by:

<radius>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<r> *parabola_effective_radius_value_real* </r>

<x> *chainage_ip_coordinate_real* **</x>**

<y> *height_ip_coordinate_real* **</y>**

</kvalue>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

parabola_effective_radius_value_real is the effective radius of the parabola on the VIP.

chainage_ip_coordinate_real is the chainage of the VIP.

height_ip_coordinate_real is the height of the VIP.

(e) A vertical intersection point (VIP) with an arc of a given radius at the VIP.

This is defined by:

<arc>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<r> *arc_radius_real* **</r>**

<x> *chainage_ip_coordinate_real* **</x>**

<y> *height_ip_coordinate_real* **</y>**

</arc>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

arc_radius_real is the radius of the arc on the VIP.

chainage_ip_coordinate_real is the chainage of the VIP.

height_ip_coordinate_real is the height of the VIP.

(f) A vertical intersection point (VIP) with an asymmetric parabola defined by the start and end chainage lengths at that VIP

This is defined by:

<asymmetric>

<id> *part_id_integer* **</id>**

time_created_block

time_updated_block

<l1> *parabola_start_chainage_length_real* **</l1>**

<l2> *parabola_end_chainage_length_real* **</l2>**

<x> *chainage_ip_coordinate_real* **</x>**

<y> *height_ip_coordinate_real* **</y>**

</asymmetric>

where

part_id_integer is a number that is unique for each horizontal and vertical part and the value is a multiple of 100.

time_created_block

is the time the super tin was originally created, This is optional. For the syntax see 1.3.7 Time Created.

time_updated_block

is the last time the super tin was last modified, This is optional. For the syntax see 1.3.7 Time Created.

parabola_start_chainage_length_real is the start chainage length of the asymmetric parabola on the VIP.

parabola_end_chainage_length_real is the end chainage length of the asymmetric parabola on the VIP.

chainage_ip_coordinate_real is the chainage of the VIP.

height_ip_coordinate_real is the height of the VIP.

Notes

1. A <length> block with *arc_length_real* equal to zero, or a <spiral> block with the *arc_radius_real*, *leading_transition_length_real* and *trailing_transition_length_real* all zero, will also represent a HIP with no arcs or transitions on it.:

   <length>

   <id> *part_id_integer* **</id>**

   time_created_block

   time_updated_block

   <l> *0* </l>

   <x> *x_ip_coordinate_real* **</x>**

   <y> *y_ip_coordinate_real* **</y>**

   </length>

   OR

2. If the VIP is the first VIP or the last VIP then no parabola or arc will be drawn even if the relevant parameters are non zero.

As an example of ***vertical_parts*** with only VIP methods:

```
<vertical_parts>
  <ip>
    <id> 600 </id>
    <x>-50.8459652 <x>
    ,<y> 59.79764161 <y>
  </ip>
  <kvalue>
    <id> 700 </id>
    <k> 1.25 </k>
    <x> 38.4627 </x>
    <y> 179.2126 </y>
  </kvalue>
    <length>
    <id> 800 </id>
    <l> 50 </l>
    <x> 172.61694837 </x>
    <y> 154.72967932 <x>
  </length>
  <asymmetric>
    <id> 900 </id>
    <l1> 25 </l1>
    <l2> 75 </l2>
    <x> 270.0182 </x>
    <y> 208.1493 </y>
  </asymmetric>
    <arc>
    <id> 1000 </id>
    <r> 1000 </r>
    <x> 424.2402 </x>
    <y>196.5637 </y>
  </arc>
  <radius>
    <id> 1100 </id>
    <r> 200 </r>
    <x> 526.7263 </x>
    <y> 201.5302 </y>
  </radius>
  <ip>
    <id> 1200 </id>
    <x> 637.69216273 </x>
    <y> 198.71894484 </y>
  </ip>
</vertical_parts>
```

1st VIP
VIP only

Unique Part id
incrementing by 100

2nd VIP
Parabola defined
by k value

3rd VIP
Parabola defined
by length

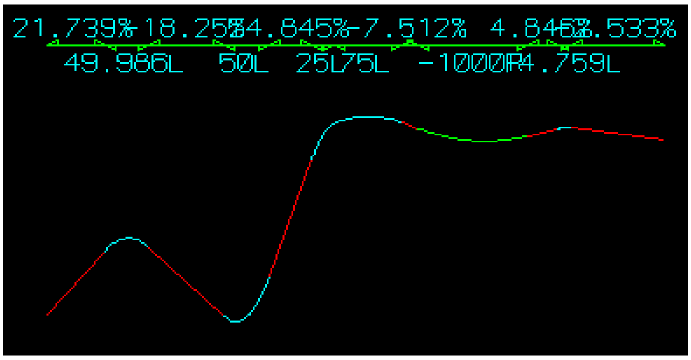
4th VIP
Asymmetric parabola defined
by two lengths

5th VIP
Arc with radius

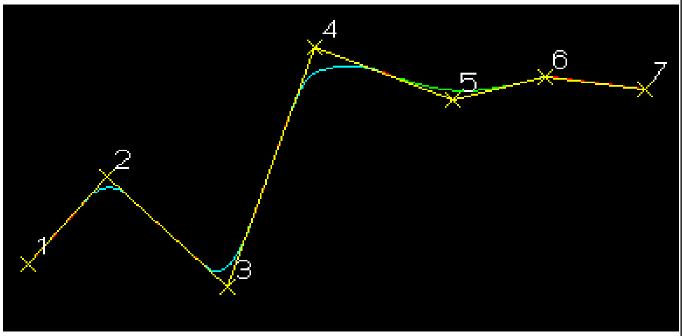
6th VIP
Parabola defined
by effective radius

7th VIP
VIP only

Section View of Super Alignment



Vertical Geometry Being Edited



Vertical Parts with IP Methods Only

Continue to 1.6.12 Text String or return to 1.6 Elements Contained in Models or 1 12d XML File Format.

1.6.12 Text String

The format for the **string_text** element is:

<string_text>

string_header_block

point_block

vertex_text_value_block

vertex_annotate_value_block

</string_arc>

where

string_header_block

the common header block for each string. for the contents and the syntax, see 1.6.3 String Header Block.

point_block

The format of the **point_block** is:

<point> *x_real y_real z_real* **</point>**

where

(*x_real,y_real,z_real*) is the vertex of the text.

vertex_text_value_block

The text for the text string.

The format of the **vertex_text_value_block** is:

<vertex_text_value**>** *characters_of_the_text* **</**vertex_text_value**>**

where

characters_of_the_text is the characters of the text with the except of some character that are special characters and are replace by something else.

For example **&** in the text is replaced **&amp** and a new line is given by **
**. See Characters "<", ">" and "&" and Escaping.

vertex_annotate_block

These are the setting for displaying text at a vertex.

The format of the **vertex_annotate_block** is:

<vertex_text_value**>**

vertex_annotation_information

</vertex_text_value**>**

where

vertex_annotation_information is the annotation to be used for drawing the text. For the definition of *vertex_annotation_information* see 1.6.4.1 Vertex Annotation Information.

For example

```
<string_text>
    <name>text</name>
    <chainage>0</chainage>
    <breakline>line</breakline>
    <colour>yellow</colour>
    <style>1</style>
    <time_created>28-Apr-2015 07:48:35</time_created>
```

```
    <time_updated>28-Apr-2015 07:49:33</time_updated>
    <point>1230.93054186 517.0328703 null</point>
    <vertex_text_value>First line&#xa;Second line</vertex_text_value>
    <vertex_annotate_value>
        <worldsize>20</worldsize>
        <textstyle>Arial</textstyle>
        <angle>45</angle>
        <x_factor>1</x_factor>
        <slant>0</slant>
        <offset>0</offset>
        <raise>0</raise>
        <text_colour>yellow</text_colour>
        <justify>middle-centre</justify>
    </vertex_annotate_value>
</string_text>
```

Continue to 1.6.13 Trimesh or return to 1.6 Elements Contained in Models or 1 12d XML File Format.

1.6.13 Trimesh

A trimesh is a type of primitive_3d object.

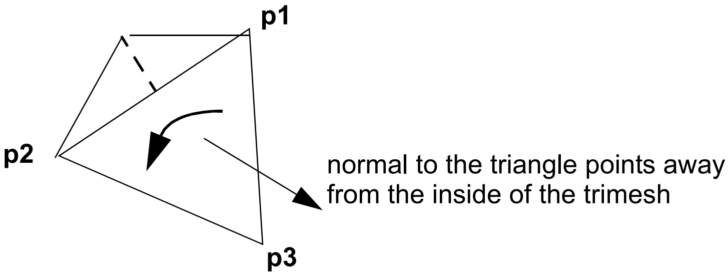
A trimesh is made up of 3D triangles and can be described by giving the list of ***m*** vertices in the trimesh and the three vertices that make up each of the ***n*** triangular faces. The normal to each triangle face points to the "outside" of the trimesh.

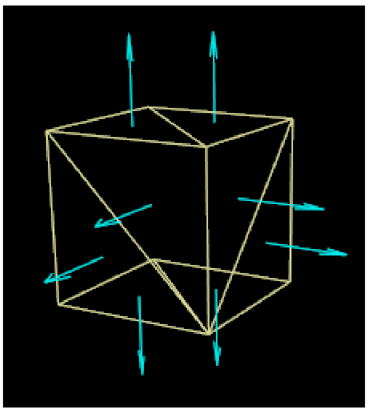
So the trimesh element contains a list of 3d points and a list of triangle faces where each triangle face is given as a triple of indices of points from the point list.

The order of the points **p1**, **p2** and **p3** in the triangle triple is important and must be such that the direction of the normal vector to each triangle points away from the inside of the trimesh.

That is, the normal vector of the triangle which is given by the cross product of the two vectors **p1p2** and **p1p3** points away from the inside of the trimesh.

Hence when looking towards the triangle from the outside, the points **p1**, **p2** and **p3** are in a **counter clockwise** order around the triangle.





The 12d XML definition of a trimesh is:

<primitive_3d>

string_header_block

trimesh_3d_block

</primitive_3d>

where

string_header_block

the header block for a trimesh is the same as the common header block for a string. For the contents and the syntax, see 1.6.3 String Header Block.

The ***colour*** in the *string_header_block* is the default colour for the triangles in the trimesh.

trimesh_block

The trimesh block gives the vertices of the trimesh and then the faces of the trimesh in terms of the vertex numbers.

<trimesh_3d>

<***vertices***>

<v> *x_value_1* *y_value_1* *z_value_1* **</v>**

<v> *x_value_2* *y_value_2* *z_value_2* **</v>**

...

<v> *x_value_n* *y_value_n* *z_value_n* **</v>**

</***vertices***>

<***faces***>

<f> *face_1_vertex_1* *face_1_vertex_2* *face_1_vertex_3* **</f>**

<f> *face_2_vertex_1* *face_2_vertex_2* *face_2_vertex_3* **</f>**

...

<f> *face_m_vertex_1* *face_m_vertex_2* *face_m_vertex_3* **</f>**

</***faces***>

<***edges***>

<e> *edge_1_vertex_1* *edge_1_vertex_2* **</e>**

<e> *edge_2_vertex_1* *edge_2_vertex_2* **</e>**

...

<e> *edge_p_vertex_1* *edge_p_vertex_2* **</e>**

</***edges***>

</trimesh_3d>

where

n is the number of vertices and (*x_value_i*, *y_value_i*, *z_value_i*) are the 3D coordinates of the ***i***'th vertex. The vertices are implicitly numbered by the order in the list (starting at vertex 1)

m is the number of faces in the trimesh and *face_j_vertex_1*, *face_j_vertex_2*, *face_j_vertex_3* are the vertex numbers of the vertices (in the ***vertices*** block) for the j'th face.

p is the number of edges in the trimesh and *edge_j_vertex_1*, *edge_j_vertex_2* are the vertex numbers of the vertices (in the ***vertices*** block) for the j'th edge.

The order of the faces in the faces block is important for many calculations, mesh properties, geometric structures.The correct order for edge in the edges block can only be formed inside ***12d Model***. For manual construction of the 12da file for trimesh, the user should leave out the edges block.

The ***vertices*** block and ***faces*** block are compulsory part of **trimesh_3d**; all other blocks (including **edges** block) in **trimesh_3d** are optional.

<trimesh_3d>

<***vertices***> </***vertices***>

<***faces***> </***faces***>

<***edges***> </***edges***>

<***info***> *info_block_contents* </***info***>

<***blend***> *blend_value_real* <***/blend***>

<***vertex_infos***> *array_infos* <***/vertex_infos***>

<***vertex_flags***> *index_array_integers* <***/vertex_flags***>

<***edge_infos***> *array_infos*<***/edge_infos***>

<***edge_flags***> *index_array_integers* <***/edge_flags***>

<***face_infos***> *array_infos* <***/face_infos***>

<***face_flags***> *index_array_integers* <***/face_flags***>

</trimesh_3d>

Information block

<***info***>

<flag> *flag_value_integer* **</flag>**

<key> *key_value_short_integer* **</key>**

<colour> *value_colour* **</colour>**

<name> *value_string_name* **</name>**

</***info***>

In a **info** block, **flag** and **key** are reserved for future development usage, the value for a key is between 0 and 255.

Note that the **colour** and **name** in the ***info*** of **trimesh_3d** block is distinct from ones of the string header block.

The value for **blend** should be a real number between 0 and 1; 0 means total transparent and; 1 (which is the default value) means total opaque.

The contents of ***vertex_infos edge_infos face_infos*** blocks are a sequences of ***info*** block (*array_infos*).

vertex_flags is a sequence (array) of **n** index integer which can refer to either: an index in the ***vertex_infos*** block (start from 1); or 0 which means there is no information on the vertex.

For example, **n** = 5

There are two kinds of information for a vertex.

<***vertex_infos***> *info1 info2* <***/vertex_infos***>

info1 <***info***> **... <colour>** *blue* **</colour> ...** </***info***>

info2 <***info***> **... <colour>** *green* **</colour> ...** </***info***>

Each of the 5 vertex has a flag number in

<***vertex_flags***> *2 0 1 2 0* <***/vertex_flags***>

The example indicates that vertices number 1 and 4 have colour green; vertex number 3 has colour blue; vertices number 2 and 5 have no information.

face_flags is a sequence (array) of **m** index integer which can refer to either: an index in the ***face_infos*** block (start from 1); or 0 which means there is no information on the face.

edge_flags is a sequence (array) of index integer which can refer to either: an index in the ***edge_infos*** block (start from 1); or 0 which means there is no information on the edge.

1.6.14 LAS Cloud String

The 12d XML format for a LAS cloud string without reference data:

<string_las_cloud_data>

string_header_block

data_block

</string_las_cloud_data>

And for a LAS cloud string with reference data:

<string_las_cloud_data>

string_header_block

ref_data_block

</string_las_cloud_data>

where

string_header_block

the header block for a trimesh is the same as the common header block for a string. For the contents and the syntax, see 1.6.3 String Header Block.

The data block contains:

<data>

category_block

format_block

range_block

points_block

</data>

The category block contains **categories** tag and a list of boolean value (true or false).

<categories>

boolean_value boolean_value ... boolean_value

</categories>

The range block contains four integer values.

<range>

<xmin> xmin_value </xmin>

<xmax> xmax_value </xmax>

<ymin> ymin_value </ymin>

<ymax> ymax_value </ymax>

</range>

The format block is.

<format>

format_name

</format>

Where *format_name* must come from the list

v10_p0 v10_p1

v11_p0 v11_p1

v12_p0 v12_p1 v12_p2 v12_p3

v13_p0 v13_p1 v13_p2 v13_p3

v14_p0 v14_p1 v14_p2 v14_p3 v14_p4 v14_p5 v14_p6 v14_p7 v14_p8 v14_p9 v14_p10

The points block must match the format given in the format block. For each format type vX_pY where X comes from the set: 10 11 12 13 14 and Y comes from the set 0 1 2 3 4 5 6 7 8 9 10; there are two choice of points data: **points_vX_pY** and **compact_points_vX_pY**.

<points_vX_pY>

point_pY

point_pY

...

...

point_yY

</points_vX_pY>

<compact_points_vX_pY>

compact_point_pY

compact_point_pY

...

...

compact_point_yY

</compact_points_vX_pY>

The point_p0 block is.

<p>

<x> x_coordinate <\x>	
<y> y_coordinate <\y>	
<z> z_coordinate <\z>	
<i> intensity <\i>	*\\ integer between 0 and 65535*
<rn> return_number <\rn>	*\\ integer between 0 and 7*
<rc> return_count <\rc>	*\\ integer between 0 and 7*
<sd> scan_direction <\sd>	*\\ integer between 0 and 1*
<fe> flight_line_edge <\fe>	*\\ integer between 0 and 1*
<cl> classification <\cl>	*\\ integer between 0 and 255*
<sr> scan_rank_angle <\sr>	*\\ integer between -128 and 127*
<ud> user_data <\ud>	*\\ integer between 0 and 255*
<id> point_source_id <\id>	*\\ integer between 0 and 65535*

</p>

The compact_point_p0 block is the same as point_p0 but without any inner tag.

<p>

x_coordinate

y_coordinate
z_coordinate
intensity \\ integer between 0 and 65535
return_number \\ integer between 0 and 7
return_count \\ integer between 0 and 7
scan_direction \\ integer between 0 and 1
flight_line_edge \\ integer between 0 and 1
classification \\ integer between 0 and 255
scan_rank_angle \\ integer between -128 and 127
user_data \\ integer between 0 and 255
point_source_id \\ integer between 0 and 65535

</p>

The point_p1 block is the same as point_p0 but with a time at the end.

<p>

<x> x_coordinate <\x>
<y> y_coordinate <\y>
<z> z_coordinate <\z>
<i> intensity <\i> \\ integer between 0 and 65535
<rn> return_number <\rn> \\ integer between 0 and 7
<rc> return_count <\rc> \\ integer between 0 and 7
<sd> scan_direction <\sd> \\ integer between 0 and 1
<fe> flight_line_edge <\fe> \\ integer between 0 and 1
<cl> classification <\cl> \\ integer between 0 and 255
<sr> scan_rank_angle <\sr> \\ integer between -128 and 127
<ud> user_data <\ud> \\ integer between 0 and 255
<id> point_source_id <\id> \\ integer between 0 and 65535
<t> gps_time <\t> \\ real number

</p>

The compact_point_p1 block is the same as point_p1 but without any inner tag.

<p>

x_coordinate
y_coordinate
z_coordinate
intensity \\ integer between 0 and 65535
return_number \\ integer between 0 and 7
return_count \\ integer between 0 and 7
scan_direction \\ integer between 0 and 1
flight_line_edge \\ integer between 0 and 1
classification \\ integer between 0 and 255

scan_rank_angle \\ *integer between -128 and 127*

user_data \\ *integer between 0 and 255*

point_source_id \\ *integer between 0 and 65535*

gps_time \\ *real number*

</p>

The point_p2 block is the same as point_p0 but with a colour (64bit integer) at the end.

<p>

<x> x_coordinate <\x>

<y> y_coordinate <\y>

<z> z_coordinate <\z>

<i> intensity <\i> \\ *integer between 0 and 65535*

<rn> return_number <\rn> \\ *integer between 0 and 7*

<rc> return_count <\rc> \\ *integer between 0 and 7*

<sd> scan_direction <\sd> \\ *integer between 0 and 1*

<fe> flight_line_edge <\fe> \\ *integer between 0 and 1*

<cl> classification <\cl> \\ *integer between 0 and 255*

<sr> scan_rank_angle <\sr> \\ *integer between -128 and 127*

<ud> user_data <\ud> \\ *integer between 0 and 255*

<id> point_source_id <\id> \\ *integer between 0 and 65535*

<c> las_colour <\c> \\ *64 bit integer*

</p>

The compact_point_p2 block is the same as point_p2 but without any inner tag.

<p>

x_coordinate

y_coordinate

z_coordinate

intensity \\ *integer between 0 and 65535*

return_number \\ *integer between 0 and 7*

return_count \\ *integer between 0 and 7*

scan_direction \\ *integer between 0 and 1*

flight_line_edge \\ *integer between 0 and 1*

classification \\ *integer between 0 and 255*

scan_rank_angle \\ *integer between -128 and 127*

user_data \\ *integer between 0 and 255*

point_source_id \\ *integer between 0 and 65535*

las_colour \\ *64 bit integer*

</p>

The point_p3 block is the same as point_p1 but with a colour (64bit integer) at the end.

<p>

<x> *x_coordinate* <\x>

<y> *y_coordinate* <\y>

<z> *z_coordinate* <\z>

<i> *intensity* <\i>	*\\ integer between 0 and 65535*
<rn> *return_number* <\rn>	*\\ integer between 0 and 7*
<rc> *return_count* <\rc>	*\\ integer between 0 and 7*
<sd> *scan_direction* <\sd>	*\\ integer between 0 and 1*
<fe> *flight_line_edge* <\fe>	*\\ integer between 0 and 1*
<cl> *classification* <\cl>	*\\ integer between 0 and 255*
<sr> *scan_rank_angle* <\sr>	*\\ integer between -128 and 127*
<ud> *user_data* <\ud>	*\\ integer between 0 and 255*
<id> *point_source_id* <\id>	*\\ integer between 0 and 65535*
<t> *gps_time* <\t>	*\\ real number*
<c> *las_colour* <\c>	*\\ 64 bit integer*

</p>

The compact_point_p3 block is the same as point_p3 but without any inner tag.

<p>

x_coordinate

y_coordinate

z_coordinate

intensity	*\\ integer between 0 and 65535*
return_number	*\\ integer between 0 and 7*
return_count	*\\ integer between 0 and 7*
scan_direction	*\\ integer between 0 and 1*
flight_line_edge	*\\ integer between 0 and 1*
classification	*\\ integer between 0 and 255*
scan_rank_angle	*\\ integer between -128 and 127*
user_data	*\\ integer between 0 and 255*
point_source_id	*\\ integer between 0 and 65535*
gps_time	*\\ real number*
las_colour	*\\ 64 bit integer*

</p>

The point_p4 block is the same as point_p1 but with a wave data at the end (not yet implemented).

The compact_point_p4 block is the same as point_p4 but without any inner tag.

The point_p5 block is the same as point_p3 but with a wave data at the end (not yet implemented).

The compact_point_p5 block is the same as point_p5 but without any inner tag.

The point_p6 block is.

<p>

<x> x_coordinate <\x>

<y> y_coordinate <\y>

<z> z_coordinate <\z>

<i> intensity <\i>	*\\ integer between 0 and 65535*
<rn> return_number <\rn>	*\\ integer between 0 and 15*
<rc> return_count <\rc>	*\\ integer between 0 and 15*
<cf> classification_flags <\cf>	*\\ integer between 0 and 15*
<sc> scanner_channel <\sc>	*\\ integer between 0 and 3*
<sd> scan_direction <\sd>	*\\ integer between 0 and 1*
<fe> flight_line_edge <\fe>	*\\ integer between 0 and 1*
<cl> classification <\cl>	*\\ integer between 0 and 255*
<ud> user_data <\ud>	*\\ integer between 0 and 255*
<sr> scan_rank_angle <\sr>	*\\ integer between -128 and 127*
<id> point_source_id <\id>	*\\ integer between 0 and 65535*
<t> gps_time <\t>	*\\ real number*

</p>

The compact_point_p6 block is the same as point_p6 but without any inner tag.

<p>

x_coordinate

y_coordinate

z_coordinate

intensity	*\\ integer between 0 and 65535*
return_number	*\\ integer between 0 and 15*
return_count	*\\ integer between 0 and 15*
classification_flags	*\\ integer between 0 and 15*
scanner_channel	*\\ integer between 0 and 3*
scan_direction	*\\ integer between 0 and 1*
flight_line_edge	*\\ integer between 0 and 1*
classification	*\\ integer between 0 and 255*
user_data	*\\ integer between 0 and 255*
scan_rank_angle	*\\ integer between -128 and 127*
point_source_id	*\\ integer between 0 and 65535*
gps_time	*\\ real number*

</p>

The point_p7 block is the same with point_p6 with a las colour (64bit integer) at the end.

<p>

<x> x_coordinate <\x>

<y> y_coordinate <\y>

<z> z_coordinate <\z>	
<i> intensity <\i>	*\\ integer between 0 and 65535*
<rn> return_number <\rn>	*\\ integer between 0 and 15*
<rc> return_count <\rc>	*\\ integer between 0 and 15*
<cf> classification_flags <\cf>	*\\ integer between 0 and 15*
<sc> scanner_channel <\sc>	*\\ integer between 0 and 3*
<sd> scan_direction <\sd>	*\\ integer between 0 and 1*
<fe> flight_line_edge <\fe>	*\\ integer between 0 and 1*
<cl> classification <\cl>	*\\ integer between 0 and 255*
<ud> user_data <\ud>	*\\ integer between 0 and 255*
<sr> scan_rank_angle <\sr>	*\\ integer between -128 and 127*
<id> point_source_id <\id>	*\\ integer between 0 and 65535*
<t> gps_time <\t>	*\\ real number*
<c> las_colour <\c>	*\\ 64bit integer*

</p>

The compact_point_p7 block is the same as point_p7 but without any inner tag.

<p>

x_coordinate	
y_coordinate	
z_coordinate	
intensity	*\\ integer between 0 and 65535*
return_number	*\\ integer between 0 and 15*
return_count	*\\ integer between 0 and 15*
classification_flags	*\\ integer between 0 and 15*
scanner_channel	*\\ integer between 0 and 3*
scan_direction	*\\ integer between 0 and 1*
flight_line_edge	*\\ integer between 0 and 1*
classification	*\\ integer between 0 and 255*
user_data	*\\ integer between 0 and 255*
scan_rank_angle	*\\ integer between -128 and 127*
point_source_id	*\\ integer between 0 and 65535*
gps_time	*\\ real number*
las_colour	*\\ 64bit integer*

</p>

The point_p8 block is the same with point_p7 with a near infrared (integer between 0 and 255) at the end.

<p>

<x> x_coordinate <\x>

<y> y_coordinate <\y>

```
    <z> z_coordinate <\z>
    <i> intensity <\i>                          \\ integer between 0 and 65535
    <rn> return_number <\rn>                    \\ integer between 0 and 15
    <rc> return_count <\rc>                     \\ integer between 0 and 15
    <cf> classification_flags <\cf>             \\ integer between 0 and 15
    <sc> scanner_channel <\sc>                  \\ integer between 0 and 3
    <sd> scan_direction <\sd>                   \\ integer between 0 and 1
    <fe> flight_line_edge <\fe>                 \\ integer between 0 and 1
    <cl> classification <\cl>                   \\ integer between 0 and 255
    <ud> user_data <\ud>                        \\ integer between 0 and 255
    <sr> scan_rank_angle <\sr>                  \\ integer between -128 and 127
    <id> point_source_id <\id>                  \\ integer between 0 and 65535
    <t> gps_time <\t>                           \\ real number
    <c> las_colour <\c>                         \\ 64bit integer
    <ir> near_infrared <\ir>                    \\ integer between 0 and 255
</p>
```

The compact_point_p8 block is the same as point_p8 but without any inner tag.

```
<p>
    x_coordinate
    y_coordinate
    z_coordinate
    intensity                                   \\ integer between 0 and 65535
    return_number                               \\ integer between 0 and 15
    return_count                                \\ integer between 0 and 15
    classification_flags                        \\ integer between 0 and 15
    scanner_channel                             \\ integer between 0 and 3
    scan_direction                              \\ integer between 0 and 1
    flight_line_edge                            \\ integer between 0 and 1
    classification                              \\ integer between 0 and 255
    user_data                                   \\ integer between 0 and 255
    scan_rank_angle                             \\ integer between -128 and 127
    point_source_id                             \\ integer between 0 and 65535
    gps_time                                    \\ real number
    las_colour                                  \\ 64bit integer
    near_infrared                               \\ integer between 0 and 255
</p>
```

The point_p9 block is the same as point_p6 but with a wave data at the end (not yet implemented).

The compact_point_p9 block is the same as point_p9 but without any inner tag.

The point_p10 block is the same as point_p8 but with a wave data at the end (not yet implemented).

The compact_point_p10 block is the same as point_p10 but without any inner tag.

The ref_data block contains:

<ref_data>

category_block // same as category in data block

<file_name> las_ref_file_name </file_name>

range_block // same as range in data block

</ref_data>

Return to 1.6 Elements Contained in Models or 1 12d XML File Format