SEMI Draft Document 3626 2003/04/23
OASIS™ - Open Artwork System Interchange Standard

1 Purpose

1.1 The purpose of this specification is to define an interchange and encapsulation format for hierarchical integrated circuit mask layout information.

1.2 *Background*—In the fall of 2001, SEMI's Data Path Task Force formed a working group to define a successor to the venerable GDSII Stream format, which had served the I.C. industry as a *de facto* standard for layout interchange for more than two decades. The old format, limited by 16-bit and 32-bit internal integer fields, by its inefficient representation of cell-native geometric figures, and by high structural overhead, was becoming difficult to use for leading-edge designs, and file sizes were becoming unwieldy, in some cases growing to many tens of gigabytes. The successor format was chartered with several overall goals:

- Achieve at least an order-of-magnitude file size improvement compared to GDSII Stream.
- Remove all 16-bit and 32-bit integer width restrictions—make the new format fully 64-bit capable.
- Efficiently represent cells with large payloads of flat native geometric figures.
- Provide a richer information palette to facilitate interchange of layout-related information between design and manufacturing.

In the months leading up to the formation of the SEMI Data Path Task Force, International Sematech sponsored a series of meetings focusing on Mask EDA issues. Many of the Task Force participants were also involved in these Sematech meetings, and carried forward much useful information from those sessions into the definition of this specification.

2 Scope

2.1 This format is designed primarily to encapsulate hierarchical mask layout for interchange between systems such as EDA software, mask writing tools, and mask inspection/repair tools.

2.2 This format is designed to be both hardware- and software-independent.

3 Limitations

3.1 Use of extension records such as XNAME, XELEMENT, and XGEOMETRY may impair interoperability between tools. It is recommended that these extensions be used primarily for prototyping, and that interoperability be maintained through the formal inclusion of extensions to this specification.

4 Referenced Standards

4.1 *IEEE Standards*[1]

IEEE 754-1985 - IEEE Standard for Binary Floating-Point Arithmetic

4.2 *ISO Standards*[2]

ISO-646-IRV - "US-ASCII" Character Set

ISO-3309 - Information technology—Telecommunications and information exchange between systems—High-level data link control (HDLC) procedures—Frame structure

4.3 *IETF Standards*[3]

RFC 1951 - DEFLATE Compressed Data Format Specification version 1.3

1. Institute of Electrical and Electronics Engineers
IEEE Operations Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, New Jersey 08855-1331, USA. Telephone: 732-981-0060; FAX: 732-981-1721.
Website: www.ieee.org

2. International Organization for Standardization
ISO Central Secretariat, 1, rue de Varembé, Case postale 56, CH-1211 Geneva 20, Switzerland. Telephone: 41-22-749-01-11; FAX: 41-22-733-34-30
Website: www.iso.ch

3. Internet Engineering Task Force
Website: www.ietf.org

5 Terminology

5.1 Abbreviations and Acronyms

5.1.1 *BNF*—Backus-Naur Form

5.1.2 *EDA*—Electronic Design Automation

5.1.3 [1]*OASIS™*—Open Artwork System Interchange Standard

5.2 Definitions

5.2.1 Most definitions of terminology specific to OASIS are found within the text of the paragraphs that contain them.

5.2.2 *Cell*—a named object in a layout hierarchy, containing native geometric information, annotation information, and/or placements of other cells.

5.2.3 *Placement*—a specification by reference that a copy of a cell is to be placed within the coordinate space of another cell at a particular location, orientation, and scale. Cell placement is the fundamental mechanism which makes hierarchy within the OASIS file possible.

5.2.4 *Geometry*—a two-dimensional geometric figure such as a polygon, rectangle, trapezoid, path, circle, etc. with inherent attributes of *layer* and *datatype*.

5.2.5 *Property*—an annotation element consisting of a name plus an optional list of values, supplying descriptive information about the characteristics of the file or one of its components.

5.2.6 *Record*—the principal data division in an OASIS file.

5.2.7 *Text Element*—an annotation element consisting of an (x,y) coordinate point and an associated string.

5.3 Symbols

5.3.1 **“->”** — indicates a mapping of an argument to its contents or its meaning.

1. Used with consent by the owner.

6 OASIS BASICS

6.1 An OASIS file is a sequence of **bytes** divided into **records**. The length of a record is discernible from its structure and is not explicit (in contrast to GDSII Stream, where all record lengths are explicit).

6.2 An OASIS file has the following overall syntax (using the modified BNF notation described in section 36 on page 30). Individual record types appear in bold uppercase and are described in more detail in following sections.

<oasis-file> -> <magic-bytes> **START** { **CBLOCK** | **PAD** | **PROPERTY** | <cell> | <name> }* **END**
<name> -> { **CELLNAME** | **TEXTSTRING** | **LAYERNAME** | **PROPNAME** | **PROPSTRING** | **XNAME** }
<cell> -> { **CELL** { **CBLOCK** | **PAD** | **PROPERTY** | **XYRELATIVE** | **XYABSOLUTE** | <element> }* }
<element> -> { <geometry> | **PLACEMENT** | **TEXT** | **XELEMENT** }
<geometry> -> { **RECTANGLE** | **POLYGON** | **PATH** | **TRAPEZOID** | **CTRAPEZOID** | **CIRCLE** | **XGEOMETRY** }

6.3 An OASIS file may represent a complete layout hierarchy, a portion of a layout hierarchy, or multiple layout hierarchies. These interpretations are not intrinsic to the format and are governed by application semantics only. Each OASIS file must be syntactically complete—it must begin with <magic-bytes> and contain at least a **START** and **END** record.

6.4 The <magic-bytes> element is a sequence of 13 ASCII characters: "`%SEMI-OASIS<CR><NL>`" where `<CR><NL>` represents the ASCII hexadecimal sequence `0D 0A`. It is provided as a recognition signature to make OASIS files easily identifiable to the UNIX *file* utility. (The intent of the carriage return and newline is to help detect corruption by FTP programs operating in non-binary mode.)

6.5 <u>EXCEPTION HANDLING</u>: OASIS processors should treat any deviation from the syntax presented in this document as a fatal error. OASIS readers are not required to implement syntax-check preprocessing in order to be considered compliant with this specification. The sequence in which exceptions are detected and reported is entirely application-dependent. In addition, for access requests which do not require the interpretation of the entire file (such as retrieval of a single cell or a subset of the cells within the file), this specification does not require OASIS readers to exhaustively check the validity of the entire file.

7 DATA CONSTRUCTS

7.1 BYTES

7.1.1 A byte is a fixed-length 8-bit value. Bit patterns for bytes are shown with the least significant bit (bit 0) on the right.

7.2 INTEGERS

7.2.1 An ***unsigned-integer*** is an N-byte (N > 0) integer value. The low-order byte appears <u>first</u> in the OASIS format. Integer byte length is variable and integers are represented as *byte-continuations* where the most significant bit of each byte except the last in the chain is a 1; the remaining seven bits in each byte are concatenated to form the actual integer value itself. There are no restrictions on integer byte length (and hence, magnitude).

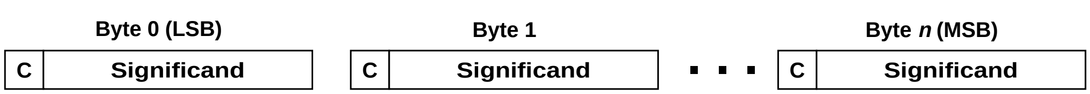

Figure 7-1
Unsigned-Integer Representation

Table 7-1: Unsigned-Integer Examples

UNSIGNED-INTEGER VALUE	BINARY REPRESENTATION
0	00000000
127	01111111
128	10000000 00000001
16,383	11111111 01111111
16,384	10000000 10000000 00000001

7.2.2 A ***signed-integer*** follows the same byte-continuation scheme as an *unsigned-integer*, and is stored in signed-magnitude form, with the *significand* left-shifted one bit and the sign bit stored in the least significant bit of the low-order (first) byte. A sign bit of 0 indicates a positive number, and a sign bit of 1 indicates a negative number. Both representations of zero (+0 and -0) should be treated as numerically equivalent for the purposes of comparison.

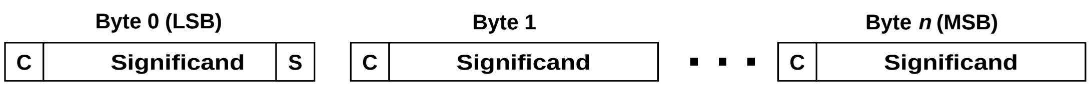

Figure 7-2
Signed Integer Representation

Table 7-2: Signed Integer Examples

SIGNED INTEGER VALUE	BINARY REPRESENTATION
0	00000000
+1	00000010
-1	00000011
+63	01111110
-64	10000001 00000001
+8,191	11111110 01111111
-8,192	10000001 10000000 00000001

7.2.3 EXCEPTION HANDLING: OASIS processors which only support integer data in a restricted space (e.g., 32-bit space) should treat any magnitude outside of this space as a fatal error.

7.3 REALS

7.3.1 A ***real*** number may be stored in one of several rational forms, or as a single-precision 4-byte (*ieee-4*) or double-precision 8-byte (*ieee-8*) floating point value. The rational forms are usually more compact than the floating-point forms, and have the advantage of being able to precisely represent many values which can only be approximated by the binary floating point representation. The type of representation is stored in an *unsigned-integer* which precedes the significant portion of the real:

Table 7-3: Real Number Types

FORMAT	MEANING
‘0’ *unsigned-integer*	Positive whole number
‘1’ *unsigned-integer*	Negative whole number
‘2’ *unsigned-integer*	Positive reciprocal
‘3’ *unsigned-integer*	Negative reciprocal
‘4’ *unsigned-integer unsigned-integer*	Positive ratio
‘5’ *unsigned-integer unsigned-integer*	Negative ratio
‘6’ IEEE-4-byte-float	Single-precision floating point
‘7’ IEEE-8-byte-float	Double-precision floating point

7.3.2 In types 0 and 1, the *real* is a whole number—its fractional portion is zero. In types 2 and 3, the *unsigned-integer* represents the denominator of a reciprocal, with an implicit numerator of 1. Types 4 and 5 are ratios, with the numerator listed first, followed by the denominator. Types 6 and 7 are binary floating point numbers in IEEE 754-1985 format, with the least significant byte of the *fraction* (byte 0) stored first.

ieee-4 float

S	Exponent	Fraction		
Byte 3		Byte 2	Byte 1	Byte 0

ieee-8 float

S	Exponent	Fraction						
Byte 7		Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0

Figure 7-3
IEEE Floating Point Formats

Table 7-4: Real Number Examples

VALUE	RATIONAL FORM	IEEE-4 FORM
0.0	00000000 00000000	00000110 00000000 00000000 00000000 00000000
1.0	00000000 00000001	00000110 00000000 00000000 10000000 00111111
-0.5	00000011 00000010	00000110 00000000 00000000 00000000 10111111
0.3125	00000100 00000101 00010000	00000110 00000000 00000000 10100000 00111110
1/3	00000010 00000011	00000110 10101011 10101010 10101010 00111110
-2/13	00000101 00000010 00001101	00000110 11011001 10001001 00011101 10111110

7.3.3 EXCEPTION HANDLING: For types 2-5, a denominator of 0 should be treated as a fatal error. A type outside the range of 0-7 should be treated as a fatal error.

7.4 STRINGS

7.4.1 A ***string*** is a sequence of zero or more bytes (“characters”) preceded by an *unsigned-integer* representing the number of characters in the string:

string -> length byte*

Strings in OASIS are further sub-typed by semantic. A ***b-string*** (“binary string”) is a string which may contain any combination of 8-bit character codes in any sequence. An ***a-string*** (“ASCII string”) may contain only *printable* ASCII character codes (hexadecimal 21-7E) plus the SP (space) character (hexadecimal 20), in any sequence. An ***n-string*** (“name string”) may contain only *printable* ASCII character codes (hexadecimal 21-7E), and must have a length greater than zero.

7.4.2 The set of *printable* ASCII characters consists of hexadecimal character codes 21-7E. In ascending order of character code, we have:

```
!"#$%&’()*+,-./0123456789:;<=>?@        [21-40]
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘        [41-60]
abcdefghijklmnopqrstuvwxyz{|}~          [61-7E]
```

This excludes space (SP), tabs (HT, VT), and all other control characters.

7.4.3 EXCEPTION HANDLING: OASIS processors should treat illegal characters in *a-strings* or *n-strings* as fatal errors. Zero-length *n-strings* should also be treated as fatal errors.

7.5 DELTAS

7.5.1 A ***delta*** represents geometric data (coordinates, vectors, planar offsets, etc.).

7.5.2 A ***1-delta*** is stored as a *signed-integer* and represents a horizontal or vertical displacement. Bit 0 encodes direction: 0 for east or north, 1 for west or south. The remaining bits are the magnitude. Horizontal or vertical alignment is implied by context.

7.5.3 A ***2-delta*** is stored as an *unsigned-integer* and represents a horizontal or vertical displacement. Bits 0-1 encode direction: 0 for east, 1 for north, 2 for west, and 3 for south. The remaining bits are the magnitude.

7.5.4 A ***3-delta*** is stored as an *unsigned-integer* and represents a horizontal, vertical, or 45-degree diagonal displacement. Bits 0-2 encode direction: 0 for east, 1 for north, 2 for west, 3 for south, 4 for northeast, 5 for northwest, 6 for southwest, and 7 for southeast. The remaining bits are the magnitude (for horizontal and vertical deltas) or the magnitude of the projection onto the x- or y-axis (for 45-degree deltas).

7.5.5 A ***g-delta*** has two alternative forms and is stored either as a single *unsigned-integer* or as a pair of *unsigned-integers*. The first form is indicated when bit 0 is zero, and represents a horizontal, vertical, or 45-degree diagonal displacement, with bits 1-3 encoding direction, and the remaining bits storing the magnitude, in the same fashion as a *3-delta*. The second form represents a general (x,y) displacement and is a pair of *unsigned-integers*. Bit 0 of the first integer is 1. Bit 1 of the first integer is the x-direction (0 for east, 1 for west). The remaining bits of the first integer represent the magnitude in the x-direction. Bit 0 of the second integer is the y-direction (0 for north, 1 for south). The remaining bits of the second integer represent the magnitude in the y-direction. Both forms may appear in a list of *g-deltas*.

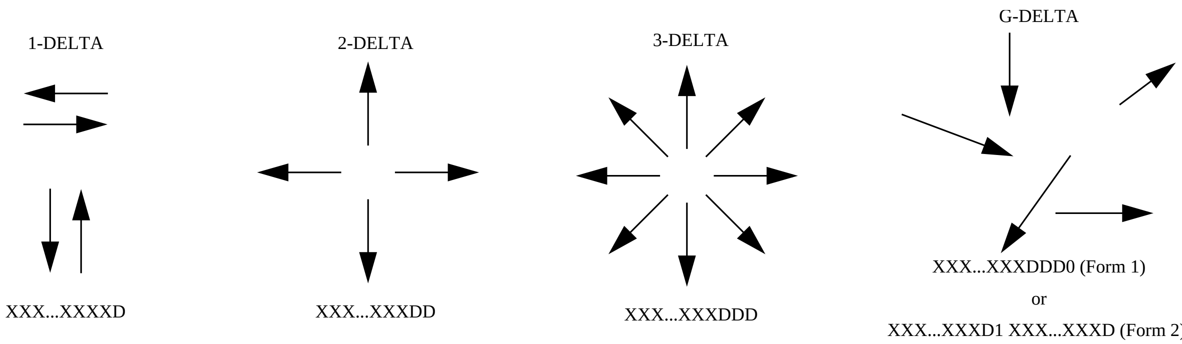

Figure 7-4
Delta Types

Table 7-5: Delta Examples

BIT PATTERN	TYPE	MEANING
11111001 00100011	1-delta	Δ = -2300
11111000 00100011	1-delta	Δ = +2300
10011000 00101010	2-delta	Δx = +1350
10011011 00101010	2-delta	Δy = -1350
11001101 00000001	3-delta	Δx = -25, Δy = +25
11010111 00000111	3-delta	Δx = +122, Δy = -122
11101001 00000011 01111010	g-delta$_2$	Δx = +122, Δy = +61
11101100 00000101	g-delta$_1$	Δx = -46, Δy = -46
10111011 00000001 10110111 00001111	g-delta$_2$	Δx = -46, Δy = -987

7.6 REPETITIONS

7.6.1 A ***repetition*** represents an “array” of cell placements, geometries, or text elements. The repetition is part of the **PLACEMENT**, **<geometry>**, or **TEXT** record itself. A repetition consists of an *unsigned-integer* which encodes the type, followed by any related repetition parameters:

Table 7-6: Repetition Types

TYPE	FORMAT
0	*re-use the previous repetition definition*
1	**x-dimension y-dimension x-space y-space**
2	**x-dimension x-space**
3	**y-dimension y-space**
4	**x-dimension x-space$_1$ ... x-space$_{N-1}$**
5	**x-dimension grid x-space$_1$ ... x-space$_{N-1}$**
6	**y-dimension y-space$_1$ ... y-space$_{M-1}$**
7	**y-dimension grid y-space$_1$ ... y-space$_{M-1}$**
8	**n-dimension m-dimension n-displacement m-displacement**
9	**dimension displacement**
10	**dimension displacement$_1$ ... displacement$_{P-1}$**
11	**dimension grid displacement$_1$ ... displacement$_{P-1}$**

x-dimension, **y-dimension**, **x-space**, **y-space**, **dimension**, **n-dimension**, **m-dimension**, and **grid** are all *unsigned-integers*. **displacement**, **n-displacement**, and **m-displacement** are *g-deltas*.

7.6.2 **TYPE 0** indicates that the previous repetition description, stored in modal variable *repetition*, is to be re-used. (See section 10 on page 12.) No additional values are stored with this type.

7.6.3 **TYPE 1** is an N-column (N > 1) by M-row (M > 1) matrix with uniform horizontal and vertical spacing between the elements. **x-dimension** is N - 2 and **y-dimension** is M - 2. The (***x-offset***, ***y-offset***) (cumulative spacing in the (horizontal,vertical) direction) of element (i,j) of the repetition (i = 0, ..., N-1 and j = 0, ..., M-1) is (i * **x-space**, j * **y-space**).

7.6.4 **TYPE 2** is an N-column (N > 1) by 1-row vector with uniform horizontal spacing between the elements. **x-dimension** is N - 2. The (***x-offset***, ***y-offset***) (cumulative spacing in the (horizontal,vertical) direction) of element i of the repetition (i = 0, ..., N-1) is (i * **x-space**, 0).

7.6.5 **TYPE 3** is a 1-column by M-row (M > 1) vector with uniform vertical spacing between the elements. **y-dimension** is M - 2. The (***x-offset***, ***y-offset***) (cumulative spacing in the (horizontal,vertical) direction) of element j of the repetition (j = 0, ..., M-1) is (0, j * **y-space**).

7.6.6 **TYPE 4** is an N-column (N > 1) by 1-row vector with (potentially) non-uniform horizontal spacing between the elements. **x-dimension** is N - 2. The (***x-offset***, ***y-offset)*** (cumulative spacing in the (horizontal,vertical) direction) of element i of the repetition (i = 0, ..., N-1) is ($\mathbf{x\text{-}space}_0$ + ... + $\mathbf{x\text{-}space}_i$, 0), with $\mathbf{x\text{-}space}_0$ = 0.

7.6.7 **TYPE 5** is identical to **TYPE 4**, except that all offset values must be multiplied by **grid** during expansion of the repetition.

7.6.8 **TYPE 6** is a 1-column by M-row (M > 1) vector with (potentially) non-uniform vertical spacing between the elements. **y-dimension** is M - 2. The (***x-offset***, ***y-offset***) (cumulative spacing in the (horizontal,vertical) direction) of element j of the repetition (j = 0, ..., M-1) is **(**0, $\mathbf{y\text{-}space}_0$ + ... + $\mathbf{y\text{-}space}_j$), with $\mathbf{y\text{-}space}_0$ = 0.

7.6.9 **TYPE 7** is identical to **TYPE 6**, except that all offset values must be multiplied by **grid** during expansion of the repetition.

7.6.10 **TYPE 8** is an N (N > 1) by M (M > 1) repetition with uniform and (potentially) diagonal displacements between the elements. **n-dimension** is N - 2 and **m-dimension** is M - 2. Defining **n-displacement** in terms of its components **nx-space** and **ny-space** (and similarly for **m-displacement**), the (***x-offset***, ***y-offset***) (cumulative spacing in the (horizontal,vertical) direction) of element (i,j) of the repetition (i = 0, ..., N-1 and j = 0, ..., M-1) is (i * **nx-space** + j * **mx-space**, i * **ny-space** + j * **my-space**).

7.6.11 **TYPE 9** is a P-element (P > 1) repetition with uniform and (potentially) diagonal displacements between the elements. **dimension** is P - 2. Defining **displacement** in terms of its components **x-space** and **y-space**, the (***x-offset***, ***y-offset***) (cumulative spacing in the (horizontal,vertical) direction) of element k of the repetition (k = 0, ..., P-1) is (k * **x-space**, k * **y-space**).

7.6.12 **TYPE 10** is a P-element (P > 1) repetition with (potentially) non-uniform and arbitrary two-dimensional displacements between the elements. **dimension** is P - 2. Defining $\mathbf{displacement}_k$ in terms of its components $\mathbf{x\text{-}space}_k$ and $\mathbf{y\text{-}space}_k$, the (***x-offset***, ***y-offset***) (cumulative spacing in the (horizontal,vertical) direction) of element k of the repetition (k = 0, ..., P-1) is ($\mathbf{x\text{-}space}_0$ + ... + $\mathbf{x\text{-}space}_k$, $\mathbf{y\text{-}space}_0$ + ... + $\mathbf{y\text{-}space}_k$) with $\mathbf{x\text{-}space}_0$ = $\mathbf{y\text{-}space}_0$ = 0).

7.6.13 **TYPE 11** is identical to **TYPE 10**, except that all offset values must be multiplied by **grid** during expansion of the repetition.

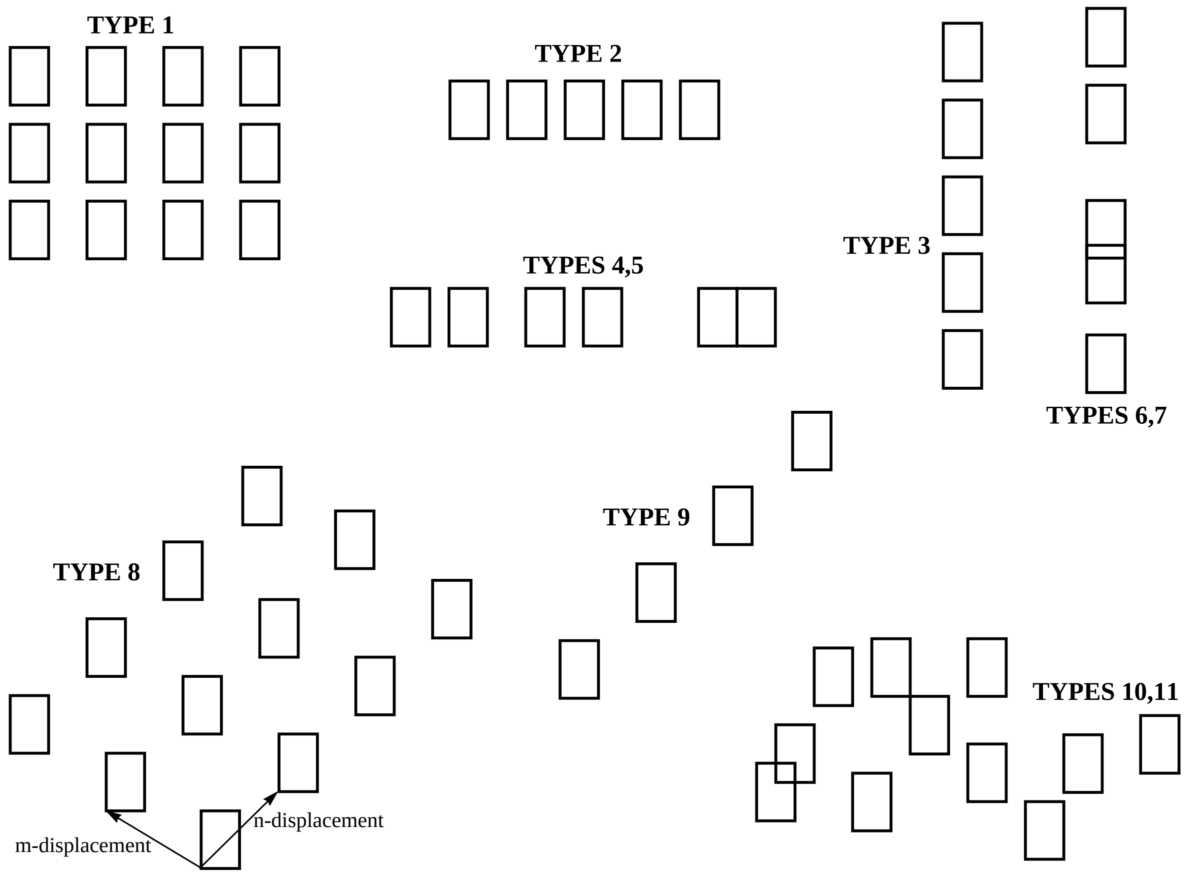

Figure 7-5
Repetition Types

7.6.14 EXCEPTION HANDLING: A *repetition* type outside the range of 0 to 11 should be treated as a fatal error. A *repetition* type of 0 may not be the first *repetition* type used within a cell.

7.7 POINT LISTS

7.7.1 A ***point-list*** represents a list of geometric coordinates for polygons or paths, and consists of an *unsigned-integer* denoting its type, followed by a list of *deltas*, in one of several formats. The initial vertex at (**x,y**) is supplied by the **POLYGON** or **PATH** record and is not part of the *point-list*; **vertex-count** (an *unsigned-integer*) is the number of points or deltas, excluding the initial vertex and any implicit vertices.

Table 7-7: Point List Types

TYPE	FORMAT	DESCRIPTION
0	**vertex-count [1-delta [... 1-delta]]**	**Implicit manhattan delta point-list (horizontal-first)**
1	**vertex-count [1-delta [... 1-delta]]**	**Implicit manhattan delta point-list (vertical-first)**
2	**vertex-count [2-delta [... 2-delta]]**	**Explicit manhattan delta point-list**
3	**vertex-count [3-delta [... 3-delta]]**	**Explicit octangular delta point-list**
4	**vertex-count [g-delta [... g-delta]]**	**Explicit all-angle delta point-list**
5	**vertex-count [g-delta [... g-delta]]**	**Explicit all-angle double-delta point-list**

7.7.2 A *point-list* of type 0 consists of a list of *1-deltas*, representing alternating horizontal and vertical relative displacements, with the first displacement implicitly horizontal. When describing a polygon *point-list* in this form, the *final two* displacements are omitted, since they can be unambiguously implied from the current point, the last edge, and the starting point. When describing a polygon, **vertex-count** must be an even number greater than or equal to 2.

7.7.3 A *point-list* of type 1 consists of a list of *1-deltas*, representing alternating vertical and horizontal relative displacements, with the first displacement implicitly vertical. When describing a polygon *point-list* in this form, the *final two* displacements are omitted, since they can be unambiguously implied from the current point, the last edge, and the starting point. When describing a polygon, **vertex-count** must be an even number greater than or equal to 2.

7.7.4 A *point-list* of type 2 consists of a list of *2-deltas*, representing a series of manhattan relative displacements. When describing a polygon *point-list* in this form, the final displacement is omitted, since the polygon is assumed to be implicitly closed, but this final implicit displacement must be a *manhattan* displacement, with either Δx=0 or Δy=0.

7.7.5 A *point-list* of type 3 consists of a list of *3-deltas*, representing a series of octangular relative displacements. When describing a polygon *point-list* in this form, the final displacement is omitted, since the polygon is assumed to be implicitly closed, but this final implicit displacement must be an *octangular* displacement at an angle that is an integral multiple of 45˚.

7.7.6 A *point-list* of type 4 consists of a list of *g-deltas*, representing a series of any-angle relative displacements. When describing a polygon *point-list* in this form, the final displacement is omitted, since the polygon is assumed to be implicitly closed.

7.7.7 A *point-list* of type 5 consists of a list of *g-deltas*, representing a series of adjustments to a relative displacement vector, with the initial vector set to (Δx=0, Δy=0). To calculate the coordinates of each successive point, the x and y components of each successive *g-delta* are added to the relative displacement vector, which in turn describes the relative displacement from the current point to the next point. When describing a polygon *point-list* in this form, the final displacement is omitted, since the polygon is assumed to be implicitly closed. This form of *point-list* is intended to allow more compact representation of polygons and paths which are approximations of large-field curvilinear figures on a fine grid, where the curvature is not extreme.

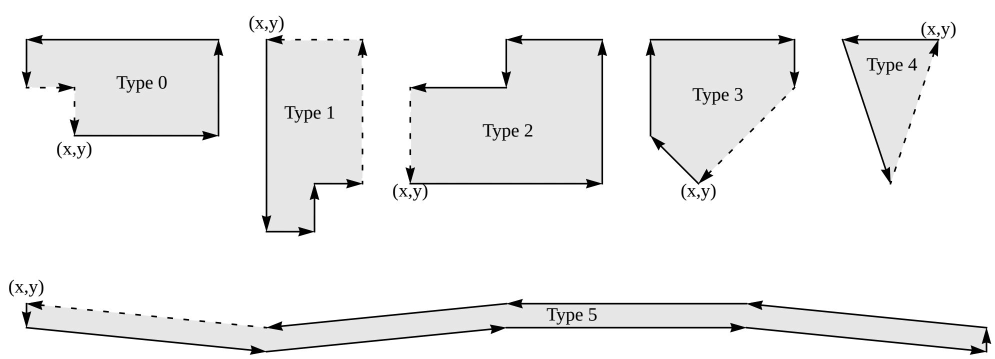

Figure 7-6
Point Lists Describing Polygons

Table 7-8: Polygon Point Lists for Figure 7-6

TYPE	BIT PATTERN
0	00000000 00000100 00001100 00001000 00010001 00000101
1	00000001 00000100 00010001 00000100 00000100 00000100
2	00000010 00000101 00100000 00011001 00010010 00001011 00010010
3	00000011 00000100 00010101 00100001 00110000 00010011
4	00000100 00000010 01000100 00001001 00001101
5	00000101 00001001 00000001 00000011 00101001 00000000 00000001 00000100 00000001 00000011 00000001 00000011 00101011 00000100 00101011 00000000 00000001 00000011 00000001 00000011

7.7.8 EXCEPTION HANDLING: A *point-list* type outside the range of 0 to 5 should be treated as a fatal error. For *point-list* types 0-1, successive coincident points and/or adjacent colinear edges are not permitted. A *non-manhattan* implicit closing vector for a polygon using *point-list* type 2, or a *non-octangular* implicit closing vector for a polygon using *point-list* type 3 should be treated as a fatal error. For polygons using *point-list* types 0-1, a vertex count which is odd or less than 2 should be treated as a fatal error.

7.8 PROPERTY VALUES

7.8.1 A ***property-value*** stores one element of a property value list. It consists of an *unsigned-integer* which encodes its type, followed by either the value itself or a reference number. Types 0-7 are *reals* which conform to the scheme described in Table 7-3 on page 5.

Table 7-9: Property Value Types

TYPE	FORMAT
0-7	**real** (see Table 7-3)
8	**unsigned-integer**
9	**signed-integer**
10	**a-string**
11	**b-string**
12	**n-string**
13	**propstring-reference-number (implied a-string)**
14	**propstring-reference-number (implied b-string)**
15	**propstring-reference-number (implied n-string)**

7.8.2 EXCEPTION HANDLING: A *property-value* type outside the range of 0 to 15 should be treated as a fatal error. Use of a **propstring-reference-number** for which there is no corresponding **PROPSTRING** record within the same OASIS file should be treated as a fatal error.

8 CELL REFERENCING

8.1 As in GDSII Stream, cells in OASIS are identified by name. The **CELL** record not only introduces a cell definition but also defines its name. **PLACEMENT** records refer by name to the cell being placed. As in GDSII Stream, there are no “anonymous” cells in OASIS.

9 LAYERS, DATATYPES, AND TEXTTYPES

9.1 As in GDSII Stream, every <geometry> has associated with it a *layer number* and a *datatype number* and every text element has associated with it a *textlayer number* and a *texttype number.*

10 MODAL VARIABLES

10.1 For compaction purposes, selected data elements in many OASIS records may be implicitly specified through the use of *modal variables* or stored state. At the beginning of the file, and whenever a **CELL** or **<name>** record is encountered, all modal variables with the exception of *placement-x*, *placement-y*, *geometry-x*, *geometry-y*, *text-x*, and *text-y*, are set to a state of *undefined*; the exceptions just mentioned are set to 0. As various elements appear in the cell's description, modal variables related to those elements are set from the elements' definitions. These modal variables can then be used implicitly by successive elements. A modal variable may hold a single value such as *geometry-w*, or a multi-variable structure such as a *repetition*.

Table 10-1: Modal Variables

MODAL VARIABLES	RELATED RECORDS
repetition	**PLACEMENT, TEXT, POLYGON, PATH, RECTANGLE, TRAPEZOID, CTRAPEZOID, CIRCLE, XGEOMETRY**
placement-x, placement-y, placement-cell	**PLACEMENT**
layer, datatype	**POLYGON, PATH, RECTANGLE, TRAPEZOID, CTRAPEZOID, CIRCLE, XGEOMETRY**
textlayer, texttype, text-x, text-y, text-string	**TEXT**
geometry-x, geometry-y	**POLYGON, PATH, RECTANGLE, TRAPEZOID CTRAPEZOID, CIRCLE, XGEOMETRY**
xy-mode	**PLACEMENT, TEXT, POLYGON, PATH, RECTANGLE, TRAPEZOID, CTRAPEZOID, CIRCLE, XGEOMETRY, XYABSOLUTE, XYRELATIVE**
geometry-w, geometry-h	**RECTANGLE, TRAPEZOID, CTRAPEZOID**
polygon-point-list	**POLYGON**
path-halfwidth, path-point-list path-start-extension, path-end-extension	**PATH**
ctrapezoid-type	**CTRAPEZOID**
circle-radius	**CIRCLE**
last-property-name, last-value-list	**PROPERTY**

10.2 Modal variable *xy-mode* governs the interpretation of the **x** and **y** fields for those related record types indicated in Table 10-1. Two interpretation modes are provided: *absolute* and *relative*. See section 21 on page 18 for a discussion of how these two modes work.

10.3 EXCEPTION HANDLING: An OASIS record which implicitly references a modal variable which is in the *undefined* state should be treated as a fatal error.

11 RECORDS

11.1 The basic unit of information in an OASIS file is a ***record***. A record consists of a single *unsigned-integer* which encodes the ***record-ID***, followed by the remainder of the record's descriptive data. In this specification, *record-ID* values are displayed as decimal numbers enclosed in apostrophes.

11.2 The **CBLOCK** record is a special case since it encapsulates a series of ordinary records in byte-compressed form. When a **CBLOCK** record is encountered while reading an OASIS file, it is first necessary to decompress its data, which will produce one or more ordinary records, which can in turn be decoded. For more information on **CBLOCK** records refer to section 35 on page 28.

11.3 Most records have an implicit length—the record must be parsed and decoded in order to determine its length. The **XNAME**, **XELEMENT**, and **XGEOMETRY** records are exceptions to this. They encapsulate all of their user-defined data in a single variable-length *b-string*, so they can be used for prototyping new record types, hiding embedded proprietary data, supporting local non-interoperable extensions, etc. without rendering an OASIS file illegible to older readers, which can simply note the string length and skip over the record.

11.4 EXCEPTION HANDLING: OASIS processors should treat the nesting of a **CBLOCK** record within another **CBLOCK** record as a fatal error.

12 PAD RECORD

12.1 A **PAD** record provides a simple way to reserve space within an OASIS file. It has the following format:

‘0’

12.2 **PAD** records may be inserted between any other two records.

12.3 EXCEPTION HANDLING: The presence of a **PAD** record before the **START** record or after the **END** record should be treated as a fatal error.

13 START RECORD

13.1 A **START** record identifies the beginning of an OASIS file, and immediately follows the <magic-bytes> sequence described in section 6.4 on page 3. It has the following format:

‘1’ version-string unit offset-flag [table-offsets]

13.2 The **version-string** is an *a-string* whose value is “1.0” for this version of the OASIS specification. Version “1.0” corresponds to the OASIS format as described in this document.

13.3 The **unit** declaration is a positive *real* number which specifies the global precision of the OASIS file’s coordinate system in grid steps per micron. The OASIS **unit** value is essentially the reciprocal of the first value in the GDSII Stream UNITS record.

13.4 **offset-flag** (an *unsigned-intege*r) is 0 when the **table-offsets** structure is stored in the **START** record; **offset-flag** is 1 when the **table-offsets** structure is instead stored in the **END** record. The option of storing **table-offsets** in the **END** record is provided to make it possible to write an OASIS file sequentially, with no seek-and-update access required, while still providing cell-level random-access capability for subsequent readers of that OASIS file.

13.5 The **table-offsets** structure consists of 6pairs of *unsigned-integers*. Each pair consists of a *flag* field, and a corresponding *byte-offset* field, in the following order:

Table 13-1: Table Offset Order

FLAG	BYTE-OFFSET
cellname-flag	cellname-offset
textstring-flag	textstring-offset
propname-flag	propname-offset
propstring-flag	propstring-offset
layername-flag	layername-offset
xname-flag	xname-offset

13.6 Each of the *flag* fields is either 1, indicating *strict* mode, or 0, indicating *non-strict* mode, for its respective table. The corresponding *byte-offset* field indicates the position of the first record of its respective table relative to the first byte (byte 0) of the OASIS file. A *byte-offset* of 0 indicates the absence of that particular table.

13.7 In *non-strict* mode, records of the corresponding type may occur anywhere in the file, even if some of them have been gathered into a table pointed to by the corresponding *byte-offset*.

13.8 In *strict* mode, all records of the corresponding type (plus any associated **PROPERTY** records) have been gathered into a single contiguous table pointed to by the corresponding *byte-offset*. **PAD** records are also permitted in *strict* mode tables. In addition, *strict* mode guarantees that all references to the corresponding class of objects (names, strings, or cells) are made exclusively by *reference-number.*

13.9 When a given *strict* mode table has been encapsulated within one or more **CBLOCK** records, the corresponding *byte-offset* should point to the first byte of the first **CBLOCK** record containing that table, and the first record of the table must be the first record which appears after decompression of the **CBLOCK** record. Adherence to this requirement means that it is not permissible to encapsulate more than one *strict* mode table within a single **CBLOCK** record, nor is it permissible to begin a *strict* mode table in the middle of a **CBLOCK** record.

13.10 EXCEPTION HANDLING: The absence of a **START** record as the first record in an OASIS file should be treated as a fatal error. A value of **unit** which is *NaN*, *Inf*, or non-positive, should also be treated as a fatal error. When a given table offset is nonzero and the table is flagged as *strict*, the presence of a “stray” record of that type located discontiguously from its tabular group should be treated as a fatal error, and any records which fail to use *reference-number* access for that class of objects should be treated as a fatal error. An OASIS reader which does not rely on any of the record grouping, *reference-number*, and *byte-offset* guarantees provided by *strict* mode is not required to detect and report any exceptions related to *strict* mode.

14 END RECORD

14.1 An **END** record identifies the end of the OASIS file. The **END** record must be the last record in the file; no trailing bytes are permitted. It has the following format:

‘2’ [table-offsets] padding-string validation-scheme [validation-signature]

14.2 The presence of the **table-offsets** structure is governed by **offset-flag** in the **START** record (see section 13). The **padding-string** (a *b-string*) must be sized and inserted by the OASIS writer so that the total byte length of the **END** record, including the *record-ID*, is exactly 256 bytes. This makes it possible for an OASIS reader to find the END record (and any **table-offsets** and **validation-signature**) using a relative seek from the logical end-of-file, avoiding the need to store a forward pointer in the **START** record. The contents of **padding-string** should be initialized to NUL characters.

14.3 **validation-scheme** is an *unsigned-integer* which selects the validation scheme used, and **validation-signature** is an optional scheme-dependent group of bytes used for validating the integrity of the OASIS file. The following validation schemes are defined:

Table 14-1: END Record Validation Schemes

SCHEME	DESCRIPTION	VALIDATION SIGNATURE LENGTH
0	No Validation	0
1	CRC32	4
2	CHECKSUM32	4

14.4 CRC32 Validation

14.4.1 The CRC32 polynomial is specified in ISO 3309:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + x^{0}$$

With the left-most bit representing the most significant bit, this corresponds to a value of:

binary 1 0000 0100 1100 0001 0001 1101 1011 0111
hexadecimal 104c11db7

14.4.2 The CRC32 value is computed using all of the bytes in the OASIS file from the first byte of the **START** record to the **END** record's **validation-scheme** integer. It is byte-order dependent. The resulting 32-bit word is stored in the last 4 bytes of the file, with the least significant byte first. This calculation is usually implemented using a table-lookup shift/XOR method. See Appendix 1 for sample C-language source code.

14.5 CHECKSUM32 Validation

14.5.1 The CHECKSUM32 validation signature is computed as a simple unsigned arithmetic summation of all of the bytes in the OASIS file from the first byte of the **START** record to the **END** record's **validation-scheme** integer. This value is then truncated to its least significant 32 bits and stored in the last 4 bytes of the file, with the least significant byte first. It is not byte-order dependent, and this characteristic makes it somewhat easier to calculate if the file is not written sequentially. It is, however, far less effective than CRC32 for detecting errors. See Appendix 1 for sample C-language source code.

14.6 EXCEPTION HANDLING: OASIS processors should treat the absence of an **END** record in an OASIS file as a fatal error.

15 CELLNAME RECORD

15.1 A **CELLNAME** record associates the name of a cell with a unique reference number. This allows **CELL** and **PLACEMENT** records, if desired, to avoid redundantly storing the actual text of the cell name and instead refer to the cell by its assigned reference number. It has the following format:

'3' cellname-string
'4' cellname-string reference-number

15.2 **cellname-string** is an *n-string* which holds the cell name. The **reference-number** is an *unsigned-integer* which is either implicitly or explicitly assigned to the cell. Implicit assignment occurs in record type '3', by assigning sequential reference numbers beginning with 0 as each successive **CELLNAME** record is encountered. Explicit assignment occurs in record type '4'.

15.3 Two standard properties, **S_BOUNDING_BOX** and **S_CELL_OFFSET** (described in section A2-2 on page 39), may be associated with each **CELLNAME** record. When all **CELLNAME** records have been grouped into a single contiguous table in *strict* mode (as described in section 13 on page 13), with an **S_CELL_OFFSET** property for every **CELLNAME** record, the table forms a complete index of all cells in the OASIS file, suitable for random access.

15.4 Record types '3' and '4' may not both be used in the same OASIS file.

15.5 EXCEPTION HANDLING: The appearance of two **CELLNAME** records in the same file with the same number but different names, or two **CELLNAME** records in the same file with the same name but different numbers, should be treated as a fatal error. The appearance of both record types '3' and '4' in the same OASIS file should be treated as a fatal error. The presence of more than one **S_CELL_OFFSET** or **S_BOUNDING_BOX** property after a given **CELLNAME** record should be treated as a fatal error.

16 TEXTSTRING RECORD

16.1 A **TEXTSTRING** record associates a text string with a unique reference number. This allows **TEXT** records, if desired, to avoid redundantly storing the actual text of the string and instead refer to the string by its assigned reference number. It has the following format:

'5' text-string
'6' text-string reference-number

16.2 **text-string** is an *a-string* which holds the text string. The **reference-number** is an *unsigned-integer* which is either implicitly or explicitly assigned to the text string. Implicit assignment occurs in record type '5', by assigning sequential reference numbers beginning with 0 as each successive **TEXTSTRING** record is encountered. Explicit assignment occurs in record type '6'.

16.3 Record types '5' and '6' may not both be used in the same OASIS file.

16.4 EXCEPTION HANDLING: The appearance of two **TEXTSTRING** records in the same file with the same number but different names, or two **TEXTSTRING** records in the same file with the same name but different numbers, should be treated as a fatal error. The appearance of both record types '5' and '6' in the same OASIS file should be treated as a fatal error.

17 PROPNAME RECORD

17.1 A **PROPNAME** record associates the name of a property with a unique reference number. This allows **PROPERTY** records, if desired, to avoid redundantly storing the actual text of the property name and instead refer to the property name by its assigned reference number. It has the following format:

'7' propname-string
'8' propname-string reference-number

17.2 **propname-string** is an *n-string* which holds the property name. The **reference-number** is an *unsigned-integer* which is either implicitly or explicitly assigned to the property name. Implicit assignment occurs in record type '7', by assigning sequential reference numbers beginning with 0 as each successive **PROPNAME** record is encountered. Explicit assignment occurs in record type '8'.

17.3 Record types '7' and '8' may not both be used in the same OASIS file.

17.4 EXCEPTION HANDLING: The appearance of two **PROPNAME** records in the same file with the same number but different names, or two **PROPNAME** records in the same file with the same name but different numbers, should be treated as a fatal error. The appearance of both record types '7' and '8' in the same OASIS file should be treated as a fatal error.

18 PROPSTRING RECORD

18.1 A **PROPSTRING** record associates a property string with a unique reference number. This allows **PROPERTY** records, if desired, to avoid redundantly storing the actual text of the property string and instead refer to the property string by its assigned reference number. It has the following format:

'9' prop-string
'10' prop-string reference-number

18.2 **prop-string** is an *a-string*, *b-string* , or *n-string* which holds the property string, depending on the referencing **PROPERTY** record. The **reference-number** is an *unsigned-integer* which is either implicitly or explicitly assigned to the property string. Implicit assignment occurs in record type '9', by assigning sequential reference numbers beginning with 0 as each successive **PROPSTRING** record is encountered. Explicit assignment occurs in record type '10'.

18.3 Record types '9' and '10' may not both be used in the same OASIS file.

18.4 EXCEPTION HANDLING: The appearance of two **PROPSTRING** records in the same file with the same number but different names should be treated as a fatal error. The appearance of both record types '9' and '10' in the same OASIS file should be treated as a fatal error.

19 LAYERNAME RECORD

19.1 A **LAYERNAME** record provides a means of mapping numeric (layer,datatype) and (layer,texttype) combinations to layer names. It has the following format:

'11' layername-string layer-interval datatype-interval
'12' layername-string textlayer-interval texttype-interval

19.2 Record type '11' maps a range of (layer,datatype) numbers to a layer name, and record type '12' maps a range of (textlayer,texttype) numbers to a layer name.

19.3 **layername-string** is an *n-string* containing the layer name.

19.4 Each of the interval fields consists of an *unsigned-integer* denoting the interval type, followed by 0, 1, or 2 *unsigned-integers* representing the bounds of that interval as follows:

Table 19-1: LAYERNAME Interval Types

TYPE	BOUNDS	IMPLIED RANGE
0		0 to ∞
1	**bound-a**	0 to **bound-a**
2	**bound-a**	**bound-a** to ∞
3	**bound-a**	**bound-a**
4	**bound-a bound-b**	**bound-a** to **bound-b**

19.5 **LAYERNAME** records may be repeated for the same layer name. The complete mapping for a layer name is formed by the union of all layer, datatype, textlayer, and texttype ranges associated with that name.

20 CELL RECORD

20.1 A **CELL** record introduces a cell definition. It has the following format:

'13' reference-number
'14' cellname-string

20.2 In record type '13', **reference-number** is an *unsigned-integer* referring to a **CELLNAME** record where the cell name is stored. In record type '14', **cellname-string** stores the cell name locally. In either representation, the cell name must be an *n-string*.

20.3 All subsequent records in the file up to the next **CELL**, **END,** or **<name>** record are considered to be part of that cell.

20.4 EXCEPTION HANDLING: Use of a **reference-number** for which there is no corresponding **CELLNAME** record within the same OASIS file should be treated as a fatal error. Multiple **CELL** records within a single file which refer to the same cell name (in effect, a duplicate cell definition) should also be treated as a fatal error.

21 XYABSOLUTE & XYRELATIVE RECORDS

21.1 The **XYABSOLUTE** and **XYRELATIVE** records control the value of modal variable *xy-mode*, which in turn governs the interpretation of the **x** and **y** values found in **PLACEMENT**, **<geometry>**, and **TEXT** records. They consist simply of a *record-ID* with no additional fields:

'15' = XYABSOLUTE
'16' = XYRELATIVE

21.2 When each **CELL** record is encountered, modal variable *xy-mode* is set to *absolute*, and related modal position variables *placement-x*, *placement-y*, *geometry-x*, *geometry-y*, *text-x*, and *text-y* are set to 0. The presence of an **XYRELATIVE** record forces modal variable *xy-mode* to *relative*, and the presence of an **XYABSOLUTE** record forces modal variable *xy-mode* to *absolute*. This mode may be changed any number of times within a cell definition.

21.3 In *absolute* mode, explicit **x** and **y** values, when present, are used directly as the actual (x,y) coordinates.

21.4 In *relative* mode, explicit **x** and **y** values, when present, are interpreted as relative displacements from the stored position information in modal variables *placement-x*, *placement-y*, *geometry-x*, *geometry-y*, *text-x*, or *text-y*, depending on the record type in which they occur. In this mode, the actual x-coordinate is computed as the sum of the **x** value and its corresponding modal position variable, and the actual y-coordinate is computed as the sum of the **y** value and its corresponding modal position variable.

21.5 In both *absolute* and *relative* modes, when an **x** or **y** value is not explicitly present in the record, the value of the corresponding modal position variable is used for the actual x or y coordinate. In both *absolute* and *relative* modes, the corresponding modal position variables are always updated with the actual (x,y) coordinate position.

21.6 The interpretation of *point-lists* and *repetitions* does not depend on *absolute* or *relative* mode. Also, even when a given element includes a *repetition*, the corresponding modal position variables (*placement-x*, *placement-y*, *geometry-x*, *geometry-y*, *text-x*, or *text-y*) are always updated with the actual (x,y) coordinate of the initial element.

22 PLACEMENT RECORD

22.1 A **PLACEMENT** record describes one or more placements of the referenced cell within the current cell. It has the following format:

'17' placement-info-byte [reference-number | cellname-string] [x] [y] [repetition]

**'18' placement-info-byte [reference-number | cellname-string] [magnification] [angle]
[x] [y] [repetition]**

22.2 In record type '17', **placement-info-byte** contains the bit pattern '**CNXYRAAF**'.

22.3 In record type '18', **placement-info-byte** contains the bit pattern '**CNXYRMAF**'.

22.4 When **C**=1, the cell reference is explicit, in which case **N**=1 means that **reference-number** (an *unsigned-integer*) is present, and refers to a **CELLNAME** record where the cell name is stored; **N**=0 means that **cellname-string** (an *n-string*) is present and stores the cell name locally. When **C**=0, **N** is ignored, and the value of modal variable *placement-cell* is used, referring to the same cell as the previous **PLACEMENT** record.

22.5 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the placement. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *placement-x* or *placement-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y**.

22.6 **R** is 1 if **repetition** is present. **F**=1 indicates reflection (or flip) about the x-axis; **F**=0 indicates no flip.

22.7 In record type ‘17’, magnification is 1.0 and rotation is a counterclockwise integral multiple of 90 degrees: **AA**=0 for 0 degrees, **AA**=1 for 90 degrees, **AA**=2 for 180 degrees, and **AA**=3 for 270 degrees.

22.8 In record type ‘18’, magnification and rotation are reals; **angle** is dimensioned in degrees, with positive values denoting a counterclockwise rotation; **magnification** is, of course, unitless. **A** is 1 if **angle** is present, otherwise the rotation defaults to 0 degrees. **M** is 1 if **magnification** is present, otherwise the magnification defaults to 1.0.

22.9 Each successive **PLACEMENT** record updates all placement-related modal variables.

22.10 EXCEPTION HANDLING: Use of a **reference-number** for which there is no corresponding **CELLNAME** record should be treated as a fatal error. Any recursive cell reference (a cell placing a copy of itself within itself) should be treated as a fatal error. Magnification values which are negative or zero should be treated as fatal errors. Floating point values of *NaN* or *Inf* for either magnification or angle should be treated as fatal errors. **PLACEMENT** records may refer to **CELL** records regardless of their relative location within the file, and may also refer to *external* cells which are not defined in the same file.

23 PLACEMENT TRANSFORM REPRESENTATION

23.1 EDA applications generally define a placement ***transform*** as a 3x3 matrix:

$$\mathbf{T} = \begin{bmatrix} \mathbf{X00} & \mathbf{X01} & \mathbf{0} \\ \mathbf{X10} & \mathbf{X11} & \mathbf{0} \\ \mathbf{X20} & \mathbf{X21} & \mathbf{1} \end{bmatrix}$$

which transforms any point (p,q) via left-multiplication by the 1x3 row matrix [p q 1]. Conversion of OASIS placement data to this form is defined as follows:

X00 = cos(angle) * magnification
X01 = sin(angle) * magnification
X10 = -f * sin(angle) * magnification
X11 = +f * cos(angle) * magnification
X20 = **x**
X21 = **y**

where f=1 if **F**=0, f=-1 if **F**=1, “angle” is the rotation angle given by either **AA** or **angle** in the **PLACEMENT** record, and “magnification” is **magnification** if specified, else 1.0. Note that if the rotation is a multiple of 90 degrees and the magnification is 1.0, then the upper 2x2 sub-matrix takes one of the following eight forms and OASIS processors may optimize accordingly:

Table 23-1: Standard Placement Values

F	angle	X00	X01	X10	X11
0	0˚	+1	0	0	+1
1	0˚	+1	0	0	-1
0	90˚	0	+1	-1	0
1	90˚	0	+1	+1	0
0	180˚	-1	0	0	-1
1	180˚	-1	0	0	+1
0	270˚	0	-1	+1	0
1	270˚	0	-1	-1	0

23.2 When **repetition** is present, the above transform is that of the first element of the repetition. In general, the transform of any element **E** of the repetition is computed by right-multiplying the transform of the first element by the matrix:

$$\mathbf{S} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \textit{x-offset} & \textit{y-offset} & \mathbf{1} \end{bmatrix}$$

to yield:

$$\begin{bmatrix} \mathbf{X00} & \mathbf{X01} & \mathbf{0} \\ \mathbf{X10} & \mathbf{X11} & \mathbf{0} \\ (\mathbf{X20} + \textit{\textbf{x-offset}}) & (\mathbf{X21} + \textit{\textbf{y-offset}}) & \mathbf{1} \end{bmatrix}$$

(Refer to section 7.6.3 and subsequent paragraphs beginning on page 8 for a discussion of how ***x-offset*** and ***y-offset*** are determined for the various **repetition** types.)

24 TEXT RECORD

24.1 A **TEXT** record represents a text element, consisting of an (x,y) coordinate point and an annotation string. It has the following format:

‘19’ text-info-byte [reference-number | text-string] [textlayer-number] [texttype-number] [x] [y] [repetition]

24.2 The **text-info-byte** contains the bit pattern ‘**0CNXYRTL**’.

24.3 When **C**=1, the text reference is explicit, in which case **N**=1 means that **reference-number** (an *unsigned-integer*) is present, and refers to a **TEXTSTRING** record where the text string is stored; **N**=0 means that **text-string** (an *a-string*) is present and stores the text string locally. When **C**=0, **N** is ignored, and the value of modal variable *text-string* is used instead.

24.4 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the text element. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *text-x* or *text-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y**.

24.5 **R** is 1 if **repetition** is present. **L** is 1 if **textlayer-number** is present. **T** is 1 if **texttype-number** is present. Both **textlayer-number** and **texttype-number** are *unsigned-integers*. When **textlayer-number** and/or **texttype-number** are unspecified, they assume the value of modal variables *textlayer* and *texttype*, respectively.

24.6 Each successive **TEXT** record updates all text-related modal variables.

24.7 EXCEPTION HANDLING: Use of a **reference-number** for which there is no corresponding **TEXTSTRING** record within the same OASIS file should be treated as a fatal error. Implicit use of modal variables *textlayer* or *texttype* when they are in the undefined state should be treated as a fatal error.

25 RECTANGLE RECORD

25.1 A **RECTANGLE** record represents a rectangular figure whose edges are parallel to the x- and y-axes. It has the following format:

'20' rectangle-info-byte [layer-number] [datatype-number] [width] [height] [x] [y] [repetition]

25.2 The **rectangle-info-byte** contains the bit pattern '**SWHXYRDL**'.

25.3 **R** is 1 if **repetition** is present. **L** is 1 if **layer-number** is present. **D** is 1 if **datatype-number** is present. Both **layer-number** and **datatype-number** are *unsigned-integers*. When **layer-number** and/or **datatype-number** are unspecified, they assume the value of modal variables *layer* and *datatype*, respectively. **W** is 1 if **width** is present. **H** is 1 if **height** is present. Both **width** and **height** are *unsigned-integers*. When **width** and/or **height** are unspecified, they assume the value of modal variables *geometry-w* and *geometry-h*, respectively.

25.4 **S** is 1 if the rectangle is a square. In this case, **H** *must* be 0, and **width**, if present, is used for both dimensions of the rectangle. When **width** is unspecified, the value of modal variable *geometry-w* is used instead.

25.5 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the lower-left corner of the rectangle. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *geometry-x* or *geometry-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y**.

25.6 Each successive **RECTANGLE** record updates all rectangle-related modal variables. (When **S**=1, both *geometry-w* and *geometry-h* are set to the rectangle's width.)

25.7 EXCEPTION HANDLING: Implicit use of modal variables *geometry-w*, *geometry-h*, *layer*, or *datatype* when they are in the undefined state should be treated as a fatal error. When **S**=1, **H**=1 should be treated as a fatal error. The interpretation of zero-area **RECTANGLE**s is application-dependent.

26 POLYGON RECORD

26.1 A **POLYGON** record represents an arbitrary polygon figure. It has the following format:

'21' polygon-info-byte [layer-number] [datatype-number] [point-list] [x] [y] [repetition]

26.2 The **polygon-info-byte** contains the bit pattern '**00PXYRDL**'.

26.3 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the initial vertex of the polygon. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *geometry-x* or *geometry-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y**.

26.4 **R** is 1 if **repetition** is present. **L** is 1 if **layer-number** is present. **D** is 1 if **datatype-number** is present. Both **layer-number** and **datatype-number** are *unsigned-integers*. When **layer-number** and/or **datatype-number** are unspecified, they assume the value of modal variables *layer* and *datatype*, respectively.

26.5 **P** is 1 if **point-list** is present. Otherwise, the value of modal variable *polygon-point-list* is used. The format of *point-lists* is defined in section 7.7 on page 9.

26.6 Each successive **POLYGON** record updates all polygon-related modal variables.

26.7 <u>EXCEPTION HANDLING</u>: Polygons with fewer than three vertices should be treated as fatal errors. Implicit use of modal variables *polygon-point-list, layer,* or *datatype* when they are in the undefined state should be treated as a fatal error. The interpretation of self-intersecting polygons, reentrant polygons, and polygons with zero-area regions is application-dependent.

27 PATH RECORD

27.1 A **PATH** record represents an arbitrary path figure, which may be thought of as a polyline with finite width. It has the following format:

> **'22' path-info-byte [layer-number] [datatype-number] [half-width]**
> **[extension-scheme [start-extension] [end-extension]] [point-list] [x] [y] [repetition]**

27.2 The **path-info-byte** contains the bit pattern '**EWPXYRDL**'.

27.3 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the initial vertex of the path centerline. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *geometry-x* or *geometry-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y.**

27.4 **R** is 1 if **repetition** is present. **L** is 1 if **layer-number** is present. **D** is 1 if **datatype-number** is present. Both **layer-number** and **datatype-number** are *unsigned-integers*. When **layer-number** and/or **datatype-number** are unspecified, they assume the value of modal variables *layer* and *datatype*, respectively.

27.5 **P** is 1 if **point-list** is present. Otherwise, the value of modal variable *path-point-list* is used. The format of *point-lists* is defined in section 7.7 on page 9.

27.6 **W** is 1 if **half-width** (an *unsigned-integer*) is present; if absent, the half-width value assumes the value of modal variable *path-halfwidth*. The path is formed by expanding the centerline (represented by line segments connecting the points) by the half-width value to each side.

27.7 **E** is 1 if **extension-scheme** is present. Otherwise, **extension-scheme**, **start-extension**, and **end-extension** are absent, and the values of modal variables *path-start-extension*, and *path-end-extension* are used instead.

27.8 When present, **extension-scheme** (an *unsigned-integer*) contains bit pattern '**0000SSEE**'. The **SS** bits govern the path starting extension, and the **EE** bits govern the path ending extension. Both **start-extension** (present only when **SS**='11') and **end-extension** (present only when **EE**='11') are *signed-integers*, as in GDSII Stream, with positive values causing the path to extend beyond its starting and/or ending vertices, and negative values causing the path to retract from its starting and/or ending vertices.

Table 27-1: Path Extension Schemes

SS BITS	DESCRIPTION
00	Use *path-start-extension* modal variable
01	Use flush (zero-length) extension at starting vertex
10	Use path-halfwidth extension at starting vertex
11	Use explicit **start-extension** at starting vertex
EE BITS	
00	Use *path-end-extension* modal variable
01	Use flush (zero-length) extension at ending vertex
10	Use path-halfwidth extension at ending vertex
11	Use explicit **end-extension** at ending vertex

27.9 Each successive **PATH** record updates all path-related modal variables.

27.10 Various types of degenerate paths, where the half-width=0, the path traces back on itself, an extension is negative with magnitude greater than its segment length, etc. are not prohibited; their interpretation is application-dependent. The path expansion scheme used at the path's interior vertices or "joints" is also application-dependent.

27.11 EXCEPTION HANDLING: Implicit use of modal variables *path-halfwidth, path-point-list, path-start-extension, path-end-extension, layer,* or *datatype* when they are in the undefined state should be treated as a fatal error.

28 TRAPEZOID RECORD

28.1 A **TRAPEZOID** record represents a trapezoid figure (a polygon with four vertices having at least two opposite sides parallel and parallel to either the x- or the y-axis). It has the following format:

'23' trap-info-byte [layer-number] [datatype-number]
[width] [height] delta-a delta-b [x] [y] [repetition]

'24' trap-info-byte [layer-number] [datatype-number]
[width] [height] delta-a [x] [y] [repetition]

'25' trap-info-byte [layer-number] [datatype-number]
[width] [height] delta-b [x] [y] [repetition]

28.2 The **trap-info-byte** contains bit pattern '**OWHXYRDL**'.

28.3 **R** is 1 if **repetition** is present. **L** is 1 if **layer-number** is present. **D** is 1 if **datatype-number** is present. Both **layer-number** and **datatype-number** are *unsigned-integers*. When **layer-number** and/or **datatype-number** are unspecified, they assume the value of modal variables *layer* and *datatype*, respectively. **W** is 1 if **width** is present. **H** is 1 if **height** is present. **Width** and **height** are *unsigned-integers* which describe the overall dimensions of the bounding box of the trapezoid as shown in Figure 28-1 on page 24. When **width** and/or **height** are unspecified, they assume the value of modal variables *geometry-w* and *geometry-h*, respectively.

28.4 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the lower-left corner of the trapezoid's bounding box. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *geometry-x* or *geometry-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y**.

28.5 **delta-a** and **delta-b** are *1-deltas*, and are both present in record type '23'. In record type '24' **delta-b** is assumed to be 0 and is omitted, and in record type '25' **delta-a** is assumed to be 0 and is omitted.

28.6 **O** is 0 if the trapezoid is horizontally-oriented, with top (PQ) and bottom (RS) sides parallel to the x-axis. In this case, **delta-a** represents (x_P - x_R) and **delta-b** represents (x_Q - x_S).

28.7 **O** is 1 if the trapezoid is vertically-oriented, with left (PQ) and right (RS) sides parallel to the y-axis. In this case, **delta-a** represents (y_P - y_R) and **delta-b** represents (y_Q - y_S).

28.8 Each successive **TRAPEZOID** record updates all trapezoid-related modal variables.

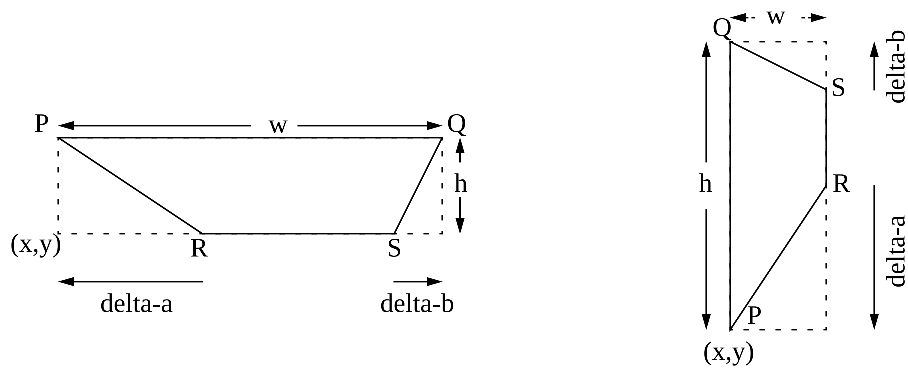

Figure 28-1
Horizontal and Vertical Trapezoids

28.9 <u>EXCEPTION HANDLING</u>: For any trapezoid, deltas of sufficient magnitude to cause segments PR and QS to cross, as well as any delta which causes either segment PR or QS not to fit diagonally within the bounding box, should be treated as fatal errors. Implicit use of modal variables *geometry-w*, *geometry-h*, *layer*, or *datatype* when they are in the undefined state should be treated as a fatal error. The interpretation of zero-area trapezoids is application-dependent.

29 CTRAPEZOID RECORD

29.1 A **CTRAPEZOID** record represents a trapezoid figure in a compact form by assuming that two sides are parallel to either the x- or the y-axis, and the remaining two sides form either a 45- or 90-degree angle with them. It has the following format:

> **'26' ctrapezoid-info-byte [layer-number] [datatype-number]**
> **[ctrapezoid-type] [width] [height] [x] [y] [repetition]**

29.2 The **ctrapezoid-info-byte** contains the bit pattern '**TWHXYRDL**'.

29.3 **R** is 1 if **repetition** is present. **L** is 1 if **layer-number** is present. **D** is 1 if **datatype-number** is present. Both **layer-number** and **datatype-number** are *unsigned-integers*. When **layer-number** and/or **datatype-number** are unspecified, they assume the value of modal variables *layer* and *datatype*, respectively. **W** is 1 if **width** is present. **H** is 1 if **height** is present. Both **width** and **height** are *unsigned-integers*, and represent the width (w) and height (h) of the trapezoid's bounding box, respectively. When **width** and/or **height** are unspecified, they assume the value of modal variables *geometry-w* and *geometry-h*, respectively.

29.4 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the lower-left corner of the trapezoid's bounding box. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *geometry-x* or *geometry-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y**.

29.5 **T** is 1 if **ctrapezoid-type** (an *unsigned-integer*) is present; otherwise it assumes the value of modal variable *ctrapezoid-type*. Types 0-25 are depicted in figure 29-1:

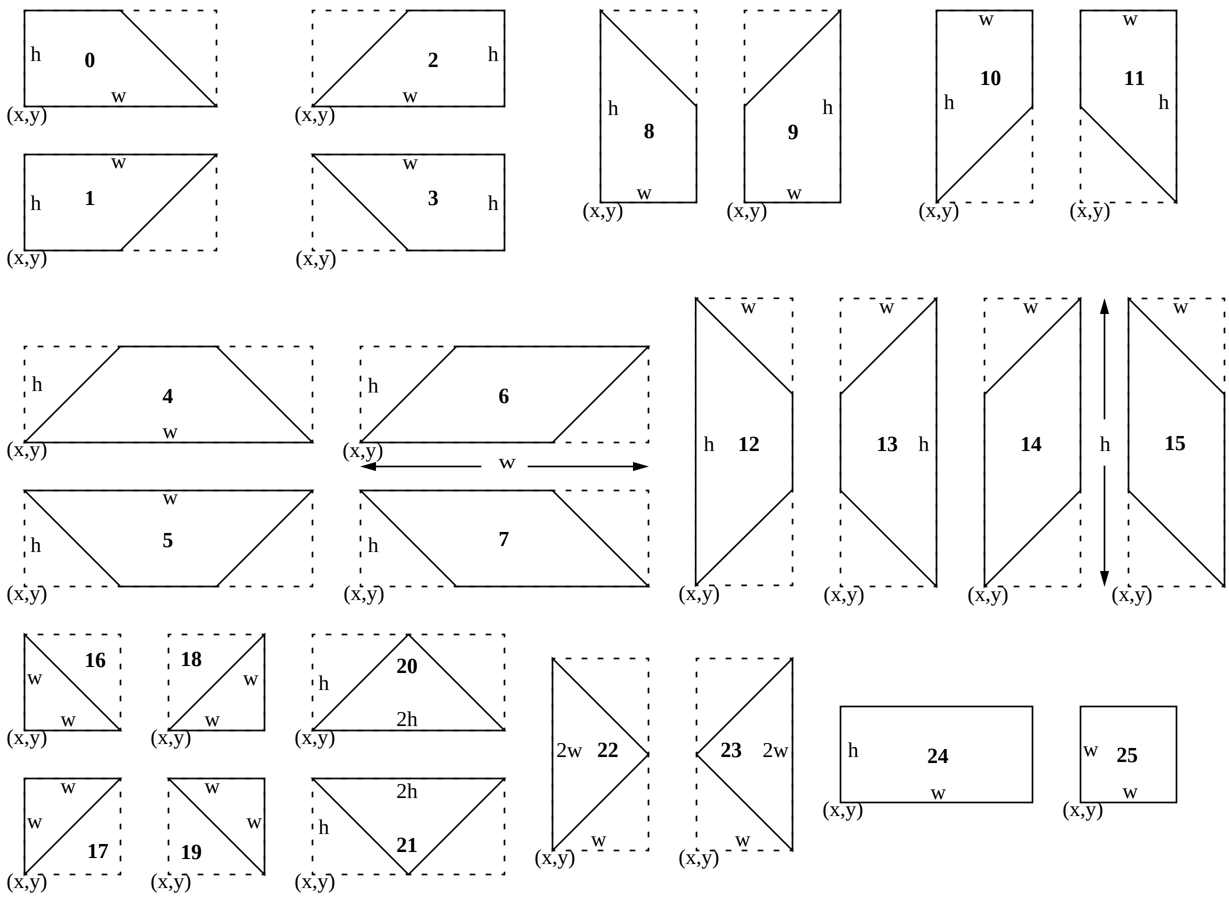

Figure 29-1
The 26 Standard CTRAPEZOID Types

29.6 The triangle, rectangle, and square forms are provided for compactness and for compatibility with some mask writing pattern file formats. For types 16-19, 22-23, and 25, **height** is not used, and **H** must be 0. For types 20-21, **width** is not used and **W** must be 0.

29.7 Each successive **CTRAPEZOID** record updates all ctrapezoid-related modal variables with the following exception: for the forms where only one of **width** or **height** is used (types 16-23 and 25), modal variables *geometry-w* or *geometry-h* are both updated to match the specified dimension.

29.8 <u>EXCEPTION HANDLING</u>: For types 0-3, (w < h) should be treated as a fatal error. For types 4-7, (w < 2h) should be treated as a fatal error. For types 8-11, (h < w) should be treated as a fatal error. For types 12-15, (h < 2w) should be treated as a fatal error. For types 16-19, 22-23, and 25, an **H** value of 1 should be treated as a fatal error. For types 20-21, a **W** value of 1 should be treated as a fatal error. A value of **ctrapezoid-type** greater than 25 should be treated as a fatal error. Implicit use of modal variables *ctrapezoid-type, geometry-w, geometry-h, layer* or *datatype* when they are in the undefined state should be treated as a fatal error. The interpretation of zero-area trapezoids is application-dependent.

30 CIRCLE RECORD

30.1 A **CIRCLE** record represents a circular figure. It has the following format:

> **'27' circle-info-byte [layer-number] [datatype-number] [radius] [x] [y] [repetition]**

30.2 The **circle-info-byte** contains the bit pattern '**00rXYRDL**'.

30.3 **R** is 1 if **repetition** is present. **L** is 1 if **layer-number** is present. **D** is 1 if **datatype-number** is present. Both **layer-number** and **datatype-number** are *unsigned-integers*. When **layer-number** and/or **datatype-number** are unspecified, they assume the value of modal variables *layer* and *datatype*, respectively.

30.4 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the circle's center. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *geometry-x* or *geometry-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y**.

30.4.1 **r** is 1 if **radius** is present, otherwise **radius** assumes the value of modal variable *circle-radius* instead.

30.5 Each successive **CIRCLE** record updates all circle-related modal variables.

30.6 <u>EXCEPTION HANDLING</u>: Implicit use of modal variables *circle-radius, layer,* or *datatype* when they are in the undefined state should be treated as a fatal error. The interpretation of zero-area **CIRCLE**s is application-dependent.

31 PROPERTY RECORD

31.1 A *property* is an annotation element consisting of a name plus an optional list of values, supplying descriptive information about the characteristics of the OASIS file or one of its components. A property may be associated with the entire OASIS file, a **<name>** record, a **CELL**, a **PLACEMENT**, or an **<element>** record within a cell. The **PROPERTY** record has the following format:

> **'28' prop-info-byte [reference-number | propname-string] [prop-value-count] [<property-value>*]**
> **'29'**

31.2 Record type '29' provides a compact way to specify a duplicate copy of the most-recently-seen property together with its value list. It makes use of modal variables *last-property-name* and *last-value-list*, which were defined by a previous **PROPERTY** record.

31.3 The **prop-info-byte** contains the bit pattern '**UUUUVCNS**'.

31.4 When **C**=1, the property name reference is explicit, in which case **N**=1 means that **reference-number** (an *unsigned-integer*) is present, and refers to a **PROPNAME** record where the property name is stored; **N**=0 means that **propname-string** (an *n-string*) is present and stores the property name locally. When **C**=0, **N** is ignored, and the value of modal variable *last-property-name* is used instead.

31.5 When **V**=0, values of **UUUU** from 0 to 14 indicate the number of **<property-value>** fields which are part of this record, and **prop-value-count** is omitted. When **V**=0 and **UUUU**=15, **prop-value-count**, an *unsigned-integer*, is present and indicates the number of **<property-value>** fields. When **V**=1, **UUUU** must be 0, and modal variable *last-value-list* supplies the value list. See section 7.8 on page 11 for a description of **<property-value>** types.

31.6 When **S**=1, a standard property is indicated; when **S**=0, a non-standard or user property is indicated. The list of OASIS Standard Properties appears in Appendix 2 on page 39. That appendix also describes how to represent GDSII-Stream-style properties using the **S_GDS_PROPERTY** standard property.

31.7 Each successive **PROPERTY** record updates modal variables *last-property-name* and *last-value-list*.

31.8 In general, **PROPERTY** records directly follow the record with which they are associated. **PROPERTY** records occurring directly after the **START** record are associated globally with the entire OASIS file. **PROPERTY** records occurring after a **CELL** record or its corresponding **CELLNAME** record pertain to that entire cell. **PROPERTY** records occurring after a **PLACEMENT** record pertain to the placement(s) it describes, including *repetitions*. **PROPERTY** records occurring after an **<element>** record pertain to that element and any *repetitions*.

31.9 **PROPERTY** records do not associate with **CBLOCK** or **PAD** records. Instead, property association occurs as though all **CBLOCK** records have been uncompressed, and all **PAD** records have been deleted.

31.10 EXCEPTION HANDLING: Implicit use of modal variables *last-property-name* or l*ast-value-list* when they are in the undefined state should be treated as a fatal error. Use of a **reference-number** for which there is no corresponding **PROPNAME** record should be treated as a fatal error.

32 XNAME RECORD

32.1 An **XNAME** record allows backward-compatible extension of OASIS **<name>** records. It associates a string with a unique reference number. It has the following format:

'30' xname-attribute xname-string
'31' xname-attribute xname-string reference-number

32.2 **xname-string** is user-defined as an *a-string*, *b-string*, or *n-string* which holds the name. **xname-attribute** is an *unsigned-integer* providing the ability to associate the **XNAME** with a user-defined class. The **reference-number** is an *unsigned-integer* which is either implicitly or explicitly assigned to the name. Implicit assignment occurs in record type '30', by assigning sequential reference numbers beginning with 0 as each successive **XNAME** record is encountered. Explicit assignment occurs in record type '31'.

32.3 Record types '30' and '31' may not both be used in the same OASIS file.

32.4 EXCEPTION HANDLING: The appearance of two **XNAME** records in the same file with the same reference number but different names should be treated as a fatal error. The appearance of both record types '30' and '31' in the same OASIS file should be treated as a fatal error.

33 XELEMENT RECORD

33.1 An **XELEMENT** record allows backward-compatible extension of OASIS **<element>** records. It has the following format:

'32' xelement-attribute xelement-string

33.2 **xelement-attribute** is an *unsigned-integer* providing the ability to associate the **XELEMENT** with a user-defined class. **xelement-string** is a *b-string* containing user-defined data.

34 XGEOMETRY RECORD

34.1 An **XGEOMETRY** record allows backward-compatible extension of OASIS **<geometry>** records. It has the following format:

> **'33' xgeometry-info-byte xgeometry-attribute**
> **[layer-number] [datatype-number] xgeometry-string [x] [y] [repetition]**

34.2 The **xgeometry-info-byte** contains the bit pattern '**000XYRDL**'.

34.3 **R** is 1 if **repetition** is present. **L** is 1 if **layer-number** is present. **D** is 1 if **datatype-number** is present. Both **layer-number** and **datatype-number** are *unsigned-integers*. When **layer-number** and/or **datatype-number** are unspecified, they assume the value of modal variables *layer* and *datatype*, respectively.

34.4 **x** and **y** are *signed-integer* coordinates representing either the *absolute* or the *relative* (x,y) location of the geometry. **X** is 1 if **x** is present, and **Y** is 1 if **y** is present. When either **x** or **y** is unspecified, the value of modal variable *geometry-x* or *geometry-y*, respectively, is used instead. Refer to section 21 on page 18 for a discussion of how *absolute* and *relative* modes affect the interpretation of **x** and **y**.

34.5 **xgeometry-attribute** is an integer providing the ability to associate the **XGEOMETRY** with a user-defined class. **xgeometry-string** is a *b-string* containing user-defined data describing the geometry.

34.6 Each successive **XGEOMETRY** record updates all **XGEOMETRY**-related modal variables.

34.7 <u>EXCEPTION HANDLING</u>: Implicit use of modal variables *layer,* or *datatype* when they are in the undefined state should be treated as a fatal error.

35 CBLOCK RECORD

35.1 A **CBLOCK** record provides a mechanism for embedding compressed data within the structure of an OASIS file for additional compactness. It has the following format:

> **'34' comp-type uncomp-byte-count comp-byte-count comp-bytes**

35.2 **comp-type** is an *unsigned-integer* describing the type of compression used for this record. **uncomp-byte-count** is an *unsigned-integer* describing the number of bytes prior to compression, and **comp-byte-count** is an *unsigned-integer* describing the number of bytes after compression. **comp-bytes** is a sequence of bytes containing the compressed byte sequence.

35.3 When **comp-type**=0, the compression scheme is the lossless DEFLATE Compressed Data Format, Version 1.3, as documented in RFC 1951 (1996). Other values of **comp-type** are reserved for future versions of the OASIS format; the intent is to be able to support a mixture of compression methods within a single OASIS file for maximum compactness.

35.3.1 One example of compression/decompression software that is compliant with RFC 1951 is found in ZLIB version 1.1.4 (March 2002). This software version can be used without any licensing or legal encumbrances. It is expected that future versions of the ZLIB software will also remain RFC-1951-compliant. Users of future releases of ZLIB are cautioned to check for continued conformance to RFC 1951 as well as any changes in the terms of use.

35.3.2 Use of the ZLIB software is *not* mandatory in order to be compliant with the OASIS specification. Any compression/decompression software that stores and processes data in conformance with RFC 1951 is OASIS-compliant. It should be noted that alternatives to the **CBLOCK** record may emerge in the future, supporting other compression mechanisms. Use of multiple compression methods within a single OASIS file is not ruled out.

35.4 The **START**, **END**, **CELL**, and nested **CBLOCK** records may not be stored within a compressed record. This maintains the ability to perform random access at the cell level within an OASIS file. A **CBLOCK** record may not encapsulate more than one "*strict* mode" name table (refer to sections 13 and 14 beginning on page 13). All other sequences of records, of any length, may be stored in a **CBLOCK** record.

35.5 EXCEPTION HANDLING: During the reading of a **CBLOCK** record, it is a fatal error if the number of bytes returned after decompression does not match **uncomp-byte-count**.

36 DETAILED BNF SYNTAX

36.1 This specification uses a modified Backus-Naur Form (BNF) notation to describe OASIS file syntax. The following table summarizes the conventions used in the modified BNF:

Table 36-1: Modified BNF Notation

SYMBOL	TERM	MEANING
ABCD	Bold Uppercase	Denotes an OASIS record name
abcd	Bold Lowercase	Denotes a fundamental data type defined in section 7
< >	Angle Brackets	Enclose an element name which is further defined elsewhere in the BNF
->	Arrow	Means "is composed of"
[ ]	Square Brackets	Enclose element(s) which are optional, and if present, occur only once
{ }	Braces	Enclose element(s) which are required
|	Vertical Bar	Indicates a choice between mutually exclusive elements within { } braces
*	Asterisk	An asterisk following an element means the element may occur zero or more times
...	Ellipsis	Appears between elements to indicate a variable-length list of like type
' '	Single Quotes	Enclose a decimal number denoting an OASIS *unsigned-integer*
" "	Double Quotes	Enclose a literal character string
"<CR>"	Control Character	Angle brackets enclose the name of an ASCII Control Character within a string
//	Double Virgule	Indicates all characters to its right are comments—not part of the syntax

36.2 The OASIS syntax is detailed as follows:

<oasis-file> -> <magic-bytes> **START** { **CBLOCK** | **PAD** | **PROPERTY** | <cell> | <name> }* **END**
<name> -> { **CELLNAME** | **TEXTSTRING** | **LAYERNAME** | **PROPNAME** | **PROPSTRING** | **XNAME** }
<cell> -> { **CELL** { **CBLOCK** | **PAD** | **PROPERTY** | **XYRELATIVE** | **XYABSOLUTE** | <element> }* }
<element> -> { <geometry> | **PLACEMENT** | **TEXT** | **XELEMENT** }
<geometry> -> { **RECTANGLE** | **POLYGON** | **PATH** | **TRAPEZOID** | **CTRAPEZOID** | **CIRCLE** | **XGEOMETRY** }

<magic-bytes> -> "`%SEMI-OASIS<CR><NL>`"

PAD -> **'0'**

START -> **'1'** <version-string> <unit> <offset-flag> [<table-offsets>]

END -> **'2'** [<table-offsets>] <padding-string> <validation-scheme> [<validation-signature>]

CELLNAME -> **'3'** <cellname-string>
CELLNAME -> **'4'** <cellname-string> <reference-number>

TEXTSTRING -> **'5'** <text-string>
TEXTSTRING -> **'6'** <text-string> <reference-number>

PROPNAME -> **'7'** <propname-string>
PROPNAME -> **'8'** <propname-string> <reference-number>

PROPSTRING -> **'9'** <prop-string>
PROPSTRING -> **'10'** <prop-string> <reference-number>

LAYERNAME -> '11' <layername-string> <layer-interval> <datatype-interval>
LAYERNAME -> '12' <layername-string> <textlayer-interval> <texttype-interval>

CELL -> **'13'** <reference-number>
CELL -> **'14'** <cellname-string>

XYABSOLUTE -> **‘15’**
XYRELATIVE -> **‘16’**

PLACEMENT -> **‘17’** <placement-info-byte> [<reference-number> | <cellname-string>]
[<x>] [<y>] [<repetition>]
PLACEMENT -> **‘18’** <placement-info-byte> [<reference-number> | cellname-string>]
[<magnification>] [<angle>] [<x>] [<y>] [<repetition>]

TEXT -> **‘19’** <text-info-byte> [<reference-number> | <text-string>] <l-t> [<x>] [<y>] [<repetition>]

RECTANGLE -> **‘20’** <rectangle-info-byte> <l-d> [<width>] [<height>] [<x>] [<y>] [<repetition>]

POLYGON -> **‘21’** <polygon-info-byte> <l-d> [<point-list>] [<x>] [<y>] [<repetition>]

PATH -> **‘22’** <path-info-byte> <l-d> [<half-width>]
[<extension-scheme> [<start-extension>] [<end-extension>]]
[<point-list>] [<x>] [<y>] [<repetition>]

TRAPEZOID -> **‘23’** <trap-info-byte> <l-d> [<width>] [<height>] <delta-a> <delta-b>
[<x>] [<y>] [<repetition>]
TRAPEZOID -> **‘24’** <trap-info-byte> <l-d> [<width>] [<height>] <delta-a>
[<x>] [<y>] [<repetition>]
TRAPEZOID -> **‘25’** <trap-info-byte> <l-d> [<width>] [<height>] <delta-b>
[<x>] [<y>] [<repetition>]

CTRAPEZOID -> **‘26’** <ctrapezoid-info-byte> <l-d> [<ctrapezoid-type>] [<width>] [<height>] [<x>] [<y>]
[<repetition>]

CIRCLE -> **‘27’** <circle-info-byte> <l-d> [<radius>] [<x>] [<y>] [<repetition>]

PROPERTY -> **‘28’** <prop-info-byte> [<reference-number> | <propname-string>]
[<prop-value-count>] [<property-value>*]
PROPERTY -> **‘29’**

XNAME -> **‘30’** <xname-attribute> <xname-string>
XNAME -> **‘31’** <xname-attribute> <xname-string> <reference-number>

XELEMENT -> **‘32’** <xelement-attribute> <xelement-string>

XGEOMETRY -> **‘33’** <xgeometry-info-byte> <xgeometry-attribute> <l-d> <xgeometry-string> [<x>] [<y>]
[<repetition>]

CBLOCK -> **‘34’** <comp-type> <uncomp-byte-count> <comp-byte-count> <comp-bytes>

<table-offsets> -> <cellname-flag> <cellname-offset>
<textstring-flag> <textstring-offset>
<propname-flag> <propname-offset>
<propstring-flag> <propstring-offset>
<layername-flag> <layername-offset>
<xname-flag> <xname-offset>

<offset-flag>, <cellname-flag>, <cellname-offset>, <textstring-flag>, <textstring-offset>,
<propname-flag>, <propname-offset>, <propstring-flag>, <propstring-offset>,
<layername-flag>, <layername-offset>, <xname-flag>, <xname-offset> -> **unsigned-integer**

<padding-string> -> **b-string**

<validation-scheme> -> **unsigned-integer**
<validation-signature> -> **byte***

<placement-info-byte>, <text-info-byte>, <rectangle-info-byte>,
<polygon-info-byte>, <path-info-byte>, <trap-info-byte>, <ctrapezoid-info-byte>,
<circle-info-byte>, <prop-info-byte>, <xgeometry-info-byte> -> **byte**

<layer-interval>, <datatype-interval>, <textlayer-interval>, <texttype-interval> -> <layer-interval>
<layer-interval> -> { <li0> | <li1> | <li2> | <li3> | <li4> }
<li0> -> **‘0’**
<li1> -> **‘1’** <bound-a>
<li2> -> **‘2’** <bound-a>
<li3> -> **‘3’** <bound-a>
<li4> -> **‘4’** <bound-a> <bound-b>
<bound-a>, <bound-b> -> **unsigned-integer**

<l-d> -> [<layer-number>] [<datatype-number>]
<l-t> -> [<textlayer-number>] [<texttype-number>]
<layer-number>, <datatype-number>, <textlayer-number>, <texttype-number> -> **unsigned-integer**

<reference-number> -> **unsigned-integer**
<cellname-string>, <propname-string>, <layername-string> -> <n-string>
<version-string>, <text-string> -> <a-string>
<prop-string>, <xname-string> -> { <a-string> | <b-string> | <n-string> }
<xelement-string>, <xgeometry-string> -> <b-string>

<a-string>, <b-string>, <n-string> -> <string-length> **byte***
<string-length> -> **unsigned-integer**

<xname-attribute>, <xelement-attribute>, <xgeometry-attribute> -> **unsigned-integer**

<property-value> -> { <pvreal> | <pv8> | <pv9> | <pv10> | <pv11> | <pv12> | <pv13> | <pv14> | <pv15> }
<pvreal> -> <real>
<pv8> -> **‘8’ unsigned-integer**
<pv9> -> **‘9’ signed-integer**
<pv10> -> **‘10’** <a-string>
<pv11> -> **‘11’** <b-string>
<pv12> -> **‘12’** <n-string>
<pv13> -> **‘13’** <reference-number> // a-string
<pv14> -> **‘14’** <reference-number> // b-string
<pv15> -> **‘15’** <reference-number> // n-string

<repetition> -> { <rep0> | <rep1> | <rep2> | <rep3> | <rep4> | < rep5> | <rep6> | <rep7> | <rep8> |
<rep9> | <rep10> | <rep11>}
<rep0> -> **‘0’**
<rep1> -> **‘1’** <x-dimension> <y-dimension> <x-space> <y-space>
<rep2> -> **‘2’** <x-dimension> <x-space>
<rep3> -> **‘3’** <y-dimension> <y-space>
<rep4> -> **‘4’** <x-dimension> <x-space> ... <x-space>
<rep5> -> **‘5’** <grid> <x-dimension> <x-space> ... <x-space>
<rep6> -> **‘6’** <y-dimension> <y-space> ... <y-space>
<rep7> -> **‘7’** <grid> <y-dimension> <y-space> ... <y-space>
<rep8> -> **‘8’** <n-dimension> <m-dimension> <n-displacement> <m-displacement>
<rep9> -> **‘9’** <dimension> <displacement>
<rep10> -> **‘10’** <dimension> <displacement> ... <displacement>
<rep11> -> **‘11’** <grid> <dimension> <displacement> ... <displacement>

<grid>, <x-dimension>, <y-dimension>, <dimension>, <n-dimension>, <m-dimension>,
<x-space>, <y-space> -> **unsigned-integer**

<displacement>, <n-displacement>, <m-displacement> -> <g-delta>

<point-list> -> { <pl0> | <pl1> | <pl2> | <pl3> | <pl4> | <pl5> }
<pl0> -> <vertex-count> <1-delta>*   // Implicit manhattan delta point-list (horizontal-first)
<pl1> -> <vertex-count> <1-delta>*   // Implicit manhattan delta point-list (vertical-first)
<pl2> -> <vertex-count> <2-delta>*   // Explicit manhattan delta point-list
<pl3> -> <vertex-count> <3-delta>*   // Explicit octangular delta point-list
<pl4> -> <vertex-count> <g-delta>*   // Explicit all-angle delta point-list
<pl5> -> <vertex-count> <g-delta>*   // Explicit all-angle double-delta point-list

<vertex-count>, <half-width>, <extension-scheme>, <ctrapezoid-type> -> **unsigned-integer**
<width>, <height>, <radius> -> **unsigned-integer**
<prop-value-count> -> **unsigned-integer**
<delta-a>, <delta-b> -> <1-delta>

<comp-type>, <uncomp-byte-count>, <comp-byte-count> -> **unsigned-integer**
<comp-bytes> -> **byte***

<x>, <y> -> **signed-integer**
<start-extension>, <end-extension> -> **signed-integer**

<unit>, <angle>, <magnification> -> <real>

<1-delta> -> **signed-integer**   // xxx...xxxd
<2-delta> -> **unsigned-integer**   // xxx...xxdd
<3-delta> -> **unsigned-integer**   // xxx...xddd
<g-delta> -> **unsigned-integer** [**unsigned-integer**]   // xxx...xxxddd0 or xxx...xxxd1 xxx...xxxd

<real> -> { <real0> | <real1> | <real2> | <real3> | <real4> | <real5> | <real6> | <real7> }
<real0> -> **‘0’ unsigned-integer**   // Positive whole number
<real1> -> **‘1’ unsigned-integer**   // Negative whole number
<real2> -> **‘2’ unsigned-integer**   // Positive reciprocal
<real3> -> **‘3’ unsigned-integer**   // Negative reciprocal
<real4> -> **‘4’ unsigned-integer unsigned-integer**   // Positive ratio
<real5> -> **‘5’ unsigned-integer unsigned-integer**   // Negative ratio
<real6> -> **‘6’ ieee-4**   // Single-precision floating point
<real7> -> **‘7’ ieee-8**   // Double-precision floating point

APPENDIX 1
CALCULATION OF VALIDATION SIGNATURES

A1-1 Sample CRC32 C-Language Source Code

The CRC32 must be calculated by processing the file contents as a single stream of bytes (CRC's are order-dependent). The CRC should be initialized by calling:

```
uint32  crc;  /* the crc value */
crc32_init(&crc);
```

As each chunk of data in written into the file, one should call :

```
byte    *buf;  /* data written to output */
size_t  len;   /* # of bytes of data written to output */

crc32_add(&crc, buf, len);
```

When the **END** record is to be written, the CRC should be calculated using the

```
<id-value> and <validation-scheme>  only.
```

The final value of the CRC32 should then be appended to the file as a 4-byte value in little-endian order.

```
#define  CHG_ENDIAN32(a)  { byte  *p, b; \
  p = (byte *) &(a); b=p[0]; p[0]=p[3]; p[3]=b; b=p[1]; p[1]=p[2]; p[2]=b; }

#ifdef BIG_ENDIAN_MACHINE
/* put calculated CRC in LITTLE_ENDIAN order (to align with byte ordering of the polynomial) */
CHG_ENDIAN32(crc);
#endif
```

```
/*
   (c) Copyright 2003 SEMI   no warranty, express or implied
   not liable for damages resulting from or in connection with use of this software
*/

#include <stdio.h>
#include <errno.h>

#define TEST


/********************/
/* basic data types */
/********************/
typedef unsigned char   byte;
typedef unsigned int    uint32;


/*************/
/* constants */
/*************/
#define  BUFFER_SZ      8 * 1024
#define  BITS_IN_BYTE   8


/**********/
/* macros */
/**********/
#define CHG_ENDIAN(a) {byte *p, t; p=(byte *) &(a); t=p[0]; p[0]=p[3]; p[3]=t; t=p[1]; p[1]=p[2]; p[2]=t;}


/*
   CRC polynomial as specified in ISO 3309 and ITU-T V.42
   used in Ethernet, FDDI, cksum, etc
   polynomial is  x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10
                       + x^8  + x^7  + x^5  + x^4  + x^2  + x^1  + x^0

   if the leftmost bit is the msb, this is
      binary  1 0000 0100 1100 0001 0001 1101 1011 0111
      hex     1 0    4    c    1    1    d    b    7
   big order bit is implicit so we have
      0x04c11db7
*/
```

```
#ifdef _ILP32
#  define  CRC32_POLY      0x04c11db7ul  /* polynomial */
#  define  CRC32_CONSTANT  0x4b90b035ul  /* constant which matches polynomial above */

#  define  LEFTMOST_BIT    0x80000000ul
#  define  ALL_BITS        0xfffffffful
#else
#  define  CRC32_POLY      0x04c11db7u   /* polynomial */
#  define  CRC32_CONSTANT  0x4b90b035u   /* constant which matches polynomial above */

#  define  LEFTMOST_BIT    0x80000000u
#  define  ALL_BITS        0xffffffffu
#endif




/* initialized to zero by the compiler */
static uint32  Crc32_tbl[256];


static void
crc32_tbl_load(void)
   {
   int     i;
   uint32  c;
   int     j;


   /* initialize auxiliary table */
   for (i = 0; i < 256; i++)
      {
      c = i << 24;

      for (j = 0; j < BITS_IN_BYTE; j++)
         c = c & LEFTMOST_BIT ? (c << 1) ^ CRC32_POLY : (c << 1);

      Crc32_tbl[i] = c;
      }
   }


void
crc32_init(uint32  *crc)
   {
   /* initialize auxiliary table (if necessary) */
   if (!Crc32_tbl[1])
      crc32_tbl_load();

   /* preload shift register, per CRC-32 spec */
   *crc = ALL_BITS;
   }


void
crc32_add(uint32  *crc,
          byte    *buf,
          size_t  len
         )
   {
   uint32  val;
   size_t  i;


   val = *crc;
   val = ~val & ALL_BITS;

   for (i = 0; i < len; i++)
      val = (val >> 8) ^ Crc32_tbl[ (val ^ buf[i]) & 0xff];

   val = ~val & ALL_BITS;
   *crc = val;
   }


main(int argc, char **argv)
   {
   char    *path;
   FILE    *fptr;
```

```
   size_t  len;
   byte    buf[BUFFER_SZ];
   uint32  crc;
   uint32  crc_to_file;


   switch (argc)
      {
      case 1 :
         path = "<stdin>";
         fptr = stdin;  /* read from standard input */
         break;

      case 2 :
         /* open input file (use the 'b' flag to read as binary rather than text) */
         path = argv[1];
         if ( (fptr = fopen(path, "rb") ) == NULL)
            {
            fprintf(stderr, "\nerror opening %s (%s)\n", path, strerror(errno) );
            exit(1);
            }
         break;

      default :
         fprintf(stderr, "\nusage: %s   pathname\n", argv[0]);
         fprintf(stderr, "       -or-");
         fprintf(stderr, "\n       %s < pathname\n", argv[0]);
         exit(1);
      }

   /* initialize */
   crc32_init(&crc);

   /* calculate crc for all data in file */
   while (len = (fread(buf, 1, BUFFER_SZ, fptr) ) )
      crc32_add(&crc, buf, len);

   if (!feof(fptr) )
      {
      fprintf(stderr, "\nerror reading %s (%s)\n", path, strerror(errno) );
      if (fptr != stdin)
         fclose(fptr);
      exit(1);
      }

   if (fptr != stdin)
      fclose(fptr);

   crc_to_file = crc;

   /* ensure CRC32 is written to Oasis file in LITTLE_ENDIAN byte order */
#ifdef BIG_ENDIAN_MACHINE
   CHG_ENDIAN(crc_to_file);
#endif

#ifdef TEST
   /* this is the crc value that should be the last 4 bytes in the file */
   printf("crc_to_file                         = 0x%08x\n", crc_to_file);

   /* assume the CRC32 value crc_to_file was appended to the end of the Oasis file */
   /* add the CRC32 (in LITTLE_ENDIAN order) to the data stream and continue CRC calculation */
   crc32_add(&crc, (byte *) &crc_to_file, sizeof(crc_to_file) );
#endif

   printf("crc_constant (should be 0x%08x) = 0x%08x\n", CRC32_CONSTANT, crc);

   exit(0);
   }
```

A1-2 Sample CHECKSUM32 C-Language Source Code

```
/*
   (c) Copyright 2003 SEMI
   no warranty, express or implied
   not liable for damages resulting from or in connection with use of this software
*/


#include <stdio.h>
#include <errno.h>


/*********************/
/* basic data types */
/*********************/
typedef unsigned char    byte;
typedef unsigned int     uint32;

#ifdef _ILP32
typedef unsigned long long  uint64;
#else
typedef unsigned long       uint64;
#endif


/*************/
/* constants */
/*************/
#define  BUFFER_SZ      8 * 1024
#define  BITS_IN_BYTE   8

/**********/
/* macros */
/**********/
#define CHG_ENDIAN(a) {byte *p, t; p=(byte *)&(a); t=p[0]; p[0]=p[3]; p[3]=t; t=p[1]; p[1]=p[2]; p[2]=t;}



void
checksum_init(uint32  *chksum)
   {
   *chksum = 0;
   }


void
checksum_add(uint32  *chksum,
             byte    *buf,
             size_t  len
            )
   {
   uint64  val;  /* could be a uint32, but overflow handling is undefined */
   size_t  i;


   val = (uint64) *chksum;

   for (i = 0; i < len; i++)
      {
      val += buf[i];      /* sum */
      val &= 0xffffffff;  /* limit to 32 bits */
      }

   *chksum = (uint32) val;
   }


main(int argc, char **argv)
   {
   char    *path;
   FILE    *fptr;
   size_t  len;
   byte    buf[BUFFER_SZ];
   uint32  chksum;
   uint32  chksum_to_file;


   switch (argc)
      {
```

```
      case 1 :
         path = "<stdin>";
         fptr = stdin;  /* read from standard input */
         break;

      case 2 :
         /* open input file (use the 'b' flag to read as binary rather than text) */
         path = argv[1];
         if ( (fptr = fopen(path, "rb") ) == NULL)
            {
            fprintf(stderr, "\nerror opening %s (%s)\n", path, strerror(errno) );
            exit(1);
            }
         break;

      default :
         fprintf(stderr, "\nusage: %s   pathname\n", argv[0]);
         fprintf(stderr, "       -or-");
         fprintf(stderr, "\n       %s < pathname\n", argv[0]);
         exit(1);
      }

   /* initialize */
   checksum_init(&chksum);

   /* calculate checksum for all data in file */
   while (len = (fread(buf, 1, BUFFER_SZ, fptr) ) )
      checksum_add(&chksum, buf, len);

   if (!feof(fptr) )
      {
      fprintf(stderr, "\nerror reading %s (%s)\n", path, strerror(errno) );
      if (fptr != stdin)
         fclose(fptr);
      exit(1);
      }

   if (fptr != stdin)
      fclose(fptr);

   chksum_to_file = chksum;

   /* ensure CHECKSUM32 is written to Oasis file in LITTLE_ENDIAN byte order */
#ifdef BIG_ENDIAN_MACHINE
   CHG_ENDIAN(chksum_to_file);
#endif

   /* this is the checksum value that should be the last 4 bytes in the file */
   printf("chksum_to_file = 0x%08x\n", chksum_to_file);

   exit(0);
   }
```

APPENDIX 2
OASIS Standard Properties

A2-1 File-Level Standard Properties

A2-1.1 Any file-level standard properties must appear immediately after the **START** record in an OASIS file. Use of file-level standard properties is optional—OASIS processors may omit/ignore any or all of them.

A2-1.2 **S_MAX_SIGNED_INTEGER_WIDTH**
This property declares the maximum number of bytes required to represent any *signed-integer* in the file, after all continuation bits have been removed and the integer has been expressed in twos-complement form. Its value list consists of a single *unsigned-integer.*

A2-1.3 **S_MAX_UNSIGNED_INTEGER_WIDTH**
This property declares the maximum number of bytes required to represent any *unsigned-integer* in the file, after all continuation bits have been removed. Its value list consists of a single *unsigned-integer.*

A2-1.4 **S_MAX_STRING_LENGTH**
This property declares the maximum number of bytes permitted in any string within the file. Its value list consists of a single *unsigned-integer.*

A2-1.5 **S_POLYGON_MAX_VERTICES**
This property declares the maximum number of vertices permitted in any polygon within the file, including any implicit vertices, but counting the initial vertex only once. Its value list consists of a single *unsigned-integer.*

A2-1.6 **S_PATH_MAX_VERTICES**
This property declares the maximum number of vertices permitted in any path within the file. Its value list consists of a single *unsigned-integer.*

A2-1.7 **S_TOP_CELL**
This property is used to declare the name of the "top cell" of a cell hierarchy. Its value list consists of a single *n-string*. It may be repeated if more than one distinct cell hierarchy exists within the OASIS file in which it appears.

A2-1.8 **S_BOUNDING_BOXES_AVAILABLE**
This property indicates whether or not S_BOUNDING_BOX properties appear in **CELLNAME** records. Its value list consists of a single *unsigned-integer.* A value of 0 means that S_BOUNDING_BOX properties are not provided. A value of 1 means that at least some S_BOUNDING_BOX properties are provided. A value of 2 means that an S_BOUNDING_BOX property is provided for every **CELLNAME** record.

A2-2 Cell-Level Standard Properties

A2-2.1 Any cell-level standard properties must appear immediately after the corresponding **CELLNAME** record in an OASIS file. Use of cell-level standard properties is optional—OASIS processors may omit/ignore any or all of them.

A2-2.2 **S_BOUNDING_BOX**

This property may occur once after each **CELLNAME** record, and declares the bounding box of that cell. Its value list consists of the following 5 fields: <flags> <lower-left-x> <lower-left-y> <width> <height>. The lower-left-x and lower-left-y fields are *signed-integers* representing the lower-left corner of the cell's bounding box. The width and height fields are *unsigned-integers* representing the width and height of the cell's bounding box. The bounding box should be calculated to cover the full extent of all geometric figures and text element (x,y) points within that cell and all of its subcells after a full expansion of any hierarchy beneath the cell.

The flags field is an *unsigned-integer.* Only the least-significant 3 bits are presently defined, and have the following meanings:

flags.bit.0: 0 = bounding box is known
1 = bounding box is unknown
flags.bit.1: 0 = bounding box is non-empty
1 = bounding box is empty
flags.bit.2: 0 = bounding box depends on no external cells
1 = bounding box depends on one or more external cells

A2-2.3 **S_CELL_OFFSET**

This property may occur once after each **CELLNAME** record. Its value list consists of a single *unsigned-integer* which declares the byte offset from the beginning of the file (byte 0) to where the corresponding **CELL** record appears in the file. An offset value of 0 denotes an *external* cell, with no corresponding **CELL** record in the same OASIS file.

A2-3 Element-Level Properties

A2-3.1 **S_GDS_PROPERTY**

This property is intended exclusively for compatibility with GDSII Stream properties. It may occur one or more times after any element within a **CELL** definition. Its value list contains exactly two values in sequence: <attribute>, an *unsigned-integer*, and <propvalue-string>, a *b-string*. These values correspond to GDSII Stream PROPATTR and PROPVALUE records, respectively.