18 Intergraph Standard File Formats (Element Structure)

The Intergraph Standard File Formats (ISFF) are the file formats common to MicroStation and Intergraph's Interactive Graphics Design System (IGDS). ISFF is now available to the public. This enables Intergraph customers and third-party developers to create custom applications for MicroStation that read and write ISFF format without a license from Intergaph.

Types of Files

ISFF consists of several types of binary files:

- **Design files** are sequential, variable-length files with variable-length records for the Design File Header (see page 18-2), file set-up information, graphic elements, and non-graphic data.

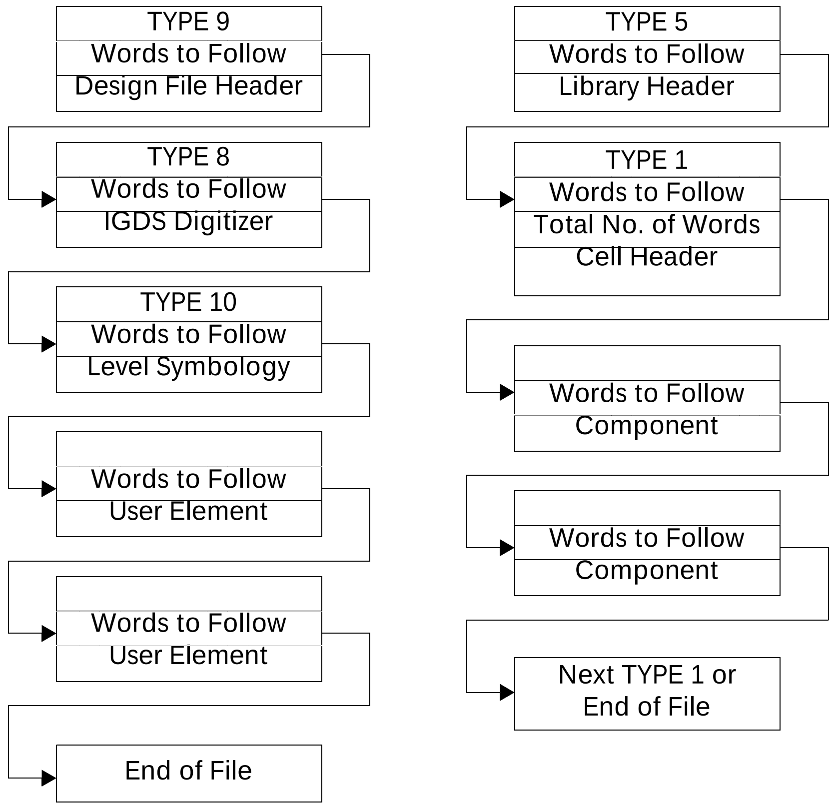
  User-defined elements begin with the fourth element.

  Design files are typically designated with the extension ".dgn."

- **Cell libraries** store cell definitions for placement in design files. A cell library consists of a file header (type 5) element followed by individual cell descriptions. Each cell is a complex element that contains a cell library header (type 1) element and component elements.

  Cell descriptions can be nested. Nested cells contain a type 2 header and component elements. If the cell library already contains the nested cell, its component elements are not repeated.

  Cell libraries are typically designated with the extension ".cel."



Design file (left) and cell library (right) structure.

Design File Header

The first three elements of the design file are called the **design file header**.

Type:	**Stores:**
8. Digitizer setup	Used only by IGDS; it is ignored by MicroStation.
9. Design file settings	Settings that are saved when FILEDESIGN is executed (File menu/Save Settings)..
10. Level symbology	The symbology (color, line style, and line weight) that elements on a level display with in a view for which Level Symbology is on.

Primitive and complex elements

Primitive elements

The primitive elements are lines, line strings, shapes, ellipses, arcs, text, and cones.

Complex elements

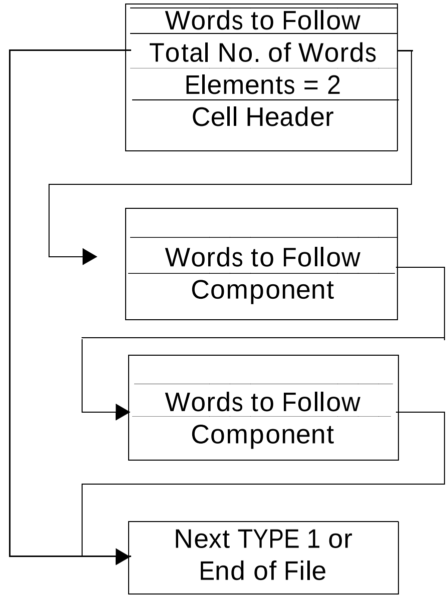
A **complex element** is a set of elements that logically forms a single entity.

Complex elements are stored in the design file as a header followed by the component elements. Text nodes, complex chains, and complex shapes are stored in the design file as shown in the illustration at right.

Other complex elements are cells, surfaces, solids, and B-splines. See the sections about those elements for information about how their headers and component elements are arranged in the design file.

The complex element header contains information about the entire set of elements, including the number of component elements. Word 19 of the header contains the total length in words of the component elements plus the number of words following word 19 in the header.

The maximum combined length of the header and all component elements cannot be greater than 65535 words.



Complex element structure

Element Representation

This appendix shows the formats for ISFF elements. Each figure shows the components of the element, the member names for each structure, and the word (1 word = 16 bits) offsets for each member. The figure represents the element **as it appears in the design file** and its internal representation on the VAX, PC (DOS), and Macintosh. This is the only figure that is important to most programmers.

✍ The in-memory format of elements on the Intergraph CLIPPER, Sun SPARC, and Hewlett Packard HP700 differs slightly from the figures in this appendix. Long integers always start on even word boundaries, and double-precision, floating point values always start on four-word boundaries in this format.

Byte ordering

Computers differ in their storage **byte order**, that is they differ in which byte they consider to be the first of a longer piece such a short or long integer.

Ordering:	**Example systems:**	**Address of long (32-bit) integer is address of:**	**Illustration:**
Big endian (left-to-right)	SPARC, HP700, Macintosh, SGI, and IBM RS6000	High-order byte	3 | 2 | 1 | 0
Little endian (right-to-left)	80x86-based PCs, CLIPPER, and DEC Alpha	Low-order byte	0 | 1 | 2 | 3

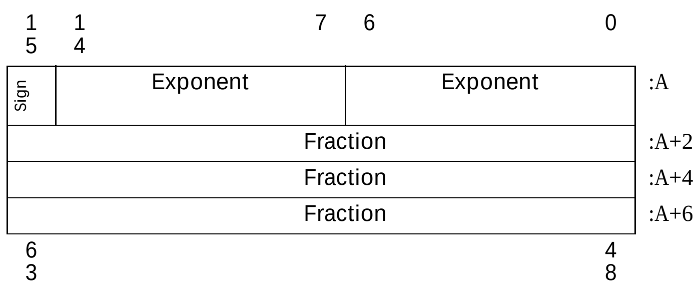
In design files:

- Short integers are stored with little-endian ordering.
- Long integers are stored with **middle-endian** byte ordering (as on the PDP-11).

2	3	0	1

Floating-point values

All floating-point variables are stored in the design file in VAX D-Float format. MDL and MicroCSL automatically convert floating-point variables to the native format of the CPU in use. Bits are labeled from the right, 0–63.



Elements not described in this appendix

These element types are not described in this appendix. They are not supported by IGDS and versions of MicroStation prior to Version 4.0. They cannot be manipulated directly and must be accessed with MDL built-in functions.

- 33. Dimension
- 34. Shared Cell Definition
- 35. Shared Cell Instance
- 36. Multi-line

All of these elements begin with the standard element header and display header. Type 34 is a complex element in which the total length of the definition is given in the word following the display header.

Common Element Parameters

The parameters that are common to one or more elements are explained here.

Element header

The first 18 words of an element in the design file are its fixed header — containing the element type, level, words to follow, and range information. The C declaration for this header is as follows:

```
typedef struct
     {
     unsigned          level:6;          /* level element is on
*/
     unsigned          :1;               /* reserved */
     unsigned          complex:1;        /* component of complex
elem.*/
     unsigned          type:7;           /* type of element */
     unsigned          deleted:1;        /* set if element is
deleted */
     unsigned short    words;            /* words to follow in
element */
     unsigned long     xlow;             /* element range - low
*/
     unsigned long     ylow;
     unsigned long     zlow;
```

```
        unsigned long       xhigh;              /* element range - high
*/
        unsigned long       yhigh;
        unsigned long       zhigh;
        } Elm_hdr;
```

In addition, the next several components of all displayable elements are identical. This additional header is defined as follows:

```
    typedef struct
        {
        unsigned  short grphgrp;            /* graphic group number */
        short     attindx;                  /* words between this and
                                                   attribute linkage
*/
        union
            {
            shorts;
            struct
                {
                unsigned       class:4;       /* class */
                unsigned       l:1;           /* locked */
                unsigned       n:1;           /* new */
                unsigned       m:1;           /* modified */
                unsigned       a:1;           /* attributes present */
                unsigned       r:1;           /* view independent */
                unsigned       p:1;           /* planar */
                unsigned       s:1;           /* 1=nonsnappable */
                unsigned       h:1;           /* hole/solid (usually)
*/
                } b;
            } props;
        union
        {
            short     s;
            Symbologyb;
            } symb;
        } Disp_hdr;
```

Here, `Symbology` is defined as:

```
    typedef struct
        {
        unsigned       style:3;             /* line style */
        unsigned       weight:5;            /* line weight */
        unsigned       color:8;             /* color */
        } Symbology;        /* element symbology word 652 */
```

Element type and level

The first word in the header defines the element’s type and level.

U	Type							C	R	Level					

The fields in the first word are:

U	clear if element is active; set if the element is deleted
Type	number that denotes the element’s type
C	set if the element is part of a complex element; otherwise clear
R	reserved (equals zero)
Level	number that indicates the element’s level (0-63)

Words to follow

Word 2 of the element header indicates the number of words in the element excluding words 1 and 2; that is the word count to the next element in the design file (commonly referred to as “words to follow” or “WTF”).

For complex elements, this defines the length of the header element only and does not include component elements.

Range

Words 3–14 of the element header contain the six long (double precision) integers that define the element’s range — its low and high x, y, and z coordinates in absolute units of resolution (UOR).

All points in an element must be completely contained in the design plane.

Graphic group number

Word 15 contains the element’s graphic group number. If zero, the element is not in a graphic group. If non-zero, the element is in a graphic group with all other elements that have the same graphic group number.

Index to attribute linkage

Word 16 defines the number of words existing between (and excluding) word 16 and the first word of the attribute data. Attribute data is optional and may or may not be present.

Properties indicator

Word 17 describes the element's properties:

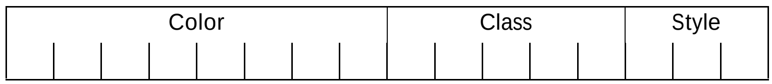
H	S	P	R	A	M	N	L	Reserved				Class			

Bit	Indicates:
H	For closed element types (shape, complex shape, ellipse, cone, B-spline surface header, and closed B-spline curve header), the H-bit indicates whether the element is a solid or a hole. • 0 = Solid • 1 = Hole For a cell header (type 2), if the H-bit is: • 0 = Header for a cell • 1 = Header for an orphan cell (created by GROUP SELECTION or application) For a line, if the H-bit is: • 0 = Line segment • 1 = Infinite-length line For a point string element, if the H-bit is: • 0 = Continuous • 1 = Disjoint The H-bit has no meaning in other elements.
S	Whether the element is snappable. • 0 = Snappable • 1 = Not snappable
P	If a surface is planar or non-planar. • 0 = Planar • 1 = non-planar
R	Element orientation. • 0 = Oriented relative to design file • 1 = Oriented relative to screen
A	Whether attribute data is present. • 0 = Attribute data not present • 1 = Attribute data present
M	Whether the element has been graphically modified. • 0 = Not modified • 1 = Has been modified
N	Whether the element is new. • 0 = Not new • 1 = New (set to 1 when the element is placed)
L	Whether the element is locked. • 0 = Not locked • 1 = Locked

Bit	Indicates:
4–7	Reserved.
Class	Represented as follows: 0. Primary; 1. Pattern component; 2. Construction element; 3. Dimension element; 4. Primary rule element; 5. Linear patterned element; 6. Construction rule element.

Element symbology

Word 18 defines the element's symbology (color, line style, and line weight).



Color	Number (0-255) that indicates the element's color
Weight	Number (0-31) that indicates line weight
Style	The line style is represented as follows: • 0. Solid (SOL) • 1. Dotted (DOT) • 2. Medium dashed (MEDD) • 3. Long-dashed (LNGD) • 4. Dot-dashed (DOTD) • 5. Short-dashed (SHD) • 6. Dash double-dot (DADD) • 7. Long dash-short dash (LDSD)

Point coordinates

MicroStation is based on a 32-bit integer design plane. Point coordinates are specified as two or three long integers (for 2D and 3D design files, respectively). Coordinate definitions are assigned by the following C structures:

2D

```
typedef struct
     {
     long      x;
     long      y;
     } Point2d;
```

3D

```
typedef struct
     {
     long      x;
     long      y;
     long      z;
     } Point3d;
```

Sometimes a point that is not within the design plane needs to be specified. For example, the center point for an arc may be far from the design plane, although the design plane must completely contain the arc. In these cases, points are specified as two or three double-precision (64-bit), floating point values:

2D

```
typedef struct
     {
     double    x;
     double    y;
     } Dpoint2d;
```

3D

```
typedef struct
     {
     double    x;
     double    y;
     double    z;
     } Dpoint3d;
```

Rotation angle (2D) and quaternion (3D)

In 2D design files, rotation is represented by a value, `angle`, that is counterclockwise from the X-axis. `Angle` is a long integer with the lower-order bit equal to .01 seconds. The conversion from `angle` to degrees is expressed as follows:

- Degrees = $^{Angle}/_{360000}$

In 3D design files, an element's orientation is represented by the transformation matrix to design file coordinates. These transformations are stored in a compressed format called **quaternions**. Quaternions store a 3×3 ortho-normal transformation matrix as four values rather than nine.

The `mdlRMatrix_toQuat` function (MDL) and the `trans_to_quat` routine (MicroCSL) generate a quaternion from a transformation matrix. The `mdlRMatrix_fromQuat` function (MDL) and the `quat_to_trans` routine (MicroCSL) generate a transformation matrix from a quaternion. See the documentation for these functions for details.

Attribute linkage data

Any element can optionally contain auxiliary data commonly referred to as **attribute data** or **attribute linkage data**. This data can consist of a link to an associated database or any other information that pertains to the element.

Attribute data that is not associated with DMRS or a MicroStation-supported database such as Oracle is referred to as a **user linkage**. A user linkage can co-exist with a database linkage or other user linkages. MicroStation does not attempt to interpret user linkages; these linkages are, however, maintained when MicroStation modifies an element. When an element with a user linkage is copied, the linkage is also copied. Therefore, multiple linkages can occur.

The format of user linkages is described below. As with other linkages, when user linkages are present, the A-bit must be set in the properties word. Individual user linkages cannot exceed 256 words. Multiple user linkages can be attached to an element. The combined length of an element and its linkages must not be greater than 768 words. Considering worst-case element lengths, the length of the linkage area should not exceed 140 words.

User linkages consist of a header word, a user ID word, and user-defined data. The
U-bit in the linkage header is set to indicate that the linkage is a user linkage. The ID word should be unique to the software package to which the linkage applies.

Level Symbology (Type 10)

Stores the symbology (color, line style, and line weight) that elements on a level display with in a view for which Level Symbology is on.

The values of the range are zero.

If the high bit in the next-to-last word of the range is set, then the low three bits are flags for selectively using the three components of the level symbology words.

- If bit 0 is clear, then use style (line code).
- If bit 1 is set, then use line weight.
- If bit 2 is set, then use color.

If the high bit in the next-to-last word of the range is clear, the color, line weight, and line style are used.

The format of each level symbology word is the same as that for Element symbology (see page 18-10).

Library Cell Header (Type 1)

Library cell header elements contain information needed to create a cell in a design file. They are found only in cell libraries.

The `celltype` member indicates the following types of cells:

0. Graphic cell
1. Command menu cell
2. Cursor button menu cell
3. Function key menu cell (not supported by MicroStation)
4. Matrix menu cell

5. Tutorial cell
6. Voice menu cell (not supported by MicroStation)

The C definition is as follows:

```
typedef struct
    {
    Elm_hdr             ehdr;               /* element header */
    short               celltype;           /* cell type */
    short               attindx;            /* attribute linkage */
    long                name;               /* Radix-50 cell name */
    unsigned short      numwords;           /* # of words in
description */
    short               properties;         /* properties */
    short               dispsymb;           /* display symbology */
    short               class;              /* cell class (always 0)
*/
    short               levels[4];          /* levels used in cell
*/
    short               descrip[9];         /* cell description */
    } Cell_Lib_Hdr;
```

Cell descriptions in cell libraries

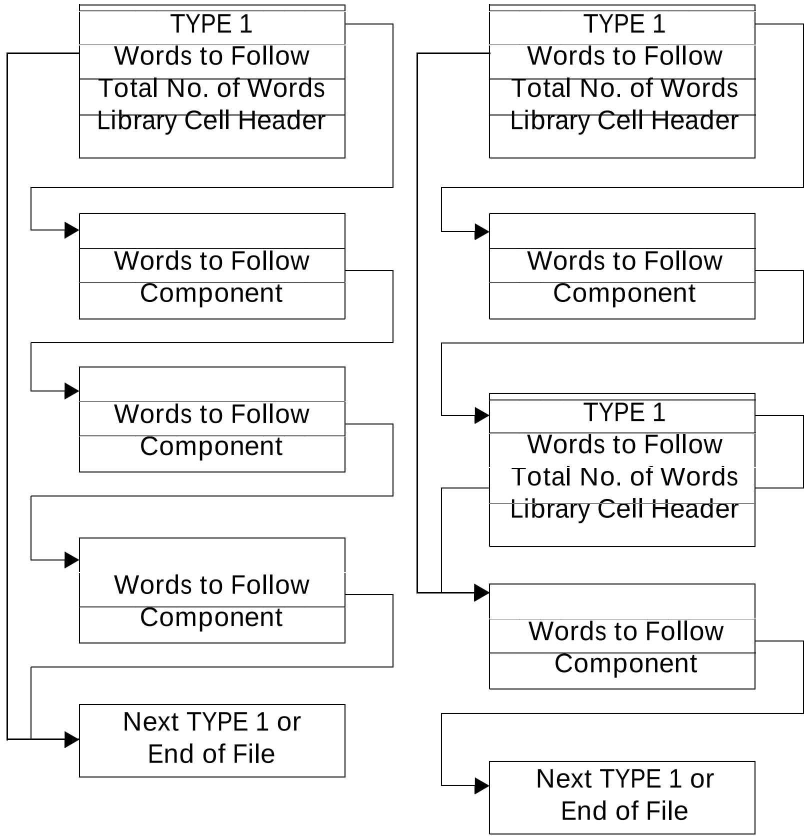
Each cell description in a cell library is a complex element that contains a library cell header (type 1) followed by component graphic elements.

A cell definition can be **nested** — included in another cell. A nested cell definition is stored as a cell header (type 2) that points to a library cell header (type 1). The component elements of a nested cell are not repeated.

When the user places a cell in the design file in IGDS and versions of MicroStation prior to Version 4.0, or as an unshared cell in MicroStation Version 4.0 or later versions, it is placed as a cell header (type 2) followed by its component elements. Each nested cell definition is placed with its cell header (type 2) followed by its component elements.

In MicroStation Version 4.0 or later versions, when the user places the cell in the design file as a shared cell:

- If there is no shared cell definition element for that cell in the design file, one is created.



Arrangement of cells and components in cell library.

Left: Cell with no nested cells.

Right: Cell with nested cell definition.

- If there is a shared cell definition element for that cell in the design file, a shared cell instance element is placed in the design.
- If the cell contains a nested cell and the nested cell is not defined as a shared cell in the design file, a shared cell definition element is created for the nested cell. If the cell contains a nested cell and a shared cell definition is in the design file, a shared cell instance element is created.

✍ Shared cell definition and shared cell instance elements are not described in this appendix. They cannot be manipulated directly and must be accessed with MDL built-in functions.

Cell Header (Type 2)

A cell header element begins:

- A nested cell definition in a cell library.
- A cell placed in a design file in IGDS and versions of MicroStation prior to version 4.0.
- An unshared cell placed in a design file in MicroStation Version 4.0 and later versions.

2D:

```
typdef struct
     {
     Elm_hdr             ehdr;          /* element header */
     Disp_hdr            dhdr;          /* display header */
     unsigned short      totlength;     /* total length of cell */
     long                name;          /* Radix 50 name */
     short               class;         /* class bit map */
     short               levels[4];     /* levels used in cell */
     Point2d             rnglow;        /* range block low */
     Point2d             rnghigh;       /* range block high */
     Trans2d             trans;         /* transformation matrix */
     Point2d             origin;        /* cell origin */
     } Cell_2d;
```

3D:

```
typedef struct
     {
     Elm_hdr             ehdr;          /* element header */
     Disp_hdr            dhdr;          /* display header */
     unsigned short      totlength;     /* total length of cell */
     long                name;          /* Radix 50 name */
     short               class;         /* class bit map */
     short               levels[4];     /* levels used in cell */
     Point3d             rnglow;        /* range block low */
     Point3d             rnghigh;       /* range block high */
     Trans3d             trans;         /* transformation matrix */
     Point3d             origin;        /* cell origin */
     } Cell_3d;
```

Each cell header contains an origin (in design file coordinates) and a transformation matrix that describe all manipulations (rotation and scaling) from the cell library definition to the current design file orientation. The transformation matrix is a 2×2 or 3×3 matrix stored as a long integer with the lower-order bit equal to 4.6566E-6 ($10{,}000 / 2^{31}$).

✍ Shared cells are stored in the design file as shared cell definition and shared cell instance elements. These elements are not described in this appendix. They cannot be manipulated directly and must be accessed with MDL built-in functions.

.

2D (left) and 3D (right) Cell Header

Word Offset	2D Cell Header	
0-17	Header	
18	Words in Description	cell_2d.totlength
19-20	Cell Name	cell_2d.name
21	Class Bit Map	cell_2d.class
22-25	Level Indicators	cell_2d.levels
26-27	Range Block Diagonal X1	cell_2d.rnglow
28-29	Y1	
30-31	X2	cell_2d.rnghigh
32-33	Y2	
34-35	Transformation Matrix T11	cell_2d.trans
36-37	T12	
38-39	T21	
40-41	T22	
42-43	X Origin	cell_2d.origin
44-45	Y Origin	
46	Attribute Linkage	

Word Offset	3D Cell Header	
0-17	Header	
18	Words in Description	cell_3d.totlength
19-20	Cell Name	cell_3d.name
21	Class Bit Map	cell_3d.class
22-25	Level Indicators	cell_3d.levels
26-27	Range Block Diagonal X1	cell_3d.rnglow
28-29	Y1	
30-31	Z1	
32-33	X2	cell_3d.rnghigh
34-35	Y2	
36-37	Z2	
38-39	Transformation Matrix T11	cell_3d.trans
40-41	T12	
42-43	T13	
44-45	T21	
46-47	T22	
48-49	T23	
50-51	T31	
52-53	T32	
54-55	T33	
56-57	X Origin	cell_3d.origin
58-59	Y Origin	
60-61	Z Origin	
62	Attribute Linkage	

Line Elements (Type 3)

Line elements consist of the header information and design plane coordinates of the line endpoints.

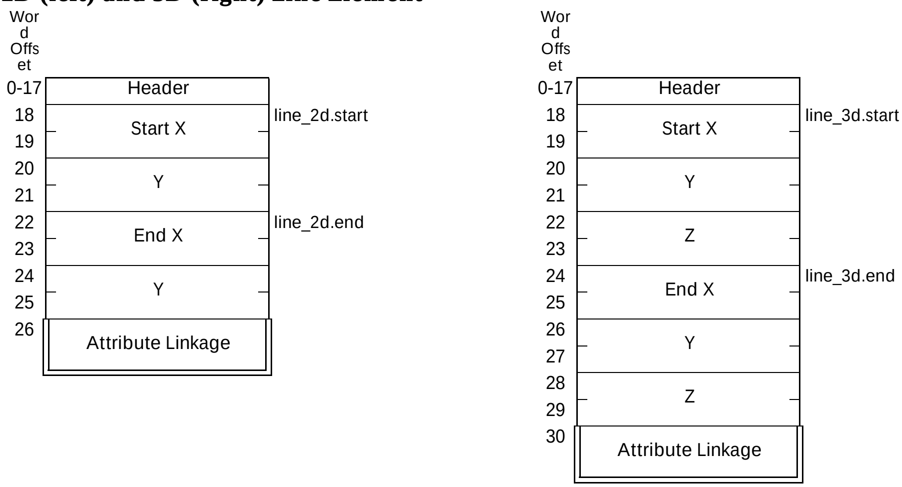
2D:

```
typedef struct
    {
    Elm_hdr        ehdr;      /* element header */
    Disp_hdr       dhdr;      /* display header */
    Point2d        start;     /* starting point */
    Point2d        end;       /* ending point */
    } Line_2d;
```

3D:

```
typedef struct
    {
    Elm_hdr        ehdr;      /* element header */
    Disp_hdr       dhdr;      /* display header */
    Point3d        start;     /* starting point */
    Point3d        end;       /* ending point */
    } Line_3d;
```

2D (left) and 3D (right) Line Element



Line String (Type 4), Shape (Type 6), Curve (Type 11), and B-spline Pole Element (Type 21)

Line string, shape, curve, and B-spline pole elements are represented similarly in the design file. The header information is followed by the number of vertices and then the coordinates of each vertex. A maximum of 101 vertices can be in an element of these types. In a shape, the coordinates of the last vertex must be the same as the first vertex. For curves, two extra points at the beginning and end of the vertex list establish the curvature at the ends. Thus, a curve can have just 97 user-defined points.

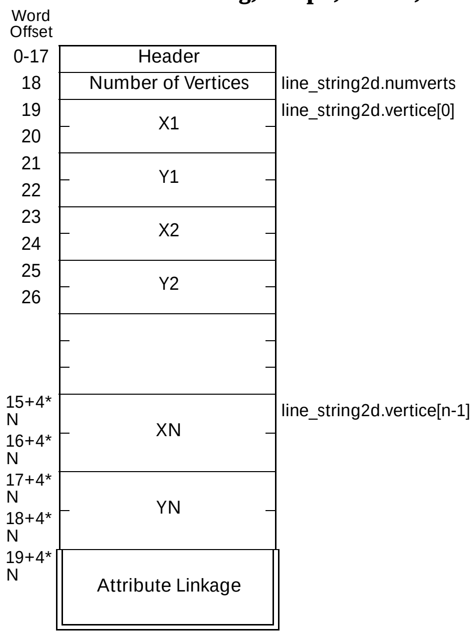
2D:

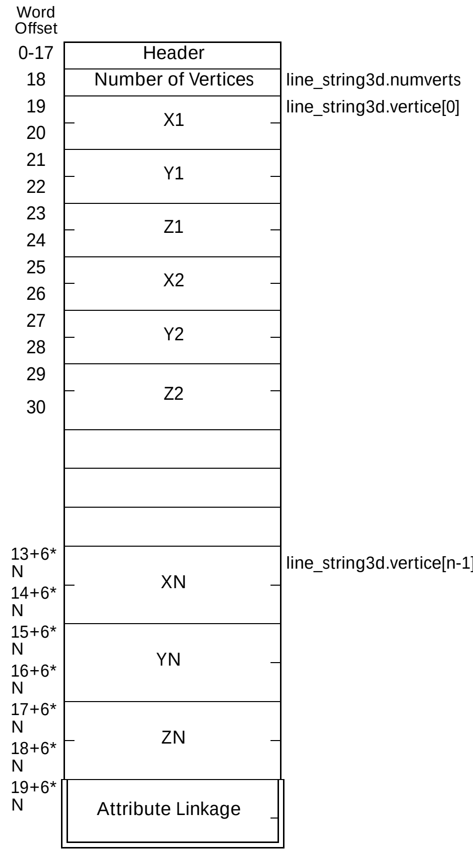
```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     short          numverts;      /* number of vertices */
     Point2d        vertice[1];    /* points */
     } Line_String_2d;
```

3D:

```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     short          numverts;      /* number of vertices */
     Point3d        vertice[1];    /* points */
     } Line_String_3d;
```

2D and 3D Line String, Shape, Curve, and B-spline Pole Elements





The curve (type 11) element is a 2D or 3D parametric spline curve completely defined by a set of n points. The first two and last two points define endpoint derivatives and do not display. The interpolated curve passes through all other points.

A curve with n points defines n-1 line segments; interpolation occurs over the middle n-5 segments. Each segment has its own parametric cubic interpolation polynomial for the x and y (and z in 3D) dimensions. The parameter for each of these polynomials is the length along the line segment. Thus, for a segment k, the interpolated points P are expressed as a function of the distance d along the segment as follows:

$$P_k(d) = \{F_{k,x}(d),\ F_{k,y}(d),\ F_{k,z}(d)\} \text{ with } 0 \le d \le D_k$$

$F_{k,x}$, $F_{k,y}$, and $F_{k,z}$ are cubic polynomials and D_k is the length of segment k. In addition, the polynomial coefficients are functions of the segment length and the endpoint derivatives of $F_{k,x}$, $F_{k,y}$, and $F_{k,z}$. The subscript $_k$ is merely a reminder that these functions depend on the segment.

The cubic polynomials are defined as follows:

$$F_{k,x} = a_x d^3 + b_x d^2 + c_x d + X_k$$

$$c_x = t_k$$

$$b_x = [3(X_k+1-X_k)/D_k - 2t_{k,x} - t_{k+1,x}] / D_k$$

$$a_x = [t_{k,x} + t_{k+1,x} - 2(x_{k+1}-x_k)/D_k] / D_k^2$$

The m variable is analogous to the slope of the segment.

If $(|m_{k+1,x} - m_{k,x}| + |m_{k-1,x} - m_{k-2,x}|) \neq 0$, then:

$$t_{k,x} = (m_{k-1,x}|m_{k+1,x}-m_{k,x}| + m_{k,x}|m_{k-1,x}-m_{k-2,x}|)/(|m_{k+1,x}-m_{k,x}| + |m_{k-1,x}-m_{k-2,x}|)$$

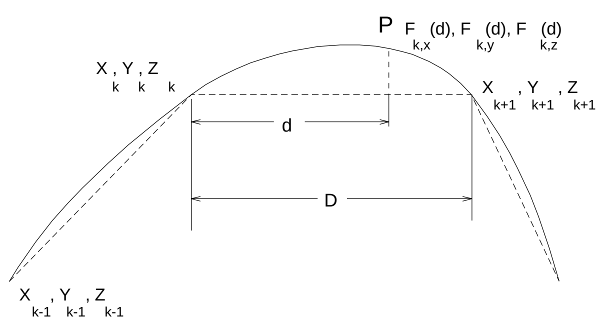
else:

$$t_{k,x} = (m_{k+1,x}+m_{k,x}) / 2$$

$$m_{k,x} = (X_{k+1} - X_k) / D_k$$

$F_{k,y}(d)$ and $F_{k,z}(d)$ are defined analogously.

Curve Definition



Text Node Header (Type 7)

Text node header elements are complex headers for groups of text elements, specifying the number of text strings, the line spacing between text strings, the origin of the text node, the node number, and the maximum number of characters in each text string.

2D:

```
typedef struct
     {
     Elm_hdr             ehdr;          /* element header */
     Disp_hdr            dhdr;          /* display header */
     unsigned short      totwords;      /* total words following */
     short               numstrngs;     /* # of text strings */
     short               nodenumber;    /* text node number */
     byte                maxlngth;      /* maximum length allowed */
     byte                maxused;       /* maximum length used */
     byte                font;          /* text font used */
     byte                just;          /* justification type */
     long                linespc;       /* line spacing */
     long                lngthmult;     /* length multiplier */
     long                hghtmult;      /* height multiplier */
     long                rotation;      /* rotation angle */
     Point2d             origin;        /* origin */
     } Text_node_2d;
```

3D:

```
typedef struct
     {
     Elm_hdr             ehdr;          /* element header */
     Disp_hdr            dhdr;          /* display header */
     unsigned short      totwords;      /* total words following */
     short               numstrngs;     /* # of text strings */
     short               nodenumber;    /* text node number */
     byte                maxlngth;      /* maximum length allowed */
     byte                maxused;       /* maximum length used */
     byte                font;          /* text font used */
     byte                just;          /* justification type */
     long                linespc;       /* line spacing */
     long                lngthmult;     /* length multiplier */
     long                hghtmult;      /* height multiplier */
     long                quat[4];       /* quaternion rotations */
     Point3d             origin;        /* origin */
     } Text_node_3d;
```

Text node number

Each text node is assigned a unique number (`nodenumber`). This number is displayed at the node origin when node display is on. Applications can use it to uniquely identify the node.

Line length

The user specifies the maximum number of characters (`maxlngth`) in a line of text when the node is created. The maximum used (`maxused`) line length indicates the number of characters currently in the longest text line.

Justification and origin

The justification defines the position of text strings relative to the origin. The origin retained in the design file is the true, user-defined origin. The following justifications are possible:

Left/Top (0)	Center/Top (6)	Right margin/Top (9)
Left/Center (1)	Center/Center (7)	Right margin/Center (10)
Left/Bottom (2)	Center/Bottom (8)	Right margin/Bottom (11)
Left margin/Top (3)		Right/Top (12)
Left margin/Center (4)		Right/Center (13)
Left margin/Bottom (5)		Right/Bottom (14)

Line spacing

This long integer indicates the number of UORs from the bottom of a text string to the top of the next string.

2D and 3D Text Node Headers

Word Offset	Field		Variable
0-17	Header		
18	Words in Description		text_node2d.totwords
19	# Text Strings		text_node2d.numstrings
20	Text Node Number		text_node2d.nodenumber
21	Max Used	Max Allowed	text_node2d.maxlngth
22	Justification	Font	text_node2d.font
23	Line Spacing		text_node2d.linespc
24			
25	Length Multiplier		text_node2d.lngthmult
26			
27	Height Multiplier		text_node2d.hghtmult
28			
29	Rotation Angle		text_node2d.rotation
30			
31	X Origin		text_node2d.origin
32			
33	Y Origin		
34			
35	Attribute Linkage		

Word Offset	Field		Variable
0-17	Header		
18	Words in Description		text_node3d.totwords
19	# Text Strings		text_node3d.numstrings
20	Text Node Number		text_node3d.nodenumber
21	Max Used	Max Allowed	text_node3d.maxlngth
22	Justification	Font	text_node3d.font
23	Line Spacing		text_node3d.linespc
24			
25	Length Multiplier		text_node3d.lngthmult
26			
27	Height Multiplier		text_node3d.hghtmult
28			
29	Rotation Quaternion Q4		text_node3d.quat
30			
31	Q1		
32			
33	Q2		
34			
35	Q3		
36			
37	X Origin		text_node3d.origin
38			
39	Y Origin		
40			
41	Z Origin		
42			
43	Attribute Linkage		

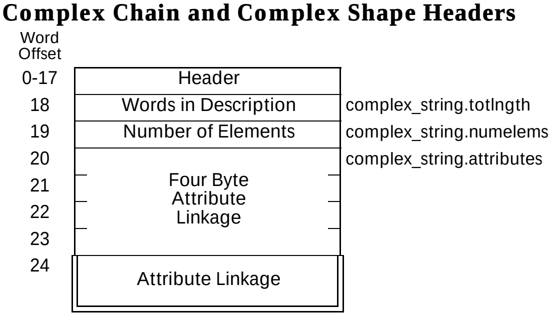
Complex Chain Headers (Type 12) and Complex Shape Headers (Type 14)

Complex chains (open) and complex shapes (closed) are complex elements formed from a series of elements (lines, line strings, arcs, curves, and open B-Spline curves). A complex chain or complex shape consists of a header followed by its component elements. These structure of the header is identical for both complex chains and complex shapes in 2D and 3D files. The element is a complex shape if the endpoints of the first and last component elements are the same.

```
typedef struct
      {
      Elm_hdr             ehdr;               /* element header */
      Disp_hdr            dhdr;               /* display header */
      unsigned short      totlength;          /* total length of surface */
      unsigned short      numelems;           /* # of elements in surface
*/
      short               attributes[4];      /* to reach min. element size
*/
      } Complex_string;
```

Four words of attribute data are included in complex chains and shapes to ensure that they are at least 24 words long, which is the minimum element length required for some Intergraph file processors.

Complex Chain and Complex Shape Headers



Ellipse Elements (Type 15)

Ellipse elements are specified with a center, rotation angle, and major and minor axes. A circle is an ellipse with the major and minor axes equal. The ellipse element is defined in C as follows:

2D:

```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     double         primary;       /* primary axis */
     double         secondary;     /* secondary axis */
     long           rotation;      /* rotation angle */
     Dpoint2d       origin;        /* origin */
     } Ellipse_2d;
```

3D:

```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     double         primary;       /* primary axis */
     double         secondary;     /* secondary axis */
     long           quat[4];       /* quaternion rotations */
     Dpoint3d       origin;             /* origin */
     } Ellipse_3d;
```

Primary and secondary axes

Ellipse axes are defined by two double-precision floating point values that specify the lengths in UORs of the semi-major and semi-minor axes. The primary axis is not necessarily the longest (semi-major) axis, but rather is the axis whose orientation is specified by the rotation angle or quaternion.

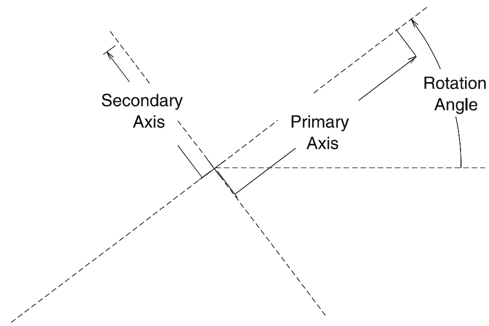
Orientation

The rotation angle or quaternion defines the orientation of the primary axis with respect to the design file coordinate system.

Origin

The origin (center) of the ellipse is expressed as double-precision floating point coordinates.

Ellipse Parameters



2D and 3D Ellipse Element

Word Offset	Field	Name
0-17	Header	
18-21	Primary Axis	ellipse_2d.primary
22-25	Secondary Axis	ellipse_2d.secondary
26-27	Rotation Axis	ellipse_2d.rotation
28-31	X Origin	ellipse_2d.origin
32-35	Y Origin	
36	Attribute Linkage	

Word Offset	Field	Name
0-17	Header	
18-21	Primary Axis	ellipse_3d.primary
22-25	Secondary Axis	ellipse_3d.secondary
26-27	Rotation Quaternion Q4	ellipse_3d.quat
28-29	Q1	
30-31	Q2	
32-33	Q3	
34-37	X Origin	ellipse_3d.origin
38-41	Y Origin	
42-45	Z Origin	
46	Attribute Linkage	

Arc Elements (Type 16)

Arc elements are defined by the center, the rotation, start, and sweep angles, and the major and minor axes. The C structure definitions are as follows:

2D:

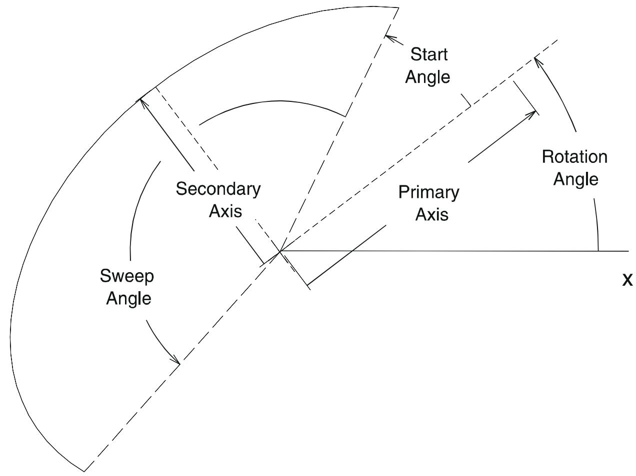
```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     long           startang;      /* start angle */
     long           sweepang;      /* sweep angle */
     double         primary;       /* primary axis */
     double         secondary;     /* secondary axis */
     long           rotation;      /* rotation angle */
     Dpoint2d       origin;        /* origin */
     } Arc_2d;
```

3D:

```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     long           startang;      /* start angle */
     long           sweepang;      /* sweep angle */
     double         primary;       /* primary axis */
     double         secondary;     /* secondary axis */
     long           quat[4];       /* quaternion rotations */
     Dpoint3d       origin;        /* origin */
     } Arc_3d;
```

Arc parameters

Arc parameters



Parameter:	**Description**
Primary and secondary axes	Defined by two double-precision floating point values that specify the lengths in UORs of the semi-major and semi-minor axes. The primary axis is not necessarily the longest (semi-major) axis, but the axis whose orientation is specified by the rotation angle or quaternion.
Orientation	Rotation angle or quaternion defines the orientation of the primary axis with respect to the design file coordinate system.
Origin (center)	Expressed as double-precision floating point coordinates. The center itself need not be within the design plane although the entire arc definition must be within the design plane.
Start angle	Expressed in the same format as a 2D rotation angle. It defines the counterclockwise angle in the plane of the arc from the primary axis to the starting point of the arc on a unit circle.
Sweep angle	Represents the sweep of the arc along a unit circle. It is in the same format as a 2D rotation angle except that the sign bit indicates the direction of sweep, 0=counterclockwise, 1=clockwise. Note that MicroStation interprets the special case of a 0° sweep angle as a 360° sweep angle.

2D and 3D Arc Element

Word Offset	Field	Name
0-17	Header	
18-19	Start Angle	arc_2d.startang
20-21	Sweep Angle	arc_2d.sweepang
22-25	Primary Axis	arc_2d.primary
26-29	Secondary Axis	arc_2d.secondary
30-31	Rotation Angle	arc_2d.rotation
32-35	X Origin	arc_2d.origin
36-39	Y Origin	
40	Attribute Linkage	

Word Offset	Field	Name
0-17	Header	
18-19	Start Angle	arc_3d.startang
20-21	Sweep Angle	arc_3d.sweepang
22-25	Primary Axis	arc_3d.primary
26-29	Secondary Axis	arc_3d.secondary
30-31	Rotation Quaternion Q4	arc_3d.quat
32-33	Q1	
34-35	Q2	
36-37	Q3	
38-41	X Origin	arc_3d.origin
42-45	Y Origin	
46-49	Z Origin	
50	Attribute Linkage	

Text Elements (Type 17)

A text element stores a single line of text. The C structures are as follows.

2D:

```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     byte           font;          /* text font used */
     byte           just;          /* justification type */
     long           lngthmult;     /* length multiplier */
     long           hghtmult;      /* height multiplier */
     long           rotation;      /* rotation angle */
     Point2d        origin;        /* origin */
     byte           numchars;      /* # of characters */
     byte           edflds;        /* # of enter data fields */
     char           string[1];     /* characters */
     } Text_2d;
```

3D:

```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     byte           font;          /* text font used */
     byte           just;          /* justification type */
     long           lngthmult;     /* length multiplier */
     long           hghtmult;      /* height multiplier */
     long           quat[4];       /* quaternion angle */
     Point3d        origin;        /* origin */
     byte           numchars;      /* # of characters */
     byte           edflds;        /* # of enter data fields */
     char           string[1];     /* characters */
     } Text_3d;
```

These parameters define the text.

Parameter:	Description
Font	A single byte is used to store the font for a text element. This number corresponds to the appropriate font definition in the font library.
Length and height multipliers	The basic character size is 6 UORs wide and 6 UORs high (4 UORs of width and 2 of spacing). The length and height multipliers specify the scale factors to be applied to the basic character size to determine the true size of the text string. The multipliers are stored as long integers with the lower order bit set. Mirrored text is identified by a negative h multiplier. The maximum multiplier value is 2,147,483.648 (2^{31}/1000). The maximum text size is therefore 12,884,898 UORs (6 × 2,147,483.648).
Orientation	The rotation angle or quaternion defines the orientation of a text element relative to the design file coordinate system.
Justification and origin	At the time of placement, the active text justification determines how text is positioned about the user-defined origin. The origin stored in a text element is always the lower left of the text element. It is necessary to use the justification value to compute the user-defined origin. There are nine possible justifications for text elements:
Enter data fields	Areas within a text element that can be easily modified by the user. Each enter data field in a text string is specified by three bytes appended to the element. The first byte specifies the character number in the string (relative to 1) that is the first character in the enter data field. The second byte specifies the number of characters in the field. The third byte defines the justification of the non-blank characters within the field (-1=left, 0=center, +1=right). Note that if the number of characters is odd, the first enter data field specification does not lie on a word boundary, and if there are no enter data fields, there are no specification bytes.

Left/Top (0)	Center/Top (6)	Right/Top (12)
Left/Center(1)	Center/Center (7)	Right/Center (13)
Left/Bottom(2)	Center/Bottom (8)	Right/Bottom (14)

2D and 3D Text Elements

Word Offset	Field	Name
0-17	Header	
18		text_2d.font
19-20	Length Multiplier	text_2d.lngthmult
21-22	Height Multiplier	text_2d.hghtmult
23-24	Rotation Angle	text_2d.rotation
25-26	X Origin	text_2d.origin
27-28	Y Origin	
29	# Ed Fields — # Chars	text_2d.numchars
30	Char 2 — Char 1	text_2d.string[0]
31	Char 4 — Char 3	
...		
30+(N-1)/2	Char N — Char N-1	
31+(N-1)/2	Len Ed #1 — Start Ed #1	
32+(N-1)/2	Start Ed #2 — Just Ed #1	
...		
	Attribute Linkage	

Word Offset	Field	Name
0-17	Header	
18		text_3d.font
19-20	Length Multiplier	text_3d.lngthmult
21-22	Height Multiplier	text_3d.hghtmult
23-24	Rotation Quaternion Q4	text_3d.rotation
25-26	Q1	
27-28	Q2	
29-30	Q3	
31-32	X Origin	text_3d.origin
33-34	Y Origin	
35-36	Z Origin	
37	# Ed Fields — # Chars	text_3d.numchars
38	Char 2 — Char 1	text_3d.string[0]
39	Char 4 — Char 3	
...		
38+(N-1)/2	Char N — Char N-1	
39+(N-1)/2	Len Ed #1 — Start Ed #1	
40+(N-1)/2	Start Ed #2 — Just Ed #1	
...		
	Attribute Linkage	

3D Surface Header (Type 18) and 3D Solid Header (Type 19)

A surface or solid is a complex 3D element that is projected or rotated from a planar boundary element (line, line string, curve, arc, or ellipse). The surface or solid header precedes an ordered set of primitive elements that define boundaries, cross sections and rule lines.

A solid (type 19) is **capped** at both ends — it encloses a volume. A surface (type 18) is not capped on the ends — it encloses no volume. Surface and solid headers are identical except for their type number. The C definition is as follows:

```
     typedef struct
          {
          Elm_hdr             ehdr;               /* element header */
          Disp_hdr            dhdr;               /* display header */
          unsigned short      totlength;          /* total length of
surface */
          unsigned short      numelems;           /* # of elements in
surface */
          byte                surftype;           /* surface type */
          byte                boundelms;          /* # of boundary
elements-1 */
     #ifdefunix
          short               filler
     #endif
          short               attributes[4];      /* unknown attribute
data */
          } Surface;
```

Method of creation

Each surface or solid header has a type number describing its method of creation.

For surfaces, the following values are used.

0=Surface of projection

1=Bounded Plane

2=Bounded Plane

3=Right circular cylinder

4=Right circular cone

5=Tabulated cylinder

6=Tabulated cone

7=Convolute

8=Surface of revolution

9=Warped surface

For solids (capped surfaces), the following values are used.

0=Volume of projection

1=Volume of revolution

2=Volume defined by boundary elements

Solid or Surface Elements

Word Offset		
0-17	Header	
18	Words in Description	surface.totlngth
19	Number of Elements	surface.numelems
20	Surface Type	surface.type
21	Attribute Linkage	

Elements in surfaces and solids

Any line, line string, curve, arc, or ellipse can be a boundary element of a surface or solid. A complex element cannot be a component of a surface or solid. Rule elements are restricted to lines and arcs.

Elements are stored in a surface or solid in a strict order. Boundary elements (class=0) appear first after the surface/solid header. The second boundary element immediately follows the first boundary and is followed by any rule lines connecting the first and second boundary. If additional boundary elements are included they should follow this same pattern with the boundary elements preceding the rule lines that connect it to the previous boundary.

Point String Elements (Type 22)

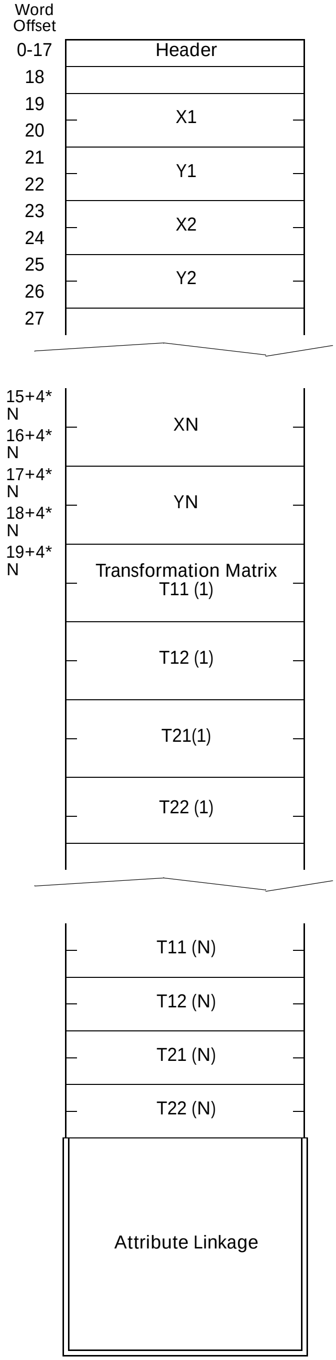
A point string element consists of a number of vertices with orientations defined at each vertex. They are useful in specialized applications that need to specify orientations as well as point locations, such as a “walk through.”

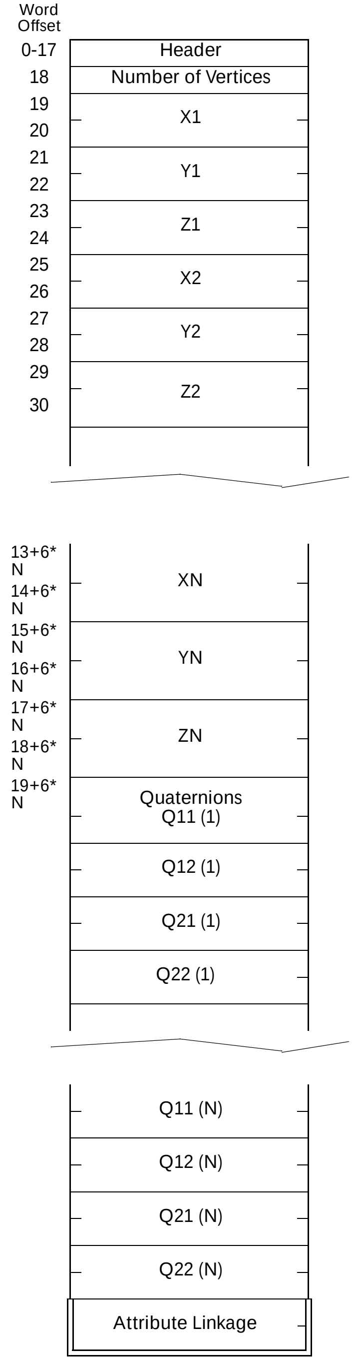
Point strings can be defined as either contiguous or disjoint. Contiguous point strings are displayed with lines connecting the vertices. Disjoint point strings are displayed as a set of discrete points. Both types are placed and manipulated in the same way, but exhibit slightly different characteristics when snapping or locating.

It is impossible to define a point string structure in C because all point locations are stored before any of the orientations.

	Description
Range	The range of the point string element is the range of the points.
Properties	The H-bit (bit 15) of the properties word indicates the type of point string (0 = continuous, 1 = disjoint) for display purposes. The setting of the planar bit indicates whether the points are coplanar.
Number of points	The maximum number of vertices allowed in a single point string is 48. A longer series of points is formed by combining multiple elements in a complex chain.
Point coordinates	An array contains the X and Y coordinates for 2D points or the X, Y, and Z coordinates for 3D points as integer values.
Point orientations	An array contains the rotation matrices (2D) or quaternions (3D) describing the points’ orientations with respect to the drawing axes. The coefficients of the matrices, as well as the quaternions, lie within the range of –1 to 1. These values are stored as signed double-precision integers with the low-order bit equal to $1/(2^{31}-1)$. Therefore, to convert these coefficients to floating point, the integers must be divided by 2^{31}-1.

2D and 3D Point String Elements





Cone Elements (Type 23)

A circular truncated cone is described by two circles lying in parallel planes in a 3D design file. If the radius of both circles is identical, the cone represents a cylinder. The cone can be skewed by adjusting the positions of the circles. The C structure is:

```
typedef struct
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     short          unknown;       /* unknown data */
     long           quat[4];       /* orientation quaternion */
     Dpoint3d       center_1;      /* center of first circle */
     double         radius_1;      /* radius of first circle */
     Dpoint3d       center_2;      /* center of second circle */
     double         radius_2;      /* radius of second circle */
     } Cone_3d;
```

Cone type

The cone type word describes characteristics of the cone. Valid cone types include:

- 0 = general (nonspecific) cone
- 1 = right cylinder
- 2 = cylinder
- 3 = right cone
- 4 = cone
- 5 = right truncated cone
- 6 = truncated cone

Bits 3-14 of the cone type word are reserved and should be set to zero. Bit 15 indicates whether the cone is a surface or a solid (0=solid, 1=surface).

Parameters

Name:	**Description:**
Orientation	The orientation for both circles is defined by a single set of quaternions.
Radii	The radii for the circles are stored as double-precision floating point values. Either of these values may be zero to cause a pointed cone.

Cone Elements

Word Offset	Field	Name
0-17	Header	
18		cone.rsrv
19-20	Quaternion Q4	cone.quat
21-22	Q1	
23-24	Q2	
25-26	Q3	
27-30	X1	cone.center 1
31-34	Y1	
35-38	Z1	
39-42	R1	cone.radius 1
43-46	X2	cone.center 2
47-50	Y2	
51-54	Z2	
55-58	R2	cone.radius 2
59	Attribute Linkage	

B-spline Elements (Type 21, 24, 25, 26, 27, 28)

Rational, non-rational, uniform, and non-uniform B-spline curves and surfaces are represented in the design file by several different element types.

B-spline curves

Four element types are used to represent B-spline curves. A B-spline Curve Header Element (type 27) stores curve parameters. A B-spline Pole Element (type 21) stores poles. If the B-spline curve is rational, the pole element is immediately followed by a B-spline Weight Factor Element (type 28) that stores the weights of the poles. If the B-spline curve is non-uniform, the knots are stored in a B-spline Knot Element (type 26) immediately following the header. The order of these elements is fixed and is:

1. B-spline Curve Header Element (type 27)
2. Optional: B-spline Knot Element (type 26) if the curve is non-uniform
3. B-spline Pole Element (type 21)
4. Optional: B-spline Weight Factor Element (type 28) if the curve is rational

B-spline surfaces

Five element types are used to represent B-spline surfaces. A B-spline Surface Header Element (type 24) stores surface parameters. Subsequent B-spline Pole Elements (type 21) store separate rows of poles. If the surface is rational, each pole element is immediately followed by a B-spline Weight Factor Element (type 28). If the surface is non-uniform, a B-spline Knot Element (type 26) immediately follows the header. Finally, if the surface is trimmed, one or more B-spline Surface Boundary Elements (type 25) precede the first pole element. The order of these elements is fixed and must follow the order specified below:

1. Uniform, non-rational surfaces (type 24, 21, 21, 21…)
2. Uniform, rational surfaces (type 24, 21, 28, 21, 28, 21, 28…)
3. Non-uniform, non-rational (type 24, 26, 21, 21, 21…)
4. Non-uniform, rational (type 24, 26, 21, 28, 21, 28, 21, 28…)

5. Boundary immediately precedes poles (all type 25s must precede type 21, but follow type 26, if present. The order of the elements is fixed and can be either:

   type 24, 25, 25, 25, …, 21, 21, 21… or
   type 24, 25, 25, 25, …, 21, 28, 21, 28… or
   type 24, 26, 25, 25, 25, …, 21, 21, 21… or
   type 24, 26, 25, 25, 25, …, 21, 28, 21, 28…

B-spline curve header (type 27)

A B-spline curve header begins the definition of a B-spline curve and defines parameters describing the curve.

```
typedef struct
     {
     Elm_hdr            ehdr;          /* element header */
     Disp_hdr           dhdr;          /* display header */
     long               desc_words;    /* # of words in descr. */
     struct
          {
          unsigned      order:4;          /* B-spline order - 2 */
          unsigned      curve_display:1;  /* curve display flag */
          unsigned      poly_display:1;   /* polygon display flag */
          unsigned      rational:1;       /* rationalization flag */
          unsigned      closed:1;         /* closed curve flag */
          unsigned      curve_type:8;     /* curve type */
          } flags;
     short              num_poles;          /* number of poles */
     short              num_knots;          /* number of knots */
     } Bspline_curve;
```

Range

The range of a B-spline curve is the range of the control polygon. All points on the stroked curve lie within this range.

Curve parameters

A word of data is included that contains various parameters. A number two less than the B-spline order is stored in bits 0-3. Bit 7 is set for closed curves and cleared for open curves. Bit 6 is set for rational B-splines and cleared for non-rational splines. If bit 6 is set, a weight factor element must be included. Bit 5 is set to indicate if display of the polygon is enabled; bit 4 is set if curve display is enabled.

Number of poles

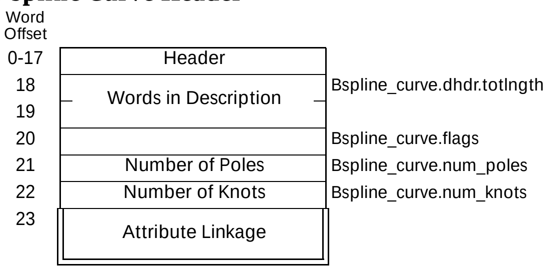
The maximum number of poles is 101.

Number of knots

For uniform B-spline curves, the number of knots is 0. The number of knots stored in the type 26 element for non-uniform B-spline curves is calculated as follows:

#KNOTS = #POLES - ORDER	(open curves)
#KNOTS = #POLES -1	(closed curves)

B-spline Curve Header



B-spline surface header (type 24)

A B-spline surface header begins the definition of a B-spline surface and defines parameters describing the surface.

```
typedef struct bspline_surface
     {
     Elm_hdr             ehdr;              /* element header */
     Disp_hdr            dhdr;              /* display header */
     long                desc_words;        /* # words in
description */
     struct
          {
          unsigned       order:4;         /* B-spline U order - 2 */
          unsigned       curve_display:1; /* surface display flag */
          unsigned       poly_display:1;  /* polygon display flag */
          unsigned       rational:1;      /* rationalization flag */
          unsigned       closed:1;        /* closed U surface flag*/
          unsigned       curve_type:8;    /* surface type */
          } flags;
     short               num_poles_u;     /* number of poles */
     short               num_knots_u;     /* number of knots */
     short               rule_lines_u;    /* number of rule lines */
     struct
```

```
     {
     unsignedshort v_order:4;            /* B-spline order - 2
                                            (v Direction) */
short              reserved1:2;         /* reserved */
     unsigned      short arcSpacing:1;  /* rule lines spaced by
                                            arc length */
     unsigned      short v_closed:1;    /*closed curve flag */
     unsigned      short reserved2:8;   /* reserved */
     } bsurf_flags;
short              num_poles_v;      /* number of poles */
short              num_knots_v;      /* number of knots */
short              rule_lines_v;     /* number of rule lines */
short              num_bounds;       /* number of boundaries */
} Bspline_surface;
```

Range

The range of a B-spline surface is the range of the control polygon. All points on the stroked surface lie within this range.

Surface parameters in U direction

A word of data is included that contains various parameters of the surface in the U direction. Parameters in the V direction are stored in a different word. A number two less than the B-spline U order is stored in bits 0-3. Bit 7 is set for surfaces closed in U and cleared for surfaces open in that direction. Bit 6 is set for rational B-splines and cleared for non-rational B-splines. If bit 6 is set, a weight factor element must follow every pole element. Bit 5 is set to indicate if display of the polygon is enabled; bit 4 is set if the surface display is enabled.

Number of poles in U direction

The maximum number of poles is 101 for each row of the surface, as each row is stored in a separate type 21 B-spline Pole element.

Number of knots in U direction

For uniform B-spline surfaces, the number of stored knots is 0. The number of knots stored in the type 26 element for non-uniform B-spline surfaces is the sum of the number of knots in the U direction plus the number of knots in the V direction. The number of knots in the U direction is calculated as follows:

#KNOTS_U = #POLES_U - ORDER_U	(surfaces open in U)
#KNOTS_U = #POLES_U -1	(surfaces closed in U)

Number of rules in U direction

The maximum number of rule lines is 256 for each direction of a B-spline surface.

Surface parameters in V direction

A word of data is included that contains various parameters of the surface in the V direction. Parameters in the U direction are stored in a different word. A number two less than the B-spline V order is stored in bits 0-3. Bit 7 is set for surfaces closed in V and cleared for surfaces open in that direction. Bit 6 is set if the rule lines are to be displayed spaced evenly by arc length. It is cleared if the rule line is to be spaced evenly throughout the parameter interval 0.0 to 1.0. The other bits in this word are reserved at this time.

Number of poles in V direction

The is no limit to the number of poles in the V direction of the surface, as each row is stored in a separate type 21 B-spline Pole element. Each row can contain a maximum of 101 poles.

Number of knots in V direction

For uniform B-spline surfaces, the number of stored knots is 0. The number of knots stored in the type 26 element for non-uniform B-spline surfaces is the sum of the number of knots in the U direction plus the number of knots in the V direction. The number of knots in the V direction is calculated as follows:

#KNOTS_V = #POLES_V - ORDER_V	(surfaces open in V)
#KNOTS_V = #POLES_V -1	(surfaces closed in V)

Number of rules in V direction

The maximum number of rule lines is 256 for each direction of a B-spline surface.

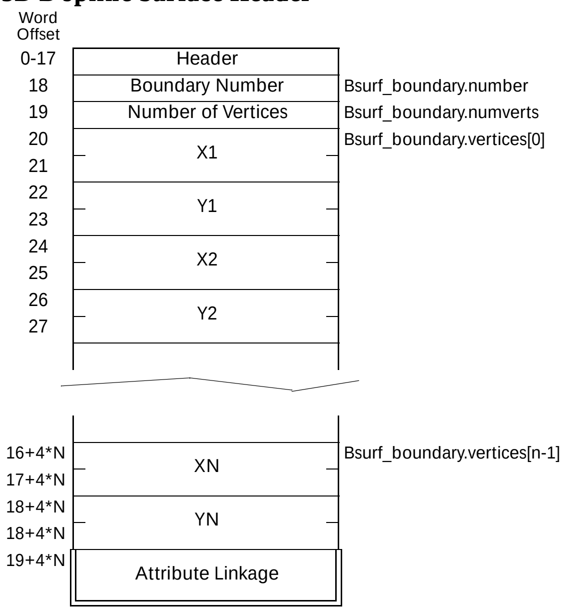
Number of boundaries

The total number of boundaries in the surface is stored in this word. This may differ from the total number of type 25 boundary elements stored with the surface as a single boundary may require more than one type 25 element to represent it.

Sense (Inner/Outer) of the boundaries

If the surface contains boundaries, the *Bspline_surface.dhdr.props.b.h* bit of the display header determines whether the area inside (*Bspline_surface.dhdr.props.b.h* is `FALSE`) or outside (*Bspline_surface.dhdr.props.b.h* is `TRUE`) of the boundaries is to be removed from the surface. This flag corresponds to the *holeOrigin* flag of the `MSBsplineSurface` data structure.

3D B-spline Surface Header



B-spline pole element (type 21)

(See “Line String (Type 4), Shape (Type 6), Curve (Type 11), and B-spline Pole Element (Type 21)” on page 18-20.)

B-spline surface boundary element (type 25)

The format of the B-spline Surface Boundary element is as follows.

```
typedef struct bsurf_boundary
    {
    Elm_hdr        ehdr;           /* element header */
    Disp_hdr       dhdr;           /* display header */
    short          number;         /* boundary number */
    short          numverts;       /* number of boundary vertices */
    Point2d        vertices[1];    /* boundary points (in UV space)*/
    } Bsurf_boundary;
```

Boundary number

This word indicates which boundary of the surface is being represented by this type 25 element. Subsequent type 25 elements may be used to define a single surface boundary by sharing the same boundary number. For example, the first and second type 25 elements in a surface may have a boundary number of 1 assigned to them, while the third, fourth, and fifth type 25 elements may have the boundary number 2 assigned to them. The represented surface would have two boundaries, one defined by two elements, the other defined by three.

Number of points

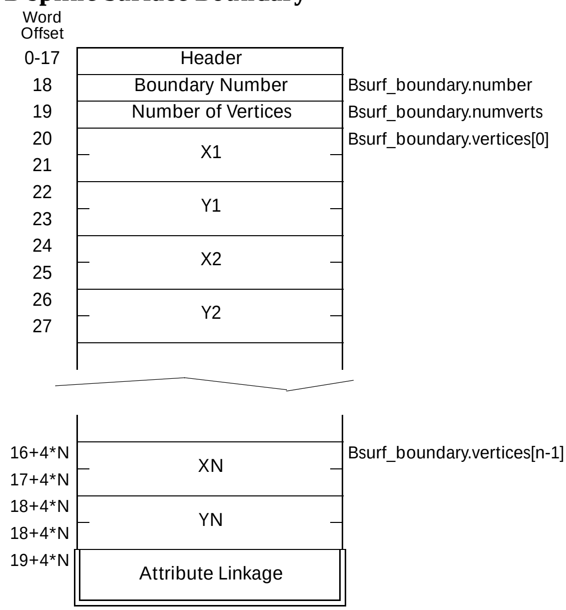
This word contains the number of points in this boundary element. The maximum number of points in any single boundary element is 151.

Vertices

The coordinates of the points defining the boundary element are stored as double-precision integer values in the U-V parameter space of the surface with the low order bit equal to $1/(2^{31}-1)$.

Only integers between 0 and 2^{31}-1 are acceptable, giving an effective range of 0.0 to 1.0 in both coordinates.

B-spline Surface Boundary



B-spline Knot element (type 26)

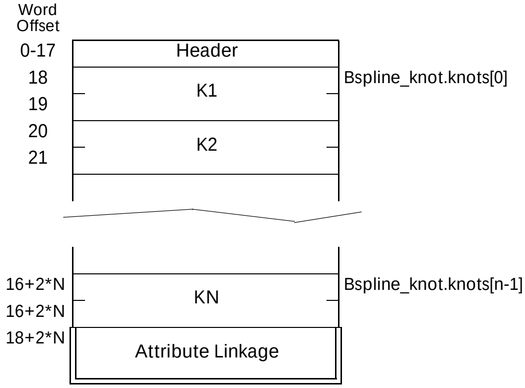
The format of the B-spline Knot element is displayed below.

```
typedef struct bspline_knot
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr  dhdr;          /* display header */
     long           knots[1]; /* knots (variable length) */
     } Bspline_knot;
```

Knots

For non-uniform B-spline curves and surfaces, the values of the interior non-uniform knots are stored as double-precision integers with the low order bit equal to $1/(2^{31}-1)$. Only integers between 0 and 2^{31}-1 are acceptable, giving an effective range of 0.0 to 1.0 for the interior knot values. For non-uniform surfaces, all the interior knots in the U direction are stored before the interior knots in the V direction.

B-spline Knot



B-spline Weight Factor element (type 28)

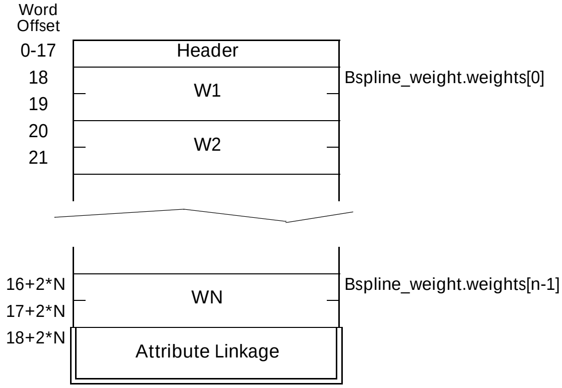
The format of the B-spline Weight Factor element is:

```
typedef struct bspline_weight
     {
     Elm_hdr        ehdr;          /* element header */
     Disp_hdr       dhdr;          /* display header */
     long           weights[1];    /* weights (variable length) */
     } Bspline_weight;
```

Knots

For rational B-spline curves and surfaces, the values of the pole weights are stored as double-precision integer values with the low order bit equal to $1/(2^{31}-1)$. Only integers between 0 and $2^{31}-1$ are acceptable, giving an effective range of 0.0 to 1.0 for the weight values. For rational surfaces, each type 21 pole element must be followed by a weight factor element giving the weights of that row of poles.

B-spline Weight Factor Element



Raster Header Element (Type 87)

Raster data consists of a complex raster header followed by scanline data in raster data elements (type 88). The header element contains the orientation of the image in the design file as well as data format information. The only difference between 2D and 3D raster header elements is the format of the vertices of the clipping polygon.

2D

```
typedef struct
     {
     Elm_hdr             ehdr;          /* element header */
     Disp_hdr            dhdr;          /* display header */
     unsigned long       totlength;     /* total length of cell */
     Raster_flags        flags;         /* misc. raster data*/
     byte                foreground;
     byte                background;
     unsigned short      xextent;
     unsigned short      yextent;
     short               reserved[2];
     double              resolution;
     double              scale;
     Point3d             origin;
     unsigned short      numverts;
```

```
        Point2d             vert2d[1];
        } Raster_hdr2d;
```

3D

```
    typedef struct
        {
        Elm_hdr             ehdr;           /* element header */
        Disp_hdr            dhdr;           /* display header */
        unsigned long       totlength;      /* total length of cell */
        Raster_flags        flags;          /* misc. raster data */
        byte                foreground;
        byte                background;
        unsigned short      xextent;
        unsigned short      yextent;
        short               reserved[2];
        double              resolution;     /* currently unused */
        double              scale;
        Point3d             origin;
        unsigned short      numverts;
        Point3d             vert3d[1];
        } Raster_hdr3d;
```

The raster flags are described in a C structure as follows:

```
    typedef struct
        {
        unsigned        right:1;
        unsigned        lower:1;
        unsigned        horizontal:1;
        unsigned        format:5;
        unsigned        color:1;            /* not used by MicroStation
*/
        unsigned        transparent:1;
        unsigned        positive:1;         /* not used by MicroStation
*/
        unsigned        unused:5;
        } Raster_flags;
```

Justification

The justification of a raster element indicates which corner of the element is the origin and the direction of the scan lines. Currently MicroStation supports only raster elements with upper left justification and horizontal scan lines.

Format

MicroStation currently supports three raster formats. Format 1 is straight binary with a single bit for each pixel and format 9 is run length encoded binary. For formats 1 and 9, a foreground and background color is stored in the element and used to determine the color for pixel values of one and zero, respectively. Format 2 stores a byte for each pixel and is used to store color images.

Transparent background

If the `transparent` bit is set, a raster element has a transparent background and pixels are not set if they are set to the background color.

Background and foreground colors

For format 1 (binary) and 9 (run length binary), the foreground and background colors indicate the color indexes for pixel values of 0 and 1, respectively.

Pixel extents

The x pixel extent is the number of pixels in the raster image along the x axis, and the y pixel extent is the number of pixels along the y-axis.

Device resolution

Unused; reserved for future use.

Pixel to UOR conversion factor

The number of UORs per pixel.

UOR origin

This defines the relative position of the raster image in the design file. It is the coordinate of the origin of the raster data.

Clip box

The clip box for a raster element is drawn prior to the display of the raster data but does not currently affect the raster data display.

The display of the clipping box can be omitted by setting the number of vertices to zero.

Raster Header Element

Word Offset	Field	Name
0-17	Header	
18–19	Words in Description	raster_hdr.totlngth
20		raster_hdr.raster_flags
21		raster_hdr.foreground.background
22	X Pixel Extent	raster_hdr.xextent
23	Y Pixel Extent	raster_hdr.yextent
24–25	Reserved	raster_hdr.reserved
26–29	Resolution	raster_hdr.resolution
30–33	Scale	raster_hdr.scale
34–35	X Origin	
36–37	Y Origin	
38–39	Z Origin	
40	Number of Vertices	raster_hdr.numverts
41–42	X1	raster_hdr.vertice(0)
43–44	Y1	
45–46	X2	
47–48	Y2	
...	...	
37+4*N–38+4*N	XN	
39+4*N–40+4*N	YN	
41+4*N	Attribute Linkage	

Raster Data Elements (Type 88)

A scan line element contains the pixel information for all or part of a single scan line of raster data. Scan line elements have the same format for the first 18 words. They differ in the actual data stored. The size of an element is limited to 768 words.

```
typedef struct
    {
    Elm_hdr             ehdr;               /* element header */
    Disp_hdr            dhdr;               /* display header */
    Raster_flags        flags;              /* flags */
    byte                foreground;
    byte                background;
    unsigned short      xoffset;
    unsigned short      yoffset;
    unsigned short      numpixels;
    byte                pixel[1];
    } Raster_comp;
```

Range

The range block for the scanline includes the spacing around the pixels. Thus, if the pixel to UOR scale is 10, there are 5 pixels in the scanline, and the origin is (0,0), the range is (-5,-5) to (45,5).

Pixel offset

These are the x and y pixel offsets from the origin. Thus, for a raster image with upper left origin and horizontal scanlines, an offset of (0,2) represents the first pixel of the third scanline.

Number of pixels

The number of pixels in the element.

Pixel data

The data that determines the color of each pixel can be stored in several formats. The same format must be used for each scanline in a raster element. The formats supported as of this writing are:

- **Bitmap**, or straight binary (type 1) — Each bit defines the color of one pixel. If the bit is set, the foreground color is used; if the bit is clear, the background color is used.
- **Byte** (type 2) — Each byte defines the color of one pixel.
- **Run length binary** (type 9) — Each word contains a run length (number of pixels). The first value is considered to be “off” so each pixel in the first run length is displayed in the

background color. The next value is “on” so each pixel in the second run length is displayed in the foreground color, etc.

Raster Data Element

Word Offset		
0-17	Header	
18		raster_comp.raster_flags
19		raster_comp.foreground.background
20	X Pixel Offset	raster_comp.xoffset
21	Y Pixel Offset	raster_comp.yoffset
22	Number of Pixels	raster_comp.numpixels
23		raster_comp.pixel
	Attribute Linkage	

Group Data Elements (Type 5)

Type 5 elements are commonly referred to as **group data elements**. They store non-graphic data such as reference file attachments and named views. The different type 5 elements are distinguished by level. The type 5 elements used by MicroStation and also supported by IGDS are documented in this section.

Reference file attachment element (type 5, level 9)

Type 5 reference file attachment elements are stored on level 9. They contain all the information necessary to define a single reference file attachment. The C structure is:

```
typedef struct
     {
     Elm_hdr    ehdr;              /* element header */
     Disp_hdr   dhdr;              /*display header */
     short      file_chars;        /* no. of chars. in file spec */
     char       file_spec[65]      /*file specification */
     byte       file_num;          /* file number */
     Fb_opts    fb_opts;           /* file builder options mask */
     Fd_opts    fd_opts;           /* file displayer options mask*/
     byte       disp_flags[16];    /* display flags */
     short      lev_flags[8][4];   /* level on/off flags */
     long       ref_org[3];        /* origin in ref file uors */
     double     trns_mtrx[9];      /* transformation matrix */
     double     cnvrs_fact;        /* conversion factor */
```

```
        long      mast_org[3];         /* origin in master file uors */
        short     log_chars;           /* characters in logical name */
        char      log_name[22];        /* logical name (padded) */
        short     desc_chars;          /* characters in description */
        char      description[42];     /* description (padded) */
        short     lev_sym_mask;        /* level symbology enable mask*/
        short     lev_sym[63];         /* level symbology descriptor */
        long      z_delta;             /* Z-direction delta */
        short     clip_vertices;       /* clipping vertices */
        Point2d   clip_poly[1];        /* clipping polygon */
        } Ref_file_type5;
    typedef struct
        {
        unsigned  multi_attach:1;      /* multi-attach */
        unsigned  one_one_map:1;       /* 1:1 mapping */
        unsigned  slot_in_use:1;       /* slot in use */
        unsigned  upd_fildgn:1;        /* update on file design */
        unsigned  db_diff_mf:1;        /* database dif than mas file */
        unsigned  snap_lock:1;         /* snap lock */
        unsigned  locate_lock:1;       /* locate lock */
        unsigned  missing_file:1;      /* missing file */
        unsigned  unused:8;            /* unused */
        } Fb_opts;
    typedef struct
        {
        unsigned  view_ovr:1;          /* view override */
        unsigned  display:1;           /* display */
        unsigned  line_width:1;        /* lines with width */
        unsigned  unused:13;           /* unused */
        } Fd_opts;
```

For information about the MicroStation reference file attachment element (type 66, level 5) and when a reference file attachment must be stored as that type rather than a type 5, see “MicroStation Application Elements (Type 66)” on page 18-58.

Named view element (type 5, level 3)

Type 5 elements on level 3 are used to store named views. A named view is created when the SAVE VIEW command is executed.

```
typedef struct
     {
     Elm_hdr   ehdr;                /* element header */
     short     grphgrp;             /* graphics group number */
     short     attindx;             /* words between this and
                                              attributes */
     short     properties;          /* property bits
                                              (always same) */
     unsigned  num_views:3;         /* number of views */
     unsigned  reserved:13;         /* reserved for Intergraph */
     char      viewdef_descr[18];   /* view definition descr. */
     byte      full_scr1;
     byte      full_scr2;
     Viewinfo  view[1];
     char      rest_of_elem[1];     /* record has variable len.
*/
     }Named_view_type5;
typedef struct
     {
     unsigned  fast_curve:1;        /* fast curve display */
     unsigned  fast_text:1;         /* fast text */
     unsigned  fast_font:1;         /* fast font text */
     unsigned  line_wghts:1;        /* line weights */
     unsigned  patterns:1;          /* pattern display */
     unsigned  text_nodes:1;        /* text node display */
     unsigned  ed_fields:1;         /* enter data field
underlines */
     unsigned  on_off:1;            /* view on or off */
     unsigned  delay:1;             /* delay on */
     unsigned  grid:1;              /* grid on */
     unsigned  lev_symb:1;          /* level symbology */
     unsigned  points:1;            /* points */
     unsigned  constructs:1;        /* line constructs */
     unsigned  dimens:1;            /* dimensioning */
     unsigned  fast_cell:1;         /* fast cells */
     unsigned  def:1;
     } Viewflags;
typedef struct
     {
     Viewflags      flags;          /* view flags */
```

```
          short          levels[4];          /* active levels (64 bit
array) */
          Point3d        origin;             /* origin (made up of longs)
*/
          Upoint3d       delta;              /* delta to other corner of
view*/
          double         transmatrx[9];      /* view transformation matrix
*/
          double         conversion;         /* conv. from digitizer to
uors */
          unsigned long  activez;            /* view active z */
          } Viewinfo;
```

Color table element (type 5, level 1)

Color table elements are type 5 elements stored on level 1. One color table element can be stored for each screen (left and right). When MicroStation opens a design file it searches for color table elements and adjusts the graphics controller(s) to match the color table found. A byte value is stored for red, green and blue intensities for each element color and the background (0=background + 255 element colors).

```
typedef struct
     {
     Elm_hdr        ehdr;                    /* element header */
     Disp_hdr       dhdr;                    /* display header */
     short          screen_flag;             /* screen flag */
     byte           color_info[256][3];      /* color table info. */
     } Color_table_type5;
```

MicroStation Application Elements (Type 66)

Type 66 elements are similar to type 5 elements in that they store non-graphical data. Since the data in a type 66 element is not associated with a level, levels distinguish between types of data. For example, level 8 in type 66 is reserved for digitizing data.

Type 66 elements are not supported by IGDS. Early versions of IGDS may report that the type 66 elements are invalid when MicroStation design files are uploaded to a VAX. This problem was corrected in later versions. In MicroStation, type 66 elements can be deleted from a design file with one of the DELETE66ELEMENTS commands.

MicroStation reference file attachment element (type 66, level 5)

Type 66 reference file attachment elements are stored on level 5. They contain all the information necessary to define a single reference file attachment. The structure is identical to that of the type 5 reference file attachment element, except that some of the values can be different.

A reference file attachment must be stored as a type 66 element if any of the following are true:

- There are already 32 or more reference file attachments.
- In MicroStation Version 4.0 and later versions, the file_num byte of the display section (high byte of word 51) can be a value from 1–255, and therefore must be treated as an unsigned byte. IGDS and versions of MicroStation prior to Version 4.0 only allow values from 1–32.
- There are multiple clipping masks in that reference file attachment.
- The clipping point array can have “disconnect” values in it. These are recognized as points where both X and Y values are 0x8000000 (-2147483648). The part of the array before the disconnect is the exterior clipping boundary. After the first disconnect there is one or more interior clipping mask boundaries, separated from each other by an additional disconnect. If there are no clipping masks, there are no disconnects. The number of clipping points stored in the element (word 246 starting from 0) includes the number of points in clipping boundary, the disconnects, and the number of points in the clipping mask boundary or boundaries. Both the clipping exterior and the clipping mask boundaries are closed (the first point equals the last point).
- It is an attachment of a 2D reference file to a 3D master design file.
- Bit 15 of the fb_opts word (word 52 starting at 0) in the display section is set if the reference file is 2D and the master file is 3D. This bit was always clear for IGDS and earlier MicroStation versions. Interpretation of the remainder of the attachment element is as for a 3D attachment. 3D elements are constructed from the 2D elements in the reference file by setting all Z values to 0.

MicroStation digitizer element (type 66, level 8)

The digitizing transformation and associated digitizing parameters are stored in a type 66 element on level 8. The C structure is:

```
typedef struct
    {
    Elm_hdr        ehdr;            /* element header */
    Disp_hdr       dhdr;            /* display header */
    Uspoint2d      extentlo;        /* bottom left of active area
of
                                       tablet */
    Uspoint2d      extenthi;        /* top right of active area
of
                                       tablet */
    Uspoint2d      origin;          /* origin for the screen-
mapped
                                       portion */
    Uspoint2d      corner;          /* corner for the screen-
mapped
                                       portion */
    short          defined;         /* TRUE = transform valid */
    double         trans[3][4];     /* trans. from digitizer to
dgn.
                                       file coords. */
    double         stream_delta;    /* sampling delta (UORs) */
    double         stream_tol;      /* shortest segment that must
                                       be saved */
    double         angle_tol;       /* angle above which point
must
                                       be saved */
    double         area_tol;        /* area above which point
must
                                       be saved */
    short          extrawords[8];   /* currently unused */
    } MicroStation_dig;
```

MicroStation extended TCB element (type 66, level 9)

A type 66 element on level 9 is used to store MicroStation-specific TCB variables (those that are not supported by IGDS). The C structure is:

```
typedef struct
     {
     Elm_hdr        ehdr;               /* element header */
     Disp_hdr       dhdr;               /* display header */
     double         axlock_angle;       /* axis lock angle */
     double         axlock_origin;      /* axis lock origin */
     Ext_viewinfo   ext_viewinfo[8];    /* ext'd view info. strucs.*/
     Ext_locks      ext_locks;          /* extended lock bits */
     long           activecell;         /* active cell (Radix 50) */
     double         actpat_scale;       /* active patterning scale */
     long           activepat;          /* active patterning cell */
     long           actpat_rowspc;      /* active patterning row
                                                    spacing */
     double         actpat_angle;       /* active patterning angle */
     double         actpat_angle2;      /* active patterning angle */
     long           actpat_colspc;      /* active patterning column
                                              spacing */
     long           actpnt;             /* active point (Radix 50) */
     double         actterm_scale;      /* active line terminator
                                                    scale */
     long           activeterm;         /* active line terminator */
     short          hilitecolor[2];     /* hilite color for two
                                                    screens */
     short          fullscreen_cursor;/* keypoint snap flag */
     short          keypnt_divisor;     /* divisor for keypoint
                                                    snapping */
     char           celfilenm[50];      /* local cell lib. name */
     short          xorcolor[2];        /* exclusive or color */
     } Mstcb_elm;
```

The `Ext_viewinfo` and `Ext_locks` structures are defined as follows:

```
typedef struct ext_viewinfo
     {
     Ext_viewflags ext_flags;           /* extended flags */
     short          classmask;          /* class masks */
     short          unused;             /* reserved for future use */
     double         perspective;        /* perspective disappearing
                                               point */
     short          padding[52];        /* reserved for future use */
     } Ext_viewinfo;
```

```
      typedef struct ext_viewflags
           {
      #ifndef (mc68000)
           unsigned       fill:1;             /* true if element fill
enabled */
           unsigned       unused1:15;         /* reserved for future use */
           unsigned       unused:16;          /* reserved for future use */
      #else
           unsigned       unused:16;          /* reserved for future use */
           unsigned       unused1:15;         /* reserved for future use */
           unsigned       fill:1;             /* true = element fill
enabled */
      #endif
           } Ext_viewflags;
      typedef struct ext_locks
           {
      #ifndef mc68000
           unsigned       axis_lock:1;        /* Axis Lock */
           unsigned       auxinp:1;           /* auxiliary input device */
           unsigned       show_pos:1;         /* continuous coordinate
                                                        display */
           unsigned       autopan:1;          /* automatic panning */
           unsigned       axis_override:1;    /* override Axis Lock */
           unsigned       cell_stretch: 1;    /* cell stretching */
           unsigned       iso_grid:1;         /* isometric grid */
           unsigned       iso_cursor:1;       /* isometric cursor */
           unsigned       full_cursor: 1;     /* full screen cursor (PC
only)*/
           unsigned       iso_plane:2;        /* 0=Top, 1=Left, 2=Right,
                                                        3=ALL*/
           unsigned       selection_set:1;    /* enable selection set */
           unsigned       auto_handles:1;     /* select elements upon
                                                   placement */
           unsigned       single_shot:1;      /* single shot tool selection
*/
           unsigned       dont_restart:1;     /* set if command doesn't
                                                        want to be
restarted */
           unsigned       view  Single_shot:1;     /* single shot view
commands */
           unsigned       snapCnstplane:1;         /* snap to construction
plane */
           unsigned       cnstPlanePerp:1;         /* snap perpendicular to
                                                   construction plane */
           unsigned       fillMode:1;              /* not currently used */
           unsigned       iso_lock:1;              /* isometric lock */
```

```
            unsigned       unused2:12;              /* reserved for future
use */
#else
            unsigned       unused2:12;              /* reserved for future
use */
            unsigned       iso_lock:1;              /* isometric lock */
            unsigned       fillMode:1;              /* not currently used */
            unsigned       cnstPlanePerp:1;         /* snap perpendicular to
                                                       construction
plane */
            unsigned       snapCnstplane:1;         /* snap to construction
plane */
            unsigned       viewSingle_shot:1;       /* single shot view
commands */
            unsigned       dont_restart:1;          /* set if command
doesn't want
                                                       to be restarted */
            unsigned       single_shot:1;           /* single shot tool
selection */
            unsigned       auto_handles:1;     /* select elements when
placed */
            unsigned       selection_set:1;         /* enable selection set
*/
            unsigned       iso_plane:2;             /* 0=Top, 1=Left,
2=Right,
                                                       3=ALL */
            unsigned       full_cursor: 1;     /* full screen cursor (PC
only)*/
            unsigned       iso_cursor:1;            /* isometric cursor */
            unsigned       iso_grid:1;              /* isometric grid */
            unsigned       cell_stretch: 1;         /* cell stretching */
            unsigned       axis_override:1;         /* override Axis Lock */
            unsigned       autopan:1;               /* automatic panning */
            unsigned       show_pos:1;              /* continuous coord.
display*/
            unsigned       auxinp:1;                /* auxiliary input
device */
            unsigned       axis_lock:1;             /* Axis Lock */
     #endif
            } Ext_locks;
```

Application startup element (type 66, level 10)

A type 66 element on level 10 is used to automatically start an application when a design file is opened. The C structure is:

```
typedef struct
    {
    Elm_hdr   ehdr;                          /* element header */
    Disp_hdr  dhdr;                          /* display header */
    char      startappcommand[256];          /* app. command line */
    short     reserved[110];
    } MicroStation_Startapp;
```

Application data element (type 66, level 20)

Application-specific data is stored in a type 66 element on level 20. The specific element format is at the discretion of the application developer. However, the maximum element size is 768 words, and word 18 must contain the signature value assigned by Bentley Systems, Inc.

The general format is shown as a C structure below.

```
typedef struct
    {
    Elm_hdr       ehdr;          /* element header */
    Disp_hdr      dhdr;          /* display header */
    short         signature;     /* assigned signature */
    .
    .                            /* application data */
    .
    } Ms_appdata;)
```