The Digital Geographic Information Exchange Standard (DIGEST)

Part 2 - Annex C
VECTOR RELATIONAL FORMAT

Edition 2.1
September 2000

*Produced and issued by the **D**igital **G**eographic **I**nformation **W**orking **G**roup (DGIWG)*

Annex C
Vector Relational Format

Annex C - Contents page

Annex C - Appendices page

Annex C - Figures page

Annex C - Tables page

C.1 GENERAL REQUIREMENTS

C.1.1 General

Vector Relational Format (VRF) is a generic geographic data model designed to be used with any digital geographic data in vector format that can be represented using nodes, edges, and faces. VRF is based upon the georelational data model, combinatorial topology, and set theory (see Appendix C1 for discussions of these concepts).

A VRF-compliant database product must include all mandatory tables and columns which are described in clause C.2. Similarly, a VRF-compliant application must properly interpret all mandatory and optional tables and columns enumerated in this document.

C.1.2 Relationship Between VRF and Specific Products

VRF establishes a standard data model and organization, providing a consistent interface to data content. Data content itself shall be defined in a product specification that determines the content of the feature tables and the relationships between them. VRF can also accommodate additional tables that are not required by VRF itself. Without further standardization these additional tables, although VRF-compliant in structure, will provide column names and attribute values that may not be understood by others. Use of tables outside of those described in this standard may limit interoperability. Figure C-1 illustrates the relationship between VRF and specific products.

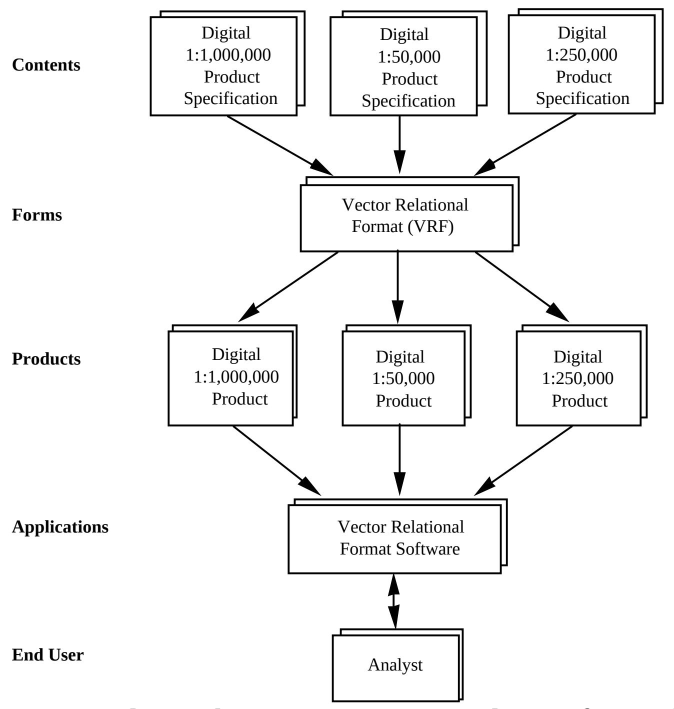

Contents
Digital 1:1,000,000 Product Specification
Digital 1:50,000 Product Specification
Digital 1:250,000 Product Specification
Forms
Vector Relational Format (VRF)
Products
Digital 1:1,000,000 Product
Digital 1:50,000 Product
Digital 1:250,000 Product
Applications
Vector Relational Format Software
End User
Analyst

Figure C-1 Relationship Between VRF and Specific Products

C.1.3 VRF Hierarchy

VRF can be viewed as a five-level hierarchy of definitions (Figure C-2) that increase in degree of abstraction from the bottom up. The bottom two levels define the physical representations of various data structures utilized in VRF. The data structure level concerns the logical representation of VRF data objects. These objects are elements in the VRF data model. The data model describes the data objects and the relationships among them. The top level, which contains the product specification, is used to tailor VRF to the requirements of the product.

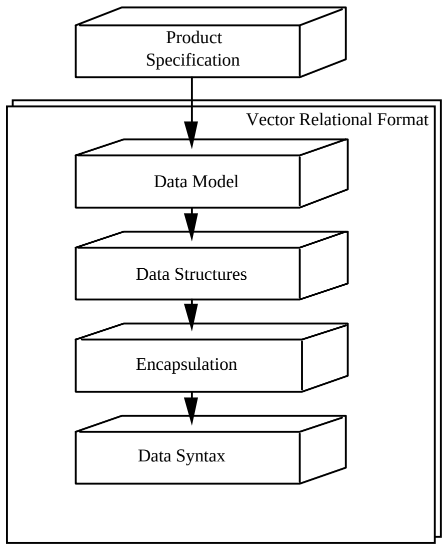


Figure C-2 Vector Relational Format Structure

This document gives a definition of VRF that encompasses the bottom four levels. A product specification is a combination of the conceptual database design and implementation details required to develop a product that is compliant with VRF. The conceptual modeling of feature classes and coverages is the first responsibility of the author of the product specification. This includes defining the list of features, attributes, attribute values, and providing their definitions. The physical design of the product database is also the responsibility of the author of the product specification. Physical design considerations include determining the tiling scheme, the topology level, the feature to / from primitive relationships (i.e., 1:1, 1:N, N:1 and N:M), the column types, and the table definitions.

C.2 DETAILED REQUIREMENTS

C.2.1 General

This clause describes the necessary components of Vector Relational Format (VRF). Clause C.2.2 discusses the conceptual components of the VRF data model (see Appendix

C1 for an overview describing the data model). Clause C.2.3 contains definitions of the data structures implemented in VRF. The encapsulation of VRF field types, table construction, and indexing structures are discussed in clause C.2.4. The encoding of the data syntax is found in clause C.2.5.

C.2.2 VRF Data Model

The discussion of the data model is broken into three subclauses: data organization, VRF data model components and data quality. Each of these subclauses addresses the data model from a different perspective. The data organization subclause (C.2.2.1) addresses VRF by defining the physical structures that make it up. Only three structures are used to implement the entire VRF data model: directories, tables and indexes. The data model component subclause (C.2.2.2) addresses VRF by defining the entities in a geographic database and describing the way in which these entities are captured through the physical VRF structures, starting with the most basic components (primitives) and continuing through the other levels (features, coverages, libraries and database). Finally, the data quality subclause (C.2.2.3) describes the options available in VRF for the maintenance of data quality information at any level in the data model.

C.2.2.1 Data Organization

All VRF data is organized in the form of files. A file is a named, sequentially ordered stream of bytes (Figure C-3). Files may be created, deleted, opened, closed, read (from byte m to byte n), and written (from byte m to byte n).

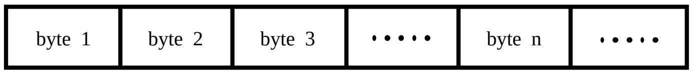


Figure C-3 Byte Stream

VRF uses only three types of files: directories, tables and indexes. All directory and file names in VRF databases are to be in lower case.

Note: It is acknowledged that, when CD-ROM output is generated by conversion software complying with ISO 9660, directory and file names are automatically converted to upper case. Case-sensitive operating systems accommodate this change through their ISO 9660 import software. However, even in this case, embedded references to directory and file names will always be in lower case.

C.2.2.1.1 Directory

The directory is a file that identifies the names of a collection of files, and their beginning addresses and lengths (Table C-1).

Table C-1 Directory Structure

Directory		
Name	Address	Length
File Name	Location on Medium	Length in Media Storage Units

VRF directories are strictly hierarchical; each file is contained in exactly one directory. File names must be unique within a directory. A file referenced by a directory is said to be contained in that directory. A file contained in a directory may be referenced by a special form of its name called a pathname (because it contains the location path to the file). A pathname has the following form:

<directory name><separator><file name>

where:

< > indicates that the enclosed name element is to be replaced with the actual text string indicated.

VRF uses the backslash character (\) as the generic pathname directory separator. For platforms requiring a different separator, software will replace the backslash with the appropriate separator character. For example, if a file named roads.lft is contained in a directory named urbareas, and the separator is (\), the resulting pathname would be:

urbareas\roads.lft

Directories are themselves files, so they may be contained in other directories. They are referenced by pathname the same way as other files. Thus, if urbareas is contained in library1, the resulting pathname would be:

library1\urbareas

Finally, pathnames may be combined. Since the directory that contains a file can be contained within a directory itself, it is necessary to have a form of file name that uniquely identifies that file contained within that directory (file names, while unique within a directory, are not unique between directories). For our example, that form would be:

library1\urbareas\roads.lft

C.2.2.1.2 Tables

In the VRF data model, the table is the organizational structure for all data content. All tables in a VRF database share a common basic structure; this structure, which is described in the VRF table components clause (C.2.2.1.3), is mandatory for all VRF tables.

By definition, a VRF table must include at least the basic structure. Optionally, a VRF table can also reference additional structures: the narrative table, thematic index(es), column narrative table(s) and value description table(s). A table can also have an associated variable-length index and a spatial index (for primitive tables). In the VRF data model, all geographic phenomena are modeled by VRF tables or by tables derived from a VRF table. A table derived from a VRF table is one that possesses all the properties of a VRF table but also has additional properties that support other specific functions.

The primitive table and the attribute table are examples of derived VRF tables. A derived table can also be further specialized to satisfy a particular need. For example, a feature table may be derived from the attribute table.

A VRF table may have an associated index file for variable-length records and a narrative table. A primitive table (discussed in clause C.2.2.2.1) may possess these two tables associated with its VRF table, but may also have a spatial index file and a minimum bounding rectangle table. An attribute table may also possess the tables associated with a VRF table, but may also have value description tables that provide the data dictionary for the table. The same data dictionary table may be shared by more than one attribute table. Finally, a feature table inherits the tables associated with an attribute table, but may also have a thematic index file. Feature tables are discussed further in clause C.2.3.3.1.

C.2.2.1.3 VRF Table Components

VRF tables consist of the following parts: a table header, a row identifier, and the table contents (under special situations (clause C.2.2.2.3.3) a table may contain only a header). The table header contains the metadata about a table and the column definitions. Columns are defined by a name and a data type; each column must have a name that is unique within the table.

Data contents in VRF tables are organized into rows and columns. All rows in a table share the same column definitions. Each row in the table is defined by a unique row identifier (row id). The row ids shall start at 1 and be sequential with no gaps in the numbering. Table C-2 depicts the principal components of a VRF table.

Table C-2 VRF Table Structure

Table Header Metadata and column definitions:	
a. Table description b. Narrative table name (optional) c. Column definitions: Column name Field type Field length Key type Column textual description Optional value description table name Optional thematic index name Optional column narrative table name	
ID	**Table of Contents**
Indicates the starting position of each row.	The data composing the table that match the column definitions.

This document describes the column definitions for all the VRF standard-specified column, and the table organization for those columns. No specific ordering of columns within a table is required. Product specifications may require a particular product-specific order. Data columns and tables described in this document are labeled either mandatory or optional. A VRF product must include all mandatory tables and columns. It is not possible to remove any mandatory column from any table. A VRF-compliant application must be able to process a VRF product and interpret all mandatory and optional columns as described in this document.

Additional product-specific columns are allowed by VRF. If present, these columns must be defined in their product specifications. Product-specific columns must not alter the use of the columns specified in this document.

C.2.2.1.4 Indexes

A table may have associated indexes. If a table contains a variable-length coordinate string column, a variable-length string column or a field type "K" (triplet id), a separate variable-length index file must be present.

In addition to variable-length indexes, VRF also supports spatial, thematic, and feature indexes. Spatial indexes contain references to row data that are based on the value of a coordinate column. Thematic indexes contain references to row data that are based on the value of non-coordinate columns.

Feature indexes have been developed to enhance processing of complex queries. They contain references linking rows in primitive tables to rows in associated feature tables.

C.2.2.1.5 Narrative Tables

Each VRF table may have an associated narrative table that provides miscellaneous information about the VRF table. The purpose of the narrative table is to provide the database designer with the ability to record comments or information pertinent to the associated feature table. The narrative table name is stored in the VRF feature table's header information. In addition, VRF provides for optional narrative tables keyed to individual columns within a table. The narrative table name is stored as the third optional entry in the column definition (see Table C-2).

C.2.2.1.6 Attribute Tables

Real-world objects are referred to as entities or features; they are modeled in tables in VRF. The properties of entities are called attributes. In an attribute table, one table column is defined for each attribute describing an object. Each object occupies a row in the table. Examples of attributes include data quality, size and name. A sample attribute table is shown in Table C-3.

A column or a group of columns that can be used to identify or select a row is called a key. A unique key is a key that uniquely identifies each row. One unique key is designated the primary key; each table has one and only one primary key. In the city attribute table (Table C-3), the built-up area column is the primary key.

Table C-3 City Attribute Table

ID	**Built-Up Area**	**State**	**Population Size**	**Median Income per Household**
implicit	*character string*	*character string*	*binary integer*	*binary integer*
UNIQUE KEY	PRIMARY KEY	NON-UNIQUE (Foreign Key)	NON-UNIQUE	NON-UNIQUE
1	Los Angeles	California	2966850	15735
2	New York	New York	7071639	13854
3	Salt Lake City	Utah	163033	13211
4	Las Vegas	Nevada	164674	17468
5	San Francisco	California	1366383	16782

A relational join is a database operation that brings together a number of tables into a new relation by using a set of common keys. The tables in such joins are called base tables. When a common key in a join is the primary key in one of the base tables but not in another, the non-primary (yet common) key is called a foreign key.

In the city attribute table (Table C-3), the state column is a foreign key; in the state attribute table (Table C-4), the state column is the primary key. In the city attribute table (Table C-3) the state column becomes a foreign key only through its reference by the state attribute table (Table C-4).

Table C-4 State Attribute Table

ID	**State**	**Area (sq.mi.)**	**Total Population**
implicit	*character string*	*binary integer*	*binary integer*
UNIQUE KEY	PRIMARY KEY	NON-UNIQUE	NON-UNIQUE
1	California	158706	26365000
2	Nevada	110561	936000
3	New York	49108	17783000
4	Utah	84899	1645000

C.2.2.2 VRF Data Model Components

The VRF data model may be considered to be layered into four structural levels (Figure C-4). At the lowest level, a VRF database consists of feature classes. In the database, these feature classes are defined using VRF primitive and attribute tables. Feature classes make up coverages, which in turn make up libraries, and finally, a database is made up of libraries.

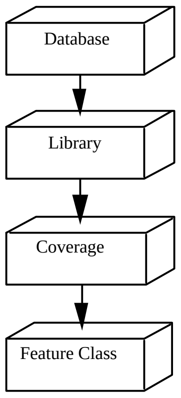


Figure C-4 VRF Structural Levels

An analogy can be drawn between VRF and written language. Letters are at the bottom of the language hierarchy. Words are made up of letters. In turn, sentences are made up of words. An essay is created from sentences, and a collection is made up of essays. Each of these entities has a distinct and different meaning not possessed by the entities below. The content of each entity, however, depends on that of the constituent entities.

Databases and libraries are used primarily to facilitate data access, whereas coverages (which incorporate topology) are used to define the relationships between features.

C.2.2.2.1 Primitives

There are three geometric primitives in VRF: nodes, edges, and faces (Figure C-5). As Figure C-5 shows, there are two types of node primitives: entity nodes and connected nodes. There is one type of cartographic primitive, text. These four primitives are combined to model any geographic phenomena using vector geometry. All primitives except text can be linked to each other by topological relationships, which are discussed further in clause C.2.2.2.3.1.

Coordinates must be consistent with the type specified in data_type in the Geographic Reference Table: Geographic Coordinates (if GEO), Grid Coordinates (if MAP), Relative Coordinates (if DIG). Units of measure are also specified in that table.

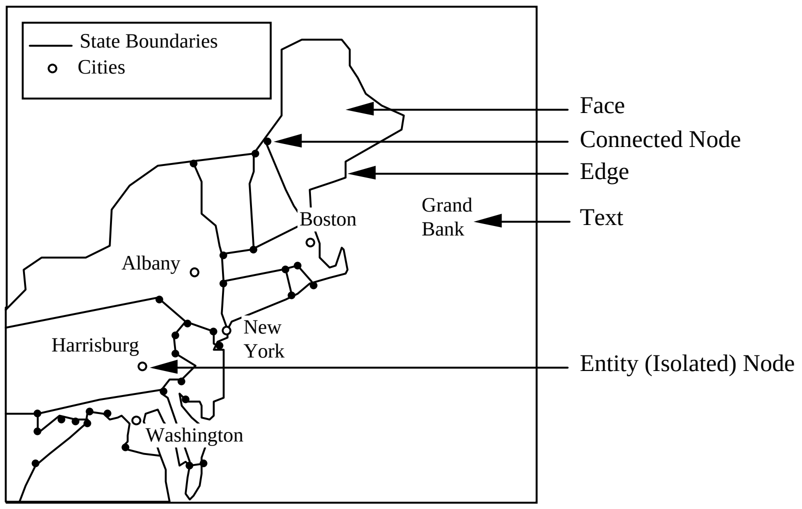


Figure C-5 Geometric and Cartographic Primitives

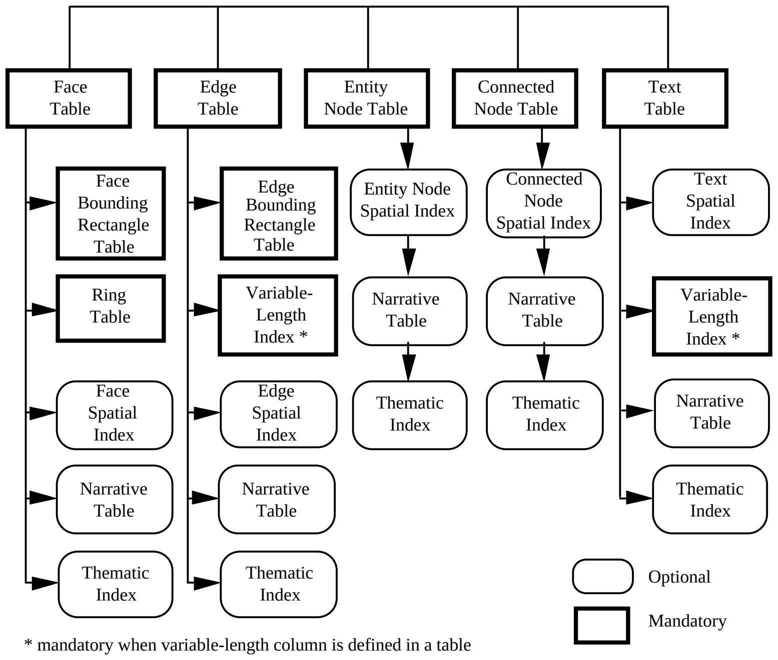


Figure C-6 Primitive Directory Contents

The following clauses summarize each of the primitives. Figure C-6 depicts each primitive and its associated tables and indexes.

C.2.2.2.1.1 Nodes

Nodes are zero-dimensional primitives that are used to store significant locations. No two nodes can occupy the same coordinate tuple. There are two types of nodes: entity nodes and connected nodes.

a. Entity Nodes are used to represent isolated features that are either truly zero dimensional, such as survey points, or too small to resolve at the collection scale, such as water towers at 1:24,000 scale. An entity node is topologically linked to its containing face when face, topology is present. An entity node cannot fall on an edge.

b. Connected nodes are always found at the ends of edges and are topologically linked to the edges. Connected nodes are used in two ways: (1) to define edges topologically and (2) to represent point features that are found at the start or end of an edge, such as overpasses, locks in a canal, or underground utility access points. Under the first usage, the connected nodes are referred to as start and end nodes. Under the second usage, attributes will be associated with the point features related to the connected nodes. All connected nodes are included in the node table. If many edges intersect a node, only one edge will be maintained per node in the node table; other edges are linked by using winged-edge topology (Appendix C2).

All connected nodes which lie on a tile boundary will have cross-tile components (tile_id and first_edge).

C.2.2.2.1.2 Edges

Edges are one-dimensional primitives that are used to represent the locations of linear features (such as roads) and the borders of faces. Edges are composed of an ordered collection of two or more coordinate tuples (pairs or triplets). At least two of the coordinate tuples must be distinct. The orientation of an edge can be recognized by the ordering of the coordinate tuples.

Edges are topologically defined by nodes at ends (levels 1-3 topology); edges, in turn, define faces (level 3 topology). In addition to the start node and end node columns, the edge primitive table contains column information (right edge, left edge, right face, left face) that is necessary to support higher levels of topology. This topology information permits the query and retrieval of features. The direction of an edge is its orientation from start node to end node. Each edge table has an associated edge bounding rectangle (ebr) table which contains the minimum bounding rectangle (mbr) for each edge. There is a one-to-one relationship between the edge table and its associated edge bounding rectangle table. Appendix C2 describes the use of winged-edge topology, which is used with edge primitives.

C.2.2.2.1.3 Faces

A face is a two-dimensional primitive enclosed by edges. Faces may be used to represent area features, (such as countries, inland water, or urban areas). Faces are non-overlapping, and the faces in a coverage completely exhaust the area of a plane. Faces are defined by topological references to a set of edges that compose the face border. A face may consist of multiple rings; there is one outer ring (traversed clockwise) and zero or more inner rings (traversed counter-clockwise). Each face table has an associated face bounding rectangle (fbr) table which contains the minimum bounding rectangle for each face. There is a one-to-one relationship between the face table and its associated face bounding rectangle table.

C.2.2.2.1.4 Text

Text is a cartographic rather than a geometric object. Text strings can be placed in specific locations in geographic space. Text can be used to associate names with regions that are vague or ill-defined, such as the Rocky Mountains. A text primitive may also be used when the name of a feature needs to be located in a specific relationship to a feature and could not otherwise be reproduced. For example, the text "Pacific Ocean" may be required for graphic display on a map, and may therefore be encoded as a text string, even though it is also being stored as an attribute of an area in a hydrographic coverage. Text primitives do not participate in topology.

C.2.2.2.2 Feature Classes

A feature class is a set of features that share a homogeneous set of attributes. Features are defined using primitive and attribute tables by means of relational modeling.

Tables are related to each other by their common keys. The relationships between tables are determined by the product specification.

C.2.2.2.2.1 Feature Definition

A feature is represented by a set of one or more primitives, a single row of attribute data in a feature table which uniquely identifies the feature, and zero or more rows of attribute data in other tables. A simple feature (e.g., spot height) may consist of one or more primitives of a single type and a single row of attribute data. A complex feature (e.g., an airport) will be identified by one row in a complex feature table, but will include the additional information contained in other feature tables. A feature may be logically related to another feature (e.g. a road connects with an interchange, a bridge is stacked on a river).

Features are grouped into feature classes. Each feature class is individually defined by a set of attributes (column definitions) and is uniquely named. The rows of features in a feature class collectively form the feature table for the feature class. Every feature class has one and only one feature table. The feature table is a special form of an attribute table because it directly references a feature. Table C-5 expresses the basic structure of a feature table in VRF.

Table C-5 Feature Table Structure

Primary Key	**Attributes**
Either a primitive row identifier or feature definition table id (may be the table id).	Attributes as specified in the product specification, or join values for reference into other attribute tables.

C.2.2.2.2.2 Feature Table Joins

Simple features may be composed of one or more primitives of a single type, while complex features may be composed of one or more simple or complex features. A feature join designates which primitives belong to which features. Feature joins may also be used to logically relate features to one another. Four types of feature joins represent all the possible relationships between features and primitives: one-to-one, many-to-one, one-to-many, and many-to-many. Appendix C3 provides a detailed discussion of these four types of join columns and feature join tables.

C.2.2.2.2.3 Feature Class Types

There are two types of feature classes in VRF: simple feature classes and complex feature classes. Figures C-7 and 7a portrays the structural schema of these feature classes.

a. Simple feature classes. A simple feature class consists of a (logically) single primitive table and a single simple feature table. There are four subtypes of the simple feature class in VRF:

(1) Point feature classes (composed of entity or connected nodes)
(2) Line feature classes (composed of edges)
(3) Area feature classes (composed of faces (level 3) or of edges (level 0-2). See C.2.2.2.3.1 for definition of topological levels)
(4) Text feature classes

A text feature class consists of a text primitive table and a text feature table. The text feature class is not a true feature class, but it is often useful to process text as if it were a feature. For instance, many maps contain text annotation that does not reference a specific geographic entity. The text "Himalaya Mountains" may not define any geometric primitive or feature, but merely provide associative information for the viewer. Using a text feature allows thematic queries on text just like other features. For instance, if a text feature has a height attribute, software can retrieve 'all text with HEIGHT > 0.5'.

b. Complex feature classes. A complex feature class consists of one or more simple feature classes, one or more complex feature classes, or both, and a single complex feature table, all within one coverage. For example, a complex watershed feature may be constructed from simple features, such as rivers, springs, and lakes.

C.2.2.2.2.4 Constructing Feature Classes

A feature class consists of a set of tables that includes at least one primitive table and one feature table and optionally, join tables and related attribute tables. The rules for constructing feature classes are stored in the feature class schema table, which describes how each table relates to each other table in the feature class (Table C-6).

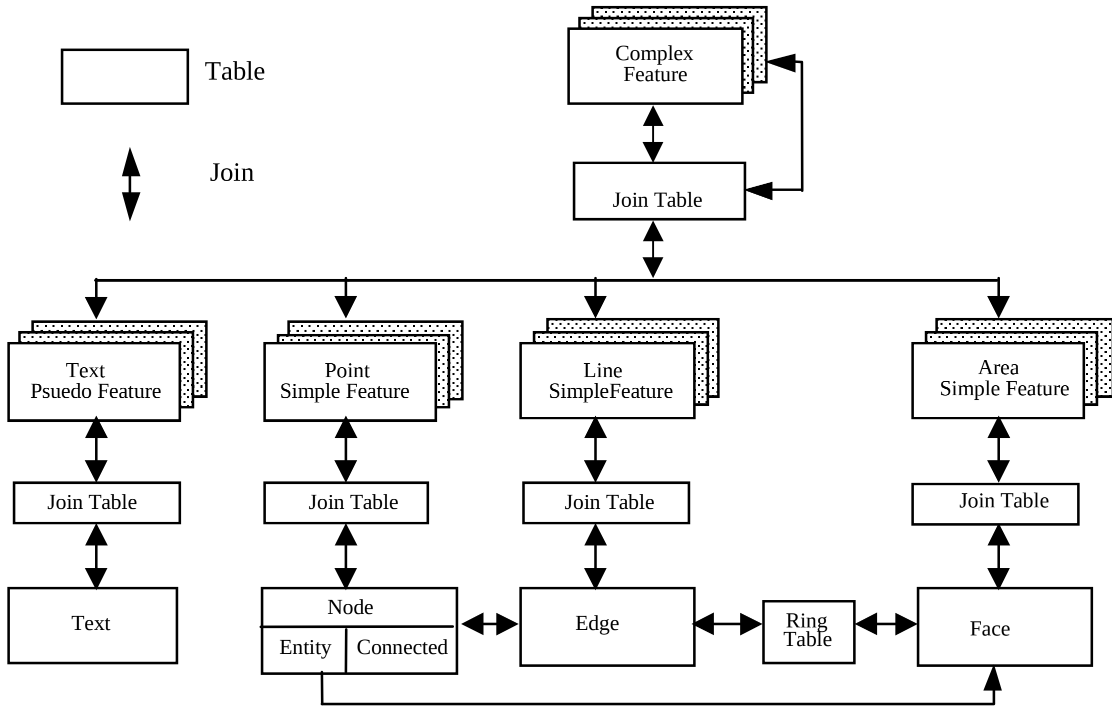


Figure C-7 Feature Class Structural Schema (level 3)

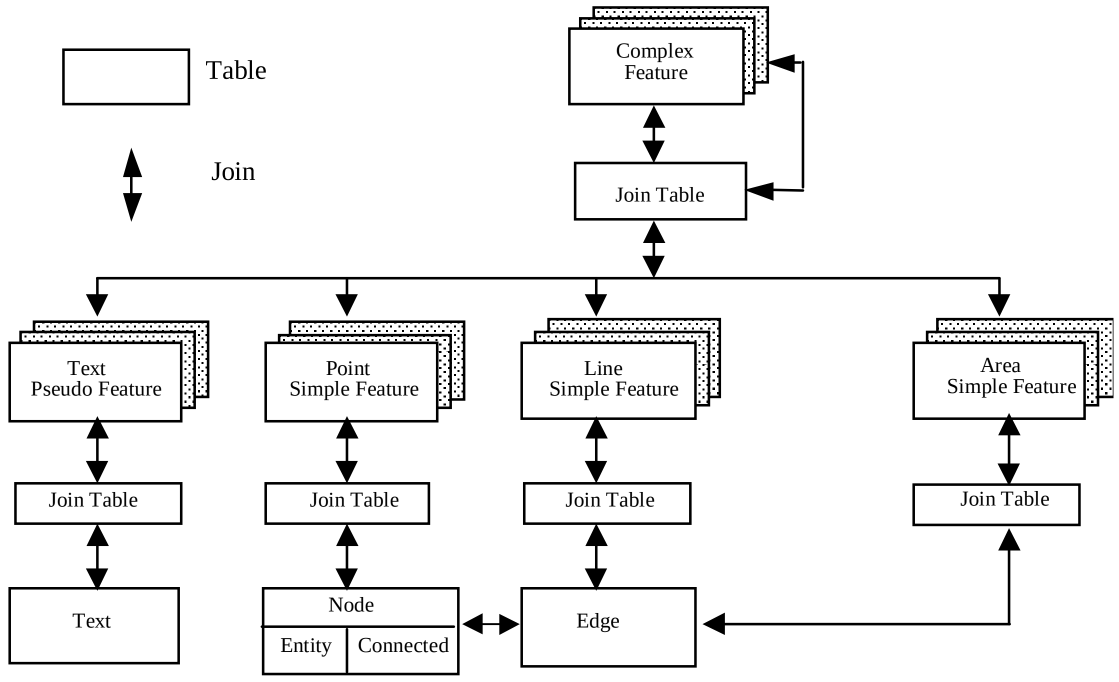


Figure C-7A Feature Class Structural Schema (level 0-2)

Table C-6 Feature Class Schema Table

Column Name	**Description**
id	Required row id
feature_class	Name of the feature class
table1	The first table name in the relationship
table1_key	Column name of table 1 join key
table2	The second table name in the relationship
table2_key	Column name of table 2 join key

Table C-7 shows a feature class schema table and an example of a simple feature class. Within the schema table, the feature class is named trnline. The first table in the relation is called trnline.lft. The second table is named edg, which is the standard label for the edge primitive. The key column, id, in trnline.lft relates to the key column, id, in edg. The trnline.lft has six attributes: f_code, bot, len, ohb, tuc, and from_to. The following attributes describe the feature: f_code, bot, len, ohb, and tuc. The attribute from_to, on the other hand, describes the geometry of the feature (see clause C.2.3.3.1). The edge primitive contains the required columns for level 2 topology (see clause C.2.2.2.3.1). Appendix C3 provides additional information on feature classes and feature joins.

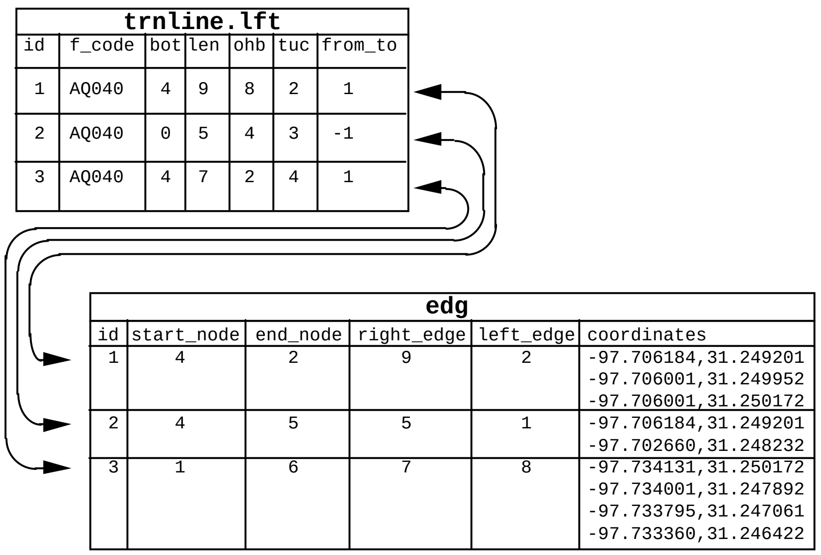
Table C-7 Feature Class Schema Table and Simple Feature Class

Feature class schema table

fcs

id	feature_class	table1	table1_key	table2	table2_key
1	trnline	trnline.lft	id	edg	id
2	trnline	edg	id	trnline.lft	id

Simple feature class



trnline.lft

id	f_code	bot	len	ohb	tuc	from_to
1	AQ040	4	9	8	2	1
2	AQ040	0	5	4	3	-1
3	AQ040	4	7	2	4	1

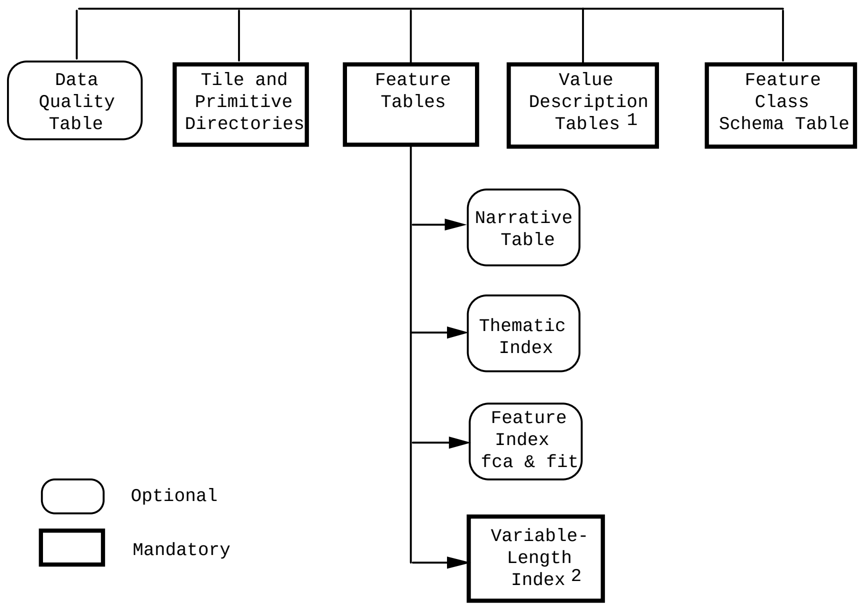
edg

id	start_node	end_node	right_edge	left_edge	coordinates
1	4	2	9	2	-97.706184,31.249201 -97.706001,31.249952 -97.706001,31.250172
2	4	5	5	1	-97.706184,31.249201 -97.702660,31.248232
3	1	6	7	8	-97.734131,31.250172 -97.734001,31.247892 -97.733795,31.247061 -97.733360,31.246422

C.2.2.2.3 Coverage

A coverage is composed of features whose primitives maintain topological relationships according to a level of topology (level 0, 1, 2, or 3) defined for the coverage. All of the file structures that make up a coverage are stored in a directory or subdirectories of that directory.

At the coverage level (see Figure C-8), there are three mandatory components: the primitive files or the subdirectories containing those primitives, the feature tables, and the feature class schema table. When join tables are implemented they will exist at the coverage level. Value description tables must be used when implementing coded attributes. A variable-length index file is mandatory whenever a variable-length column is defined in a table. An MBR is required for each face and edge table. Spatial indexes are optional for each primitive table. A feature minimum bounding rectangle table may be included for each feature table. Maintaining a data quality table at the coverage level is optional. Feature index tables may be used to support quick retrieval of feature information for a selected primitive. Feature class attribute tables are required to support feature index tables. When tile directories exist, the primitive tables are placed in the tile directories. Tile directories are mandatory for a tiled coverage.



1. Mandatory when coded attributes are used
2. Mandatory when variable-length column is defined in a table

Figure C-8 Coverage Contents

C.2.2.2.3.1 VRF Topology

There are four recognized levels of topology in VRF coverages, ranging from level 3, where all topological connections are explicitly present, to level 0, where no topological information is explicitly present. Figure C-9 summarizes the characteristics of these levels and gives an example of each.

Since text does not have any topological relationships, it is not listed in Figure C-9. Text may be included with other primitives at any topological level, even though it does not have any topology.

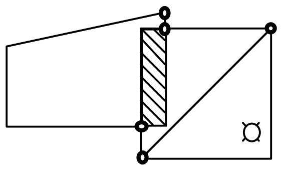
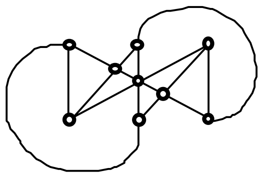
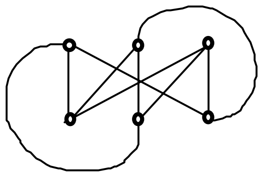
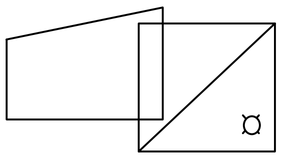
Level	Name	Primitives	Description	Example
3	Full topology	Connected nodes, entity nodes, edges, and faces	The surface is partitioned by a set of mutually exclusive and collectively exhaustive faces. Edges meet only at nodes.	
2	Planar graph	Connected nodes, entity nodes, and edges	A set of edges and nodes where, when projected onto a planar surface, the edges meet only at nodes. .	
1	Non- planar graph	Connected nodes, entity nodes, and edges	A set of entity nodes and edges that may meet at nodes.	
0	Boundary represen-tation (spaghetti)	Entity nodes, and edges.	A set of entity nodes and edges. Edges contain only coordinates, not start and end nodes.	

Figure C-9 Levels Of Topology in VRF Coverages

The columns carried in the edge and node tables, which determine connectivity and adjacency for the topology, depend on the level of topology. For instance, the edge table in Table C-7 does not contain the level 3 topology columns right_face and left_face, because faces do not exist in level 2 topology. Table C-8 shows the columns that are mandatory in each primitive table for the required level of topology. The characteristics of these columns are specified in the primitive definitions found in clause C.2.3.2.

Table C-8 Columns Required to Define Topology in VRF Coverages

Level	**Primitive**	**Mandatory Columns**
3	Face	ring_ptr
3	Ring Table	fac_id, start_edge
3	Edge	start_node, end_node right_face, left_face, right_edge, left_edge
3	Node	containing_face, first_edge
3	Entity Node*	containing_face
3-1	Connected Node*	first_edge
2-1	Edge	start_node, end_node right_edge, left_edge
2-1	Node	first_edge
2-0	Entity Node*	(none)
0	Edge	(none)
0	Node	(none)

* Note: The Node represents the optional single node table. If not used, a coverage will use separate entity and connected node tables.

Figures C-10, C-11, and C-12 use entity relationship (ER) diagrams to portray the primitives and their relationships for each level of topology.

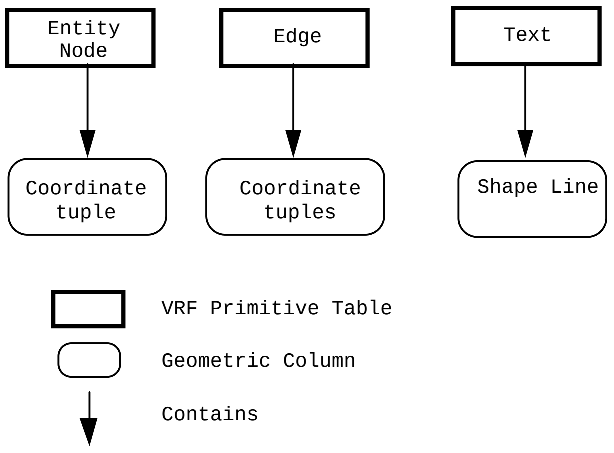


Figure C-10 Level 0 Topology

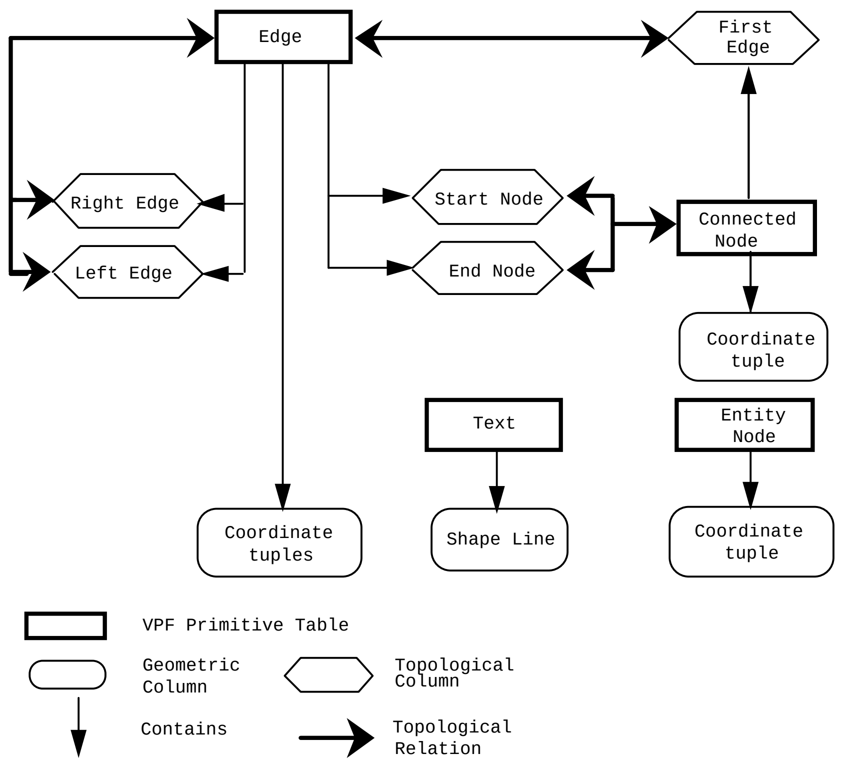


Figure C-11 Level 1 and Level 2 Topology

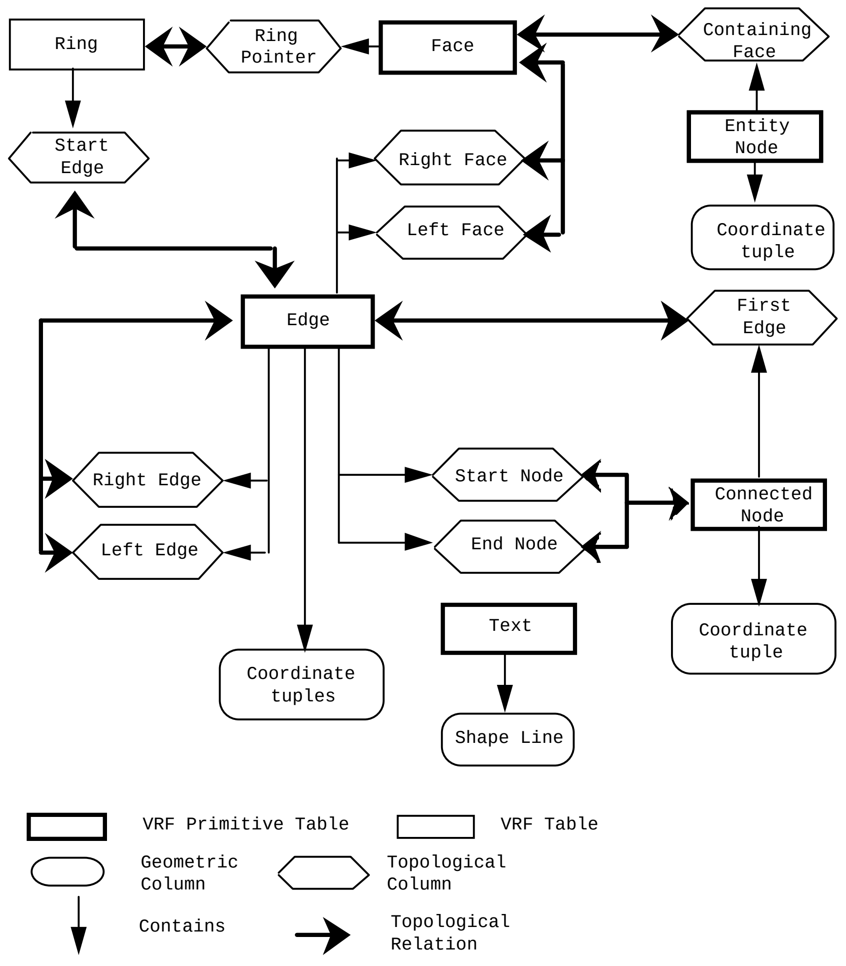


Figure C-12 Level 3 Topology

C.2.2.2.3.2 Value Description Tables

A value description table (vdt) is provided to describe coded attributes. There are three types of attribute values: distinct values, integer value codes, and character value codes.

a. Integer value codes. In many cases, the values entered in an attribute column are only codes designed to facilitate data processing and transmission. Numerical codes and their corresponding descriptions are maintained in the integer vdt.

b. Character value codes. For alphanumeric codes, there is a character vdt similar to the integer vdt. For instance, feature attribute coding systems generally use a five-character-string feature coding scheme.

c. Distinct values. Distinct values are attribute values that can be directly interpreted. Measurements of length or elevation are examples of distinct values. The interpretation of distinct values does not require a value description table.

C.2.2.2.3.3 Tiled Coverages

Tiling is geographically subdividing a coverage solely for the purpose of enhancing data management; a coverage subdivided in such a manner is then referred to as a tiled coverage. A tiled coverage contains the same attribute information as an untiled coverage. The logical interpretation of a tiled coverage is identical to that of an untiled one. Each tile will be a separate subdirectory under the coverage directory and contain separate primitive tables for those features contained within the tile. A tiled coverage will contain a single feature table for each feature class. Features in this table are joined with their corresponding primitives using a combination of tile_id and prim_id. If the entire coverage is devoid of data, then no coverage directory is necessary. Tiles do not contain feature attribute or schema tables. These tables belong to the coverage as a whole.

A tiled coverage is physically subdivided into tiles according to a tiling scheme. The tiling scheme (tile boundaries and size of tiles) and the handling of the features that lie on tile boundaries and text primitives that cross borders are all defined by a product specification. Each tile in a tiling scheme has a unique tile identifier. Figure C-14 shows a tiling scheme that uses regular rectangular tiles. Appendices C2 (Winged-edge topology) and C4 (Tiling) contain more information on tiling and its impact.

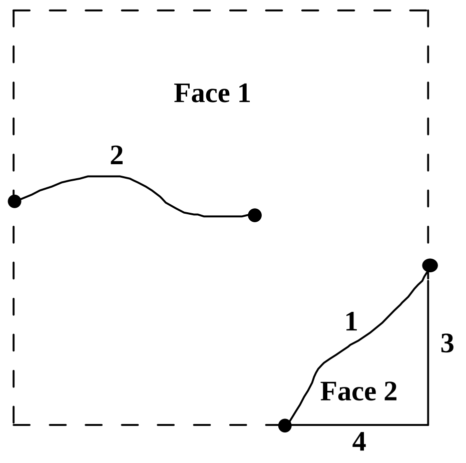
Primitive definition occurs wholly within a tile. The following paragraphs address the effect of tiling:

a. Edges: When an edge is broken by a tile boundary a connected node is placed at the edge-tile intersection. The identical (in terms of a geographical coordinate tuple) connected node occurs in both tiles forming the boundary. All edges which lie along a tile boundary will have cross-tile topology. The identical (in terms of a geographical coordinate tuple) edge occurs in both tiles forming the boundary.

b. Faces: A face broken by a tile boundary has a new edge constructed and inserted at the boundary for each tile to close the face internal to the tile. These edges take part in cross-tile topology.

c. Face 1: Face 1 (universe face) represents a special case for tile boundaries. In those cases where face 1 is the only face being broken, actual tile boundaries will not be stored. For example, where face 2 is broken by the tile boundary and the rest of the tile is defined by face 1, only the tile boundary edges necessary to close face 2 are stored (Figure C-13).

d. Connected Nodes: All connected nodes which lie on a tile boundary will have cross-tile components (tile_id and first_edge).



Note: Face 1 is the universe face. The tile's edge table will only store edges 1,2, 3 and 4. The dashed edges for the universe face are implied, but not stored.

Figure C-13 Storage of Tile Boundaries

Two other situations to consider are that of a tile of a level 3 topology coverage which contains only point features or no features. In these cases, the tile contains either entity node primitives and face 1 or simply face 1, respectively. Level 3 topology requires inclusion of a face, ring, edge, node, face bounding rectangle table, and an edge variable-length index (Table C-8). The face, ring and face bounding rectangle tables will reference the universe face (face 1) only. The edge table must exist since it is referenced by the ring table. The node table must exist since it is referenced by the edge table. The existence of an edge table requires an edge variable-length index and the existence of a face table requires a face bounding rectangle table. The edge and node table and the edge variable-length index will contain no records.

See Table C-9 for this scenario:

Table C-9 Tiled Coverages Sample Tables

fac table		
id	dnarea.aft_id	ring_ptr
1	(NULL)	1

rng table		
id	fac_id	start_edge
1	1	(NULL)

fbr table				
id	x min	y min	x max	y max
1	(null)	(null)	(null)	(null)

edg table						
id	dnline.lft_id	sn	en	rf	lf	coordinates
(no records---header information only)						

nod table				
id	dnpoint.pft_id	containing_face	first_edge	coordinates
1	1	1	null	37.5,-76.5
2	5	1	null	39.0, -80.0

Note: The node table will have no records if no entity nodes are present. If using end and cnd tables, point features (if present) will be linked to the entity node table and there will be an empty connected node table (containing only a header and no records).

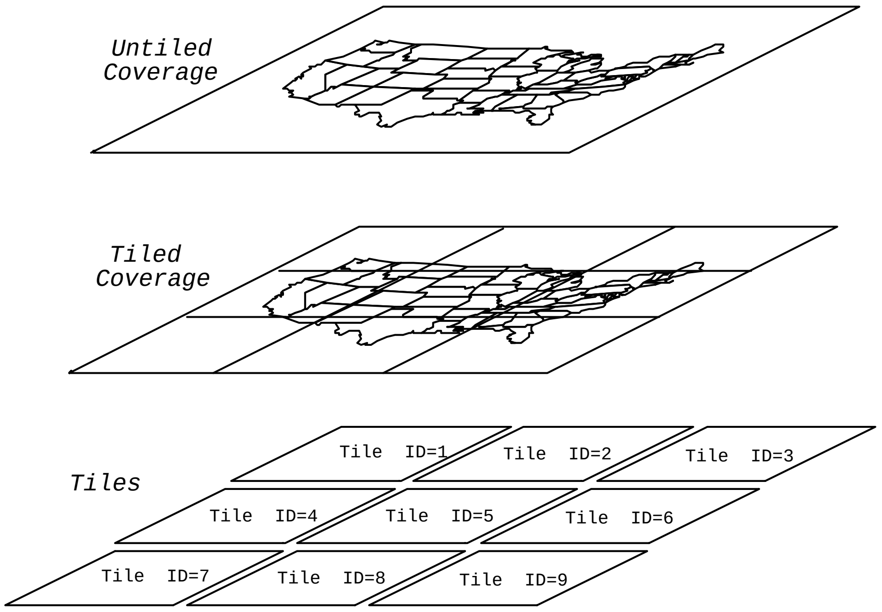


Figure C-14 A Tiling Scheme

C.2.2.2.3.4 Cross-Tile Keys

VRF provides a mechanism for maintaining geographic features in a logically continuous spatial database, whether or not a tiling scheme is present. Since the primitives in each tile of a tiled coverage are managed separately from those in other tiles, labels given to primitives are unique only within a tile. In order to support a logically continuous spatial database, a triplet id can be used instead of an integer key to reference primitives across multiple tiles. The triplet id augments the key of a primitive with the key of the tile in which the primitive falls. Appendix C2 contains a discussion that fully describes this concept.

a. For an edge primitive, the triplet id is used to maintain cross-tile topology. The left face, right face, left edge, and right edge columns are defined as triplet ids to support tiled coverages. The triplet id contains a reference to the internal topology within the current tile; the two other components reference the external tile directory and the primitive within that tile. For example, for a face divided by a tile boundary, the external id portion of the left face field in Figure C-15 would include the continuing face in the other tile. This inclusion of internal and external tile references allows software to detect tile borders and continue operations across boundaries or to operate only within the current tile. If a coverage is untiled, the left face, right face, left edge, and right edge columns may be defined as integer columns; otherwise the external tile id and primitive id sub-fields of the triplet id will not exist (see C.2.4.6)

b. For a connected node in a tiled coverage the triplet id is applied to the first_edge column.

c. Cross-tile topology only occurs between tiles within a library. Cross-tile components will only be populated for edges intersecting tile boundaries within a library. There is no cross-tile topology between tiles in different libraries.

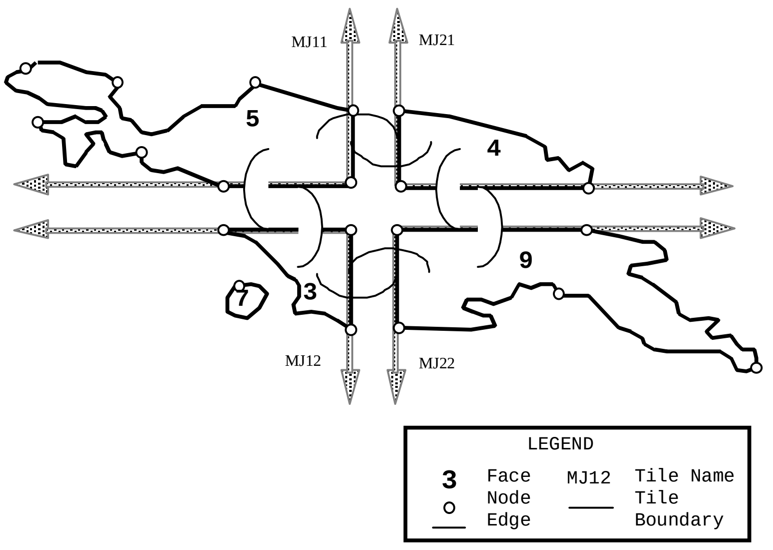


Figure C-15 Face Cross-Tile Matching

C.2.2.2.4 Library

A library is a collection of coverages that share a single coordinate system and scale, have a common thematic definition, and are contained within a specified spatial extent. If any of the coverages composing the library are tiled, then all other coverages must either use the same tiling scheme, or be untiled. The contents and organization of the libraries are determined by a product specification. All of the tables and coverages making up the library are contained within a single master directory (Figure C-16).

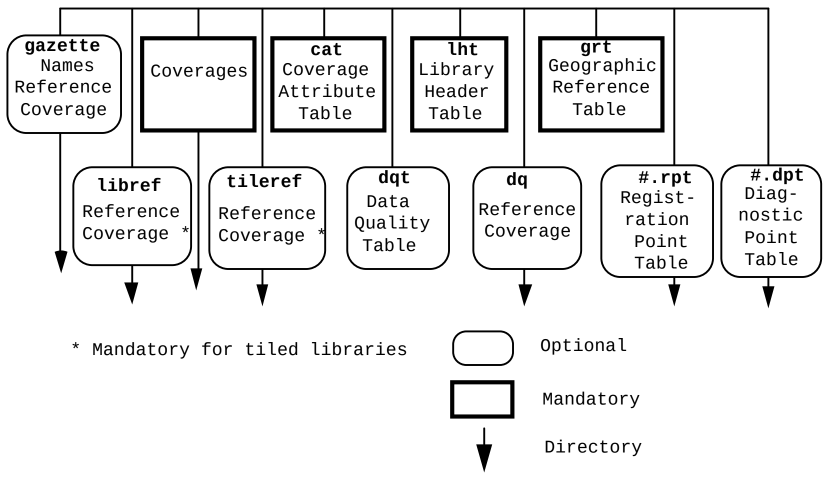

gazette Names Reference Coverage
Coverages
cat Coverage Attribute Table
lht Library Header Table
grt Geographic Reference Table
libref Reference Coverage *
tileref Reference Coverage *
dqt Data Quality Table
dq Reference Coverage
#.rpt Regist-ration Point Table
#.dpt Diag-nostic Point Table
* Mandatory for tiled libraries
Optional
Mandatory
Directory

Figure C-16 Library Directory

C.2.2.2.4.1 Tile Reference Coverage (tileref)

A tile reference coverage is mandatory if a library contains tiled coverages. The spatial extent of the library and its tiling scheme are represented in the tile reference coverage. A library cannot contain partial tiles. This reference coverage contains a set of faces and area features identifying the tiles that the library uses to subdivide the region of interest. The universe face always has a face id that equals one. The inner ring for face 1 in tileref defines the library's spatial extent. For irregularly shaped libraries, this will be a smaller total area than the bounding rectangle defined in the lat. The tile reference coverage is a standard untiled coverage with level 3 topology.

C.2.2.2.4.2 Library Attributes

General information about a library is stored in a variety of metadata tables, (i.e. library header tables, lht, coverage attribute tables, cat, geographic reference tables, grt, and data quality tables, dqt).

C.2.2.2.4.3 Library Coordinate System

The coordinate system of a library is defined by a geographic reference table. An example of a geographic reference table would document the projection used, its base parameters, and the values used to define the size of the Earth. This information (values for the semi-major and the semi-minor axis of an ellipsoid, other projection datum information, the false origin of a projection, and so forth) is necessary to understand a coordinate system in a VRF library.

C.2.2.2.4.4 Library Reference Coverage (libref)

When tiles exist in the library, a library reference coverage must exist. This coverage is spatially registered to the tile reference coverage to provide a preliminary view of the data contained within the library to use for such functions as "zoom out." The contents of this coverage will be a generalized map of the coverage considered to be most significant to the library. For example, if a library contains the rivers, transportation, and political boundaries of the Australian continent, a generalized map of the political boundaries might be considered appropriate for the library reference coverage.

C.2.2.2.4.5 Data Quality Reference Coverage

It is possible to include a data quality coverage at the library level. This coverage is spatially registered to the tile reference coverage. Its purpose is to record data quality information that pertains to the entire library. Appendix C5 contains more detailed information about the contents of this coverage.

C.2.2.2.4.6 Names Reference Coverage (gazette)

The names reference coverage provides the user with a way to locate a place in a library by using a place name. This is a special type of thematic query. The most common use of the names reference coverage is to enter a query string (for instance "London"), have the software locate all the places that are "London," and display their geographic locations and names on the display device. The name feature class contains a point feature table and a node primitive table.

C.2.2.2.5 Database

A database is a collection of related libraries and additional tables. The library attribute table acts as a table of contents for the database. Database information is contained in a database header table. Database level data quality information can be maintained in the data quality table. Appendix C5 contains more detailed information about the content of this table. Figure C-17 illustrates the arrangement of database tables and coverages.

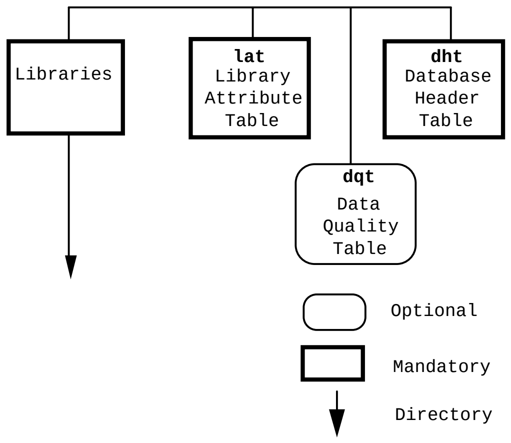


Figure C-17 Database Directory

C.2.2.3 Data Quality

VRF allows for the storage of data quality information to permit the evaluation of the data for particular applications. Although the exact form of the data quality information supplied for a database is set by a product specification, VRF supports incorporation of data quality information at each structural level in the database. Data quality information may be stored at any VRF level. When it exists at a given level, it applies to all data at or below that level. However, when data quality information exists at multiple levels, the information stored at lower levels always takes precedence over that at the higher levels.

C.2.2.3.1 Types of Data Quality Information

A VRF database may contain seven types of data quality information: source, positional accuracy, attribute accuracy, date status, logical consistency, feature completeness, and attribute completeness. Definitions of these quality types are provided in Appendix C5. The extent of data quality information contained in a product and the types of data quality to be included are determined by the product specification.

C.2.2.3.2 Data Quality Encoding

Data quality information can be represented as an attribute or as a coverage. In the case of attributes, data quality information may be added to an existing VRF table, stored in a separate table, or stored in the data quality table discussed in clause C.2.3.7. Appendix C5 describes data quality encoding in more detail.

C.2.3 Implementation

The following paragraphs describe the implementation requirements of the VRF data structures. Discussion covers the primitive, feature class, coverage, library, and database levels. A description of a data quality table and the narrative table is also provided.

C.2.3.1 General Implementation Information

In order to fully explain the content of each data structure, each table is given a text description, definition table, and an example.

C.2.3.1.1 Table Definitions

These column descriptions define the contents of each table. Each description example contains five entries: column name, description, column type, key type, and whether the column is optional or mandatory (Op/Man; see Table C-13). The asterisk (*) in the column name item is a substitute for an associated feature or primitive table name that is provided by the product specification. "Null" in example tables refers to a valid VRF null for that column type. Table C-10 is an example of the table definition style.

Table C-10 Table Definition Example

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
*.pft_id	Feature id	I	N	OF
containing_face	Face containing entity node	I	N	M3
first_edge	Edge id	X	N	O
coordinate	Coordinates	C	N	M

Schema descriptions identify the following columns.

a. Column type. The column type column in the definition table expresses the type of data the column must contain. The encapsulation of these types is discussed in additional detail in clause C.2.4. For the purposes of this section, Table C-11 identifies field types. When the number of elements is not specified as part of a column type definition described in this document for any VRF table header, it is assumed to be 1.

b. Key type. VRF provides three key types. They are primary keys, unique keys, and non-unique keys. Columns identified as non-unique in this document may be changed to unique by a product specification. Table C-12 lists the key types and the codes used in table definitions. Any primary key may be referred to as a foreign key in another table.

Table C-11 Column Types

Column Type	Description
T,n	Fixed-length text
T,*	Variable-length text
L,n	Level 1 (Latin 1 - ISO 8859) Fixed-length text
L,*	Level 1 (Latin 1 - ISO 8859) Variable-length text
N,n	Level 2 (obsolete - retained for backward compatibility)
N,*	Level 2 (obsolete - retained for backward compatibility)
M,n	Level 3 (Multilingual - ISO 10646) Fixed-length text
M,*	Level 3 (Multilingual - ISO 10646) Variable-length text
F	Short floating point (4 bytes)
R	Long floating point (8 bytes)
S	Short integer (2 bytes)
I	Long integer (4 bytes)
C,n	2-coordinate array - short floating point
C,*	2-coordinate string - short floating point
B,n	2-coordinate array - long floating point
B,*	2-coordinate string - long floating point
Z,n	3-coordinate array - short floating point
Z,*	3-coordinate string - short floating point
Y,n	3-coordinate array - long floating point
Y,*	3-coordinate string - long floating point
D	Date and time
X	Null field
K	Triplet id
G,n	2-coordinate array - short integer
G,*	2-coordinate string - short integer
H,n	2-coordinate array - long integer
H,*	2-coordinate string - long integer
V,n	3-coordinate array - short integer
V,*	3-coordinate string - short integer
W,n	3-coordinate array - long integer
W,*	3-coordinate string - long integer

Note: The asterisk (*) indicates variable-length string. n indicates a fixed-length array; n is defined by the product specification. The product specification can change columns of type * to n. Type characters are case sensitive when used in table definitions.

Table C-12 Key Types

Key	Description
P	Primary key
U	Unique key
N	Non-unique key

c. Optional/mandatory. The optional/mandatory column indicates whether the column is optional or mandatory for a VRF table. For each column, there are several mandatory conditions, as shown in Table C-13. The code OF is used on a primitive table when direct pointers to the feature table are desired to improve performance.

Table C-13 Optional/Mandatory Conditions

Code	Description
O	Optional
OF	Optional feature pointer
M	Mandatory
M<n>	Mandatory at level n topology (0–3)
MT	Mandatory if tiles exist

C.2.3.1.2 Reserved Table Names and Extensions

Each VRF table name consists of a reserved name or extension. Table C-14 lists the tables whose names cannot be modified or changed.

There are a few reserved directory names at the library and database levels. These names are listed in Table C-15.

In a coverage directory, there are many feature class tables that have reserved suffixes. The product specification may define any eight-character prefix, following the naming conventions detailed in clause C.2.4.5. Table C-16 lists the table suffixes.

Table C-14 Reserved File Names

File Name	Description
cat	Coverage Attribute Table
cnd	Connected Node Primitive Table
csi	Connected Node Spatial Index
dht	Database Header Table
dqt	Data Quality Table
ebr	Edge Bounding Rectangle Table
edg	Edge Primitive Table
end	Entity Node Primitive Table
esi	Edge Spatial Index
fac	Face Primitive Table
fbr	Face Bounding Rectangle Table
fca	Feature Class Attribute Table
fcs	Feature Class Schema Table
fsi	Face Spatial Index
grt	Geographic Reference Table
lat	Library Attribute Table
lht	Library Header Table
nod	Node Primitive Table
nsi	Node or Entity Node Spatial Index
pes	Primitive Expansion Schema Table
rng	Ring Table
txt	Text Primitive Table
tsi	Text Spatial Index
char.vdt	Character Value Description Table
int.vdt	Integer Value Description Table

Table C-15 Reserved Directory Names

Directory Name	Description
libref	Library reference coverage
dq	Data quality coverage
tileref	Tile reference coverage
gazette	Names reference coverage

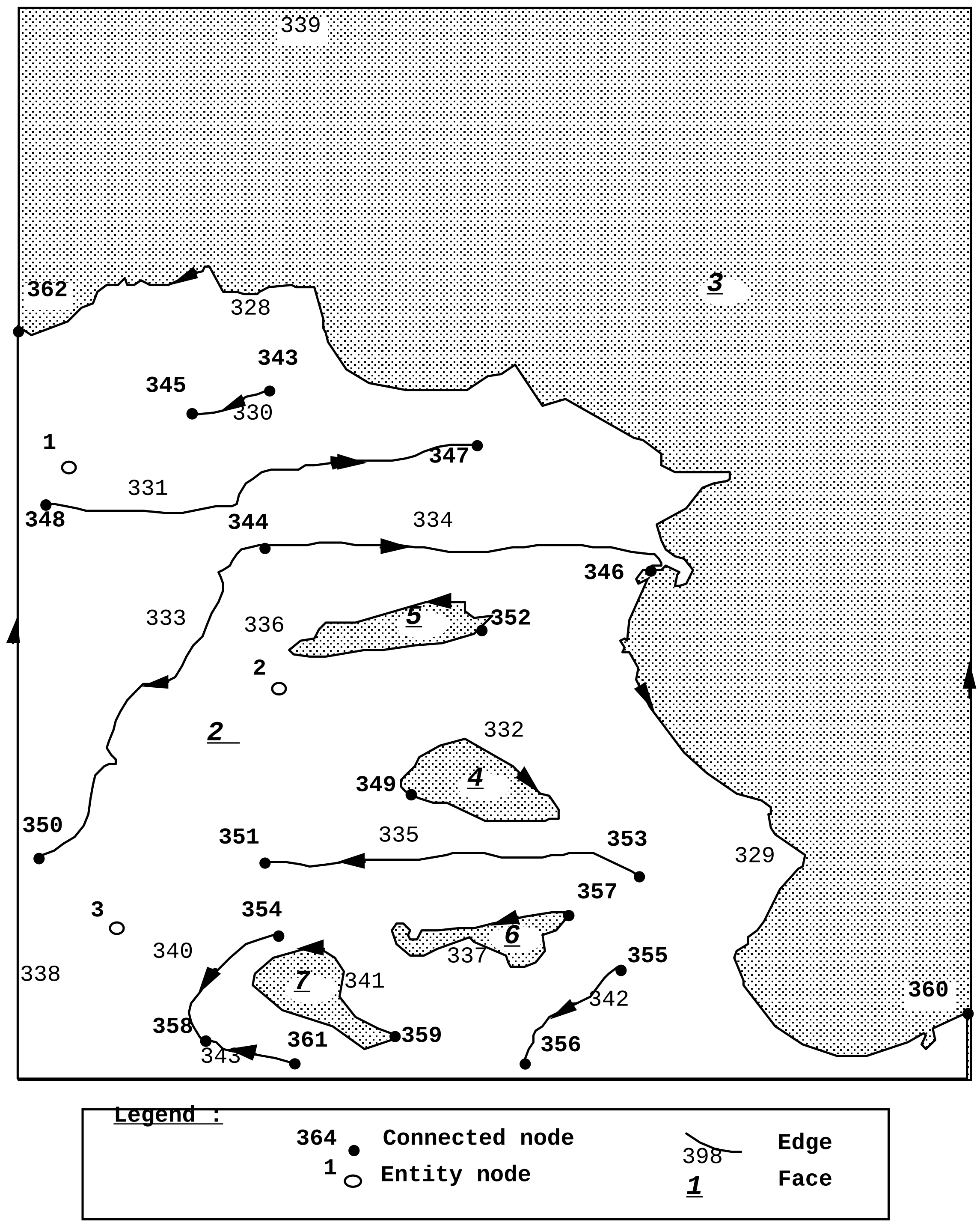
Table C-16 Reserved Table Name Extensions

File Name Suffix	**Description**
.abr	Area Bounding Rectangle Table
.aft	Area Feature Table
.ajt	Area Join Table
.ati	Area Thematic Index
.cbr	Complex Bounding Rectangle Table
.cft	Complex Feature Table
.cjt	Complex Join Table
.cti	Complex Thematic Index
.doc	Narrative Table
.dpt	Diagnostic Point Table
.fit	Feature Index Table
.fjt	Feature Relations Join Table
.fti	Feature Index Table Thematic Index
.jti	Join Thematic Index
.lbr	Line Bounding Rectangle Table
.lft	Line Feature Table
.ljt	Line Join Table
.lti	Line Thematic Index
.pbr	Point Bounding Rectangle Table
.pft	Point Feature Table
.pjt	Point Join Table
.pti	Point Thematic Index
.rat	Related Attribute Table
.rpt	Registration Point Table
.tbr	Text Bounding Rectangle Table
.tft	Text Feature Table
.tjt	Text Feature Join Table
.tti	Text Thematic Index

Any table that contains variable-length records must have a variable-length index associated with it. The index file shall have the same file name as the table, except that the last character will end with "x". For example, a variable-length record road line table, road.lft, would have a variable-length index road.lfx. The one exception to this convention is for the fcs, whose variable-length index shall be named fcsx. Prior to DIGEST Edition 2.0, June 1997, the variable-length index for the fcs table was named fcz.

C.2.3.2 Primitives

As discussed in clause C.2.2.2.1, there are three types of geometric primitives in VRF: nodes, edges, and faces. There are two classes of nodes: entity nodes and connected nodes. In addition, text is used as a cartographic primitive. These four primitives, with the addition of feature tables, allow the modeling of geographic phenomena requiring vector geometry. If the optional feature pointer is used, edge, face, and node primitives that are not features will carry a null in that field. Figure C-18 illustrates the various types of primitives. Columns can only be added to primitive tables to handle source, positional accuracy, up-to-dateness, security, and releasability.



(Note: This figure represents a partial database. Therefore only a subset of primitives are shown. A complete set of primitives would have sequentially numbered IDs beginning with 1.)

Figure C-18 Node, Edge, and Face Primitives

C.2.3.2.1 Node Primitives

The node primitive (composed of the entity nodes, which are free floating, and the connected nodes, which occur only at edge ends) can be modeled as either a single node table (nod) or as separate entity node (end) and connected node (cnd) tables. All nodes represent zero-dimensional locations.

a. Entity node primitive. The entity node primitive table (end) is composed of three columns: a primary key, a foreign (to the face table) key, and the node coordinates. The first_edge null column is included to maintain compatibility with the connected node primitive so that the formats for both classes of node primitive conceptually remain the same. The containing_face column is only required for level 3 topology to maintain a topological relationship to the face that contains the node. Table C-17 defines the meaning of the entity node primitive. Table C-18 illustrates an entity node table; the entity nodes described are those in Figure C-18.

Table C-17 Entity Node Definition

Column Name	**Description**	**Column Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
*.pft_id	Feature id	I	N	OF
containing_face	Face containing the entity node	I	N	M3
first_edge	(Null)	X	N	O
coordinate	Coordinates	C/Z/B/Y/G/ H/V/W	N	M

Note: The asterisk (*) indicates a placeholder for the point feature class name. There may be more than one *.pft_id.

Table C-18 Entity Node Record Example

id	**dnpoint.pft_id**	**containing_face**	**first_edge**	**coordinate**
1	936	2	Null	10.56 37.91
2	937	2	Null	10.36 37.72
3	953	2	Null	10.15 37.86

b. Connected node primitive. The connected node primitive table (cnd) is composed of three columns: a primary key, a foreign key (to the edge table), and the node coordinates. The containing_face null column is included to maintain compatibility with the entity node primitive. The first_edge column is a foreign key required for level 1 and higher topology levels to maintain a topological relationship to the edges that include the node. The complete set of edges around a connected node may be assembled by following the topology of the connected node until the first edge reappears.

Refer to Appendix C2 for more discussion of this algorithm. A connected node table is required for any coverage of topology level 1-3 containing an edge table. If the optional feature pointer is used, connected nodes that are not features will carry a null in that field. Table C-19 defines the connected node primitive. Table C-20 illustrates a connected node table with data for four of the connected nodes shown in Figure C-18.

Table C-19 Connected Node Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
*.pft_id	Feature id	I	N	OF
containing_face	(Null)	X	N	O
first_edge	Edge id	K/I	N	M1–3
coordinate	Coordinates	C/Z/B/Y/G/ H/V/W	N	M

Note: The asterisk (*) indicates a placeholder for the point feature class name. There may be more than one *.pft_id.

Table C-20 Connected Node Record Example

id	dn.pft_id	containing_face	first_edge	coordinate
. . .	. . .	. . .	. . .	. . .
343	42	Null	330	10.63 37.72
344	Null	Null	333	10.49 37.73
345	Null	Null	330	10.61 37.80

c. Node primitive. The node primitive table (nod) is an optional method of modeling the entity and connected nodes within a single table. The node primitive contains up to four mandatory columns, depending on the level of topology. The mandatory id column contains the row id and is the primary key. The containing_face column is a foreign key to the face table for topology level 3. It is used for all entity nodes associated with point features which do not fall on an edge. The first_edge column is a foreign key to the edge table for topology levels 1-3. It is used for all connected nodes. Within the node table any single record will contain a valid value for either the containing_face or first_edge. One of these values must be null. The coordinate column is mandatory. A node table is required for any coverage of topology level 1-3 containing an edge table. Table C-21 defines the node table. Table C-22 illustrates a node table.

Table C-21 Node Table Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
*.pft_id	Feature id	I	N	OF
containing_face	Face containing node	I	N	M3
first_edge	Edge id	K/I	N	M1-3
coordinate	Coordinates	C/Z/B/Y/G/ H/V/W	N	M

Note: The asterisk (*) indicates a placeholder for the point feature class name. There may be more than one *.pft_id.

Table C-22 Node Record Example

id	dnpoint.pft_id	containing_face	first_edge	coordinate
1	936	2	Null value	10.56 37.91
2	937	2	Null value	10.36 37.72
3	953	2	Null value	10.15 37.86
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
343	42	Null value	330	10.63 37.72
344	Null value	Null value	333	10.49 37.73
345	Null value	Null value	330	10.61 37.80

C.2.3.2.2 Edge Primitive

The edge primitive table includes up to eight mandatory columns, depending on the level of topology. The mandatory id column contains the row id and is the primary key. An optional edge_type column allows for the encoding of defined circular arcs. The start_node and end_node columns are foreign keys to the node primitive and are mandatory for levels 1-3. The right_face and left_face columns are foreign keys to the face table and are mandatory for level 3 topology. The right_edge and left_edge columns are foreign keys to the edge table and are mandatory for levels 1-3. The coordinates column is mandatory. For simplicity in drawing edges, the coordinate string includes the node coordinates at each end, regardless of the existence of a node primitive. Thus, the minimum length of the coordinate string, is two pairs, except for defined circular arcs where there must be three and only three coordinate pairs. Appendix C2 describes winged-edge topology in more detail.

a. Node information. The foreign keys to the node table are required for level 1,2 and 3 topology to maintain a topological relationship to the node connected to the edge. The start node indicates the beginning of the edge, and the relationship of the start node to the end node defines the edge orientation.

b. Edge information. Two foreign keys, right and left edge, are required for level 1, 2, and 3 topology, establishing connectivity between each edge and its neighboring edges in the coverage network. The right and left edges establish winged-edge topology for both line networks and faces. If all the edges incident at a node are sorted according to the bearing each edge radiates from that node, the right edge of a particular edge is the first edge encountered, counterclockwise in the sort order, around the end node of that particular edge. Similarly, the left edge is the first edge encountered around the start node.

c. Face information. When faces are present (level 3 topology), the right and left face columns are added to the edge primitive table. Depending on the edge direction, the face columns are assigned. When a face is split by a tile boundary, the internal tile boundary is used to close the face on each tile. The tile id and external face id are also maintained in the triplet id.

Table C-23, defines the edge table. Table C-24 illustrates an edge table created from data shown in Figure C-18.

Table C-23 Edge Table Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
**.lft_id	Feature id	I	N	OF
edge_type	Defines edge type	I	N	O
start_node	Start node id	I	N	M1–M3
end_node	End node id	I	N	M1–M3
right_face	Right face id	K/I	N	M3
left_face	Left face id	K/I	N	M3
right_edge	Right edge id from end node	K/I	N	M1-M3
left_edge	Left edge id from start node	K/I	N	M1-M3
coordinates	Coordinates	C/Z/B/Y/G/ H/V/W,*	N	M

Notes: The (**) indicates a placeholder for the line feature class name.
There may be more than one **.lft_id.

The edge_type is used to differentiate those edges whose coordinates are to be joined by straight lines (column is absent or an edge_type value of '1' is used) and those that are formed by defined geometric circular curves (edge_type 2 or3).

In the latter case there must be three coordinates only, whose meanings are defined as follows:

edge_type	Co-ordinate 1	Co-ordinate 2	Co-ordinate 3
2	start point	centre point	end point
3	start point	mid-point	end point

Note that though the strict geometry only requires that the second coordinate for edge-type=3 lies anywhere on the arc to be drawn, the accuracy of the arc will be maximized if it lies close to the mid point of the arc. This is shown in figure C-18A.

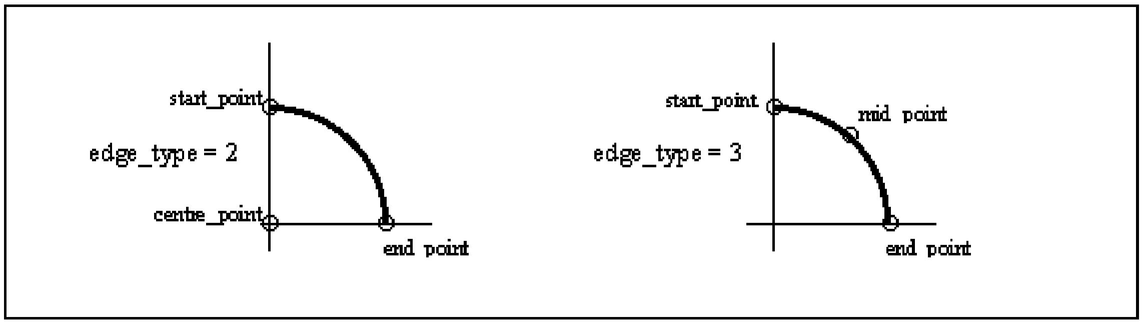


Figure C-18A Geometry for Geometric Circular Arcs

For edge_type=2, the arc must be drawn in a clock-wise manner from the start point until the end-point is reached. The distance from centre_point to start_point is to be taken as the radius of the arc. The end_point must be located on the arc. In the event of a discrepancy, the arc should be terminated where it crosses the straight line vector produced by joining the centre_point and end_point, extended as necessary. A complete circle will be drawn if the start_point and end_point are coincident.

For edge_type=3, the direction of the arc is inherent in its definition. A complete circle will be defined by coincident start_point and end_point coordinates with the mid_point lying at the diametrically opposite point on the circle.

Note that where a feature utilizes any geometric arc the generated vector should be used when calculating the necessary bounding rectangle. Implementers should note that extremely shallow curves defined by edge_type=2 may be difficult to represent correctly given the precision of calculation available. Similarly, computational problems may occur if the midpoint for edge_type=3 is positioned too close to the start or end point of a curve.

Table C-24 Edge Record Example

id	line	et	sn	en	rf	lf	re	le	**Coordinates** start node...end node
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
328	24500	1	346	362	3	2	339	334	10.46 37.38...10.68 37.00
329	24502	1	346	360	2	3	338	328	10.46 37.38...10.06 37.09
330	24505	1	343	345	2	2	330	330	10.63 37.72...10.61 37.80
331	24524	1	348	347	2	2	331	331	10.53 37.93...10.58 37.53
332	24534	1	349	349	4	2	332	332	10.26 37.36...10.26 37.36
333	24569	1	344	350	2	2	333	334	10.49 36.73...10.20 36.94
334	24573	1	344	346	2	2	329	333	10.49 36.73...10.46 37.38
335	24575	1	353	351	2	2	335	335	10.18 36.38...10.20 36.73
336	24581	1	352	352	2	5	336	336	10.20 36.81...10.20 36.81
337	24585	1	357	357	2	6	337	337	10.15 36.46...10.15 36.46
338	-32767	1	360	362	2	1	328	339	10.06 37.09...10.68 37.00
339	-32767	1	360	362	1	3	338	329	10.06 37.09...10.68 37.00
340	24601	1	354	358	2	2	343	340	10.13 36.72...10.04 36.78
341	24603	1	359	359	2	7	341	341	10.04 36.61...10.04 36.61
342	24612	1	355	356	2	2	342	342	10.10 36.40...10.02 36.50
343	24626	1	361	358	2	2	340	343	10.02 36.71...10.04 36.78
344	24635	2	384	385	9	8	392	396	10.04 36.31, 10.04 36.27, 10.08 36.27
345	24639	3	393	397	10	11	381	386	10.25 37.37, 10.42 37.30, 10.49 37.13

Note**:** line = **.lft_id, sn = start_node, en = end_node, rf = right_face, lf = left_face, re = right_edge, le = left_edge. Only start and end node coordinates are shown, although all coordinates would actually be present in this variable-length column. For values of edge_type equal to 2 or 3 there must be three and only three values in the coordinates column.

C.2.3.2.3 Face Primitive

Faces are defined as planar regions enclosed by an edge or a set of edges except for the universe face which is unbounded. All faces are defined by one or more rings, which are connected networks of edges that compose the face border. Each ring starts with a reference to a particular edge, and is defined by traveling in a consistent direction. The left and right edge columns on the edge primitive are traversed, always keeping the face being defined on one side, until the ring returns to its starting edge. All faces must have one and only one outer ring, which bounds the exterior. A face may require inner rings to represent areas belonging to other faces that it encloses totally. Inner rings and outer rings are disjointed. There is no upper limit on the number of inner rings. A ring table (see below) is defined to handle these disjointed rings. A ring within a ring contained within a face (e.g., a lake within an island which is contained within a larger lake) has no direct topological relation to the outer face (the larger lake). Face primitives are implemented as follows.

a. Face table. The face table contains two columns, where the primary key id identifies the face and the ring_ptr column points to the outer ring in the ring table. Face id 1 is always reserved for the universe face in a face table; it will never correspond to a feature in the feature table.

The outer ring of the universe face is a topological artifact which does not have a geometric representation. The outer ring cannot be displayed. The common boundary between the universe face and all other faces constitutes the inner ring or rings of the universe face. Inner rings of the universe face behave the same as the rings of other faces. Table C-25 defines the face table. Table C-26 depicts an example of the face table for three faces from Figure C-18.

Table C-25 Face Table Definition

Column Name	**Description**	**Column Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M3
*.aft_id	Feature id	I	N	OF
ring_ptr	Ring id	I	N	M3

Notes: The asterisk (*) indicates a placeholder for the area feature class name.
There may be more than one *.aft_id.

Table C-26 Face Record Example

id	**dnarea.aft_id**	**ring_ptr**
1	Null	1
2	4571	3
3	4572	13
4	4573	14

b. Ring table. The ring table contains one reference to the edge table for each ring of a face. The first row in the ring table for each face refers to the outer ring of that face. Because the outer ring for face 1 (universe face) is a topological artifact, its start edge will be null. Each inner ring has one reference to a first edge on that ring. The ring table maintains an order relationship for its rows. The first record of a new face id will always be defined as the outer ring. Any repeating records with an identical face value will define inner rings. Table C-27 defines ring table structure, and Table C-28 depicts eight rings from Figure C-18 and follows the face example in Table C-26.

Table C-27 Ring Table Definition

Column Name	**Description**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M3
fac_id	Face id	I	N	M3
start_edge	Edge id	I	N	M3

Table C-28 Ring Record Example

id	**fac_id**	**start_edge**
1	1	Null
2	1	339
3	2	328
4	2	330
5	2	331
6	2	336
7	2	332
8	2	335
9	2	337
10	2	342
11	2	341
12	2	343
13	3	339
14	4	332

C.2.3.2.4 Text Primitive

Text is implemented to allow the representation of names associated with vague or ill-defined regions, such as the Appalachian Mountains or by reference to other features. The text primitive is normally composed of three items: a primary key, the text string, and a coordinate string defining a shape line. Optional attributes may also be associated with the primitive, such as text color or font height.

The shape line must contain at least one coordinate pair. If the shape line contains only one coordinate pair, the coordinate pair is considered to represent the lower left coordinate, and the default orientation for the shape line will be assumed (minimum readable text and parallel to X axis.) In order to specify orientation, two coordinate pairs must be entered. The second coordinate pair defines the lower right of the string. Some fonts have ascenders and descenders that extend above or below the shape line. Third and subsequent coordinate pairs define control points in a shape line. The control points of a shape line define a continuous function. Characters in a text string are individually oriented along the shape line.

Table C-29 defines the text primitive table structure. Table C-30 is a hypothetical example of a text primitive table.

Table C-29 Text Primitive Structure Table

Column Name	**Description**	**Column Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
**.tft_id	Feature id	I	N	OF
string	Text string	T/L/M/N,*	N	M
shape_line	Coordinates	C/Z/B/Y/G/ H/V/W,*	N	M

Note: The (**) indicates a placeholder for the text feature class name. There may be more than one **.tft_id.

Table C-30 Text Primitive Record Example

id	**dntext.tft_id**	**string**	**shape_line**
1	529	Fiume Salso	14.41,37.74 14.45,37.73 14.52,37.69 14.60,37.69
2	530	Simeto	14.80,37.71 14.81,37.68 14.80,37.66 14.81,37.65
3	531	Belice	12.91,37.69 12.93,37.71 12.95,37.73
4	532	Dittaino	14.51,37.56 14.54,37.55 14.58,37.56 14.62,37.57

C.2.3.2.5 Minimum Bounding Rectangle Table

A minimum bounding rectangle record is required for each record in an edge or face primitive table. Since the outer ring of the universe face is a topological artifact which does not have a geometric representation, the fbr record for face 1 should contain nulls for xmin, ymin, xmax and ymax. The definition found in Table C-31 is used for both the face and edge minimum bounding rectangle tables. Coordinate values are as defined in the Geographic Reference Table (grt), Table C-42. Table C-32 is an example of the face minimum bounding rectangle table used for Figure C-18.

Table C-31 Minimum Bounding Rectangle Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
xmin	Minimum x coordinate	F/R/S/I	N	M
ymin	Minimum y coordinate	F/R/S/I	N	M
xmax	Maximum x coordinate	F/R/S/I	N	M
ymax	Maximum y coordinate	F/R/S/I	N	M

Table C-32 Face Bounding Rectangle Record Example

id	xmin	ymin	xmax	ymax
1	Null	Null	Null	Null
2	12.31	37.97	13.31	37.99
3	14.54	37.83	14.58	37.85

C.2.3.2.6 Z-value Differences at Edge Intersections

The 2D-planar data model used by VPF requires the existence of a node at a single-coordinate intersection (x,y or x,y,z) where edges cross. The following tables provide a method of storing the true Z-value for edges intersecting where disjoint Z-values occur. The use of these tables is limited to two specific occurrences:

- edges cross at different elevations, with or without a node feature (e.g., an overpass or bridge over a road) at the intersection
- edges cross at grade, single elevation, and a node feature exists at a different elevation than the edges (e.g., a hanging stop light in the middle of an intersection)

When either case occurs, the coordinate Z-component of the connected node and the associated first or last coordinate Z-component in the intersecting edges will be filled with the null value for that coordinate type (e.g., NaN for floating point or the integer null value). The true Z-value will be described in a related attribute table associated with the edge and connected node tables. The related attribute table will be stored at the same directory level as its associated edge and connected node tables.

The capability defined in this clause specifically applies only to instances of single-point edge intersections and is implemented at the geometry level. When the proper VPF structures are implemented, it continues to support the storage of coincident features where the features are associated with a single primitive at a single Z-value. It does not support the storage of coincident features at different Z-values.

a. Edge node Z's. Z-values for intersections (start node or end node) where disjoint Z-values occur will be stored in a table named zcoin.rat (Table C-32A). Only those edges associated with a disjoint Z-value at a node will be linked to the zcoin.rat. Each record in the zcoin.rat will contain the true edge elevation at the node and a reference to the connected node. There will be a record in the zcoin.rat for each elevation at the connected node where disjoint Z-values occur. Thus, a 4-edge intersection representing a road overpass would have two entries in the zcoin.rat (Table C-32B), one for each elevation for the intersecting edges. Each record in the join table linking the edge table to the zcoin.rat will contain the edge id and the zcoin.rat id where that edge's start or end point elevation value is defined (Table C-32C). The edge table start or end node coordinate Z-value will be filled with null for those edge nodes included in the zcoin.rat.

b. Connected Node Z's. Z-values for connected nodes at locations where disjoint Z-values occur will be stored in the zcoin.rat (Table C-32A). Each connected node requiring an elevation will be linked to a single record in the zcoin.rat through a join table (cnd.rjt). Each record in the join table linking the connected node table to the zcoin.rat will contain the connected node id and the zcoin.rat id (Table C-32D). Only one entry will be recorded in the cnd.rjt for each connected node. The elevation referenced by the connected node may either coincide with one of the edge nodes or it may represent a feature with an elevation different from any of the intersecting edges. Each connected node referenced in the zcoin.rat will have a null value in its cnd table coordinate Z-component.

Table C-32A zcoin Related Attribute Table Definition

Column Name	**Description**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
cnd_id	Connected node id	I	N	M
z_value	Value of true Z	F/R/S/I	N	M

Table C-32B zcoin Related Attribute Table Example

id	**cnd_id**	**z_value**
1	15	10.1234
2	15	1.23456
.		.
.		.

Table C-32C Edge start/end Node Join Table for Disjoint z-values.

Column Name	**Description**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
edg_id	Edge id	I	N	M
zcoin.rat_id	zcoin id	I	N	M

Table C-32D Connected Node Join Table for Disjoint z-values.

Column Name	**Description**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
cnd_id	Connected node id	I	N	M
zcoin.rat_id	zcoin id	I	N	M

c. Primitive Expansion Schema Table. When the zcoin.rat table is used to store Z-values for edges and connected nodes, a Primitive Expansion Schema Table (pes) is required at the coverage level to define the additional relationship of the primitive tables to the related attribute table. Only the primitive-to-related attribute table relationships are stored in the Primitive Expansion Schema Table. The Primitive Expansion Schema Table is defined in Table C-32E below. An example Primitive Expansion Schema Table is shown in Table C-32F.

Table C-32E Primitive Expansion Schema Table Description.

Column Name	**Description**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
prim_table	Primitive table name	T,8	N	M
table1	First table in a relationship	T,12	N	M
table1_key	Join column in the first table	T,*	N	M
table2	Second table in a relationship	T,12	N	M
table2_key	Join column in the second table	T,*	N	M

Table C-32F Primitive Expansion Schema Table Example

id	**prim_table**	**table1**	**table1_key**	**table2**	**table2_key**
1	cnd	cnd	id	cnd.rjt	cnd_id
2	cnd	cnd.rjt	zcoin.rat_id	zcoin.rat	id
3	cnd	zcoin.rat	id	cnd.rjt	zcoin.rat_id
4	cnd	cnd.rjt	cnd_id	cnd	id
5	edg	edg	id	edg.rjt	edg_id
6	edg	edg.rjt	zcoin.rat_id	zcoin.rat	id
7	edg	zcoin.rat	id	edg.rjt	zcoin.rat_id
8	edg	edg.rjt	edg_id	edg	id

C.2.3.3 Feature Class

A feature class is composed of a variety of tables containing geometric, topological, and attribute data. Complex feature relationships can be defined. Feature-to-feature logical relationships can be defined. Some features may require a single feature and an associated primitive table, while others may need multiple tables linking features into a complex hierarchy. Attribute data may be extended by the use of related attribute tables (rat) associated with feature or primitive tables. An additional use of related attribute tables would be to provide information on sources used for features and attributes within a coverage using a source.rat. Implementation (type of information stored per source) is product specific.

A feature minimum bounding rectangle table may be included for each feature table as defined in Table C-31. The feature class definition is provided by a product specification.

Constructing feature classes requires the use of feature tables. If necessary, the feature class definition may require the use of a feature join table to accurately construct the feature in relational form. Appendix C3 describes feature class relationships in greater detail.

C.2.3.3.1 Feature Tables

A feature table consists of a feature identifier and one or more attribute columns. The specific nature of the attribute shall be described in a product specification.

Table C-33 defines feature table contents for simple features that are linked to only one spatial element (i.e. a single primitive record). If the coverage is tiled, the feature table, for simple features, maintains two columns that identify the tile id and primitive id that are related to the feature. This primitive_id column is named after the primitive table it relates to, followed by "_id". The tile id is named tile_id. The column from_to is optional and is used to provide directionality of a line feature with respect to its edge primitive(s).

A from_to value of 1 means the feature has the same orientation as its related primitive(s); a value of -1 means opposite orientation.

Table C-33 Feature Table Definition (1:1 or N:1)

Column Name	**Description**	**Column Type**	**Key Type**	**Op/Man**
id	Feature primary key	I	P	M
tile_id	Tile id	S	N	MT [1]
*_id [2]	Primitive id	S/I/K	N	M
<attribute n>	(nth description)	Any	Any	M [3]
from_to	Line feature orientation	S	N	O[4]

Notes:
1. If primitive id column is of type K, then no tile reference id column is required.
2. The asterisk (*) indicates a placeholder for the primitive table name: edg, fac, nod, or txt.
3. A minimum of one attribute is required for any table. If the table does not contain tile_id and/or *_id columns, an attribute column will be mandatory. Any additional attributes are optional.
4. Optional for line feature tables only.

Table C-33A defines feature table contents for compound simple features that are linked to one or many spatial elements (i.e. one or many primitive records) through a feature join table. The three fields, tile_ id, *_id, and from_to, of table C-33A belong then to the Feature Join Table.

Table C-33A Feature Table Definition (1:N or N:M)

Column Name	**Description**	**Column Type**	**Key Type**	**Op/Man**
id	Feature primary key	I	P	M
<attribute n>	(nth description)	Any	Any	M [1]
**.ajt_id[2]	"Area to Edge" Feature Join Table id[3]	S/I	N	O[3]

Notes:
1. A minimum of one attribute is required for any table. Any additional attributes are optional.
2. The asterisk (**) indicates a placeholder for the join table name: *.ajt

3. For area feature tables only. A link to a special feature join table that defines edges composing the boundary(ies) of area features in topology levels 0,1,2 and provides an efficient way to identify boundaries for multi-faced areas in topology level 3. Refers to the first record describing the area feature. Mandatory for area features in topology levels 0, 1, and 2, the tile_id and *_id fields are then omitted. Optional for area in topology level 3.

Table C-33B defines feature table contents for complex features that are linked to one or many feature tables through one or many feature join tables.

Table C-33B Complex Feature Table Definition

Column Name	**Description**	**Column Type**	**Key Type**	**Op/Man**
id	Feature primary key	I	P	M
<attribute n>	(nth description)	Any	Any	M [1]

All feature class definitions fall into one of five categories: area, line, point, text, and complex features.

[Area feature - Level 3] An area feature is composed of one or more face primitives. The Area_to_Edge join table is an optional index that speeds area cycles when areas consist of many faces, possibly in more than one tile. The primitive id column will be fac_id.

[Area feature - Levels 2, 1, 0] An area feature is defined by a set of edges which are related to the area feature through a specific area feature join table called the Area_to_Edge join table. This table lists the edges in sequence first, moving in a clockwise direction around the single outer most boundary of the area, and then moving counterclockwise around the boundaries of any inner areas or voids. The Area Feature Table will be as defined in table C-33A even if all area features are described by a single outer edge.

Table C-34 Area_to_Edge Join Table Definition

Column Name	**Description**	**Column Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
*.aft_id [1]	Feature id	I	N	M
tile_id	Tile id	S	N	MT
edg_id	Signed Edge Primitive ID or area flag [2]	S/I	N	M

Notes:

1. The asterisk (*) is a placeholder for the area feature class name.
2. The edg_id field is assigned a positive or negative value to indicate the orientation of the edge in sequencing around the boundary of an area feature. The absolute value is used for the edge reference. If inner areas or voids are present, the edg_id field is assigned a zero (0) value to indicate when an area boundary cycle has been completed. The next edge value, after a zero flag, will be the first edge of an inner area or void.

Line feature. Line features are composed of edge primitives. The primitive id column name will be edg_id. A linear river feature will typically contain attributes such as; jurisdiction, depth, and width. from_to is an optional column used to provide directionality of a line feature with respect to its edge primitive(s). A from_to value of 1 means the feature has the same orientation as its related edge(s). A value of -1 means the feature has the opposite orientation.

Point feature. A point feature class contains node primitives, but can have numerous attributes describing the feature. The primitive column name will be nod_id (end_id or cnd_id if nodes are modeled as separate tables). An airport point feature, for example may contain many attributes such as the number of runways, number of terminals, elevation, and size.

Text feature. A text feature class is composed of a text primitive. The primitive column name will be txt_id. The text feature usually contains additional attributes, such as font, color, and font size. Font type, color, and size are sometimes included in a related attribute table.

Complex feature. Complex features can be constructed from any combinations of features within a coverage. The previous airport example could be redefined to include point, line, and area feature classes, when this composition is required for specific applications. Complex features cannot be recursively defined.

Features may have distinct relations such as stacked-on (STK), stacked-under (STU), conjunction (CON), disjunction (DIS), or alternative (ALT) representations (refer to Part 2, Clause One, Paragraph 5.1.1.3 for an explanation of relations between features). When preserving this type of information, an optional feature relations join table (fjt) may be invoked. Table C-35 defines the fjt implementation required to maintain feature relations in a coverage.

Table C-35 Feature Relations Join Table (Coverage Level) Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Record id number	I	P	M
*_id [1]	id of feature which has a relation	I	N	M
rel_type	type of relationship (STK or STU, CON or DIS, ALT)	T,3	N	M
**_id [2]	id of feature to which the relationship applies.	I	N	M

Notes:
1. The asterisk (*) indicates a placeholder for the feature class name of the first table in the relationship.
2. The double asterisk (**) indicates a placeholder for the feature class name of the second table in the relationship. Separate feature relations join tables will be required for each two table combination which have these relations

The above feature relationships may be maintained across coverages. This requires a Feature Relations Join Table at the library level. This table will store the type of feature relation in addition to the coverage id, feature class id and feature id for the respective features. Table C-35A defines the fjt implementation required to maintain cross coverage feature relations.

Table C-35A Feature Relations Join Table (Library Level) Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Record id number	I	P	M
cov1_id	Coverage id (from Coverage Attribute Table)	I	N	M
fc1_id	Feature Class id (from Feature Class Attribute Table in coverage 1)	I	N	M
fea1_id [1]	id of feature which has a relation	I	N	M
rel_type	Type of relationship (STK or STU, CON or DIS, ALT)	T,3	N	M
cov2_id	Coverage id (from Coverage Attribute Table)	I	N	M
fc2_id	Feature Class id (from Feature Class Attribute Table in coverage 2)	I	N	M
fea2_id [2]	id of feature to which the relationship applies.	I	N	M

Table C-35B Feature Relations Join Table (Library Level) Example

id	**cov1_id**	**fc1_id**	**fea1_id**	**rel_type**	**cov2_id**	**fc2_id**	**fea2_id**
.	.	.	.	.	.	.	.
5	9	12	23	STK	4	6	67
6	9	12	1124	STK	2	15	154
7	9	13	9898	STK	4	6	67
.	.	.	.	.	.	.	.
22	7	4	34	STU	8	3	11240
23	7	11	445	STU	9	22	667
.	.	.	.	.	.	.	.

Features may have alternative relationships when they are represented in the different libraries. For example, a bridge may be a line feature in a relatively high resolution library, and as a point in a lower resolution library. A Unique Identification (UID) may be assigned as an attribute of this particular bridge. Whenever the bridge occurs in a library, the UID signifies that it is that particular bridge.

C.2.3.3.2 Feature Join Tables

Join tables are used to implement one-to-many relationships and/or many-to-many relationships between tables. They allow a variety of constructs to be made: between a feature table and a primitive table, between feature tables (for complex features), and between a feature table and an attribute table. A join table is used to relate one column in a table with a column in a second table. This is common when a 1:N relationship exists between two tables. Thus, for example, if a complex feature has four 1:N relations with four simple feature classes, four join tables will be used. The content and number of join tables depend on the coverage design and is defined in the Product Specification. The feature class schema table (fcs) documents all of the relationships for each feature class.
Unlike the feature table format that is dependent on the category (area, line, point, text, and complex), the same join table format is applicable to all feature classes. In the description below, a feature to primitive join relationship is modeled using the structure presented in Table C-36. Together with the Feature Class Schema (fcs) table, this table structure can be utilized to represent other relationships; such as, relationships defined among features, among primitives, and so forth. Since joins are fully described in the fcs, there is usually no predetermined requirement on the join table columns. Furthermore, unless one of the tables in the join relation is a primitive table in a tiled coverage, the tile_id column is generally not necessary. Table C-36 defines join table contents. The second column contains the key of the first table in the join. The name of this column is the table's name followed by "_id." The fourth column contains the key of the second table in the join; it is similarly labeled using the table name and the "_id" suffix. For instance, in a feature table (sdrpoint.pft) relating to a primitive (nod), the key for the first table is sdrpoint.pft_id and the key for the second table is nod_id. The feature class schema table is used to express the column names for the join table. The tile_id column is mandatory when tiled directories exist in the coverage and the join table relates a feature table and a primitive table.

Refer to clause C.2.3.3.3 concerning these columns when tile directories exist in the coverage.

Table C-36 Feature Join Table Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
*_id [1]	Table 1 id	I	N	M
tile_id	Tile id	S	N	MT/ O [3]
**_ id [2]	Table 2 id	I/S/K	N	M
from_to	Linear feature orientation	S	N	O[4]
edit_date	Date of collect/edit operation	D	N	O[5]
attribute[6]	Affected attribute name	T,*	N	O[5]

Notes:

1. The asterisk (*) indicates a placeholder for the name of the first table in the join, usually a feature table name.
2. The (**) indicates a placeholder for the table name of the second table in the join, usually one of edg, fac, nod, or txt.
3. If primitive id column is of type K, then no tile reference id column is required. Tile_id is also not required if this is a join between two feature tables, as in complex features, or between a feature table and an attribute table.
4. Optional for line features only.
5. Optional for join tables linking feature tables to source.rat.
6. A null value in this field signifies that the entire feature is affected.

C.2.3.3.3 Feature-to-Primitive Relations on Tiled Coverages

If the coverage is tiled, the feature or join table maintains a triplet id column that identifies the tile id and primitive id that are related to the feature. Alternatively, the tile id and the primitive id can be stored in separate columns. The triplet id column is named after the primitive table it relates to, followed by "_id". The first field in the triplet id is null. The second field is the tile id (found in the tile reference coverage); the third field is the primitive row id in that tile. If the relation between the primitive and the feature is made using a join table, then this relation will be stored in the join table.

C.2.3.3.4 Feature-to-Feature Connectivity

VRF supports the identification of feature-to-feature connectivity for network analysis. This is implemented using a separate point feature table (related to the connected node) which links to a connectivity related attribute table through a join table. The point feature table will have: row_id and tile_id/cnd_id (or nod_id). The related attribute table will contain: row_id, fc_id/feature_id/true azimuth for the first feature, fc_id/feature_id/true azimuth for the second feature and a connectivity code for the two feature relation. Connectivity can be: F - forward only (only from the first feature to the second feature), R - reverse only (only from the second feature to the first feature) or B - bi-directional.

At each connectivity point feature, a wagon-wheel progression about the first_edge for the associated cnd will identify all possible feature pairs (combinations) of possible connectivity about the node. There will be a record in the related attribute table for each viable combination about the point. There will be no records in the related attribute table for combinations which have no connectivity or which have no line feature association. NOTE: Product specifications may reverse the connectivity option to show only cases where there is no connectivity.

C.2.3.3.5 Feature Stacked-on/Stacked-under

Stack-on/stacked-under relations are described in C.2.3.3.1 and are implemented through the use of a feature relations join table (fjt). Refer to table C-35A and C-35B for a description and example of a fjt.

C.2.3.3.6 Multi-Value Attributes

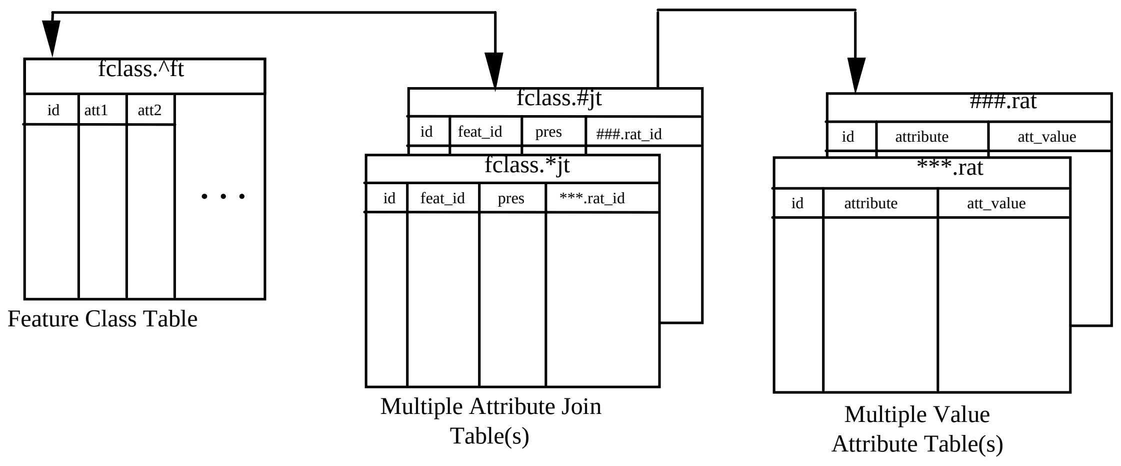
The VRF encapsulation supports multi-value attributes by the use of join tables linking related attribute tables to feature tables.

A join table may be used to link a feature class table to a multi-value attribute related attribute table (*.rat). When attributes with different types (ex. char and int) have multi-attribute values within a coverage, separate rats are required for each attribute type. Standardized coded values meaning "multiple" are used in place of the actual values which are stored in the *.rat (see Clause C.2.3.4.4). Naming for rats is equivalent to defining names for feature tables within a product (ex: names used for *.lft in a product). Therefore, rat table use and naming is defined in the product specifications.

These rat tables will contain an attribute value column and an optional attribute column. The attribute column will contain the attribute name, and is used to allow multi-attributes of the same attribute value type (e.g. integer) in a single rat table.

Since order is important for some types of multi-value attributes, a mechanism is provided to identify their precedence. An optional column is defined in a join table used to join a feature table to a rat table that defines this precedence.

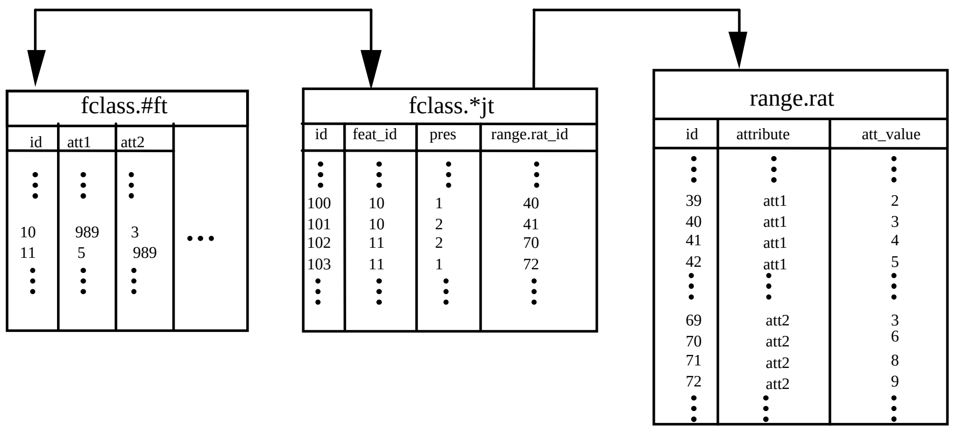
The following illustrates these attribute relationships:



Note: ^,*,# Actual names and extensions defined in product specification

Figure C-19 Multiple-value Attribute Relationships

The following is an example of a multi-value attribute relationship.



fclass.#ft

id	att1	att2	
⋮	⋮	⋮	
10	989	3	...
11	5	989	
⋮	⋮	⋮	

fclass.*jt

id	feat_id	pres	range.rat_id
⋮	⋮	⋮	⋮
100	10	1	40
101	10	2	41
102	11	2	70
103	11	1	72
⋮	⋮	⋮	⋮

range.rat

id	attribute	att_value
⋮	⋮	⋮
39	att1	2
40	att1	3
41	att1	4
42	att1	5
⋮	⋮	⋮
69	att2	3
70	att2	6
71	att2	8
72	att2	9
⋮	⋮	⋮

Figure C-20 Multiple-value Attribute Relationship Example

C.2.3.4 Coverage

Each coverage has one set of topological primitives and a collection of feature tables based upon these topological objects. All these tables are stored in one directory and are associated by file naming conventions (see Table C-16). The coverage may contain a data dictionary for all feature tables in the coverage.

An optional data quality table is allowed at the coverage level.

C.2.3.4.1 Coverage Relationships

The general description of a coverage is stored in the coverage attribute table of its library. The relationships between the tables in a coverage are described by the mandatory feature class schema table.

C.2.3.4.2 Feature Class Schema Table

A feature class schema table defines the feature classes that are contained within the coverage. Each record in the table specifies a feature class name, the name of the two tables involved in the join, and the names of the columns used in the join. The feature class name must be repeated to specify all the relationships in the feature class schema table. In the case of complex features, all relationships defining the component features must be specified with the feature class column referencing the complex feature class. This includes relationships between simple features and their primitives, even though these relationships are defined in other parts of the feature class schema table. Paragraph C3.4.7.3 of Appendix C3 contains an example of relationships in a coverage with several simple features and a complex feature that consists of one point feature class and two line feature classes. The topological relationships between primitive tables need not be specified, since they are implied by table types.

The feature class schema table must be used in conjunction with the tileref area feature table to fully define feature classes in tiled coverages.

If a key in the join is a compound key, the column names will be listed, separated by a backslash character (\). For example, a primary key composed of two columns would be specified as "name\type." Table C-37 defines feature class schema contents, and Table C-38 is an example of a feature class schema table.

Table C-37 Feature Class Schema (Coverage Level) Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
feature_class	Feature class name	T,8	N	M
table1	The first table in a relationship	T,12	N	M
table1_key	Join column in the first table	T,*	N	M
table2	The second table in a relationship	T,12	N	M
table2_key	Join column in the second table	T,*	N	M

Table C-38 Feature Class Schema (Coverage Level) Example

id	feature_class	table1	table1_key	table2	table2_key
1	sdrpoint	sdrpoint.pft	id	nod	id
2	sdrpoint	nod	id	sdrpoint.pft	id
3	sdrarea	sdrarea.aft	id	fac	id
4	sdrarea	fac	id	sdrarea.aft	id

A feature class schema table is required within the data library if "cross coverage feature relations" are defined (see C.2.3.3.1, Table C-35A, B). The feature class schema in this specific application is similar to the one used to store feature class relationships within a coverage. The coverage for the feature relations are required columns in order to link the respective feature classes to the appropriate coverage. The entry in the feature_class column will be the feature class represented by the first feature table in the relationship. Table C-38A defines the feature class schema table for library level relations.

Table C-38A Feature Class Schema (Library Level) Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
feature_class	Feature class name	T,8	N	M
cov1	The coverage of the first table in a relationship	T,8	N	M
table1	The first table in a relationship	T,12	N	M
table1_key	Join column in the first table	T,*	N	M
cov2	The coverage of the second table in a relationship	T,8	N	M
table2	The second table in a relationship	T,12	N	M
table2_key	Join column in the second table	T,*	N	M

Table C-38B Feature Class Schema (Library Level) Example

id	feature_class	cov1	table1	table1_key	cov2	table2	table2_key
1	roadl	trn	roadl.lft	id	hyd	bridgep.pft	id
2	roadl	hyd	bridgep.pft	id	trn	roadl.lft	id
3	towerp	uti	towerp.pft	id	cul	buildp.pft	id
4	towerp	cul	buildp.pft	id	uti	towerp.pft	id

C.2.3.4.3 Value Description Table

A value description table relates to associated feature class tables within a coverage. Vdts are required when coded attribute values are implemented within a coverage. The two types of vdts are integer vdt and character vdt. There will be no more than one of each per coverage. If a column in a feature table requires a vdt, then every unique coded value will have an entry in the vdt. Certain values (e.g., null or unknown) that are globally defined for all columns may not appear in a vdt, however, they will be documented in the product specifications. Table C-39 defines the format of a vdt, and Table C-40 provides an example.

Table C-39 Value Description Table Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
table	Feature table name	T,12	N	M
attribute	Column name	T,n	N	M
value	Unique attribute value	S/I/T,n	N	M
description	Attribute description	T,*	N	M

Table C-40 Integer Value Description Record Example

id	table	attribute	value	description
1	sdrpoint.pft	mcc	0	unknown
2	sdrpoint.pft	mcc	21	concrete
3	sdrpoint.pft	mcc	30	earthen
4	sdrpoint.pft	hyc	0	unknown
5	sdrpoint.pft	hyc	6	non-perennial/intermittent/fluctuating
6	sdrpoint.pft	hyc	8	perennial/permanent

C.2.3.4.4 Coded Values

The table below defines common standardized coded values typically associated with feature tables. Refer to clause C.2.5 for additional information on data syntax. Coded character values and coded integer values are listed in the proper value description tables. Integer values and floating point values store a "null" within the feature table. The related information is store in a Standardized Attribute Value (sat.rat) Table and uses a join table to link the coded value to the appropriate description. See tables C-40B, C.

Table C-40A Standardized Coded Values

Attribute Type	Null/No Value	Unknown	Multiple	Unpopulated	Not Applicable	Other
Text (T)						
Fixed Length	N/A[1]	UNK	MUL	N_P	N_A	OTH
Variable Length	0 Length	UNK	MUL	N_P	N_A	OTH
Coded	-32768[2]	0	989	997	998	999
Integer						
Short Integer	-32768[2]	-32768	-32768	-32768	-32768	-32768
Long Integer	-2147483648[3]	-2147483648	-2147483648	-2147483648	-2147483648	-2147483648
Floating Point						
Single Precision	NaN	NaN	NaN	NaN	NaN	NaN
Double Precision	NaN	NaN	NaN	NaN	NaN	NaN

Notes: The value for "Null" for each data type is defined in Table C-67.

1. If the length is one or two "-" or "--" should be used instead (refer to Clause C.2.5.4)

2. The Null value for a short integer is defined to be the bit pattern 10000000 00000000, which is equivalent to the maximum negative number in "two's complement number format." Therefore for a 16-bit length number, the corresponding value for Null is -32768.

3. The Null value for a long integer is defined to be the bit pattern 10000000 00000000 0000000 0000000, which is equivalent to the maximum negative number in "two's complement number format." Therefore for a 32-bit length number, the corresponding value for Null is -2147483648.

Table C-40B Standardized Attribute Value Join Table

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
fea_id	Feature id	I	N	M
attribute	Attribute Name	T,*	N	M
sat.rat_id	Standardized Attribute Value id	I	N	M

Table C-40C Standardized Attribute Value (sav)

sav code	description
1	NULL (Empty)
2	Unknown
3	Multiple
4	Unpopulated
5	Not Applicable
6	Other

C.2.3.5 VRF Library

The function of a VRF library is to organize collections of coverages that pertain to one geographic region. Each library must manage its own spatial extent. VRF uses a minimum bounding rectangle to define this geographic extent. VRF also supports a coverage, the library reference coverage, which describes the library region in a graphic manner. All reference coverages must be spatially registered and be in the same coordinate system.

Within each library, there are three mandatory tables and seven optional tables. The library header table defines the contents of the library. The geographic reference table contains information pertaining to the geographic location of the library. A coverage attribute table provides a list of and descriptions for the coverages contained in the library.

If the library is tiled, there will be an untiled tile reference coverage. There will also be an untiled library reference coverage that gives a general overview of the information contained in the library. In addition, it is possible to include a data quality coverage. The format for the data quality and library reference coverage follow the same rules as any normal VRF coverage. See Appendix C5 for information concerning recommended data quality coverage contents. Multiple records in these tables are used to describe multiple sources, updates and maintenance issues.

C.2.3.5.1 Library Header Table

The library header table contains information identifying the library, general information about the contents, security, and source. Table C-41 describes the group of entities that compose the library header table.

Table C-41 Library Header Table

Column Name	**Description of Content**	**Column Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
product_type	Series designator of product type	T,12	N	M
library_name	Name of library	T,12	N	M
description	Text description of library	T,100	N	M
data_struct_code	Highest level code for the library 5 = Level 0 topology 6 = Level 1 topology 7 = Level 2 topology 8 = Level 3 topology	T	N	M
scale	Source scale of the library (i.e. 200) using the representative fraction denominator	I	N	M
source_series	Series designator (e.g. 1501)	T,15	N	M
source_id	Source id - number or name that when used in conjunction with the series and edition will identify a unique source	T,30	N	M
source_edition	Source edition number	T,20	N	M
source_name	Full name of source document	T,100	N	M
source_date	Significant date. A designed date that most accurately describes the basic date of the product for computation of the probable obsolescence date. It can be the compilation date or the revision date or other depending on the product and circumstances	D	N	M

security_class	Security classification
	T = TOP SECRET
	S = SECRET
	C = CONFIDENTIAL
	R = RESTRICTED (or alternatively "FOR OFFICIAL USE ONLY" administrative classification only)
	U = UNCLASSIFIED
		T	N	M

downgrading	Originator's permission for downgrading required (yes or no)	T,3	N	M
downgrading_date	Date of downgrading (null if answer to previous entity is yes)	D	N	M
releasability	Releasability restrictions	T,20	N	M
edition_number	Edition number for this library	T,10	N	O

edition_date	Creation date of this library	D	N	O

encapsulation	Coverage encapsulation
	
	BASIC TEXT (L)
	Code identifying the encapsulation primarily used for the transmission of this Dataset [Library]
	A = ISO 8211 (Annex A)
	B = ISO 8824 (Annex B)
	C = VRF (Annex C)
	D = IIF (Annex D)
	X = Mixed encapsulations
	Must be present when the encapsulation is not homogeneous within the library.
		T	N	O

C.2.3.5.2 Geographic Reference Table

This table (Table C-42) contains four groups of fields that define the geographic parameters of the library. These field groups are the geographic parameters, projections, registration points table name, and diagnostic points table name. The geographic parameters in this table serve two purposes. Firstly, they are for descriptive documentation of the coordinate reference system used in the database. Secondly, they allow multiple separately published libraries to be inversely projected into a common coordinate system for display and query. If the database is maintained in unprojected geographic coordinates, the projection code and its corresponding projection parameters need not be included in the table. (Refer to Part 3 clause 6 for valid values in datum codes, projection codes, and unit of measure codes.)

Table C-42 Geographic Reference Table

Column Name	**Description of Contents**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
data_type	Type of coordinate data in the library	T,3	N	M
units	Units of measure code for X and Y coordinates in library (Units must be consistent with the data_type)	T,3	N	M
ellipsoid_name	Name of ellipsoid of the library	T,*	N	M[1]
ellipsoid_code	Ellipsoid code	T,3	N	M[1]
ellipsoid_detail	Details about library ellipsoid	T,50	N	O
vert_datum_name	Name of vertical datum reference	T,*	N	M
vert_datum_code	Code of vertical datum reference	T,4	N	M
vert_units	Units of measure code for Z values	T,3	N	O
sound_datum_name	Name of sounding datum	T,*	N	M
sound_datum_code	Code of sounding datum reference	T,4	N	M
sound_units	Units of measure code for soundings	T,3	N	O
geo_datum_name	Name of geodetic datum	T,*	N	M
geo_datum_code	Code of geodetic datum	T,4	N	M
projection_name	Name of the projection	T,*	N	O[2]
projection_code	Code of the projection (null if geographic coordinates)	T,2	N	O[3]
parameter1	Projection parameter 1	F	N	O[3]
parameter2	Projection parameter 2	F	N	O[4]
parameter3	Projection parameter 3	F	N	O[4]
parameter4	Projection parameter 4	F	N	O[4]
false_origin_x	False Easting origin of projection	F	N	O[5]
false_origin_y	False Northing origin of projection	F	N	O[5]
false_origin_z	False origin for Z values	F	N	O[5]
reg_pt_table	Registration point table	T,12	N	O
diag_pt_table	Diagnostic point table	T,12	N	O

Notes**:**
1. Either one or the other of ellipsoid_code or ellipsoid_name is mandatory.
2. Mandatory for data types MAP and DIG. Not used for data type GEO.
3. Mandatory only when projection_name is present.
4. May be used when projection_name is present
5. Present only when relevant.

Data_type in this table is type of coordinate system (Part 2 – 10.1.2)
See Part 3 Clause 6 for further details on geodetic parameters.

C.2.3.5.3 Coverage Attribute Table

This table contains the coverage name and topology level for each coverage in the library. Only those coverages which actually exist within a specific library will be listed in that library's coverage attribute table. The topological level associated with each coverage determines the nature of geometric and topological information available on that coverage. Table C-43 defines coverage attribute table entities. Table C-44 is an example of a coverage attribute table.

Table C-43 Coverage Attribute Table Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	U	M
coverage_name	The coverage name	T,8	P	M
description	Description string	T,*	N	M
level	The topological level	I	N	M
encapsulation	Coverage encapsulation BASIC TEXT (L) Code identifying the encapsulation primarily used for the transmission of this Dataset [Library] A = ISO 8211 (Annex A) B = ISO 8824 (Annex B) C = VRF (Annex C) D = IIF (Annex D) X = Mixed encapsulations Must be present when the encapsulation is not homogeneous within the library.	T	N	O

Table C-44 Coverage Attribute Table Example

id	coverage_name	description	level
1	ind	Industry	3
2	veg	Vegetation	3
3	aer	Aeronautical	3
4	iwa	Inland Water	3
5	gtr	Ground Transportation	3
6	phy	Physiography	3

C.2.3.5.4 Tile Reference Coverage

A library which contains tiled partitions will require a tile reference coverage, tileref, with associated area feature and attribute tables. Thus, if the tile reference coverage does not exist in a library, then no tiling scheme exists. The tile reference coverage consists of a set of faces identifying the tiles that the library uses to subdivide the region of interest. Tile attributes are used to optimize retrievals. At the library level, the tile geometry is simple, describing the location of the tile boundaries.

C.2.3.5.4.1 Tile Attributes

The tile attributes are located in the area feature table of the tile reference coverage. This table maintains a unique id and its associated tile directory name. This directory name contains the path name relative to the coverage level. Additional columns may exist for every feature class or coverage located in the library, but this is optional.

When the tile_id column is used in a table, the value relates to the id column in the tileref.aft.

Table C-45 defines the tile reference coverage attribute columns. Table C-46 is an example of a tile reference coverage area feature table.

Table C-45 Tile Reference Area Feature Table Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
tile_name	Tile name	T,n (n<=64)	U	M
fac_id	Face id	S/I/K	N	M
origin	Coordinates of tile origin	C/Z/B/Y/G/H/V/W	N	O[1]
scale	Scale factor for tile coordinate conversion	S/I	N	O

Note:
1. Present if primitive coordinates are relative coordinates and are offsets to this origin.

Table C-46 Tile Area Feature Record Example

id	tile_name	fac_id
1	MJ12	5
2	MJ22	7
3	MJ23	2
4	MJ24	3

C.2.3.5.5 Registration Point Table

The registration point table includes columns for the longitude (or easting), latitude (or northing) and elevation of ground points selected as registration points. This table also includes columns for the corresponding local x, y and z coordinates of registration points. Each registration point in this table is identified by a row id and a registration point id. The geographic reference table contains an optional column containing the registration point table name, which carries the file extension .rpt. (See Table C-47.)

Table C-47 Registration Point Table

Column Name	**Description of Contents**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
reg_pt_id	Registration point id	I	N	M
reg_long	Longitude (or Easting) of registration point	F/R/S/I	N	M
reg_lat	Latitude (or Northing) of registration point	F/R/S/I	N	M
reg_z	Elevation of registration point	F/R/S/I	N	M
reg_local_x	Local X coordinate of registration pt	F/R/S/I	N	M
reg_local_y	Local Y coordinate of registration pt	F/R/S/I	N	M
reg_local_z	Local Z coordinate of registration pt	F/R/S/I	N	M

Notes**:**

Coordinates derived from digitizing equipment can be considered a subset of local coordinates.

If elevations are not included, columns reg_z and reg_local_z may be null-filled.

C.2.3.5.6 Diagnostic Point Table

The diagnostic point table includes columns for the longitude (or easting), latitude (or northing) and elevation of ground points selected as diagnostic points. This table also includes columns for the corresponding local x, y and z coordinates of diagnostic points. Each diagnostic point in this table is identified by a row id and a diagnostic point id. The geographic reference table contains an optional column containing the diagnostic point table name, which carries the file extension .dpt. (See Table C-48.)

Table C-48 Diagnostic Point Table

Column Name	**Description of Contents**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
diag_pt _id	Diagnostic point id	I	N	M
diag_long	Longitude (or Easting) of diagnostic point	F/R/S/I	N	M
diag_lat	Latitude (or Northing) of diagnostic point	F/R/S/I	N	M
diag_z	Elevation of diagnostic point	F/R/S/I	N	M
diag_local_x	Local X coordinate of diagnostic pt	F/R/S/I	N	M
diag_local_y	Local Y coordinate of diagnostic pt	F/R/S/I	N	M
diag_local_z	Local Z coordinate of diagnostic pt	F/R/S/I	N	M

Notes:

Coordinates derived from digitizing equipment can be considered a subset of local coordinates.

If elevations are not included, columns diag_z and diag_local_z may be null-filled.

C.2.3.6 Database

Information that applies to the whole collection of data belongs at the database level. Structurally, a database consists of a set of libraries. It is possible to include a data quality table.

There are two mandatory tables: the library attribute table and the database header table. These two tables are described below. Multiple records in each table are used to describe multiple sources, updates, and maintenance issues.

C.2.3.6.1 Library Attribute Table

The library attribute table contains the name and extent for each library in the database. The library minimum bounding rectangle shall be the maximum and minimum coordinates. Table C-49 defines library attribute entities. Table C-50 is an example of a library attribute table.

Table C-49 Library Attribute Table Entity Definitions

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	U	M
library_name	Library name	T (L/M/N)[1],8	P	M
xmin	Westernmost Longitude (or Easting)	F/R	N	M
ymin	Southernmost Latitude (or Northing)	F/R	N	M
xmax	Easternmost Longitude (or Easting)	F/R	N	M
ymax	Northernmost Latitude (or Northing)	F/R	N	M

Note: 1. Column type for library_name is T. Types L, M, N are maintained for backward compatibility.

Table C-50 Library Attribute Table Example

id	library_name	xmin	ymin	xmax	ymax
1	noamer	-172.00	31.167000	-57.00	31.250000

C.2.3.6.2 Database Header Table

The database header table (Table C-51) contains information that defines database content and security information.

Table C-51. Database Header Table Definition

Column Name	Description of Contents	Column Type	Key Type	Op/Man
id	Row id	I	P	M
vrf_version	VRF version number	T,10	N	M
database_name	Directory name of the database	T,8	N	M
database_desc	Text description of the database	T,100	N	M
media_standard	Media standard used for the database	T,20	N	M
originator	Text for title and address of originator (a backslash "\" is used as a line separator)	T,*	N	M
addressee	Text for title and address of addressee (a backslash "\" is used as a line separator)	T,*	N	M
media_volumes	Number of media volumes comprising the database	T,*	N	M
seq_numbers	Sequential number(s) for each media volume in this database	T,*	N	M
num_data_sets	Number of libraries within database	T,*	N	M

security_class	Security classification of database (the highest security classification of the information package)
	T = TOP SECRET
	S = SECRET
	C = CONFIDENTIAL
	R = RESTRICTED (or alternatively "FOR OFFICIAL USE ONLY")
	U = UNCLASSIFIED
		T,1	N	M

downgrading	Originator's permission for downgrading required (yes or no)	T,3	N	M

downgrade_date	Date of downgrading	D	N	M
releasability	Releasability restrictions	T,20	N	M
other_std_name	Free text, note of other standards compatible with this database	T,50	N	O
other_std_date	Publication date of other standard	D	N	O
other_std_ver	Other standard amendment number	T,10	N	O
transmittal_id	Unique id for this database	T,*	N	M
edition_number	Edition number for this database	T,10	N	M
edition_date	Creation date of this database	D	N	M

C.2.3.7 Data Quality

The data quality table may be stored at the database, library, or coverage level. It contains information on the completeness, consistency, date status, attribute accuracy, positional accuracy, and other miscellaneous quality information. It also contains information about the source from which the geographic data was derived. Lineage information should be included in the associated narrative table, named "lineage.doc." While the contents and location of a data quality table within a VRF database are product specific, it is highly recommended that at least one table exist at the library level. Table C-52 defines the contents of the data quality table. Appendix C5 contains more information about storing data quality information in VRF.

Table C-52 Data Quality Table Definition

Column	**Name Description of Contents**	**Column Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
vrf_level	Either DATABASE or LIBRARY or COVERAGE	T,8	N	M
vrf_level_name	Directory name of database or library or coverage	T,8	N	M
feature_complete	Feature completeness percent	T,*	N	M
attrib_complete	Attribute completeness percent	T,*	N	M
logical_consist	Logical consistency	T,*	N	M
edition_num	Edition number	T,8	N	M
creation_date	Date of creation	D	N	M
revision_date	Date of revision	D	N	M
spec_name	Name of product specification (e.g. DCW)	T,*	N	M
spec_date	Date of product specification	D	N	M
earliest_source	Date of earliest source	D	N	M
latest_source	Date of latest source	D	N	M
quant_att_acc	Standard deviation of quantitative attributes	T,*	N	O
qual_att_acc	Percent reliability of qualitative attributes	T,*	N	O
collection_spec	Name of collection specification	T,*	N	M
source_file_name	Name of included source file	T,12	N	O
abs_horiz_acc	Absolute horizontal accuracy of database or library or coverage	T,*	N	M
abs_horiz_units	Unit of measure for absolute horizontal accuracy	T,20	N	M
abs_vert_acc	Absolute vertical accuracy of database or library or coverage	T,*	N	M
abs_vert_units	Unit of measure for absolute vertical accuracy	T,20	N	M
rel_horiz_acc	Point-to-point horizontal accuracy of database or library or coverage	T,*	N	M
rel_horiz_units	Unit of measure for point-to-point horizontal accuracy	T,20	N	M
rel_vert_acc	Point-to-point vertical accuracy of database or library or coverage	T,*	N	M
rel_vert_units	Unit of measure for point-to-point vertical accuracy	T,20	N	M
comments	Miscellaneous comments - free text	T/L/M,*	N	M

Multiple records in each table are used to describe multiple sources, updates, and maintenance issues.

C.2.3.8 Narrative Table

The narrative table (Table C-53) will contain any descriptive information concerning its associated table. The contents of the text column will be product specific.

Table C-53 Narrative Table Definition

Column Name	Description	Column Type	Key Type	Op/Man
id	Row id	I	P	M
text	Text information	T/L/M/N,*	N	M

C.2.3.9 Names Reference Coverage

The names reference coverage is optional at the library level. The names reference coverage must maintain the same bounding rectangle as the other library coverages.

The names reference coverage will contain at least the following: a point feature table; a node primitive table; and a thematic index created on a fixed-length text column in the point feature table. The column's name will be "place_name". Other feature classes may exist, but are not required.

C.2.4 VRF Encapsulation

Encapsulation defines the structure of data fields and the grouping of these fields. A simple table-oriented data encapsulation system is defined for VRF which allows the use of binary coded numeric data as well as text data and which uses references to identify the location of data elements within numerous related tables stored in files.

C.2.4.1 Table Definition

A VRF table consists of a header and at least one record. See C.2.2.1.3 and C.2.2.2.3.3 for the exceptions to this rule. Records can be of variable-length; indexes can be used to directly access the files as needed. A table will begin with a header defining the table contents, followed by a series of records (rows). Where table records are not of fixed length, an external index containing record offsets and length (in bytes) will be used to provide direct access (see Figure C-21).

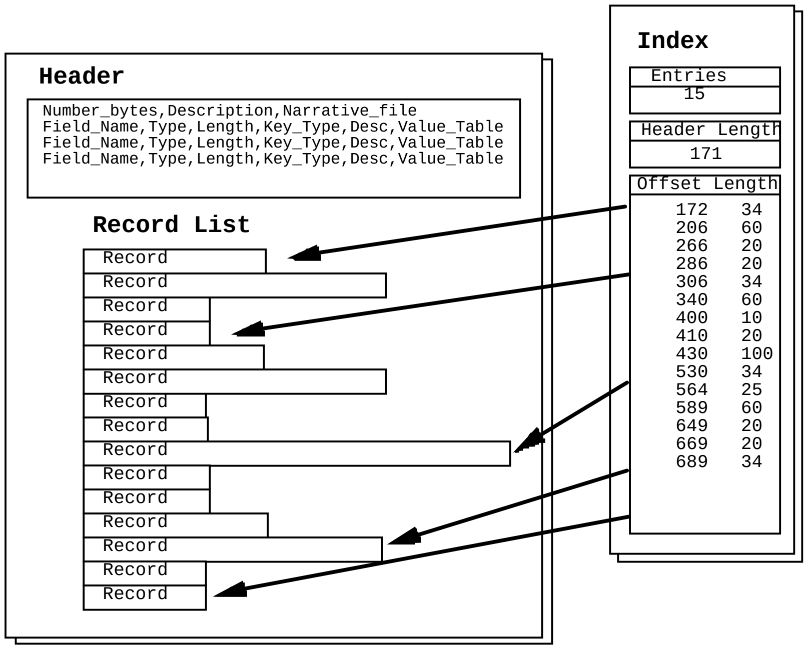

Header
Number_bytes,Description,Narrative_file
Field_Name,Type,Length,Key_Type,Desc,Value_Table
Field_Name,Type,Length,Key_Type,Desc,Value_Table
Field_Name,Type,Length,Key_Type,Desc,Value_Table
Record List
Record
Index
Entries
15
Header Length
171
Offset Length
172 34
206 60
266 20
286 20
306 34
340 60
400 10
410 20
430 100
530 34
564 25
589 60
649 20
669 20
689 34

Figure C-21 Table Structure

C.2.4.1.1 Header

A table header (described in Table C-54) is composed of two parts. The first part is a 4-byte integer that indicates the length of the following text string, which defines the table. To accommodate different hardware architecture, after the length field a byte order field may be inserted. A value of "L" in the byte order field indicates a least-significant-byte-first encoding; an "M" indicates a most-significant-byte-first encoding. Least-significant-byte-first encoding is assumed if the byte order field is not present

The second part, the header definition text string, contains three components, each of which is separated by a semicolon. The semicolon (;) indicates the end of the component.

Table C-54 Header Field Definitions

Field	**Description**	**Column Type**
Header length	Length of ASCII header string (i.e., the remaining information after this field)	I
Byte order	Byte order in which table is written: L - least-significant-first M - most-significant-first	T,1
	(semicolon separator)	T,1
Table description	Text description of the table's contents	T,n (n<=80)
	(semicolon separator)	T,1
Narrative table	An optional narrative file which contains miscellaneous information about the table	T,n (n<=12)
	(semicolon separator)	T,1
Column definitions	The following fields repeat for each column contained in the table:	
Name	Name of the column	T,n (n<=16)
	(equal sign separator)	T,1
Data type	One of the field types found in table C-52	T,1
	(comma separator)	T,1
Number [1]	Number of elements	T,n (n<=3)
	(comma separator)	T,1
Key type	Key type	T,1
	(comma separator)	T,1
Column description	Text description of the column's meaning	T,*
	(comma separator)	T,1
Value description table name	Name of an associated value description table	T,n (n<=12)
	(comma separator)	T,1
Thematic index	Name of thematic indexes.	T,n (n<=12)
	(comma separator)	T,1
Narrative table name	Name of an associated narrative table	T,n (n<=12)
	(comma separator)	T,1
End of column	(colon separator)	T,1
	... (repeat for each column)	
End of header	(semicolon separator)	T,1

Note:
1. This field contains the number of occurrences of the data type specified, not the number of bytes. For example, if there is only one integer value in the field, the header will contain the number "1" in that field. For text fields only, the value indicates the maximum of bytes allowed for that column. For example, if a maximum of 12 characters are allowed in the field, then the number of elements is specified as "12". The number of bytes specified for particular text fields are shown in subsequent tables in this specification.

The first component is the table description. The second component is a file reference to a narrative text, if available. If no narrative file exists, the dash symbol (-) is used for the file reference. The final component is the column definition substring. The column definitions are separated by colons (:), which indicate the end of the subcolumn definition. If an entry is not applicable to the field (i.e., a thematic index does not exist), the dash symbol (-) is used to indicate a null value. Trailing null field entries need not be included. For clarity in documentation, these trailing null fields should be listed, however. For each column in the table, there will be:

a. Column name, followed by an equal sign (=)
b. Data type indicator, followed by a comma (,)
c. Number of data type elements, followed by a comma
d. Key type indicator, followed by a comma
e. Column description, followed by a comma
f. Value description table name, if any, followed by a comma
g. Thematic index name, if any, followed by a comma
h. Column narrative file name, if any, followed by a comma

The character used as a separator for a particular field will not appear in that field except as a separator. For example, the Table Description header field will never include an imbedded semi-colon because a semi-colon is the specified separator. However, this field may legally contain imbedded colons or commas because neither of these characters are the separator for Table Description.

Table C-55 displays an example of a text header for an area feature table. For presentation purposes, each component in the table definition string is listed on a separate line in Table C-56. No new line, space, or tab character should be inserted after the field and component separators in the actual table definition string. Furthermore, the example does not show the 4-byte definition string length and the byte order character.

Table C-55 shows a header definition string for a Surface Drainage Area area feature table. A narrative file, sdr.doc, is attached to this table in the VRF database. The table has 12 columns, with the column ID being the primary key column.

Table C-55 Text Header Example

```
Surface Drainage Area;
sdr.doc;
id=I,1,P,Row ID,-,-,-,:
f_code=T,5,N,FACC Code,char.vdt,f_code.ati,f_code.doc,:
bgl=S,1,N,Bank Gradient Left (%),-,-,-,:
bgr=S,1,N,Bank Gradient Right (%),-,-,-,:
exs=S,1,N,Existence Category,int.vdt,-,-,:
hyc=S,1,N,Hydrological Category,int.vdt ,-,-,:
hfc=S,1,N,Hydrographic Form Category,int.vdt ,-,-,:
mcc=S,1,N,Material Composition Category,int.vdt,-,-,:
nam=T,*,N,Name,char.vdt,-,-,;
wid=S,1,N,Width (meters),-,-,-,:
wda=S,1,N,Water Depth Average (meters),int.vdt ,-,-,:
wva=S,1,N,Water Velocity Average (m/sec),int.vdt,-,-,:;
```

C.2.4.1.2 Record List

The body of data contained within the table is the record list; the header and the table index serve only to define the contents and provide effective access to this list. These records can be of fixed or variable-length, as needed.

C.2.4.1.3 Variable-length Index File

The variable-length index is a separate file that is mandatory when a VRF table contains variable-length records (as indicated by a field length "*" or a field type "K"). As shown in Table C-56, the file has two parts: a header and a data array. Each entry in the data array relates to a record in the VRF table.

Table C-56 Components of Variable-length Index File

File Component	Content of Component	Data Type	Field Length
Header	Number of entries (N) in index (which also matches the number of records in the associated table)	Integer	4 bytes
	Number of bytes in VRF table header	Integer	4 bytes
Data array	A two-dimensional array of N records	Integer	8N bytes

The data array identifies the location of every record in the variable-length file by containing the following entries for each record:

[n][0]	=	Byte offset from beginning of file
[n][1]	=	Number of bytes in table record

where n is an integer from 1 to N. The term byte offset refers to a location with respect to the beginning of a file. The first byte of a file has an offset of zero.

Thus, if the software requires the location of record 45 in a VRF table, the index file can be used to locate the exact position of the record without sequentially searching for the match. The entry for record 45 in the variable-length index would indicate the byte offset in the VRF table to the position of record 45 and the number of bytes in record 45.

C.2.4.2 Spatial Index Files

For each primitive (fac, edg, cnd, end, nod, and txt), there can exist a spatial index file that will accelerate queries by software. Although these files are optional, they are recommended, especially for large libraries. These indexes are indirectly created on the coordinate column or the minimum bounding rectangle of each primitive; Appendix C6 contains more information.

The format of the spatial index is as follows:

a. Header record. The header will contain one integer defining the number of primitives (NUMPRIM) and another integer defining the number of nodes (NNODE) in the index. Between the two integer fields are four (xmin, ymin, xmax, ymax) short floating point coordinates defining the minimum bounding rectangle. Table C-57 shows the layout for the spatial index file header record.

Table C-57 Spatial Index File Header Record Layout

Byte Offset	Width	Type	Description
0	4	Integer	Number of primitives
4	4	Floating point	MBR x1
8	4	Floating point	MBR y1
12	4	Floating point	MBR x2
16	4	Floating point	MBR y2
20	4	Integer	Number of nodes in tree

b. Bin array record. This record is a two-dimensional array the length of which is NNODE, described in the header record. The structure of this record is shown in Table C-58. This array maintains a long integer offset that points to the beginning of the bin data record and a long integer primitive count for each bin. The offset for the first bin always has a value of zero. For bins that contain no primitives, the value assigned to the count variable is zero and the offset value is zero.

Table C-58 Structure of the Bin Array Record

Byte Offset	Width	Type	Description
HDR + 8*n	4	Integer	Offset of primitive list for node n
HDR + 8*n + 4	4	Integer	Count in integer units

Note: n is {0 ... number of nodes-1}; the n value for the first node is 0. HDR is the length of the index file header record.

c. Bin data record. There are NUMPRIM records where each record contains four 1-byte integers defining the mbr for the primitive and one long integer (4 bytes) for the primitive id. These primitive ids point into the associated primitive table. Table C-59 shows the structure of the bin data record.

Table C-59 Structure of the Bin Data Record

Byte Offset	**Width**	**Type**	**Description**
HDR + BIN + OS + 8*c + 0	1	byte	MBR x1
HDR + BIN + OS + 8*c + 1	1	byte	MBR y1
HDR + BIN + OS + 8*c + 2	1	byte	MBR x2
HDR + BIN + OS + 8*c + 3	1	byte	MBR y2
HDR + BIN + OS + 8*c + 4	4	integer	Primitive id

Note: c is {0 ... number of primitives for a node - 1}; the c value for the first primitive is 0. HDR is the length of the index file header record. BIN is the summed length of all the bin array records. OS is the value of the offset variable in the corresponding bin array record (as shown in the byte offset calculation in Table C-58).

C.2.4.3 Thematic Index Files

A thematic index may be created for any column in a table. There are two types of indexes, depending on the data content in a column: an inverted list thematic index or a bit array thematic index.

For categorical data or data with few distinct values, such as soil features where numerous faces are assigned soil class designations from a relatively small number of classes, an inverted list is used. One entry in the index file is created for each distinct value in the column; correspondingly, a list of table record ids is stored with the value.

If the data in a column is all unique, especially in the case of an index for character strings, a bit array can be stored for each unique byte/character in the column. Each bit in the bit array represents a row in the indexed table. An ‘ON’ bit at a particular position means that the corresponding row in the table contains a specific byte/character pattern.

The character string form of the thematic index is used for names placement index implementations.

The thematic index file may be partitioned into three data groups: a fixed-length header, a variable number of index (or directory) entries (another index within an index), and a set of rows. Each row contains VRF record ids stored either as a list or as a bit array.

Each directory entry describes the element being indexed and the location of the row containing the list (or set) of record ids related to the element.

a. Header. The thematic index header contains 60 bytes of information that pertain to the type of index it is, the table it is associated with, and the column in that table. The layout of the header record is shown in Table C-60. An optional ordering of the entries in the index directory can be specified using the ordering flag at offset 56 in the index header. An "S" in the ordering flag indicates an ascending sort order in the index directory. Entries in the directory are assumed not to have any specific ordering otherwise.

b. Index directory. The index directory contains repeating records for each distinct element being indexed. The structure of an index directory record is shown in Table C-61. The number of entries is stored in the header record. Entries in the index directory give an offset at which the actual data are stored. There is also a count indicating the number of items maintained for a particular index value. If the count field in an index directory entry has a value of zero, the offset field contains the actual data; otherwise, the offset field contains an indirect address for the indexed data.

Table C-60 Thematic Index file Header Record Layout

Byte Offset	Width	Type	Description
0	4	Integer	Combined length of the index file header and the index directory.
4	4	Integer	Number of directory entries. This is the number of items being indexed by a particular index file.
8	4	Integer	Number of rows in the data table from which this index file was derived.
12	1	Character	Type of index file; either "I" for inverted list index, or "B" for bit array index.
13	1	Character	Type of the data element being indexed; one of: I" - 4-byte integer "T" - character string "S" - 2-byte integer "F" - 4-byte floating point "R" - 8-byte floating point
14	4	Integer	Number of data elements comprising one directory entry. This field will usually have a value of 1; an exception is a thematic index built on a text field.
18	1	Character	Type specifier for the data portion of an index file. Record ids in inverted list index can be stored by using either a 2-byte integer (type "S") or a 4-byte one (type "I"). Bit array index files always use type "S."
19	12	Character	The name of the VRF table from which the index file has been derived; no path information is included.
31	25	Character	The name of the column in the VRF table from which index entries have been pulled.
56	1	Character	Ordering flag ("S" indicates an ascending order in the index directory; no specific sort order otherwise).
57	3	Character	Unused and reserved.

Table C-61 Structure of Index Directory Record

Byte Offset	**Width**	**Type**	**Description**
HDR +n*(d+8)	d	Character string 2-byte integer 4-byte integer 4-byte floating point 8-byte floating point	The element being indexed.
HDR +n*(d+8)+d	4	Integer	The offset from the beginning of the file to the location of the row associated with this index entry.
HDR +n*(d+8)+d+4	4	Integer	The number of indexed records associated with this entry. (For bit array index files, this is the number of bytes.)

Note: n is {0 ... number of index entries-1}, and d = size of (indexed type). HDR is the length of the index file header record.

c. Index data. For each index entry there exists a data record consisting of either a list of row ids from the indexed file or a bit array.

The following example shows an inverted list thematic index created on a feature table column of data type 'S' (short integer). The name of the column is use_code. The name of the table is cularea.aft. The index table header shown in Table C-62 is thus,

Table C-62 Thematic Index Header Example

Byte Offset	**Width**	**Type**	**Value**
0	4	Integer	90 (header length + index directory length)
4	4	Integer	3 (number of directory entries)
8	4	Integer	293 (number of indexed rows)
12	1	Character	I (inverted list)
13	1	Character	S (short integer source data)
14	4	Integer	1 (data length is 1)
18	1	Character	S (short integer indexed id)
19	12	Character	cularea.aft.
31	25	Character	use_code
56	1	Character	S (sorted index directory)
57	3	Character	" "

The value at byte offset 60 is the value of the element being indexed. The number of rows from the table cularea.aft is contained at address HDR+n*(d+8)+d+4 (in this case at byte offset 66). The first entry in the directory has a use_code value of 2, and there are 5 rows that contain the value. The cularea.aft rows can be found at address 90. Another index directory entry starts at offset 70. This entry has a count of zero, indicating that the offset field contains the row number for the cularea.aft table. The index directory shown in Table C-63 is thus,

Table C-63 Thematic Index Directory Example

Byte Offset	Width	Type	Value	
60	2	Short Integer	2	(index value)
62	4	Integer	90	(offset)
66	4	Integer	5	(count)
70	2	Short Integer	3	
72	4	Integer	20	(direct row id)
76	4	Integer	0	(count of "zero")
80	2	Short Integer	4	
82	4	Integer	100	
86	4	Integer	4	

The index data shown in Table C-64 are thus,

Table C-64 Thematic Index Data Example

Byte Offset	Count	Row numbers
90	5	8 9 10 11 12
100	4	22 23 24 25

C.2.4.3.1 Feature Index

Feature indices may be created for any VRF coverage. These are join indices that have been developed to enhance processing of complex queries in relational databases. This is particularly significant in tiled VRF coverages with a number of feature classes. The DIGEST VRF Standard specifies a set of join indices for feature/primitive joins. One feature join index can be defined for each of the five primitive types in a coverage. For example, an edge feature join index, edg.fit, can be defined for edge primitives and the corresponding line and complex features that reference those primitives.

Feature join indices are optional. Feature indices are composed of: (a) a feature class attribute table (fca) and (b) a number of feature index tables (fit). Feature index tables allow quick retrieval of feature information when given a selected primitive.

They bypass the normal relational operations usually required and restore the results of feature-to-primitive and primitive-to-feature relations. As indices, they may be deleted at any time and the relationships between tables will be maintained provided the associated join tables have been defined. Any updates to the data will cause the indices to be rebuilt. The feature index tables may not be referenced in the fcs. There could be one fit for each primitive type in a coverage. For a given coverage, if a feature index is defined for a primitive type, all feature classes associated with that primitive type must be indexed. All fca and fit's reside at the coverage level.

A VRF coverage can optionally contain a Feature Class Attribute table (fca). This table should minimally have the following columns: a feature class id column (id), a feature class name column (fclass), the feature type (type) and a feature class description column (descr). A feature class attribute table definition example is shown in Table C-65.

Table C-65 Feature Class Attribute Table Definition

Column Name	**Description**	**Field Type**	**Key Type**	**Op/Man**
id	Row id	I	P	M
fclass	Feature class name	T,8	U	M
type	Feature type (P-point, L-line, A-area, T-text, C-complex)	T,1	N	M
descr	Description	T/L/M/N,*	N	M

Every primitive/feature reference, both directly and indirectly, as in the case of complex features, results in one entry in the appropriate fit for that primitive and the corresponding feature. If a feature is composed of multiple primitives, each of those feature/primitive relationships is recorded. Conversely, if a primitive is applicable for more than one feature, multiple relationships are similarly maintained. When a primitive is referenced by a complex feature via an intermediate feature, the relationship between the primitive and the complex feature, as well as that between the primitive and the intermediate feature, are recorded in the fit's.

Feature Index Tables (fit) are made up of two compound keys, the feature class id (fc_id) and the feature id (feature_id) to properly identify an individual geographic feature, and the tile id (tile_id) and primitive id (prim_id) for a primitive. Available fit names are: edg.fit, nod.fit, end.fit, cnd.fit, fac.fit and txt.fit. An example of a feature index table definition is shown in Table C-66.

Table C-66 Feature Index Table Definition

Column Name	Description	Field Type	Key Type	Op/Man
id	Row id	I	P	M
prim_id	Primitive id (foreign key to primitive table)	I	N	M
tile_id	Tile reference id	S	N	MT
fc_id	Feature class id (foreign key to fca)	I	N	M
feature_id	Feature id (foreign key to feature table)	I	N	M

C.2.4.4 Allowable Field Types

The field types depicted in Table C-67 are allowed and provide the ability to encode any data set. All variable-length types include a count item "n" (as depicted in Table C-67) preceding the data. The count is a 4-byte integer. This count item contains the number of bytes in text strings and the number of tuples in coordinate strings.

Table C-67 Allowable Field Types

Abbrv	Column Type	Null/No Value	Length (bytes)
T,n	Fixed-length text	"N/A"	n
T,*	Variable-length text	Zero length	n + 4
L,n	Level 1 (Latin 1 - ISO 8859) Fixed-length text	“N/A”	n
L,*	Level 1 (Latin 1 - ISO 8859) Variable-length text	zero length	n+4
N,n	Level 2 (obsolete retained for compatibility)	“N/A”	n
N,*	Level 2 ((obsolete retained for compatibility)	zero length	n+4
M,n	Level 3 (Multilingual - ISO 10646) Fixed-length text	“N/A”	n
M,*	Level 3 (Multilingual - ISO 10646) Variable-length text	zero length	n+4
F	Short floating point	NaN (not a number)	4
R	Long floating point	NaN (not a number)	8
S	Short integer	bit pattern 10000000 00000000	2
I	Long integer	bit pattern 10000000 00000000 00000000 00000000	4

C,n	2-coordinate array, short floating point	Both coordinates equal to null	8n
C,*	2-coordinate string	Length = 0	8n + 4
B,n	2-coordinate array, long floating point	Both coordinates equal to null	16n
B,*	2-coordinate string	Length = 0	16n + 4
Z,n	3-coordinate array, short floating point	All three coordinates equal to null	12n
Z,*	3-coordinate string	Length = 0	12n + 4
Y,n	3-coordinate array, long floating point	All three coordinates equal to null	24n
Y,*	3-coordinate string	Length = 0	24n + 4
D	Date and time	Space character filled	20
X	Null field	-	-
K	Triplet id	Type byte = 0	1–13
G,n	2-coordinate array - short integer	Both coordinates equal to null	4n
G,*	2-coordinate string - short integer	Length = 0	4n + 4
H,n	2-coordinate array - long integer	Both coordinates equal to null	8n
H,*	2-coordinate string - long integer	Length = 0	8n + 4
V,n	3-coordinate array - short integer	All three coordinates equal to null	6n
V,*	3-coordinate string - short integer	Length = 0	6n + 4
W,n	3-coordinate array - long integer	All three coordinates equal to null	12n
W,*	3-coordinate string - long integer	Length = 0	12n + 4

Note: For data types Y,Z,V and W, if the elevation (Z) field is not populated due to source restrictions, it will be filled with the appropriate NULL value.

C.2.4.5 Naming Conventions

The following define the naming conventions for VRF file and column names. (See also the VRF reserved names in tables C-14, C-15, and C-16.)

a. All naming will use ISO 646 (ASCII) characters.

b. All file names and all references to file names shall be lowercase. This includes file references within table headers, attribute values, and indices. All references to database, library, coverage, and feature class names shall be in lowercase where they occur. For example, feature class names in fcs and fca tables will be lowercase. For file names, the first character must be an alpha character from a to z. The remaining 7 characters can be alphanumeric, including the underscore ('_') character.

A file name may have a trailing period with a 3-character suffix. The suffix may contain only characters from a to z, except that x is reserved for variable-length indexes.

Any table with variable-length records will maintain a variable-length index file with the same file name as its associated table except that the last character will be x (with one exception; see paragraph C.2.3.1.2).

c. All names are to be in lowercase.

d. Directory names (names for libraries, databases, and coverages) are restricted to the same rules as for file names, except that there will be no suffix.

e. Column names follow the same restrictions as file names, but they can be 16 characters in length. The dollar sign ('$'), number sign (also known as gate, hash, pillow, pound and octothorpe) ('#'), dash ('-'), period ('.'), and slash ('/') are allowable characters.

The column name ("id") is reserved and must be used to identify each table record.

f. If a column is defined with a triplet id, the fields within the triplet id will be named as:

Field one	id	The internal tile primitive id
Field two	tile_id	The tile reference id
Field three	ext_id	The external tile primitive id

The (\) will be used as a separator between the column name and the triplet id field. Thus, when referring to the internal primitive id within a triplet id column (left_face) the column name will be named "left_face\id".

C.2.4.6 Triplet id Field Type

As discussed in cross-tile keys (clause C.2.2.2.3.4), a triplet id can be used to reference primitives from multiple tiles in a tiled coverage. This field type replaces the integer foreign key used in untiled coverages. The first component of a triplet is an 8-bit type byte. The type byte is broken down into four 2-bit pieces; each of these 2-bit pieces describes the length of a succeeding field. Table C-68 lists the possible values for these 2-bit field descriptors. Only the first three fields are currently being used. The fourth field is reserved. Figure C-22 is an example of the triplet id field.

Table C-68 Type Byte Definitions

Bit Count	Number Bits in Field
0	0
1	8
2	16
3	32

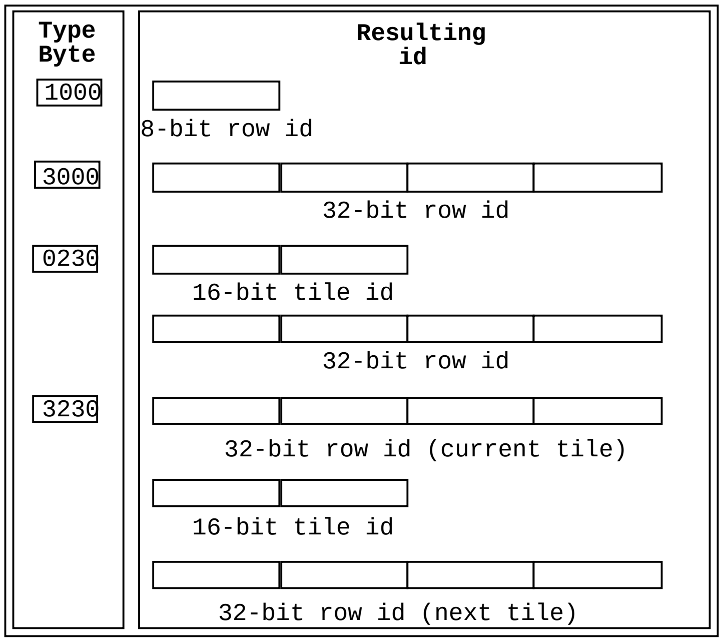


Figure C-22 Examples of the Triplet id

The first field (referred to as id) generally is used to store a primitive id without a tile id predicate. The tile reference id, the second field (tile_id), and the external primitive id, the third field (ext_id), together store an augmented primitive id for cross-tile topology.

C.2.5 Data Syntax Requirements

VRF requires the use of numeric, textual, coordinate, and date syntax items. These items comprise the lowest level of the VRF design. In order to utilize these items, a number of basic data types are required. These are integer or real numbers, strings of text, and coordinate data types. The coding of these data types is defined in terms of a number of international standards. VRF products may have a byte order specified in the product specification and the table header. Five categories of data syntax items are required in VRF. These are integer numbers, real numbers, text strings, coordinates, and date.

C.2.5.1 Integer Numbers

The two fixed-length integer data fields are 16 bits and 32 bits; hereafter they are termed "short" and "long" precision, respectively. Integers are binary encoded using the 2's complement scheme. For integer number formats, the null value consists of the sign bit set to one and all trailing bits set to zero. See Part 3 Clause 5.1.

Different length integer numbers may be required in different situations. For example, the number 32,000 can be handled in 2 bytes of data, whereas the number 2,147,483,647 will fill 4 bytes of data. The general structure and several examples of the integer number format can be found in Figure C-23.

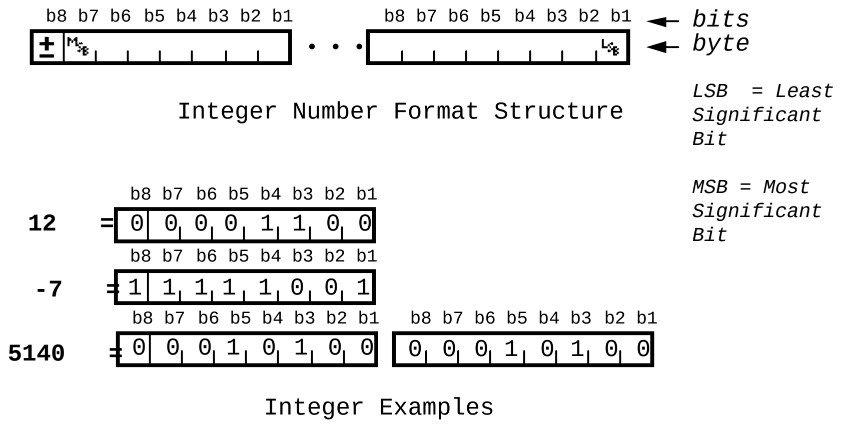


Figure C-23 VRF Integer Number Syntax

C.2.5.2 Real Numbers

Real numbers are needed to carry parametric information. VRF uses the IEEE floating-point real number format (ANSI/IEEE 754) in both 32-bit (short) and 64-bit (long) form. See Part 3 Clause 5.1.

Numbers in the single and double formats are composed of the following three fields:

s	1-bit field for sign
e	Biased exponent field (equals exponent *E* plus bias)
f	Fraction field (mantissa)

a. Range. The range of the unbiased exponent includes every integer value between $E_{\min}$ and $E_{\max}$, inclusive, and also two other reserved values, $E_{\min}$ - 1 and $E_{\max} + 1$, to encode certain special states as described below. Figure C-24 illustrates real number syntax.

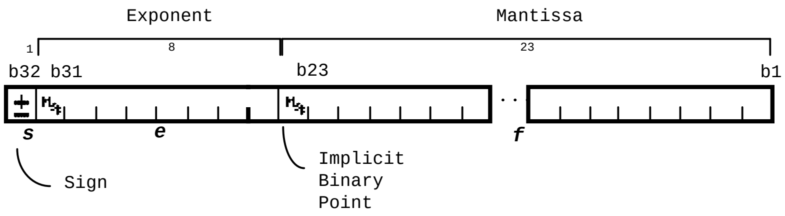
b. 32-bit format. For a 32-bit single format number, the value *v* is inferred from its constituents thus:

(1) If $e = 255$ and f is non-zero, then v is NaN,
(2) If $e = 255$ and $f = 0$, then $v = (-1)^s$ infinity,
(3) If $0 < e < 255$, then $v = (-1)^s \, 2^{e-127} \, (1.f)$,
(4) If $e = 0$ and f is non-zero, then $v = (-1)^s \, 2^{e-127} \, (0.f)$ (denormalized numbers),
(5) If $e = 0$ and $f = 0$, then $v = (-1)^s \, 0$ (zero).

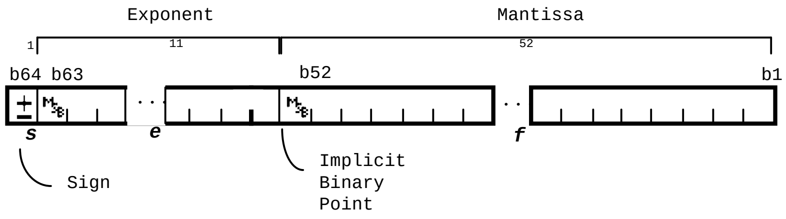
c. 64-bit format. For a 64-bit double format number the value *v* is inferred from its constituents, thus:

(1) If $e = 2047$ and f is non-zero, then v is NaN,
(2) If $e = 2047$ and $f = 0$, then $v = (-1)^s$ infinity,
(3) If $0 < e < 2047$, then $v = (-1)^s \, 2^{e-1023} \, (1.f)$,
(4) If $e = 0$ and f is non-zero, then $v = (-1)^s \, 2^{e-1023} \, (0.f)$ (denormalized numbers),
(5) If $e = 0$ and $f = 0$, then $v = (-1)^s \, 0$ (zero).

Note: The "." in equations (3) and (4) above corresponds to a decimal point.



IEEE Short Real Number Format Structure (32 bit)



IEEE Long Real Number Format Structure (64 bit)

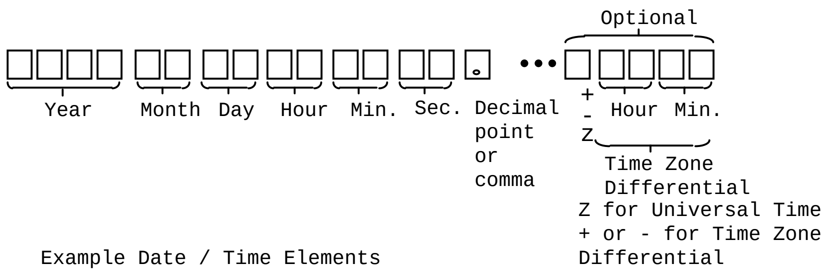
Where :
- The sign bit is 0 for positive and 1 for negative.
- The exponent is 8 bits long for short real numbers and 11 bits for long real numbers. The exponent is biased by 81 hexadecimal for short real numbers and 401 for long.
- The remaining bits are the mantissa. Since the first significant bit is known to be set (since the mantissa is normalized), it is not stored. The length is 23 bits for short real numbers and 52 bits for long real numbers.

Figure C-24 Real Number Syntax

C.2.5.3 Date and Time Syntax

The generalized time data type consists of a string of international reference version (IRV) characters where the calendar date is specified in ISO 8601. See Part 3 Clause 5.1.4 to 5.2.3.

The coding of generalized time data in accordance with ISO 8601 in VRF consists of a four-digit representation of the year, a two-digit representation of the month, and a two-digit representation of the day. This is followed by the time of day (as specified in ISO 8601) consisting of a string of digits containing a two-digit representation of the hour (based on the 24-hour clock), a two-digit representation of the minute, and a two-digit representation of the second, followed by a decimal point (or decimal comma) and an arbitrary number of digits of fractions of a second. This may be followed by the letter Z to represent coordinated Universal Time rather than local time, or be followed by a time differential from UT in accordance with ISO 8601 (see Figure C-25). In a fixed data field usage, date and time elements will be 20 bytes long, allowing for the specification of date and time with time zone differentials (or optionally fractional seconds). Unfilled digits will be filled with space characters. A null date time specification will consist of a string of space characters.



" "	• No date/time given, 20 space characters
1992	• Year is only value given, no time. Pad with space characters.
199210	• Year/month given, no time. Pad with space characters
19870205160627.	• Local time 6 minutes, 27 seconds after 4 pm on 5 February 1987
19870205210627.Z	• As above but Universal Time (Greenwich)
19870205160627.-0500	• Local Time as above. The local time is 5 hour time behind UTC

Figure C-25 Date and Time Syntax

C.2.5.4 Text Syntax

Textual information can be either variable-length or fixed length. The null state of a variable-length text string is of zero length. The null state of a fixed-length text string requires that a specific code be selected. The character string "N/A" should be used, padded if necessary. If the length is one or two, "-" or "--" should be used instead. The character space (code table position 2/0) should be used as the "space" or "blank" character, and as the padding character. The character code NUL C0 control set (code table position 0/0) and a number of other CO control characters may have special meaning on some computer systems and should not appear in any text strings. A NUL or a SUB (^Z) in a file is an end of file mark on some computers.

Two types of text strings are supported in VRF:

1. Basic text string. These strings make use of characters only from the IRV (ASCII) primary code table and the subset of the CO table identified above.

2. General text strings. These strings are composed of characters from the ISO 8859-1 Latin Alphabet 1 or the ISO 10646 (Unicode) repertoire.

The text representation and coding is described in DIGEST Part 3 clauses 5.1.4 to 5.2.3.

C.2.5.5 Coordinate Syntax

A coordinate specifies a position in the Cartesian unit coordinate space as a vectorial displacement from the origin of the coordinate space. A coordinate parameter value takes the form of a short or long floating point value or as a short or long integer displacement from a local origin.

C.2.5.6 Coordinate Strings

Two types of coordinate strings are defined for use in VRF. These consist of coordinate tuples (pairs or triplets). All coordinate strings are constructed out of the number and coordinate formats defined in the previous subclauses. A coordinate string consists of a sequence of coordinate parameter values corresponding to coordinate tuples.

C.3 Notes

(This clause contains information of a general or explanatory nature that may be helpful, but is not mandatory.)

C.3.1 Intended Use

This standard is designed to define the methods and provide guidance for creating and using digital geographic databases in Vector Relational Format.

Appendix C1 - Introduction to the VRF Data Model

C1.1 GENERAL

This appendix provides information, discussion, and examples concerning the VRF data model.

C1.2 APPLICABLE DOCUMENTS

This clause is not applicable to this appendix.

C1.3 DEFINITIONS

For purposes of this appendix, the definitions in Part 2 clause 4.1 of the main document shall apply.

C1.4 GENERAL INFORMATION

C1.4.1 Introduction

VRF is a general, user-oriented data format for representing large spatially referenced (geographic) databases. VRF is designed to be used directly; that is, software can access the data without time-consuming conversion processing. VRF is designed to be compatible with a wide variety of users, applications, and products.

To achieve its generality and user orientation, VRF uses a georelational data model that provides a flexible but powerful organizational structure for any digital geographic database in vector format. VRF defines the format of data objects, and the georelational data model supporting VRF provides a data organization within which software can manipulate the VRF data objects.

The following paragraphs discuss in general the data model that serves as the basis of VRF. Clause C1.4.2 discusses the basic concepts that form the foundation for all geographic data models. Clause C1.4.3 describes the relational model, while clause C1.4.4 describes the planar topology model.

C1.4.2 Data Model Concepts

A model is a fundamental description of a system that accounts for all known properties of that system. The system is a view of geographic reality, or information tied to specific locations in coordinate space. Since this particular model is stored in a computer, it is called a data model. A data model provides the most abstract representation of a system; it describes a collection of entities, including their relationships and semantics. The purpose of a data model is to define and capture a view of reality in a consistent and uniform manner. It provides a framework to visualize the structure and behavior of these entities in a system.

A model of a database requires three components: the definition of the data objects, data operations, and the rules of data integrity. These components are defined in the following three subparagraphs. Objects identify how the user perceives the data and its structure. The operations define how the user may manipulate the objects. Integrity rules bind the objects and operations, and establish a well-defined behavior that provides accurate information in a predictable manner. The fourth sub-paragraph deals with the purpose and functionality behind a database.

C1.4.2.1 Data Objects

Data objects identify how the user perceives the data and its structure. These objects also define the most primitive partitions of the data model architecture. For instance, an international database may contain a wide variety of objects concerning nations. The relevant objects could be areas, civil divisions, populations, resources, products, and water bodies. Relationships can be defined for these objects where populations and resources are grouped by civil divisions.

C1.4.2.2 Data Operations

Data operations define how a user may manipulate the objects; where, for instance, an object's attributes may be displayed, or new attributes could be defined, or, perhaps, new objects created. In most cases, an algebra is defined that accurately manipulates the data objects and binds the scope of operations within the data model. Classical database operations include retrieval, creation, deletion, and modification. More specific operations are defined for applications using the database.

C1.4.2.3 Data Rules

The rules of data integrity constrain the operations on objects in order to preserve overall stability. The goal is to prevent operations that yield corrupt, incorrect, or ambiguous results. Integrity rules constrain the set of valid states of databases that conform to the data model. These rules define the accuracy of the database.

C1.4.2.4 Database Purpose

One function of a database is to provide centralized control of operational data that is vital to an organization. A data model attempts to closely mesh the data objects, operations, and integrity rules into a cohesive system with optimal performance. The advantages of centralized database control are well established, and the use of a data model allows a database to provide the following functionality to an organization:

a. Consistency. The database will provide access to data in a formal manner. This will establish a consistent view of the data, enabling efficient data exchange.

b. Simplicity. A basic objective is to provide an intuitive, straightforward, and understandable interaction between the user and the data.

c. Non-redundancy. Duplication of data will be avoided wherever possible, especially when repetition provides little additional information.

d. Multiple applications. The data model needs to support multiple end-user applications because user views of the database will be different.

e. Flexibility. A critical requirement of an effective database is the ability to accept new data. A database needs to have the dynamic flexibility to grow with the needs and requirements of its users.

f. Integrity. The integrity rules should be defined in a consistent manner for all data objects and operations. The use of well-defined rules prevents operations that lead to data corruption and misinformation.

In summary, a data model provides a powerful approach to achieving optimum centralized control of critical data within a system. Using a variety of integrity rules and established operations upon defined data objects, the database provides users with the necessary tools for extracting data in support of many applications.

C1.4.3 Relational Data Model Concepts

The relational data model provides a powerful architecture for database design because of its ability to handle a wide variety of data and applications.

C1.4.3.1 Relational Data Objects

The relational model uses simple tabular data structures to portray the data in a natural, well-defined manner. These data structures contain columns and rows, where columns define attributes, the values for which are taken from a range of data defined across a given domain. The content of the rows represents the actual data entities. The strength of the relational model is its ad hoc ability to establish meaningful data, transforming data into information as a function of user perspective. For instance, a relational table "roads" can be defined as having three columns: name, class, and structure. The rows that compose the roads table would contain distinct information about each road in each field in the database. The six following properties distinguish relational tables from nonrelational data objects:

a. Entries in columns must be single-valued; a field may not contain a list of attributes.

b. Entries within a column must be of the same data type.

c. Each row must be unique, and duplicate rows are not permitted.

d. Columns may appear in any order.

e. Rows may also appear in any order.

f. Each column must have a unique name.

C1.4.3.2 Relational Data Operations

The relational model supports eight set operations: select, project, product, join, union, intersection, difference, and division. Since the relational model is founded on set theory, the operations themselves are based on fundamental mathematical principles. These operations allow data objects to be manipulated and created in a specific manner, producing stable results.

C1.4.3.3 Relational Data Rules

The integrity rules constrain operations performed on objects in order to preserve stability. The goal is to prevent operations that yield corrupt, incorrect, or ambiguous results. The entity integrity rule requires the entry of a null value in columns in relational tables for which the value is always known or understood. The referential integrity rule ensures that a foreign key referenced into another table stays within recognizable bounds. For example, a foreign key is not permitted that references a record number of 500 in a table that only contains 300 records. Additional, more subtle domain rules can also be defined to constrain entries in and operations on the database.

C1.4.4 Plane Topology Model Concepts

Conceptually, plane topology can be defined as a planar graph, where geographic reality is decomposed into a finite set of 0 cells (nodes), 1 cells (edges), and 2 cells (faces). This terminology is defined by an algebraic topology that establishes rules for decomposing continuous three-dimensional objects into representations of finite models. Once this topological mapping has been performed, a system can be modeled in a way that permits more complex relationships between objects to be established.

The purpose of topology is to capture and retain knowledge concerning a cell's spatial and thematic relationships with its neighboring cells. For a topological model to be valid, these relationships must remain constant regardless of changes in scale, shape, or size. With topology embedded in a data model, very useful relations can be established, such as adjacency and connectivity. Topological and geometric relationships (such as size, angle, and shape) provide powerful resources that allow geographic reality to be fully modeled.

C1.4.4.1 Topological Objects

Plane topology extends graphic models of nodes and edges to the development of a more powerful and expressive model that contains spatial relationships. In addition to metric capabilities (distance, shape, or size), topology determines spatial neighbor relations.

By defining more rigorous and complex relationships in the data model, the true properties of a system can be more effectively represented. Various mathematical models are available. In addition, simpler models can be used to create more complex ones. The most simple is the graphic model. More complex and useful surface-based models are built upon the graphic, using topology to capture additional analytical information.

A graphic model represents geometric information based on node and edge primitives. This model provides the base for other models that define more complex relationships. The node primitive is composed of a location in an established coordinate space. Most representations are two dimensional such as (x,y) or three dimensional such as (x,y,z). An edge is composed of a minimum of two nodes, with more details concerning linear or spline interpolation between the end points.

Because of the complexity of geographic information and the limitations of a graphic model, plane topology provides a better model for defining relationships. The use of a planar topology model (based on the two-dimensional manifold), for instance, maps more concisely in the two-dimensional space that current computing systems can manage. VRF provides four distinct levels of topology: full planar topology (level 3); a linear planar graph (level 2); a nonplanar linear graph (level 1); and no topology defined, indicating a geometric model (level 0). VRF uses the notion of topology as a constraint to enforce integrity rules upon the feature definitions. As the entities require fewer topological relationships, the rules can be relaxed. For instance, if linear features in a transportation network are being modeled, then the requirement of full planar topology may be relaxed because it is not necessary.

Plane topology defines relations between cells without modifying the underlying geometry. The concept of a cell can be visually retained in graphic models, but only allows one view of the data. Topology establishes a framework that provides more information for analysis. For instance, in figure C1-1, the edges 1, 2, 3, 4, and 5 are grouped together into face 'a.' Another face known as 'b,' including the edges 5 through 12, can be defined. Topology defines relationships on each 0-, 1-, and 2-cell in a model. Face 'b' is defined to be "left-of" edge 5; edge 7 can be defined to follow edge 6 in a cycle. This topological information provides the power to determine orientation, adjacency, and connective relationships between objects.

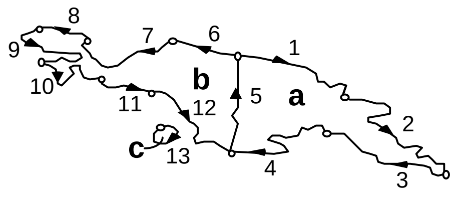


Figure C1-1 The Definition of Faces

The concept of topology held by the geometric primitives is carried upward to features and their associated thematic information. An area feature is usually labeled, or contains information pertaining to the enclosed region. For instance, an area can have a category (soil class, surface material type) or a numeric value (population size, number of airports). Topology is used to provide operations and information to distinguish between these thematic objects. Thematic relationships can exist between features without requiring the primitive geometry. For instance, a set of the islands in the Pacific Ocean (Oahu, Maui, Hawaii, Molokai, etc.) can be defined as different features with differing geometric primitives, but they can be related to one another as the Hawaiian Islands by means of a complex feature.

C1.4.4.2 Topological Operations

Topological operations are based on the single notion of adjacency—that is, if two objects are next to each other, it is necessary to maintain the adjacent relationship between them. To distinguish the topological aspects from the geometric aspects of geography, we are only concerned with whether two objects, A and B, are adjacent to each other and not with whether A is bigger than B, or one is to the north of the other, or the length of their common boundary.

Many complex topological operations can be derived from adjacency alone. In the georelational data model implemented for VRF, two topological operations are paramount: boundary and coboundary. For example, an edge has a start node and an end node; the nodes are the boundary for the edge. The edge, in turn, is the coboundary of the node. Of course, the coboundary of a node can have more than one edge if many edges meet at a node. Similarly, faces have edges as boundaries. The coboundaries of edges are maintained in the left and right faces.

C1.4.4.3 Topological Rules

The integrity rules of the topological model are contained in the definition of the objects themselves. A plane model restricts itself to planar geometry, where all entities must lie in the same plane. In addition, all faces must be mutually exclusive and non-overlapping. These constraints allow the objects to be defined in context and allow operations to be performed in a consistent manner. While these rules may seem to restrict the system model, they define the data model's domain, taking advantage of the underlying structures. By restricting the faces to be constructed of non-overlapping regions, powerful set operations can be applied to the objects (such as union, intersection, or join).

C1.4.4.4 The VRF Georelational Data Model

VRF uses a combination of the relational and the planar topological data models to provide a hybrid model for geographic data management, analysis, modeling, and display. The georelational model provides the data structure foundations for a spatial database, and software provides the rules and operators that manipulate topology, geometry, and relational objects in the form of tables. Whenever an operation requires thematic information, the use of relational and topological table operations are used to supply the result. If the operation is spatially related, geometry and topology together will be used. This triad of categories (geometry, topology, and relational tables) provides a robust database architecture.

VRF adheres to the georelational data model, but only defines the objects and the data structures that compose the objects. The georelational operations and algebra are not part of the standard, but rather are implemented in software. Every VRF object is described in the form of a relational table, composed of columns defining the syntax of each field and rows that contain the actual data.

[This page intentionally left blank]

Appendix C2 - Winged-Edge Topology

C2.1 GENERAL

This appendix provides information, discussion, and examples concerning winged-edge topology.

C2.2 APPLICABLE DOCUMENTS

This clause is not applicable to this appendix.

C2.3 DEFINITIONS

For purposes of this appendix, the definitions of Part 2 clause 4.1 of the main document shall apply.

C2.4 GENERAL INFORMATION

C2.4.1 Winged-Edge Topology

Winged-edge topology is an essential part of the VRF data model. The function of winged-edge topology is to provide line network and face topology and also to maintain seamless coverages across a physical partition of tiles. The following clauses define the components of winged-edge topology, the algorithm used to traverse the winged-edge network, and cross-tile topology.

C2.4.2 Components of a Winged Edge

Winged-edge topology uses three specific components (columns) on an edge primitive table to provide connectivity between nodes, edges, and faces. Given level 1, 2, or 3 topology, the edge primitive will contain specific columns for each topological level. As shown by Figure C2-1, there are three topological constructs: node, edge, and face information on each edge. These constructs are formally defined in clause C.2.3. A brief summary of the definition is repeated below.

a. Node information. Each edge will contain a start node and an end node column. This topological information is used to define an edge direction (digitizing direction).

b. Edge information. Right and left edges connect an edge to its neighbor edges (thus the term "winged edge"). The right edge is the first edge connected to the end node that is encountered when cycling around the node in a counterclockwise direction. The left edge is the first edge connected to the start node that is encountered when cycling around the node in a counterclockwise direction.

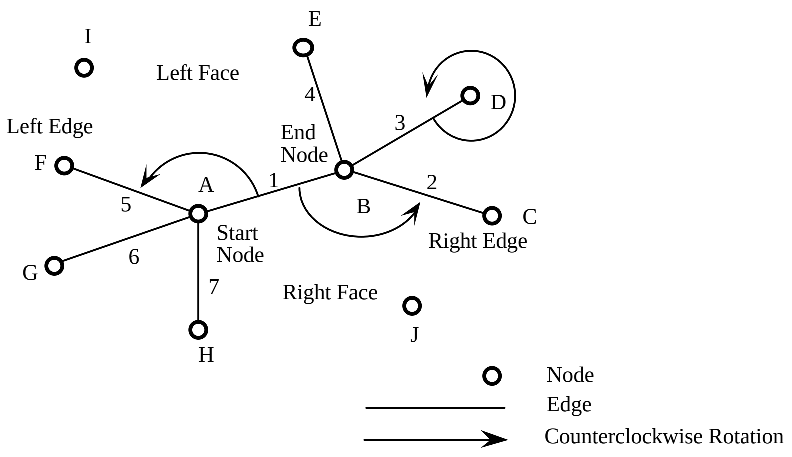
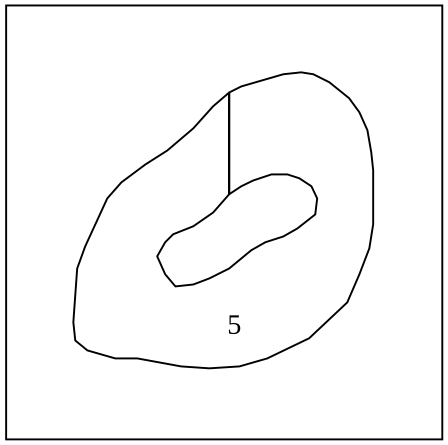


Figure C2-1 Winged-Edge Components

c. Face information. With level 3 topology specified, each edge will contain a left and right face. Left and right face are determined solely by the edge direction. This information allows an edge to know its neighboring faces. The universe face is always regarded as face 1.

C2.4.2.1 Inner and Outer Rings

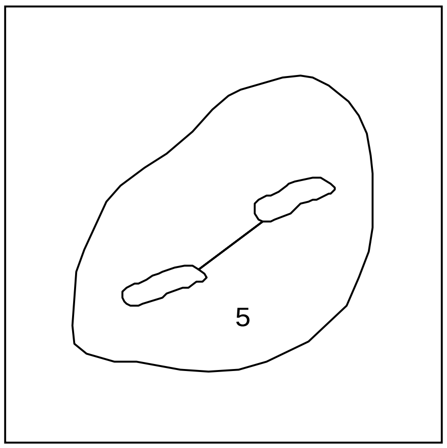
The composition of a face's outer and inner rings are governed by the rules of winged-edge topology. In addition, since edges are never considered inside a face but, rather, borders of faces, floating edges within a face will be treated as inner rings. Figures C2-2, C2-3 and C2-4 illustrate some cases of outer and inner rings.



Note:

There is no inner ring

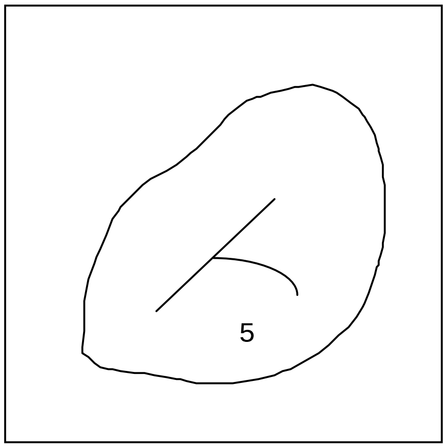
Figure C2-2 Face 5 is Represented as a Single Ring in the Ring Table



Note:

The two inner faces and their connected edge compose a single inner ring and are identified by a single ring in the ring table.

Figure C2-3 Face 5 is Represented as Two Rings in the Ring Table



Note:

The second ring in the ring table identifies the floating edge within the face

Figure C2-4 Face 5 is Represented as Two Rings in the Ring Table

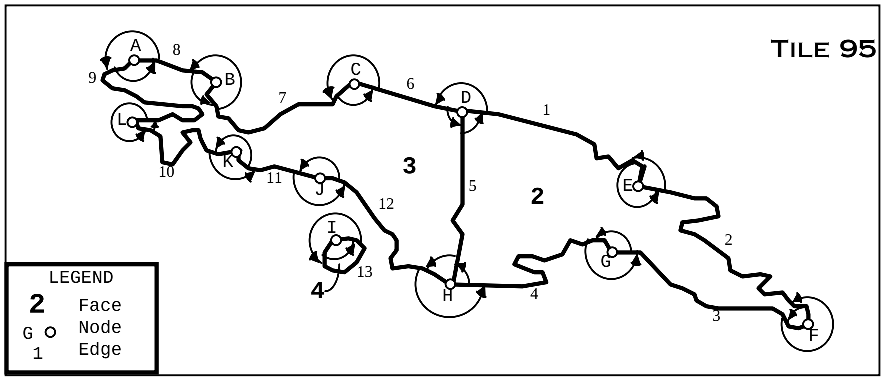
C2.4.3 Winged-Edge Algorithm

Given the definition of a winged edge, every coverage containing faces and edges is created in the same way. With the enforcement of a planar topological model, a consistent navigation algorithm can be applied.

Figure C2-5 depicts a collection of faces and accompanying edges. To navigate a face, the following algorithm is used:

a. Determine which face to draw. Determining which face to draw would typically be driven by the selection of area features that have the attributes desired; then key through the face and ring tables associated with the area.

An example of an area feature table consistent with Figure C2-5 is shown in Table C2-1.



id	start_node	end_node	right_face	left_face	right_edge	left_edge	coordinates
1	D	E	2	1	2	6	
2	E	F	2	1	3	1	
3	F	G	2	1	4	2	Not
4	H	G	1	2	3	5	Shown
5	H	D	2	3	1	12	
6	D	C	1	3	7	5	
7	C	B	1	3	8	6	
8	B	A	1	3	9	7	
9	A	L	1	3	10	8	
10	K	L	3	1	9	11	
11	K	J	1	3	12	10	
12	J	H	1	3	4	11	
13	I	I	4	1	13	13	

Figure C2-5 Winged-Edge Example, with Drawing Completely Contained within a Tile

Table C2-1 Sample Area Feature Table for Figure C2-5

id	tile_id	fac_id	<attribute 1>...< >
.	.	.	.
.	.	.	.
.	.	.	.
436	95	3	1 ...
.	.	.	.
.	.	.	.
.	.	.	.

Table C2-1 identifies tile 95 with face 3 as the primitive corresponding to area feature 436. The face table for tile 95 for Figure C2-5 (Table C2-2) yields the following:

Table C2-2 Sample Face Table for Figure C2-5

id	*.aft_id	ring_ptr
.	.	.
.	.	.
.	.	.
3	436	7

The ring table for Figure C2-5 (Table C2-3) yields the following:

Table C2-3 Sample Ring Table for Figure C2-5

id	fac_id	start_edge
.	.	.
.	.	.
.	.	.
7	3	12

b. Now identify the start edge (in this case, 12).

c. Travel to the left edge to trace the left face; the right edge for the right face. Because face 3 is the left face of edge 12, read the left edge record (edge 11). Edge 11 leads to edge 10, edge 10 to edge 9, edge 9 to edge 8, edge 8 to edge 7, edge 7 to edge 6, edge 6 to edge 5, and edge 5 to edge 12.

d. Edge 12 is the start edge, so the cycle is complete.

The same figure can be navigated as a linear network, regardless of the existence of face topology (ring list, left face, right face). Assume that a line feature table exists for this example and that every line feature has an attribute column that is used to determine the correct edge selection in the network (this table is as an Area to Edge Table). The value of a line attribute will determine the traversal criteria and also determine when it will end. The starting point and endpoint of the traversal are dependent upon the user's application. A network can be traversed with various strategies. The example below illustrates a depth first search. The algorithm is as follows:

a. Locate current edge with user application (edge 12).

b. Read end node of current edge and gather all edges incident at the node.

c. For each of the edges incident at the node, read the attribute value from the associated line feature. Does the attribute have the desired value? If so, continue with step e.

d. Go to step c. Repeat with next edge.

e. Decision. Has the network been completely traversed? If so, exit with complete network. If not, go to step b.

C2.4.4 Cross-Tile Topology

Network navigation using the winged-edge topology can be extended to cross over physical tile partitions, if they exist in the coverage. By using the information in the previous examples, it becomes possible to introduce cross-tile constructs. Assume that Figure C2-5 has been intersected with tile boundaries and that the new coverage in Figure C2-8 has been created (with generalized edges along edge 5 in Figure C2-2).

Figure C2-8 depicts a single face broken into four faces by the intersection of four tiles. The following discussion identifies several different occurrences at the tile boundaries and covers retrieval of the original (untiled) face extent from the tiled faces through winged edge topology.

When creating cross-tile topology, the following rules apply:

a. An edge is always broken when it intersects a tile boundary by placing a connected node at the intersection in all adjacent tiles. See Figure C2-7, B, C, G and H. All edges terminated by this connected node will have cross-tile topology if an edge exists in the adjoining tile. See Figure C2-6, A through F.

b. The cross-tile edge will be the first edge in the adjacent tile, counterclockwise from the referencing edge at the node. See Figure C2-6, A through F.

c. All edges which are coincident with a tile boundary (all coordinates for the edge are on the boundary) occur in both tiles (see Figure C2-7, A, D, E and F) and have cross-tile left or right face topology and have cross-tile left and right edge topology. See Figure C2-6, A, B, D, E and F.

d. When a face is broken by a tile boundary, multiple faces are created by closing the face along the tile boundary. See Figure C2-7, A. The edges used to close faces on the boundary are treated as in c. above. See Figure C2-6, B and E.

e. Connected nodes which occur on tile boundaries exist in all adjacent tiles (see FIGURE C2-7, B, C, G and H), and reference both an internal and external first edge, if an edge exists. The first edge is selected arbitrarily in both internal and external tiles. If more than one tile is adjacent, the external first edge is chosen arbitrarily from the first tile counterclockwise containing one or more edges.

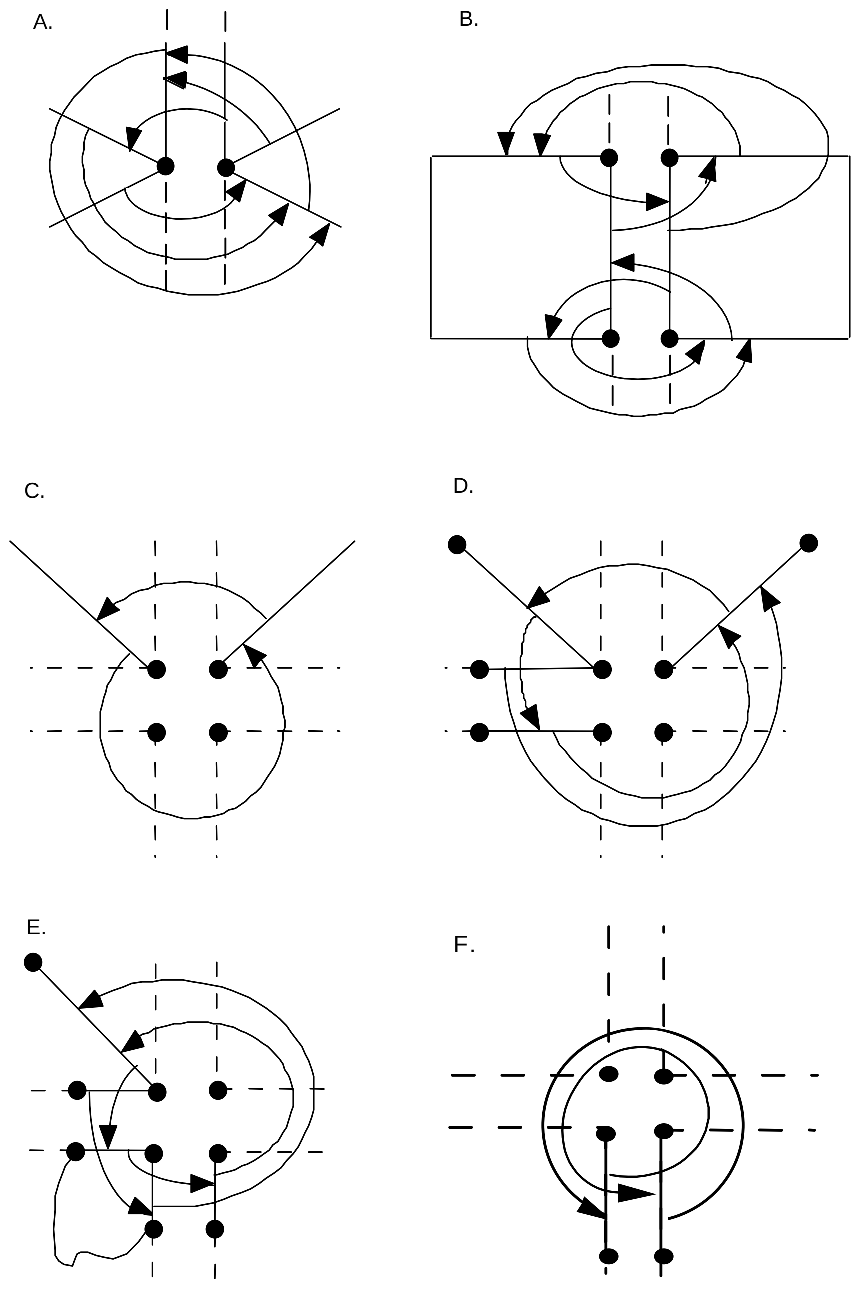


Figure C2-6 Cross-Tile Edge Rules

The following are additional examples of tile boundary primitive behavior:

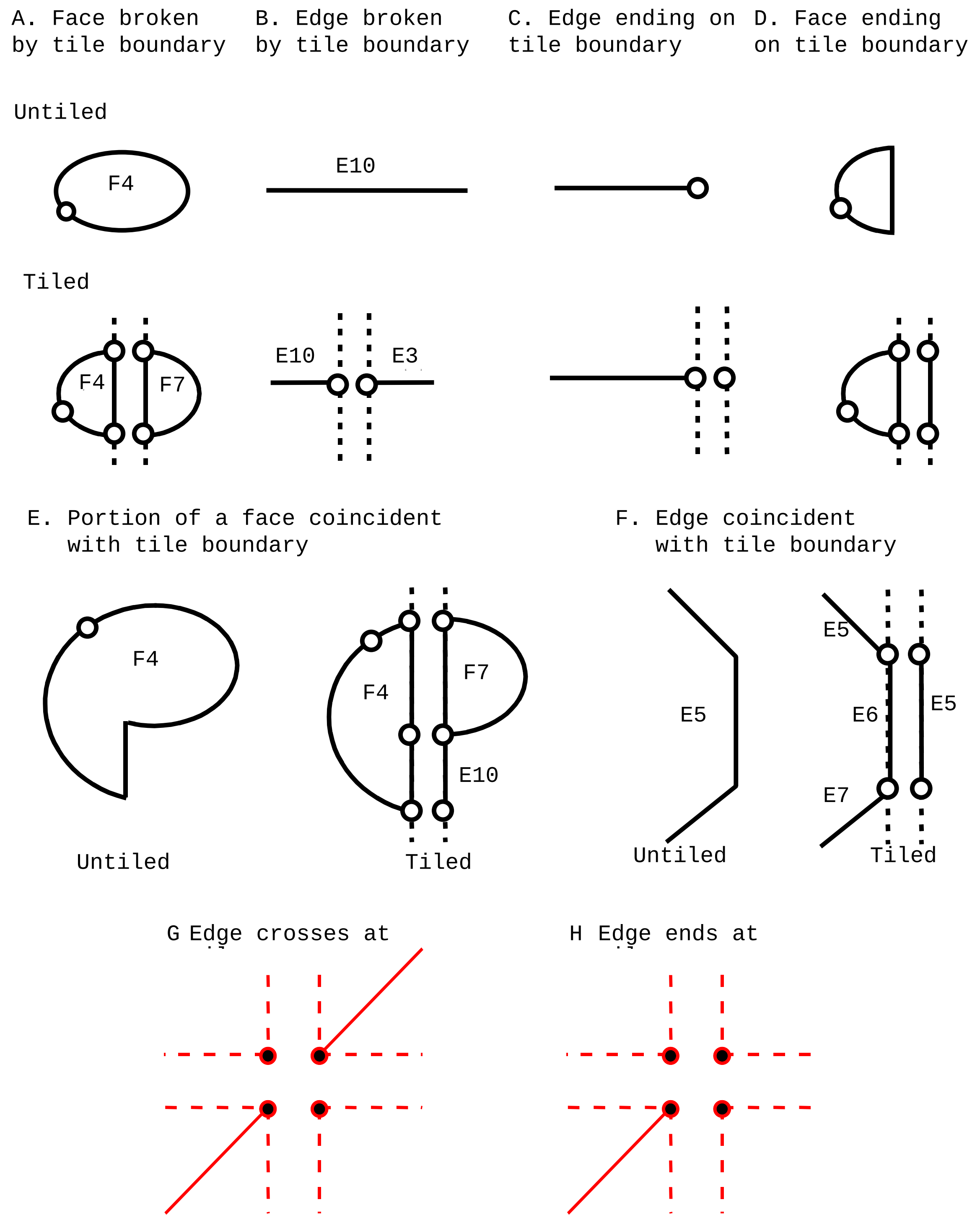
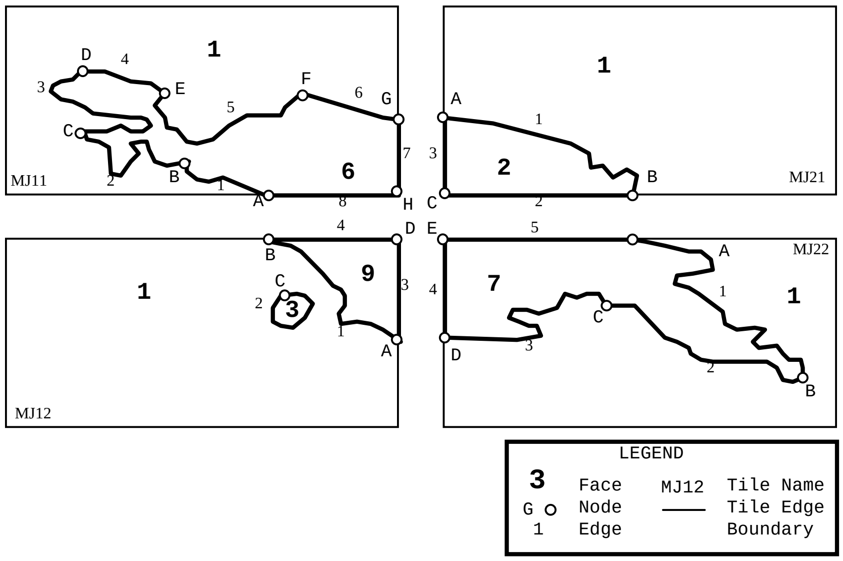

A. Face broken by tile boundary
B. Edge broken by tile boundary
C. Edge ending on tile boundary
D. Face ending on tile boundary
Untiled
F4
E10
Tiled
F4
F7
E10
E3
E. Portion of a face coincident with tile boundary
F. Edge coincident with tile boundary
F4
F4
F7
E10
E5
E5
E6
E5
E7
Untiled
Tiled
Untiled
Tiled
G Edge crosses at
H Edge ends at

Figure C2-7 Tile Boundary Primitive Behavior



Tile MJ21

id	start node	end node	right face face,tile,exface	left face face,tile,exface	right edge edge,tile,exedge	left edge edge,tile,exedge	coordinates
1	A	B	2,-,-	1,-,-	2,MJ22,5	3,MJ11,6	Not Shown
2	B	C	2,-,-	1,MJ22,7	3,MJ11,7	1,MJ22,1	
3	C	A	2,-,-	1,MJ11,6	1,MJ11,6	2,MJ11,8	

Tile MJ22

id	start node	end node	right face face,tile,exface	left face face,tile,exface	right edge edge,tile,exface	left edge edge,tile,exedge	coordinates
1	A	B	7,-,-	1,-,-	2,-,-	5,MJ21,1	Not Shown
2	B	C	7,-,-	1,-,-	3,-,-	1,-,-	
3	D	C	1,-,-	7,-,-	2,-,-	4,MJ12,3	
4	D	E	7,-,-	1,MJ12,9	5,MJ21,2	3,MJ12,1	
5	E	A	7,-,-	1,MJ21,2	1,MJ21,1	4,MJ21,3	

Note: Tile names are shown for clarity. The triplet id actually contains the tile id.

Figure C2-8 Cross-Tile Edge Example

a. Start with tile MJ21, edge 1. Read the left edge. Choose to cross into tile MJ11, edge 6.
b. Chain from edge 6, 5, 4, 3, 2, and 1 within tile MJ11.
c. From edge 1 in MJ11, go across to tile MJ12, edge 1.
d. From edge 1 in MJ12, cross tiles into tile MJ22, edge 3.
e. Chain through edges 3, 2, and 1.
f. From edge 1 in MJ22, cross into tile MJ21, edge 1.
g. When the end of the face cycle is reached, exit.

Appendix C3 - Feature Class Relations

C3.1 GENERAL

This appendix provides information, discussion, and examples concerning feature class

C3.2 APPLICABLE DOCUMENTS

This clause is not applicable to this appendix.

C3.3 DEFINITIONS

For purposes of this appendix, the definitions of Part 2 Clause 4.1 of the main document shall apply.

In this appendix, "primitive_id" (alternatively "prim_id") is used as a generic replacement for "fac_id", "edg_id", etc. In practice, "primitive" or "prim" are replaced with the actual primitive table name.

C3.4 GENERAL INFORMATION

C3.4.1 Overview

One of the most important parts of designing a VRF database is the implementation of a conceptual feature class model into feature class tables. The heart of this implementation concerns the relationships with the feature class tables and their composing primitive tables.

The structure of feature classes, and the use of join tables, thematic indexes etc. is set out in the appropriate product specification.

Depending on the design, the relationships between the features and primitives can be one-to-one (1:1), one-to-many (1:N), many-to-one (N:1), or many-to-many (N:M). This appendix is intended to help designers implement their conceptual feature classes with three major implementation constraints: software performance, tiled coverages, and indexing.

C3.4.1.1 Software Performance

Software performance is a leading consideration in a VRF database design. VRF is designed to support interactive software use. In order for software to respond optimally, the designer must determine the intended use of the database product. If intended for interactive use, the designer should design for optimal performance. If the database is strictly for exchange purposes and not interactive use, a simpler implementation is recommended.

C3.4.1.2 Tiled Coverages

Tiled coverages require a triplet id column in the feature table or associated join table to define the relationship between features and tiled primitives. Alternatively, the tile and primitive id components of the triplet id can be maintained as separate columns in the feature or join tables. Depending on the feature table type, Table C3-1 lists the column names that shall be used as the join column name.

Table C3-1 Feature Table Join Column Definitions

Area Feature	fac_id
Line Feature	edg_id
Point Feature	nod_id[1]
Text Feature	txt_id
Tile ID	tile_id[2]

Notes:
1 If modeled as separate tables this will be cnd_id or end_id.
2. The tile_id column is required in tiled coverages when the primitive id column is not defined as data type K (triplet id)

C3.4.1.3 Indexing

VRF is designed for interactive use, and the following indexing recommendations apply to help software users achieve these goals.

C3.4.1.3.1 Thematic Indexes

Implementation of thematic indexes is generally recommended on columns which are likely candidates for thematic queries. For example, thematic indexes on the primitive_id (i.e. fac_id, edg_id, nod_id, or cnd_id or end_id, txt_id) and tile_id columns in feature or join tables can improve software searches based on features located per tile. Note that VRF does not support the creation of thematic indexes on columns defined as data type K (triplet id). See Table C-67 of the DIGEST VRF Standard for a list of allowable data types.

C3.4.1.3.2 Spatial Indexes

All primitive tables in a VRF database should carry associated spatial indexes. The software performance can be improved significantly.

C3.4.1.3.3 Feature Indexes

Feature indexes enhance the retrieval of feature information when given a selected primitive. Implementation of feature indexes is generally recommended when multiple feature classes of the same primitive type exist within a single coverage (i.e. multiple line feature tables within the Ground Transportation coverage; roadl.lft and railroadl.lft). Implementation of thematic indexes on the following columns in feature index tables (*.fit) is generally recommended to further enhance performance: primitive_id, tile_id, fc_id, and feature_id.

C3.4.2 Feature and Primitive Table Relationships

The following subclauses provide a number of implementations for feature and primitive table relationships.

Each subclause is based on the five types of relationships (1:1, 1:N, N:1, N:M, and complex). Within each subclause, there are a variety of implementations available, depending on tiling, performance, and index support.

C3.4.3 One-to-One Relationships

When there is a 1:1 relationship between the feature and primitive tables, the relations are relatively straightforward. Note that for this discussion, a one-to-one relationship between features and primitives means that a feature is composed of one and only one primitive. It does not, however, imply that every primitive is associated with one and only one feature. For example, in a level three coverage with area features, the universe face (face 1) should not be associated with any feature since the outer ring is undefined. Since the Digital Geographic Information Exchange Standard (DIGEST) mandates definition of the feature-to-primitive relation, a primitive_id column (i.e. fac_id) is added to the feature table to define the relationship to the appropriate primitive. Only in very unique circumstances (i.e. An untiled, level one or two coverage with one feature class for each primitive type) can the row id in a feature table be used as a foreign key into the row id of the associated primitive table. The reverse relation (primitive-to-feature), although optional for DIGEST-compliant data, is recommended for performance reasons.

C3.4.3.1 1:1 Feature Class in an Untiled Coverage

The untiled 1:1 design (Figure C3-1) is the simplest implementation. Its performance is optimal going from the feature to its primitive. The primitive_id column in the feature table acts as a foreign key, providing a direct link to the associated record in the primitive table. However, the relationship going from the primitive back to the feature is referred to as an indirect link since the primitive_id column of all records in the feature table must be examined sequentially to find a match for a selected primitive. Such indirect links provide poor performance and are generally not recommended.

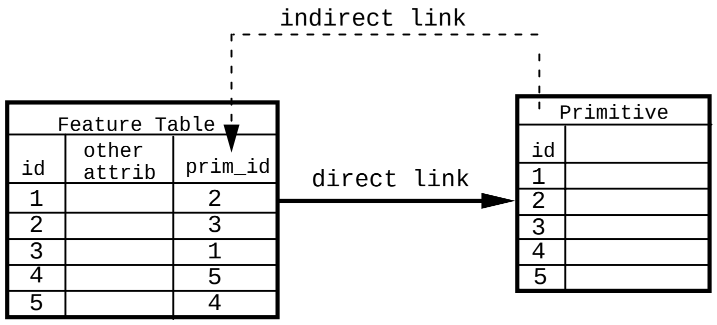


Figure C3-1. Implementation of a 1:1 Feature Class in an Untiled Coverage

- Direct links provide good performance going from the feature to its primitive.
- However, indirect links going from the primitive back to the feature provide poor performance.

C3.4.3.2 1:1 Feature Class in a Tiled Coverage

The tiled 1:1 design (Figure C3-2) is required to maintain direct relations between one feature and one primitive stored in only one tile. The performance of this implementation is good going from the feature to its primitive. However, performance going from the primitive back to the feature is poor because of the indirect link. For example, the primitive_id and tile_id columns of all records in the feature table must be examined to find all matches for a selected primitive.

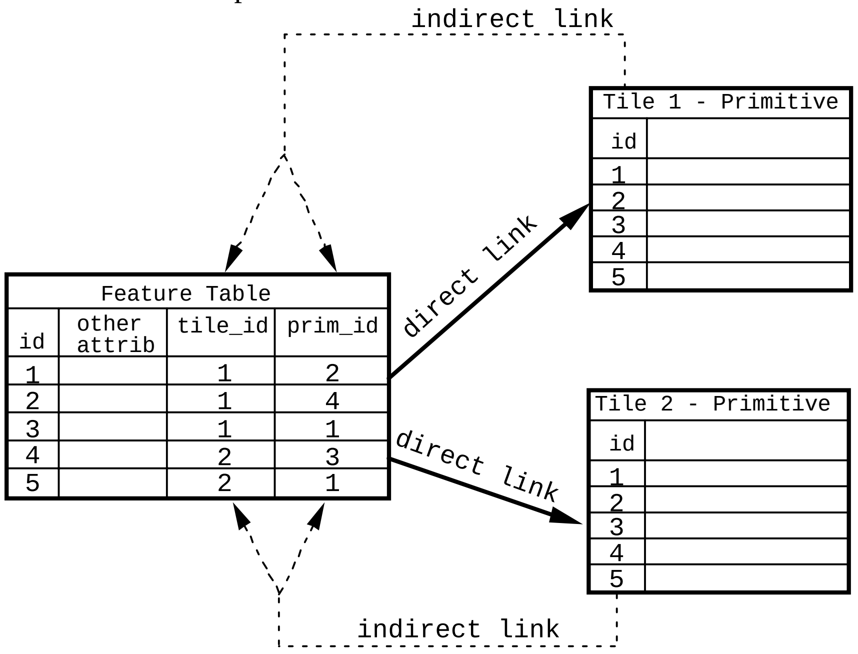


Figure C3-2. Implementation of a 1:1 Feature Class in a Tiled Coverage

- Direct link between feature and primitive provide relatively good performance.

- However, indirect links going from the primitive back to the feature provide poor performance.

C3.4.3.3 1:1 Feature Class in a Tiled Coverage with Thematic Indexes

Implementation of thematic indexes on the tile_id and/or primitive_id columns in the feature table described in C3.4.3.2 (Figure C3-3) can improve the performance when going from feature to primitive and is generally recommended. For example, a thematic index on the tile_id column in the feature table would provide a list of all records for a given tile_id value and thus improve performance on a query of all features within a selected tile. Recall (see C3.4.1.3.1), that thematic indexes cannot be implemented on a primitive_id column defined as a data type K (triplet id). Performance of the indirect link between primitive and feature is also enhanced by implementation of thematic indexes on the tile_id and primitive_id columns of feature tables. Without indexes on these columns, a sequential search of all records in the feature table would have to be performed to find all matching ids. The indexes improve performance by providing a list of the records with matching ids.

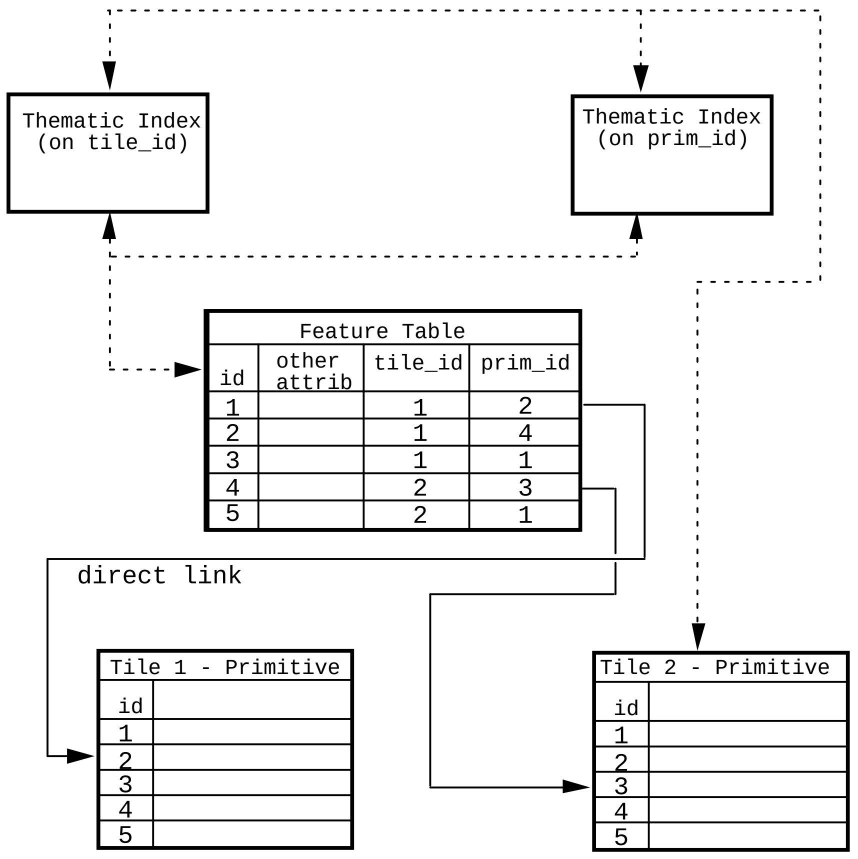
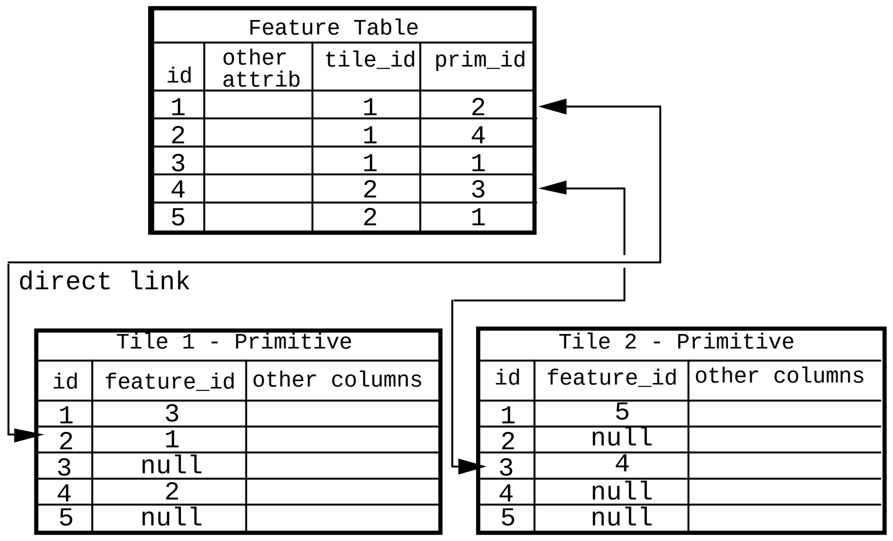


Figure C3-3 Implementation of a 1:1 Feature Class in a Tiled Coverage with a Thematic Index

- Implementation of thematic indexes improves performance of the direct link between feature and primitive.
- Performance of the indirect link between primitive and feature is also improved with the implementation of thematic indexes on the tile_id and primitive_id columns in feature tables.

C3.4.3.4 1:1 Feature Classes in a Tiled Coverage with feature_id Pointers in the Primitive Tables

One way to improve the performance of the implementation described in C3.4.3.3 is by adding a feature_id column to the primitive tables (Figure C3-4) This improves performance by adding a direct link between the primitive and associated feature. However, database designers and software developers should be aware of some of the shortcomings and/or limitations of this feature class design. Implementation of this design in a coverage with multiple feature classes for a given primitive type (i.e. roadl.lft, railroadl.lft, tunnell.lft in a Ground Transportation Coverage) results in an unnormalized primitive table (many of the feature_id columns will contain null values). Secondly, this design does not allow for the existence of coincident features (N:1 relations) in the same feature class (the feature_id column can only contain one reference to a feature in a given feature class).



Feature Table

id	other attrib	tile_id	prim_id
1		1	2
2		1	4
3		1	1
4		2	3
5		2	1

Tile 1 - Primitive

id	feature_id	other columns
1	3	
2	1	
3	null	
4	2	
5	null	

Tile 2 - Primitive

id	feature_id	other columns
1	5	
2	null	
3	4	
4	null	
5	null	

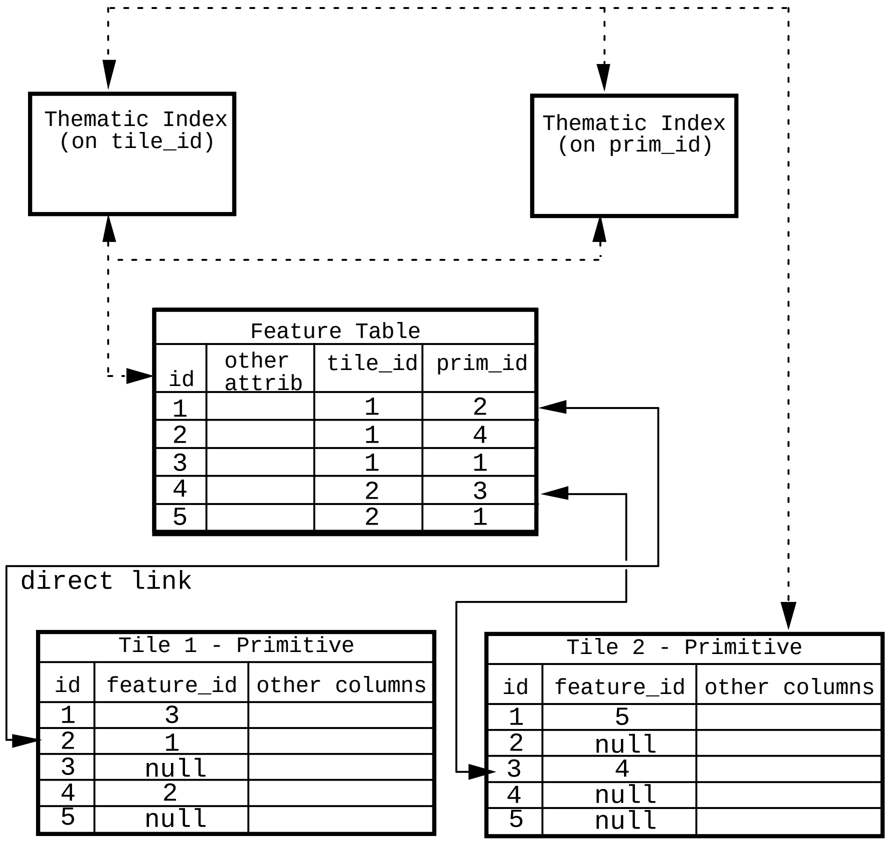
Figure C3-4 Implementation of a 1:1 Feature Class in a Tiled Coverage with feature_id Columns Added to the Primitive Tables

- Feature-to-Primitive and Primitive-to-Feature performance is good because of the implementation of direct links

- Unnormalized tables and limitations in coincident features represent disadvantages of this type of design.

C3.4.3.5 1:1 Feature Classes in a Tiled Coverage with feature_id Pointers in the Primitive Tables and Thematic Indexes

Addition of thematic indexes to the implementation described in C3.4.3.4. improves the performance in both the Feature-to-Primitive and Primitive-to-Feature directions (Figure C3-5). See C3.4.3.3. for a discussion of how thematic indexes improve performance. Thematic indexes are generally recommended on the tile_id and primitive_id columns in feature table(s).



Feature Table

id	other attrib	tile_id	prim_id
1		1	2
2		1	4
3		1	1
4		2	3
5		2	1

Tile 1 - Primitive

id	feature_id	other columns
1	3	
2	1	
3	null	
4	2	
5	null	

Tile 2 - Primitive

id	feature_id	other columns
1	5	
2	null	
3	4	
4	null	
5	null	

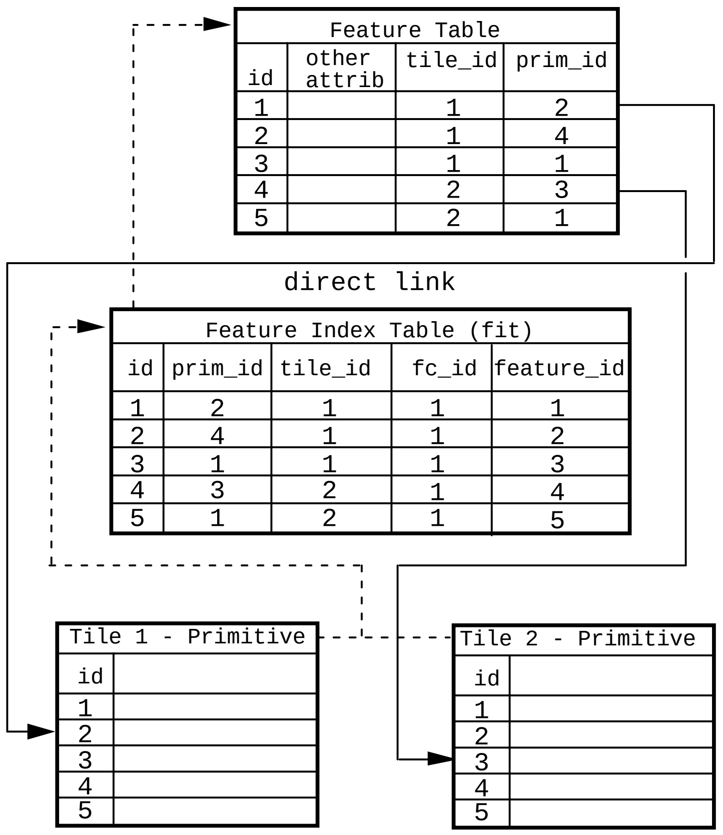
Figure C3-5 Implementation of a 1:1 Feature Class in a Tiled Coverage with feature_id Columns in the Primitive Tables and Thematic Indexes

- Feature-to-Primitive and Primitive-to-Feature performance is very good because of the implementation of direct links as well as the addition of thematic indexes.

- Unnormalized tables and limitations in coincident features represent disadvantages of this type of design (see C3.4.3.4).

C3.4.3.6 1:1 Feature Classes in a Tiled Coverage with Feature Indexes

Implementation of Feature Indexes (Figure C3-6) to improve Primitive-to-Feature performance is an alternative to adding feature_id pointers to the primitive tables. This type of implementation is probably best suited for coverages with multiple feature classes of the same primitive type since it eliminates non-normalized primitive tables. It also allows for the existence of coincident features within the same feature class. Implementation of thematic indexes on the following columns in the Feature Index Table (fit) is recommended: primitive_id, tile_id, fc_id, and feature_id.



Feature Table

id	other attrib	tile_id	prim_id
1		1	2
2		1	4
3		1	1
4		2	3
5		2	1

Feature Index Table (fit)

id	prim_id	tile_id	fc_id	feature_id
1	2	1	1	1
2	4	1	1	2
3	1	1	1	3
4	3	2	1	4
5	1	2	1	5

Tile 1 - Primitive

id	
1	
2	
3	
4	
5	

Tile 2 - Primitive

id	
1	
2	
3	
4	
5	

Figure C3-6 Feature-to-Primitive and Primitive-to-Feature Linkage

- Feature-to-Primitive and Primitive-to-Feature performance is very good because of the implementation of direct links as well as the addition of a feature index. Performance of this type of design is further enhanced with the implementation of thematic indexes on columns in the Feature Index Table (fit) and Feature Tables.

- This design eliminates unnormalized primitive tables and allows the creation of coincident features in the same feature class.

C3.4.4 One-to-Many Relationships

The next relation type is one in which a single feature is composed of many primitives. This type of feature is often referred to as a compound feature and requires implementation of a join table to define the one-to-many relationship between a feature and its associated primitives.

C3.4.4.1 1:N Feature Class in an Untiled Coverage Using Join Tables

As stated in C3.4.4, a join table is required to define the one-to-many relationship between a feature and its associated primitives. The primitive_id column in the join table acts as a foreign key and provides a direct link to the primitive(s) associated with a selected feature. The reverse relationship (primitive-to-feature) is an indirect link and provides poor performance. For example, the primitive_id column of all records in all feature join tables would have to be examined to find the feature associated with a selected primitive.

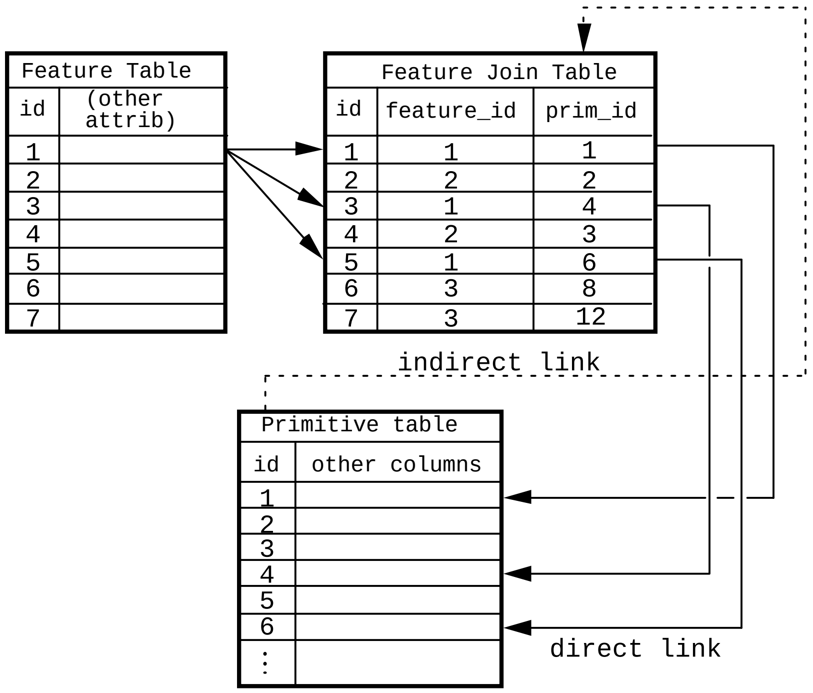


Figure C3-7 Implementation of 1:N Feature Class in an Untiled Coverage with a Join Table

- Although a direct link between feature and primitive(s) is provided by the primitive_id column in the join table, a sequential search of the feature_id column must still be performed to find <u>all</u> primitives associated with a selected feature. As a result of the sequential search, performance going from the feature to primitive is relatively slow.

- The indirect link going from the primitive back to the feature provides poor performance.

C3.4.4.2 1:N Feature Class in an Untiled Coverage with a Thematic Index

Addition of a thematic index to the implementation described C3.4.4.1 can improve the performance when going from feature-to-primitive. For example, a thematic index on the feature_id column in the join table will improve performance on a query which needs to find all primitives associated with a selected feature. Note that primitive to feature performance is still poor because of the indirect link.

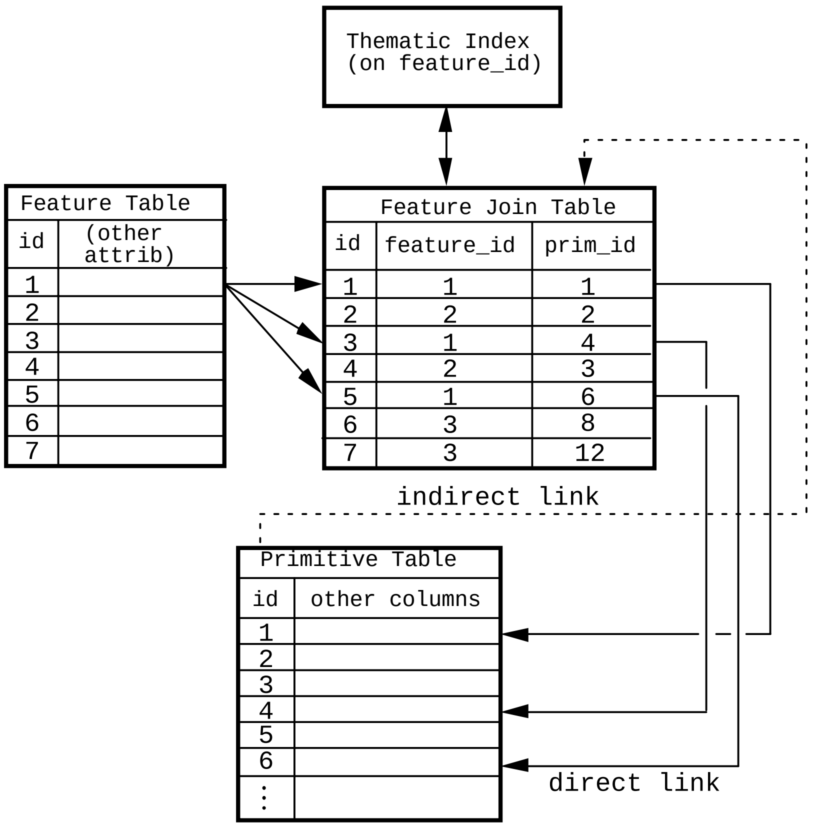


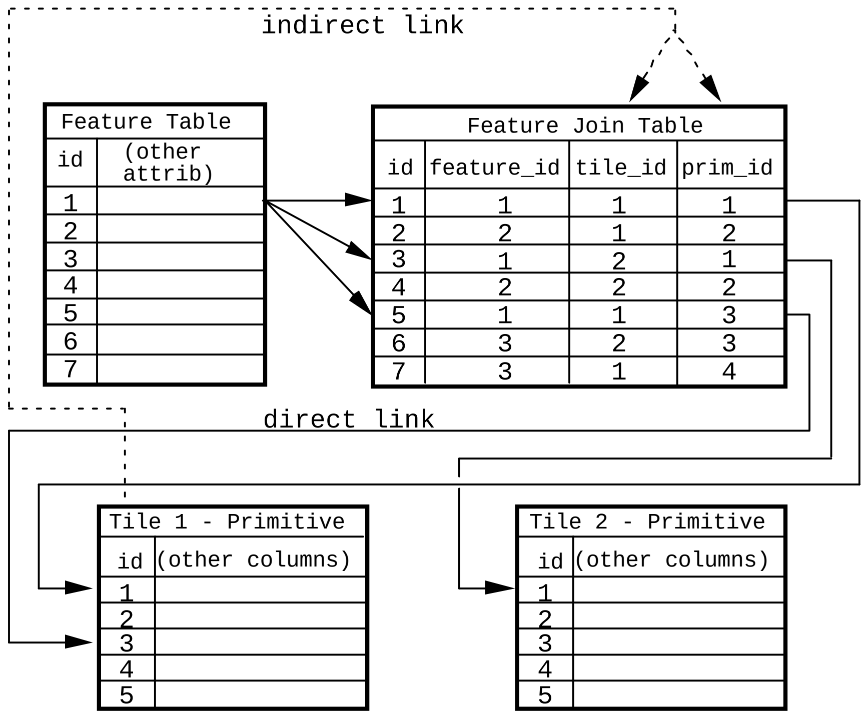
Figure C3-8 Implementation of a 1:N Feature Class in an Untiled Coverage using Join Tables and Thematic Indexes

- Addition of a thematic index to the feature_id column in the join table improves the performance of this feature class design when going from the feature to its primitive(s).

- However, the indirect link going from the primitive back to the feature provides poor performance.

C3.4.4.3 1:N Feature Class in a Tiled Coverage

The addition of tiles in a 1:N relationship makes for a complex implementation. The implementation shown in Figure C3-9 is conceptually clean but will perform poorly with software because of the lack of thematic indexes.



Feature Table

id	(other attrib)
1	
2	
3	
4	
5	
6	
7	

Feature Join Table

id	feature_id	tile_id	prim_id
1	1	1	1
2	2	1	2
3	1	2	1
4	2	2	2
5	1	1	3
6	3	2	3
7	3	1	4

Tile 1 - Primitive

id	(other columns)
1	
2	
3	
4	
5	

Tile 2 - Primitive

id	(other columns)
1	
2	
3	
4	
5	

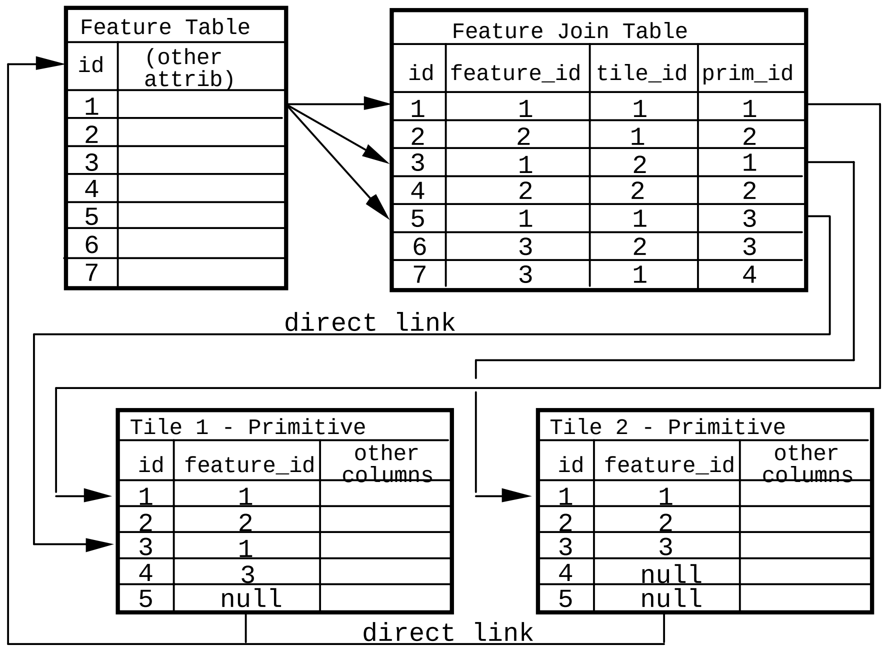
Figure C3-9 Implementation of a 1:N Feature Class in a Tiled Coverage

- Although a direct link between feature and primitive(s) is provided by the tile_id and primitive_id columns in the join table, a sequential search of the feature_id column must still be performed to find <u>all</u> primitives associated with a selected feature. As a result of the sequential search, performance going from the feature to primitive is relatively slow.

- The indirect link going from the primitive back to the feature provides poor performance.

C3.4.4.4 Tiled 1:N Coverages with feature_id Pointers in the Primitive Tables

The addition of a feature_id column to the primitive tables (Figure C3-10) provides good performance from the primitive to the feature. However, keep in mind the disadvantages and/or shortcomings of this method as identified in C3.4.3.4. Performance from the feature to the join table to the primitive is relatively poor without the implementation of thematic indexes.



Feature Table	
id	(other attrib)
1	
2	
3	
4	
5	
6	
7	

Feature Join Table			
id	feature_id	tile_id	prim_id
1	1	1	1
2	2	1	2
3	1	2	1
4	2	2	2
5	1	1	3
6	3	2	3
7	3	1	4

Tile 1 - Primitive		
id	feature_id	other columns
1	1	
2	2	
3	1	
4	3	
5	null	

Tile 2 - Primitive		
id	feature_id	other columns
1	1	
2	2	
3	3	
4	null	
5	null	

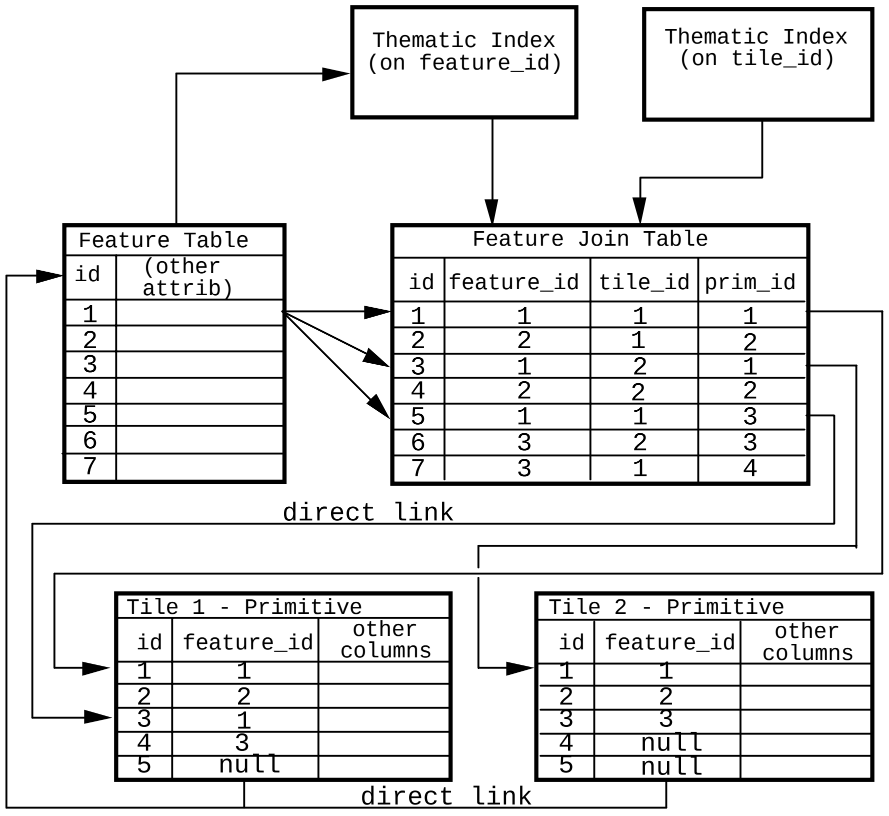
Figure C3-10 Implementation of a 1:N Feature Class in a Tiled Coverage that includes feature_id Columns in Primitive Tables

- Primitive-to-Feature performance is good because of the implementation of direct links. However, be aware of the shortcomings and disadvantages of the implementation of feature_id pointers in primitive tables (see C3.4.3.4)

- Although a direct link between feature and primitive(s) is provided by the tile_id and primitive_id columns in the join table, a sequential search of the feature_id column must still be performed to find all primitives associated with a selected feature. As a result of the sequential search, performance going from the feature to primitive is relatively slow.

C3.4.4.5 1:N Feature Class in a Tiled Coverage with feature_id Pointers in the Primitive Tables and Thematic Indexes

The addition of thematic indexes to columns in the join table (Figure C3-11) improves performance going from the feature to primitive(s). Thematic indexes are generally recommended on the feature_id and tile_id columns in join tables. Implementation of feature_id pointers in primitive tables provides good performance going from the primitive to feature. However, keep in mind the disadvantages and/or shortcomings of this method as identified in C3.4.3.4.



Feature Table

id	(other attrib)
1	
2	
3	
4	
5	
6	
7	

Feature Join Table

id	feature_id	tile_id	prim_id
1	1	1	1
2	2	1	2
3	1	2	1
4	2	2	2
5	1	1	3
6	3	2	3
7	3	1	4

Tile 1 - Primitive

id	feature_id	other columns
1	1	
2	2	
3	1	
4	3	
5	null	

Tile 2 - Primitive

id	feature_id	other columns
1	1	
2	2	
3	3	
4	null	
5	null	

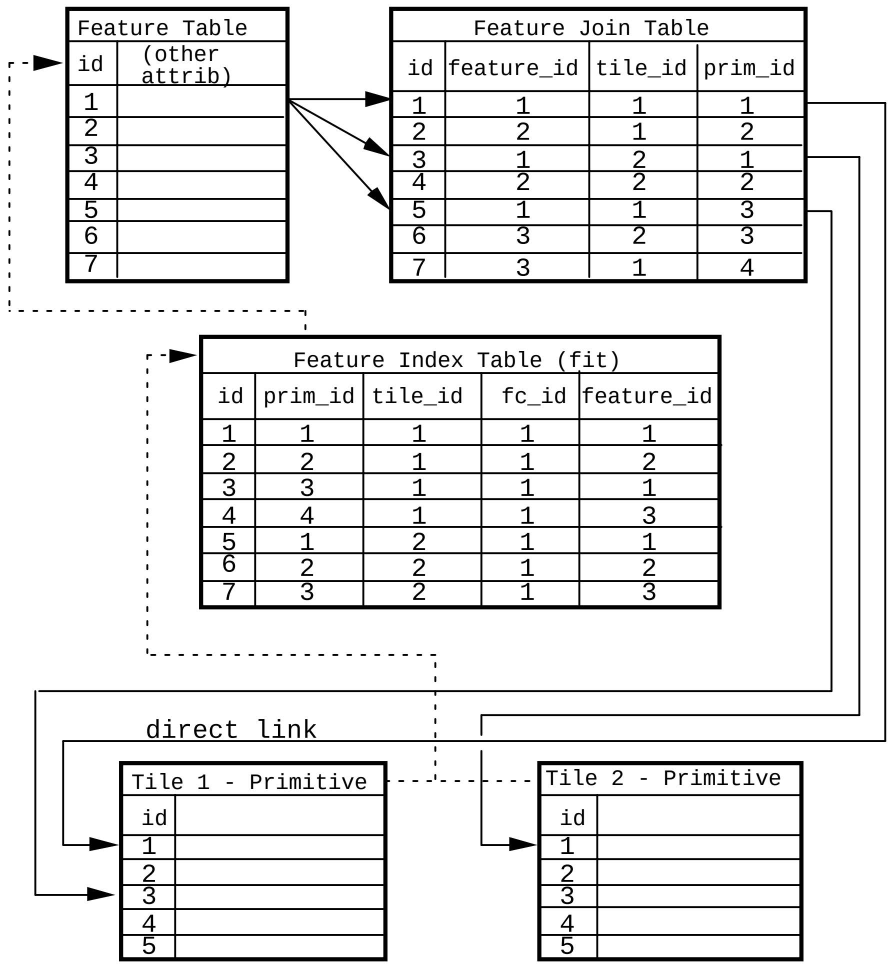
Figure C3-11. Implementation of a 1:N Feature Class in a Tiled Coverage that includes feature_id Columns and Thematic Indexes

- Feature-to-Primitive and Primitive-to-Feature performance is good because of the implementation of direct links as well as the addition of thematic indexes

- Be aware of the shortcomings and disadvantages of the implementation of feature_id pointers in primitive tables (see C3.4.3.4)

C3.4.4.6 1:N Feature Class in a Tiled Coverage with Feature Indexes and Thematic Indexes

As stated in C3.4.3.6, implementation of feature indexes to improve Primitive-to-Feature performance is an alternative to adding feature_id pointers to primitive tables. It is probably best suited for coverages with multiple feature classes of the same primitive type since it eliminates non-normalized primitive tables. Implementation of thematic indexes on the feature_id and tile_id column in the join tables as well as the following columns in the Feature Index Tables (fit's) is generally recommended: primitive_id, tile_id, fc_id, and feature_id.



Feature Table

id	(other attrib)
1	
2	
3	
4	
5	
6	
7	

Feature Join Table

id	feature_id	tile_id	prim_id
1	1	1	1
2	2	1	2
3	1	2	1
4	2	2	2
5	1	1	3
6	3	2	3
7	3	1	4

Feature Index Table (fit)

id	prim_id	tile_id	fc_id	feature_id
1	1	1	1	1
2	2	1	1	2
3	3	1	1	1
4	4	1	1	3
5	1	2	1	1
6	2	2	1	2
7	3	2	1	3

Tile 1 - Primitive

id	
1	
2	
3	
4	
5	

Tile 2 - Primitive

id	
1	
2	
3	
4	
5	

Figure C3-12 Implementation of a 1:N Feature Class in a Tiled Coverage with Feature Indexes

Feature-to-Primitive performance is good because of the direct link and thematic indexes.

Primitive-to-Feature performance is good because of the implementation of feature indexes.

C3.4.5 Many-to-One Relationships

When multiple features are associated with the same primitive, the relationship is often referred to as coincident features or N:1. For the sake of brevity, no examples of N:1 relations are provided. Instead, refer to clause C3.4.3 (1:1 relations) for the different types of performance enhancers available, keeping in mind the shortcomings/disadvantages identified in C3.4.3.4.

C3.4.6 Many-to-Many Relationship

In many-to-many (N:M) relationships, many features can relate to many primitives and the reverse. Many features can relate to one primitive, and many primitives can relate back to one feature. This relation is established using a join table. Many-to-many relations are typical and often unavoidable in an integrated coverage database design. Again, for the sake of brevity, no examples are provided for N:M relations. Instead, refer to clauses C3.4.3 and C3.4.4 for the type of performance enhancers available, making note of the advantages and disadvantages of each.

C3.4.7 Complex Feature Relationships

A complex feature may be constructed from simple features only, or from other complex features. This forms a hierarchical feature relationship. The need for complex features arises when a group of features requires different attributes than that of other features.

C3.4.7.1 One Complex Feature Composed of Simple Features from Multiple Feature Classes

As Figure C3-13 shows, a complex feature can be composed of simple features from multiple feature classes (in this case, an area feature class and line feature class). The implementation shown in Figure C3-13 is useful when not all simple features are part of a complex feature or when the complex feature is created after the simple features. This type of complex feature design has its limitations. Because of the implementation of feature_id foreign keys (i.e. *.aft_id and *.lft_id) in the complex feature table, a complex feature cannot be composed of more than one simple feature from a given feature class. For example, this type of design does <u>not</u> support the creation of an Interstate Hwy. complex feature composed of a number of road line simple features. Implementation of a complex feature join table which defines the complex feature to simple feature relations is generally considered a better design (see C3.4.7.2.)

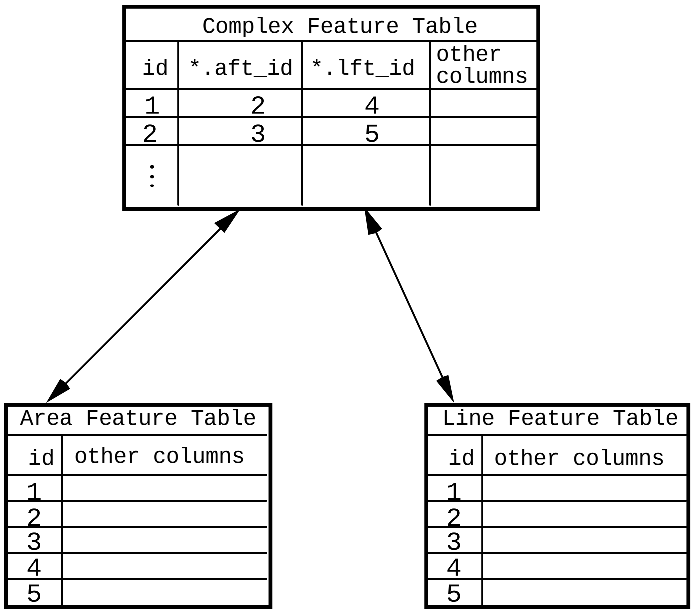
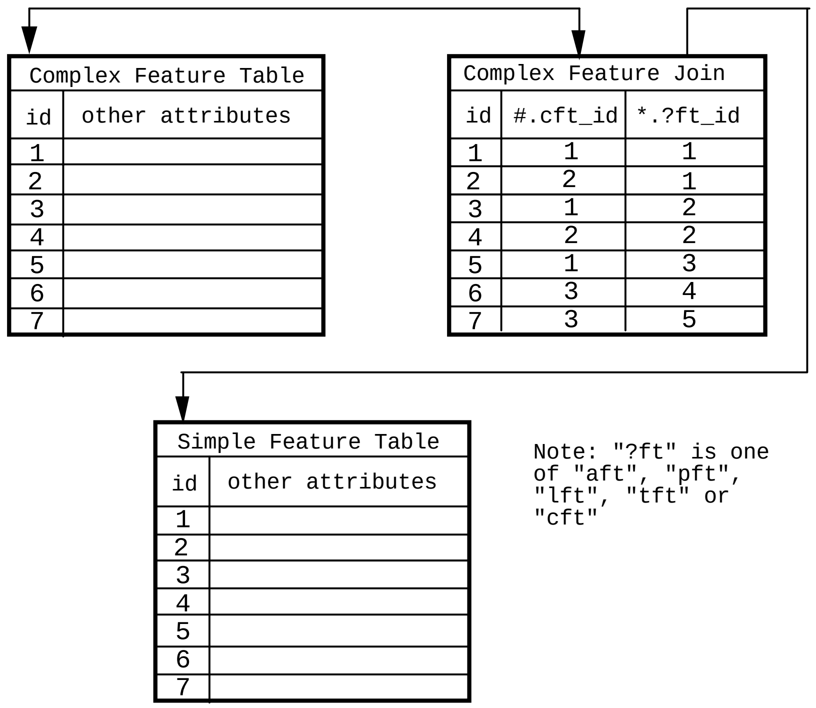


Figure C3-13. Implementation of a Complex Feature composed of Simple Features in Separate Tables

C3.4.7.2 Many Complex, Many Simple Features

Many complex features may be made up of many simple features in one feature table. The implementation shown in Figure C3-14 requires complex features and simple features to be created at the same time. It also requires the implementation of a complex feature join table to describe the complex feature to simple feature relations. Performance can be improved by adding thematic indexes on both the columns in the join table.



Complex Feature Table

id	other attributes
1	
2	
3	
4	
5	
6	
7	

Complex Feature Join

id	#.cft_id	*.?ft_id
1	1	1
2	2	1
3	1	2
4	2	2
5	1	3
6	3	4
7	3	5

Simple Feature Table

id	other attributes
1	
2	
3	
4	
5	
6	
7	

Note: "?ft" is one of "aft", "pft", "lft", "tft" or "cft"

Figure C3-14 Implementation of a Complex Feature Relationship in which Many Complex Features are Made Up of Many Simple Features in One Feature Table

C3.4.7.3 An Example of a Coverage with Simple and Complex Features

Table C3-2 is the feature class schema table from the Terminal Procedure Coverage of DFLIP Prototype No. 2. The terminal procedure complex feature (termpc) consists of action points (pactc) which are related to connected nodes, Instrument Landing System (ILS) line features (ilsl) and other route line features (seg2l). Records 3 and 4 show the relationships for the action points (pactc) and their connected nodes (cnd), records 7 to 10 show how the ILS lines (ilsl) are joined to multiple edges, and records 15 to 18 show how route lines (seg2l) are joined to multiple edges. The complex feature (termpc) realtionships begin with record 19 and continue through record 40. Figure C3-15 is helpful in tracing each record to the first and second table columns of the record.

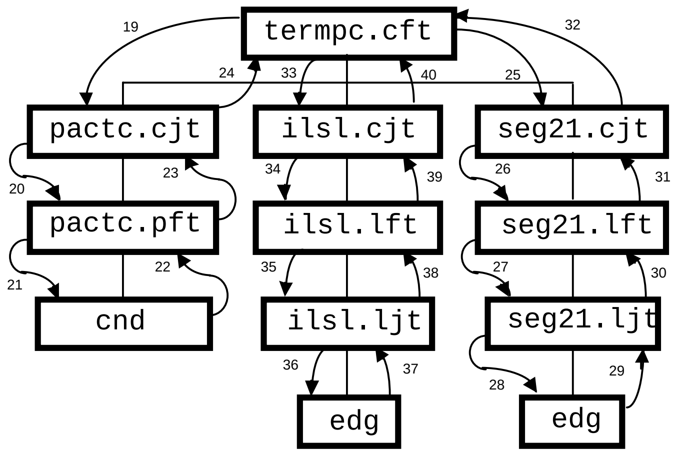


Figure C3-15 fcs Record Numbers Linking Tables for Complex Feature

Table C3-2 Content and Format for Terminal Procedures
Coverage Feature Class Schema Table

{Header length}L;
Terminal Procedures Feature Class Schema Table;-;
id=I,1,P,Row Identifier,-,-,-,:
feature_class=T,7,N,Name of Feature Class,-,-,-,:
table1=T,12,N,First Table in a Relationship,-,-,-,:
table1_key=T,16,N,Column Name in First Table,-,-,-,:
table2=T,12,N,Second Table in a Relationship,-,-,-,:
table2_key=T,16,N,Column Name in Second Table,-,-,-,:;

1	nav2c	nav2c.pft	cnd_id	cnd	id
2	nav2c	cnd	id	nav2c.pft	cnd_id
3	pactc	pactc.pft	cnd_id	cnd	id
4	pactc	cnd	id	pactc.pft	cnd_id
5	hold2c	hold2c.pft	cnd_id	cnd	id
6	hold2c	cnd	id	hold2c.pft	cnd_id
7	ilsl	ilsl.lft	id	ilsl.ljt	ilsl.lft_id
8	ilsl	ilsl.ljt	edg_id	edg	id
9	ilsl	edg	id	ilsl.ljt	edg_id
10	ilsl	ilsl.ljt	ilsl.lft_id	ilsl.lft	id
11	rab2l	rab2l.lft	id	rab2l.ljt	rab2l.lft_id
12	rab2l	rab2l.ljt	edg_id	edg	id
13	rab2l	edg	id	rab2l.ljt	edg_id
14	rab2l	rab2l.ljt	rab2l.lft_id	rab2l.lft	id
15	seg2l	seg2l.lft	id	seg2l.ljt	seg2l.lft_id
16	seg2l	seg2l.ljt	edg_id	edg	id
17	seg2l	edg	id	seg2l.ljt	edg_id
18	seg2l	seg2l.ljt	seg2l.lft_id	seg2l.lft	id
19	termpc	termpc.cft	id	pactc.cjt	termpc.cft_id
20	termpc	pactc.cjt	pactc.pft_id	pactc.pft	id
21	termpc	pactc.pft	cnd_id	cnd	id
22	termpc	cnd	id	pactc.pft	cnd_id
23	termpc	pactc.pft	id	pactc.cjt	pactc.pft_id
24	termpc	pactc.cjt	termpc.cft_id	termpc.cft	id
25	termpc	termpc.cft	id	seg2l.cjt	termpc.cft_id
26	termpc	seg2l.cjt	seg2l.lft_id	seg2l.lft	id
27	termpc	seg2l.lft	id	seg2l.ljt	seg2l.lft_id
28	termpc	seg2l.ljt	edg_id	edg	id
29	termpc	edg	id	seg2l.ljt	edg_id
30	termpc	seg2l.ljt	seg2l.lft_id	seg2l.lft	id
31	termpc	seg2l.lft	id	seg2l.cjt	seg2l.lft_id
32	termpc	seg2l.cjt	termpc.cft_id	termpc.cft	id
33	termpc	termpc.cft	id	ilsl.cjt	termpc.cft_id
34	termpc	ilsl.cjt	ilsl.lft_id	ilsl.lft	id
35	termpc	ilsl.lft	id	ilsl.ljt	ilsl.lft_id
36	termpc	ilsl.ljt	edg_id	edg	id
37	termpc	edg	id	ilsl.ljt	edg_id
38	termpc	ilsl.ljt	ilsl.lft_id	ilsl.lft	id
39	termpc	ilsl.lft	id	ilsl.cjt	ilsl.lft_id
40	termpc	ilsl.cjt	termpc.cft_id	termpc.cft	id

[This page intentionally left blank]

Appendix C4 - Tiling

C4.1 GENERAL

This appendix provides information and discussion concerning tiling for a VRF database.

C4.2 APPLICABLE DOCUMENTS

This clause is not applicable to this appendix.

C4.3 DEFINITIONS

For purposes of this appendix, the definitions of Part 2 clause 4.1 of the main document shall apply.

C4.4 GENERAL INFORMATION

C4.4.1 Rationale

Global scale databases inevitably consist of large amounts of data. In processing geographic data, entire files often need to be managed in memory, imposing a definite limit on database size. Tiling is the method used to break up geographic data into spatial units small enough to fit within the limitations of the desired hardware platform and media. VRF libraries are partitioned into a tile structure defined in a particular product specification and tileref coverage. The naming convention is based on the geographic reference system GEOREF. The actual tile size is a product-specific question dependent upon the minimum hardware configuration and distribution media. The following paragraphs describe how VRF implements tiling to subdivide a database.

C4.4.2 Cross-Tile Topological Primitives

One shortcoming of past tiling implementations occurs when primitives are split up into different files, which also removes the topological connectivity of the feature. If a lake feature's primitive is split into two separate tiles, the topological connection between the primitives of the lake is lost. They will still appear together when both tiles are drawn on the screen, but any analysis that tries to follow the original connectivity of the lakeshore will have to do a lot of extra processing and searching. It would thus be advantageous for the primitives of the tiled lake shoreline to refer to each other. There is a need for primitives that cross tile boundaries to refer both to the tile boundary itself (in order to maintain tile topology) and to its cross-tile continuation in order to simplify retrieval of the original feature. VRF meets this need by referring to edges using a triplet id that contains an internal reference to a boundary or edge within the current tile; when appropriate, the triplet id also contains an external reference to an edge in a neighboring tile.

C4.4.3 Feature Classes

Tiling introduces a constraint on feature classes as well as primitives. Tiling is a low-level implementation issue related to hardware and storage limitations, and should have no effect upon conceptual structures like feature classes. Unfortunately, when primitives are broken up into separate tables for tiling, their corresponding attributes are broken up as well. VRF resolves this by maintaining the attribute tables and feature class tables unbroken and structuring them so that they can be processed sequentially rather than all at once, thus obviating the need to break them up to meet hardware limits. These tables are stored within a coverage, and the actual file subdivisions representing the tiles appear as directories underneath the coverage.

The problem concerns connecting a primitive, now stored in one of many smaller files, to its attributes, now stored in a single large file. This is done in accord with standard relational design rules by adding a column for each feature class onto the primitive file, containing the row id from the master table that has the attributes for that primitive. If five feature classes are derived from the primitive topological layer, then five extra columns will be added to that primitive table. Adding another column for each feature class could make for a very large table since there can be many feature classes in a coverage.

Feature classes apply to all three primitive dimensions or to all products, or to all coverages, but for vertically or thematically integrated data the number could still easily approach 100. The reason for a pointer back to the feature is merely for performance issues. Please refer to feature class construction issues in Appendix C3.

VRF handles tiling and data partition problems at the primitive level by means of the triplet id to maintain cross-tile topology and the extra feature class columns on the primitive to maintain links to the more logically consistent single attribute table. There is also the need to maintain the tiles themselves and to store reference data about the tiles, their size, scheme, and so forth. This is accomplished with the tile reference coverage.

C4.4.4 Tile Reference Coverage

The tile coverage is a level 3 topology coverage representing only the tiles. No other data besides tile boundaries (defined by faces) and tile labels is used. This is stored as a separate coverage at the library level and acts as a graphic index to the tiling scheme, showing all of the tiles and only those tiles in the library, their names, and their relation to each other. The area feature table of this coverage plays a very important role. Since it can store attributes about each tile, it can be used to hold data density figures, tile data volume, summary contents for each feature class, and other metadata to assist in managing the database at a coarse tile-by-tile level.

The tile size, actual tile layout, and handling of text that crosses tile borders are not addressed in this standard since they are all product-specific questions.

Appendix C5 - Data Quality

C5.1 GENERAL

This appendix provides information, discussion, and examples concerning data quality issues in a VRF database.

C5.2 APPLICABLE DOCUMENTS

This clause is not applicable to this appendix.

C5.3 DEFINITIONS

For purposes of this appendix, the definitions of Part 2 clause 4.1 of the main document shall apply.

C5.4 GENERAL INFORMATION

C5.4.1 VRF Data Quality

This appendix describes basic strategies for storing data quality (dq) information within VRF databases. The following subclauses discuss general data quality concepts, implementation of the data quality table, and data quality coverages within VRF.

C5.4.2 General Concepts

The multilevel structure (i.e., primitive, feature , feature class, coverage, library, and database) of a VRF database affects the strategies used to store dq information. Data Quality information is often available at varying levels of specificity, from individual feature attributes to expressions of quality relevant to an entire database. Therefore, it is necessary for the VRF data producer to determine the appropriate level in the hierarchy for the various types of dq information present. When developing an implementation strategy, the data producer should also review the available dq information within the context of coverage (thematic) associations as well as spatial extent, as these will influence the use of standard data quality tables and/or the development of separate data quality coverages.

When dq information is stored at multiple levels in a VRF database, lower level information always takes precedence over that at higher levels. Data producers should account for this when compiling dq information to be stored in VRF. For example, a data producer may make a general statement at the library level that a coverage has been derived from a range of sources, and specify at the primitive level what a given feature's exact origins are. Alternatively, some variations may be organized by simple spatial divisions, such as source map boundaries. In such a case, feature level source information would be highly repetitious, and a data quality coverage might be most effective.

C5.4.3 Data Quality Tables

The data quality table (Table C-52) is a device for storing standard and nonstandard types of dq information. Standard information is explicitly defined within the dq table. Nonstandard information is that which is not accounted for in the dq table and must be stored outside the standard fields. The dq table can be implemented on the database, library, or coverage level, depending upon the uniformity of the affected data with respect to known dq characteristics. Key characteristics are source, positional accuracy, attribute accuracy, logical consistency, completeness, resolution, and lineage. All of these characteristics can be documented within standard dq table fields, with the exception of lineage.

C5.4.3.1 Lineage

Lineage information must be stored within the dq table narrative file, which should be given the name "lineage.doc." The lineage file is an important component of the dq documentation system, since the data producer must inevitably make key decisions affecting the data's fitness for use that cannot be described in standard fields. Lineage information may also change significantly from coverage to coverage, even when all of the data is derived from a single source. At a minimum, the lineage file should contain information on processing tolerances, interpretation rules applied to source materials, and basic production and quality assurance procedures. All lineage information available through the source should also be incorporated here.

C5.4.3.2 Placement of DQ Table

When implementing dq tables, the user must determine at what level within the VRF hierarchy the dq table(s) should be established. This will vary depending upon the specific nature of the dq information. For example, entire libraries originating from a single source may be best served with a single table at that level, augmented by a series of data producer-defined tables implemented at the coverage level to document more specific information. The dq table can also be implemented on multiple levels simultaneously, with general (broad) information provided at the higher levels, and progressively more detailed information specified at the lower levels. At a minimum, some form of dq information should be present at the database or library levels to provide an overview of characteristics and to describe the techniques used to store dq information within the database as a whole. With respect to VRF tables in general, the data producer is not restricted to the standard dq table. The table can be augmented as needed with producer-defined related files within the hierarchy.

C5.4.4 Data Quality Coverages

Data quality coverages delineate spatial variations in dq information across a database or a library by assigning unique quality characteristics to areas. Within this context, the database and library levels must be viewed as having distinct spatial extents to which attributes can be assigned, while data quality coverages are created to document spatial variations on a more detailed level.

Data quality coverages are very useful as visual reference data that allow data producers to capture anomalous data behavior within the context of known spatial variations. They are, by definition, level 3 topology coverages. They can be physically stored at any level (above the feature level) within the VRF hierarchy, independent of the level of information being represented.

The manner in which the dq information is structured within the coverage will be dependent upon the relationships between spatial variations, data sources, and lineage information. Edges, as well as areas, can also be attributized within dq coverages, for example, to document the interaction of disparate data sets that are not well reconciled. A more specific example is where contour lines from different data sources meet along a common edge and fail to match positionally. In this case, the data producer could document this discrepancy as an attribute of the seam (edge) between the adjacent sources.

Unlike tabular dq information, the data producer should refrain from creating "nested" dq coverages at multiple levels, as these can greatly complicate the interpretation of the information. Rather, the user is encouraged to adopt a more integrated approach, where general information is carried on lower-level faces in addition to level-specific information.

C5.4.4.1 Coverage Components

VRF provides a variety of mechanisms for storing dq information within coverages, including attribute tables, standard data quality tables, and optional user-defined relational tables. The mechanisms employed vary depending primarily upon the types of information being stored, rather than the VRF level at which it will reside.

Data quality coverage attribute tables offer the highest degree of flexibility in storing dq information, since users can design their own table formats and specifically code those components that vary spatially across the data set. Standard dq tables (Table C-52) are particularly useful for data sets where characteristics change radically (with respect to source) from one area to another. In this application, data quality tables are stored as multiple records, with each complete dq table record corresponding to a face. Within this context, it may be particularly appropriate for data producers to implement subsets or supersets of the standard dq table. Additional relational tables are useful in normalizing complex data, a technique that is particularly helpful when addressing thematically based variations in dq information (clause C5.4.4.2.2 of this appendix). Finally, text information is useful for describing phenomena without well-defined spatial extents or for annotating special conditions that do not occur with sufficient frequency to justify creating attribute fields to describe them.

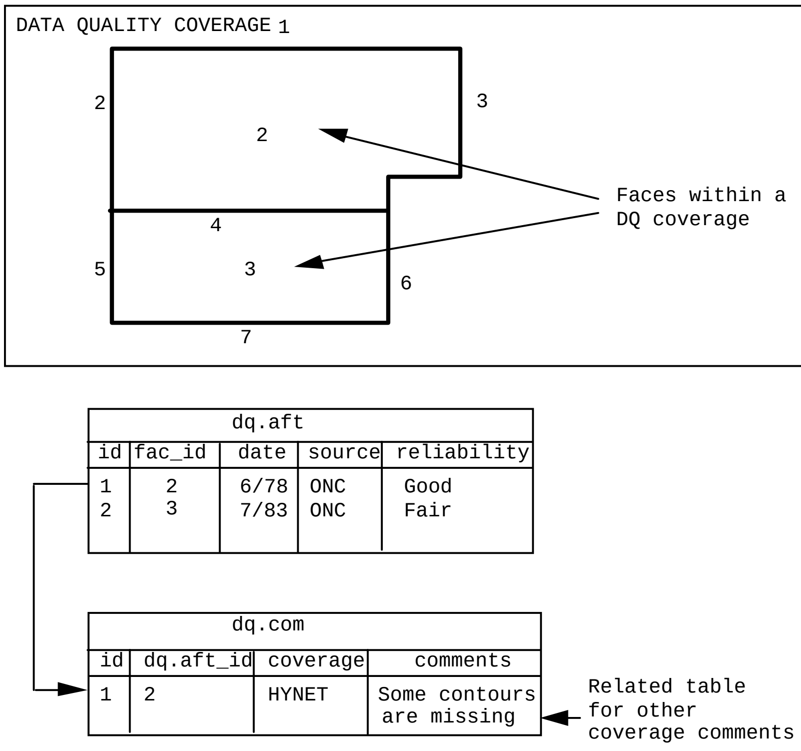
C5.4.4.2 Coverage Examples

The following clauses provide examples of how dq coverages may be designed under a variety of conditions. The user is encouraged to adopt these approaches when appropriate and to modify them when necessary.

Whenever possible, the producer should use coverage level descriptive tables to document the strategy employed in designing and developing dq coverages for a database.

C5.4.4.2.1 Shared Regions and Common Attributes

The most simple dq coverage is one where spatial variation of dq information is shared by all coverages within a library, and dq attributes are well defined. Recognizing that some nonstandard dq information may be associated with certain regions, simple relational tables with fields keyed to coverage identifiers can be constructed (Figure C5-1). Line feature attributes may also be stored in a separate table.


DATA QUALITY COVERAGE 1
2
3
2
Faces within a DQ coverage
4
5
3
6
7
Related table for other coverage comments

dq.aft				
id	fac_id	date	source	reliability
1	2	6/78	ONC	Good
2	3	7/83	ONC	Fair

dq.com			
id	dq.aft_id	coverage	comments
1	2	HYNET	Some contours are missing

Figure C5-1 Data Quality Coverage Design-1

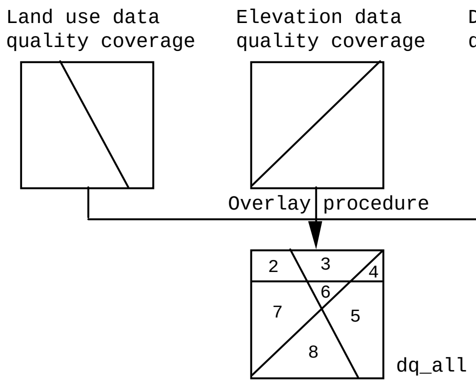
C5.4.4.2.2 Coverage-Specific Information

Clause C5.4.4.2.1 describes the basic constructs for describing characteristics of any data set with common spatial components to the reliability information. However, in some cases, the data producer may wish to describe characteristics within a single coverage, where quality information has spatial extents that vary from coverage to coverage. Figure C5-2 describes a scenario for organizing information under these conditions. The basic approach is to develop a single integrated coverage where the smallest faces are the product of the intersection of the various coverage-based data.

Coverages organized in this manner are relatively easy to maintain, particularly for selective updating where new faces affect more than one primary data coverage. An alternative approach would be to maintain a series of separate coverages.

C5.4.5 Conclusions

VRF provides a number of options for encoding data quality information. The information itself can be encoded at any level within the VRF structure depending upon its basic thematic and spatial characteristics. VRF data producers are encouraged to make use of the data quality table whenever possible. In instances where dq characteristics vary spatially, the use of data quality coverages is strongly recommended.


Land use data quality coverage
Elevation data quality coverage
Drainage data quality coverage
Overlay procedure
2
3
4
6
7
5
8
dq_all

dq_all.aft

id	fac_id	land_ date	land_ source	land_ reliab.	drn_ date	drn_ source	drn_ reliab	elev_ date	elev_ source	elev_ reliab.
1	2	6/78	ONC	Good	6/78	ONC	Good	8/80	JNC	Good
2	3	7/78	ONC	Fair	6/78	ONC	Good	8/80	JNC	Good
3	4	7/78	ONC	Fair	8/81	Photo	High	8/80	JNC	Good
4	5	7/78	ONC	Fair	8/81	Photo	High	9/78	ONC	Good
5	6	7/78	ONC	Fair	6/78	ONC	Good	9/78	ONC	Good
6	7	6/78	ONC	Good	6/78	ONC	Good	9/78	ONC	Good
7	8	6/78	ONC	Good	8/81	Photo	High	9/78	ONC	Good

Figure C5-2 Data Quality Coverage Design-2

[This page intentionally left blank]

Appendix C6 - Spatial Indexing

C6.1 GENERAL

This appendix provides information and discussion concerning spatial indexes in a VRF database.

C6.2 APPLICABLE DOCUMENTS

This clause is not applicable to this appendix.

C6.3 DEFINITIONS

For purposes of this appendix, the definitions of Part 2 clause 4.1 of the main document shall apply.

C6.4 GENERAL INFORMATION

C6.4.1 Introduction

Spatial queries are queries in which the user points at a specific position on a display device containing a graphic representation of the data and asks (for example) "What is this line?" In order to answer the spatial query, any software that conducts a spatial query on a VRF database must search the edge primitive table for an exact match with "this line." Without a spatial index, the software would have to search every vertex of every edge sequentially for the correct response.

The purpose of a spatial index is to improve the speed with which software can retrieve a specific set of row ids from a primitive table. If the database contains spatial indexes, the software, when given a spatial query like, "What are the features within this bounding region?" can quickly retrieve the primitives that match the query. For each primitive (face, edge, entity node, connected node, and text), there can exist a spatial index file: fsi, esi, nsi, csi, or tsi (see clause C.2.4.2).

C6.4.2 Categories of Spatial Decomposition

The spatial index is the second of four categories of spatial decomposition of a VRF database. The other three are the tile directory, the minimum bounding rectangle of the edge and face primitives, and the primitive coordinates. All four categories of spatial decomposition are described below.

C6.4.2.1 Tile Directory

Tiles in an implementation of VRF maintain spatially distributed primitives in separate directories. Thus, software developed for a tiled VRF database can search for data in only the relevant tile after the appropriate tile has been identified.

C6.4.2.2 Spatial Index

The second step in a typical software query is to use the appropriate index file (if one has been created within the database design). It is recommended that spatial index files associated with the primitives be created for every product implementation of VRF. Spatial indexes are discussed further below.

C6.4.2.3 Minimum Bounding Rectangle (MBR)

VRF requires that face and edge primitives have associated bounding rectangle table files—FBR and EBR. These tables allow the rapid retrieval of the primitives' spatial extent and are used by the software after the spatial index routine generates the primitive ids for the current spatial query. The bounding rectangle coordinates are typically used by the software to check the validity of the primitive ids in satisfying the query.

C6.4.2.4 Primitive Coordinates

It is necessary for software to exhaustively check nodes and text primitives for satisfaction of a spatial query, since these primitives do not have associated minimum bounding rectangles. The coordinates of the primitive are thus used to ensure the accurate retrieval of primitive ids output from the spatial index.

C6.4.3 VRF Spatial Index File

The spatial index file internal structure in VRF is based on an adaptive grid binary tree. This method is powerful because it can handle all types of spatial queries (point, line, and area). The input primitives are broken down into a grid-based binary tree. At each cell (of the tree) there is a list of primitive MBRs and a list of the primitive ids that are found at this level of the tree.

The tree is created by storing primitive ids at a cell of the tree. "Bucket size" is the number used to determine when to split a cell and is defined by the product specification. A typical bucket size is eight (8). If the cell fills to the bucket size, then the cell is split into right and left (or top and bottom) children of the cell. The primitive ids are then distributed down into either child depending on the primitive MBRs. Only if a primitive MBR intersects the adjoining child's cell, will the primitive id remain in the parent cell. Since primitives cannot be split between two cells, the number of primitives stored in a cell may exceed the bucket size. The process of splitting cells and distributing primitives based on each primitive's MBR and the bucket size continues until no cell requires splitting or the subdivision process reaches single dimensioned cells (max and min x, y are equal). See C6.4.3.2 below for further detail on the spatial index coordinate system. Product Specifications may limit the number cells in the grid-based binary tree for performance reasons (see C6.4.4.1)

When examined spatially, the spatial index divides a tile into subelements (the cells of the tree); Figure C6-1. Each split results in dividing the parent cell into half. The first split is into right and left halves, with the left half x-axis ranging from 0 to 127 and the right half x-axis ranging from 128 to 255. The next split is into top and bottom halves, with the bottom half y-axis ranging from 0 to 127 and the top half y-axis ranging from 128 to 255. Splits then continue, left/right and top/bottom, until the tree is complete.

The actual format of the spatial index file consists of the following:

a. A header containing the number of primitives, the minimum bounding rectangle of the entire spatial extent of the tree, and the number of cells in the tree. Note that, in tiled coverages, the MBR of the entire spatial extent of the tree will coincide with the tile boundary in topology level three coverages for face spatial indices (fsi) edge spatial indices (esi), and connected node spatial indices (csi), and text spatial indices (tsi). In untiled topology level three coverages, the entire spatial extent of the tree will coincide with the coverage extent. However, there is no reason to force the tile boundary as the spatial extent of the grid-based binary tree for entity node spatial indices (nsi) or for coverages with topology levels less than three. In fact, using the primitives MBR as the spatial extent of the tree rather than the tile boundary provides for a more efficient spatial index. Although the example spatial index described in this Appendix is for a tiled coverage, the concepts would apply to an untiled coverage as well.

b. A bin array of the tree. Each bin contains two items. A beginning location (offset from the end of the BIN Array Record) for the cell's data and the number of primitives in the cell. All intermediate cells are listed, even if empty. The offset for an empty cell equals zero. The last entry in the bin array record is always a populated cell. The final level of the tree is not forced to be balanced by creating empty cells.

c. Data records for each primitive in the tree. There is one record for each primitive in the tree. Each record contains four 1-byte integers defining the MBR for a primitive and that primitive's id.

C6.4.3.1 Tree Navigation

For any cell, the cell from which it was generated is the integer value obtained by dividing by two. Thus, cell 3 points back to cell 1 [INT(3/2)], as does cell 2.

New cells created by splitting are numbered by multiplying the current cell by two and adding one for the second new cell. Thus, cell 2 becomes cells 4 and 5, and cell 3 becomes cells 6 and 7.

C6.4.3.2 Spatial Index Coordinate System

The coordinate system for the spatial index is based upon 1-byte integers, so a primitive's MBR must be converted to the spatial index coordinate system. All coordinates are relative to the southwest corner of the tile or of the MBR of the data extent for the primitive type, and will range from 0 to 255. The minimum X and Y axis coordinate for each cell will be zero (0) or an even integer. The maximum X and Y axis coordinate for each cell will be an odd integer. The only exception to this rule is at level 16, where x-min = x-max and y-min = y-max. Therefore, for any level of cell decomposition a single integer value will fall in only one cell.

There is no single-line boundary between cells. The number of cell dimensions at each decomposition level are shown below.

Table C6-0A Cell Decomposition Levels

Decomposition Level	Bins at Level	Total bins in Index	Subdivision	Cell X, Y Dimension
Level 0	1	1	No partition	256 X 256
Level 1	2	3	Vertically	128 X 256
Level 2	4	7	Horizontally	128 X 128
Level 3	8	15	Vertically	64 X 128
Level 4	16	31	Horizontally	64 X 64
Level 5	32	63	Vertically	32 X 64
Level 6	64	127	Horizontally	32 X 32
Level 7	128	255	Vertically	16 X 32
Level 8	256	511	Horizontally	16 X 16
Level 9	512	1023	Vertically	8 X 16
Level 10	1024	2047	Horizontally	8 X 8
Level 11	2048	4095	Vertically	4 X 8
Level 12	4096	8191	Horizontally	4 X 4
Level 13	8192	16383	Vertically	2 X 4
Level 14	16384	32767	Horizontally	2 X 2
Level 15	32768	65535	Vertically	1 X 2
Level 16	65536	131071	Horizontally	1 X 1*

* This represents a single dimensioned cell with the min x,y equal to the max x,y.

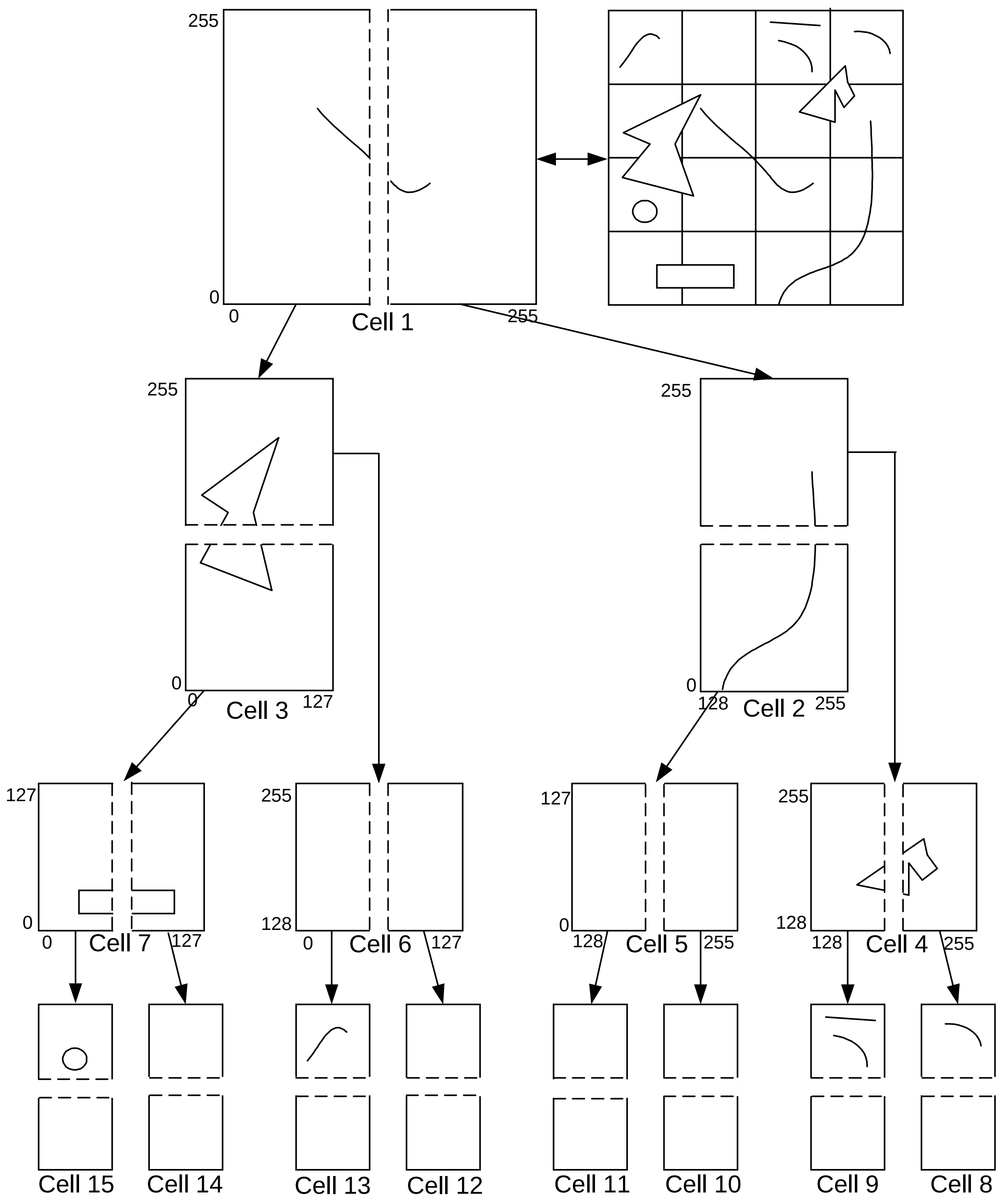


Figure C6-1 Spatial Index Cell Decomposition

C6.4.4 Examples of Spatial Index Creation

Table C6-1 Minimum and Maximum Coordinates for 19 Primitives in a Tile. Universe is primitive number 1.

Primitive ids	**x_1 (deg)**	**y_1 (deg)**	**x_2 (deg)**	**y_2 (deg)**
1	Null	Null	Null	Null
2	-5.00	54.63	-3.57	55.00
3	-5.00	52.00	-2.74	55.00
4	-5.00	54.91	-4.99	54.94
5	-5.00	54.76	-4.99	54.77
6	-4.80	54.06	-4.31	54.42
7	-3.28	54.05	-3.17	54.15
8	-0.71	53.54	0	53.74
9	-0.57	53.68	-0.53	53.69
10	-4.60	53.13	-4.05	53.43
11	-4.71	53.24	-4.56	53.33
12	-4.80	52.75	-4.78	52.77
13	-5.00	50.53	-2.35	51.82
14	-4.71	51.63	-4.68	51.65
15	-4.68	51.16	-4.65	51.20
16	-1.03	50.78	-0.95	50.84
17	-1.59	50.58	-1.08	50.77
18	-1.99	50.69	-1.96	50.70
19	-5.00	50.16	-4.99	50.17

Table C6-1 is a listing of the minimum (x_1, y_1) and maximum (x_2, y_2) coordinates of the MBRs of 19 face primitives. The coordinate values in Table C6-1 are all in degrees. The primitives are all located within a 5-by-5-degree tile that has an MBR of (-5, 50), (0, 55). The MBR coordinates can be converted to the spatial index coordinate system as follows:

For minimum (x_1,y_1) and maximum (x_2,y_2), each new coordinate is obtained from the integer truncation of

255*(Original coordinate - minimum)/(maximum - minimum), which must be in the range 0 to 255.

In our example with MBR (-5,50), (0,55), each new coordinate is given by

y = 255*(Latitude - 50)/(55 - 50) truncated to an integer

and

x = 255*(Longitude + 5)/(0 + 5) truncated to an integer

The results of this conversion are listed in Table C6-2.

Table C6-2 Minimum and Maximum Spatial Index Coordinates

Primitive ids	x_1	y_1	x_2	y_2
2	0	236	72	255
3	0	102	115	255
4	0	250	0	251
5	0	242	0	243
6	10	207	35	225
7	87	206	93	211
8	218	180	255	190
9	225	187	227	188
10	20	159	48	174
11	14	165	22	169
12	9	140	11	141
13	0	27	135	92
14	14	83	16	84
15	16	59	17	61
16	202	39	206	42
17	173	29	199	39
18	153	35	155	35
19	0	8	0	8

Note: Since face 1 (universe face) is a topological artifact (i.e. no outer ring), the MBR in normalized coordinates cannot be calculated. Therefore face 1 is not included in the bin data record portion of the index. Further, no fsi is built for a face table containing only the universe face.

When coordinates are stored as short floating point (data types C and Z), different computer systems can generate slightly different values due to conversion to long floating point for normalization computations. When this occurs very close to a cell break, it can cause the primitive MBR normalized coordinates to fall in an incorrect cell. To alleviate this problem, the following process should be followed for computing normalized coordinate for any short floating point coordinate value: after conversion to a long floating point, truncate the coordinate value following the third (3^{rd}) decimal place before computing the normalized coordinate.

If the MBRs of each primitive are plotted, they appear as shown in Figure C6-2. In Figure C6-3, dividing lines have been added to Figure C6-2 to show that primitive 13 is present in both the left and right halves of the tile, and that primitive 3 is present in both the top and bottom halves of the tile.

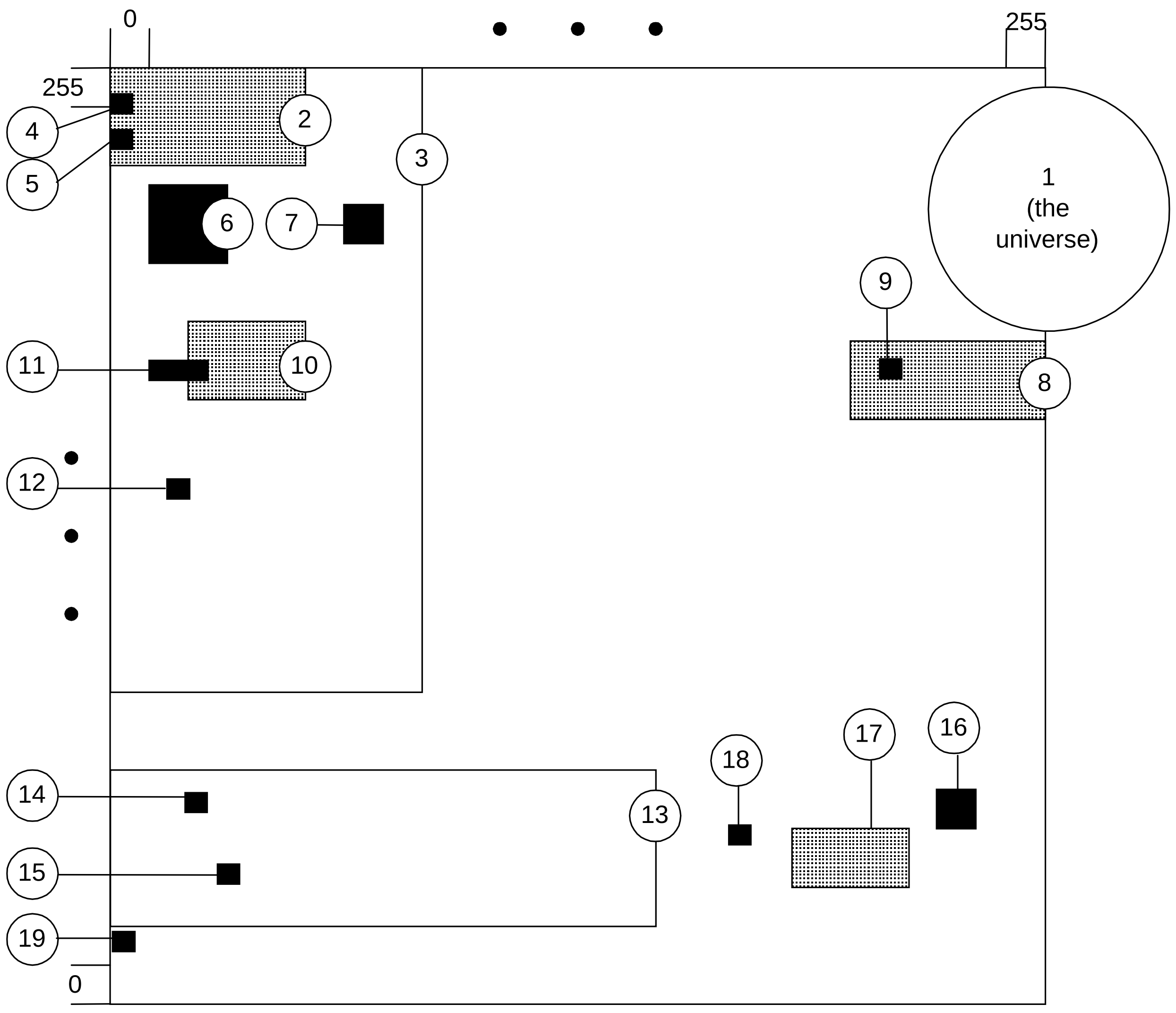


Figure C6-2 Location of MBRs in Tile

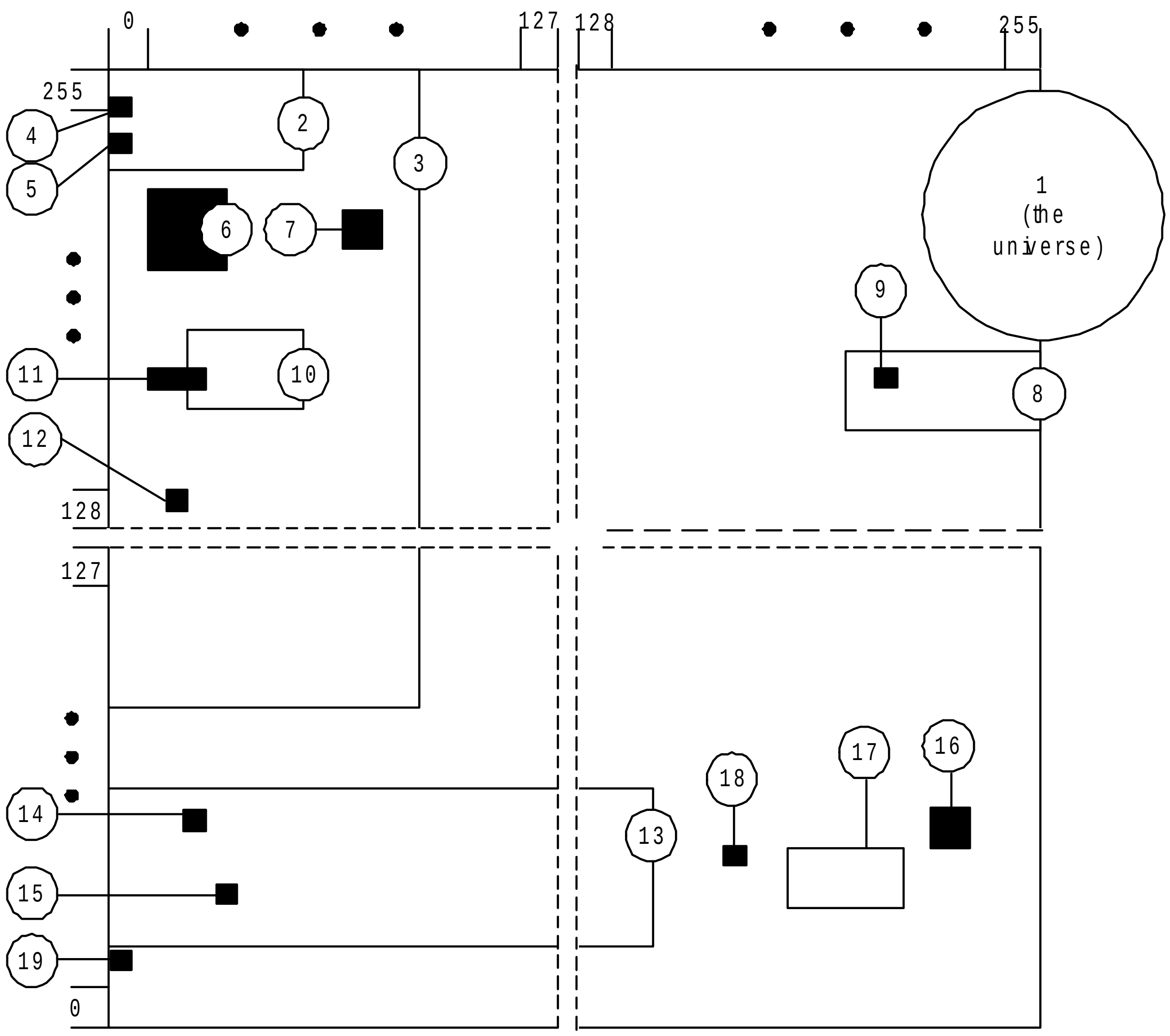


Figure C6-3 Tile Content Divided in Four Quarters

C6.4.4.1 Example of Tree Creation

The bucket size for this example is set at eight. If a parent cell contains nine or more primitives that can be propagated to the next level, the parent splits into two children. The first split of the tile puts primitives 8, 9, 16, 17, and 18 into cell 2 and places primitive 3 into cell 3 (Figure C6-4). Primitive 13 must be held in cell 1 (Figure C6-5) because neither of the split cells contains the entire MBR for primitive 13.

Cell 3 (before the split into cells 6 and 7) contains twelve primitives: 2, 3, 4, 5, 6, 7, 10, 11, 12, 14, 15, and 19.

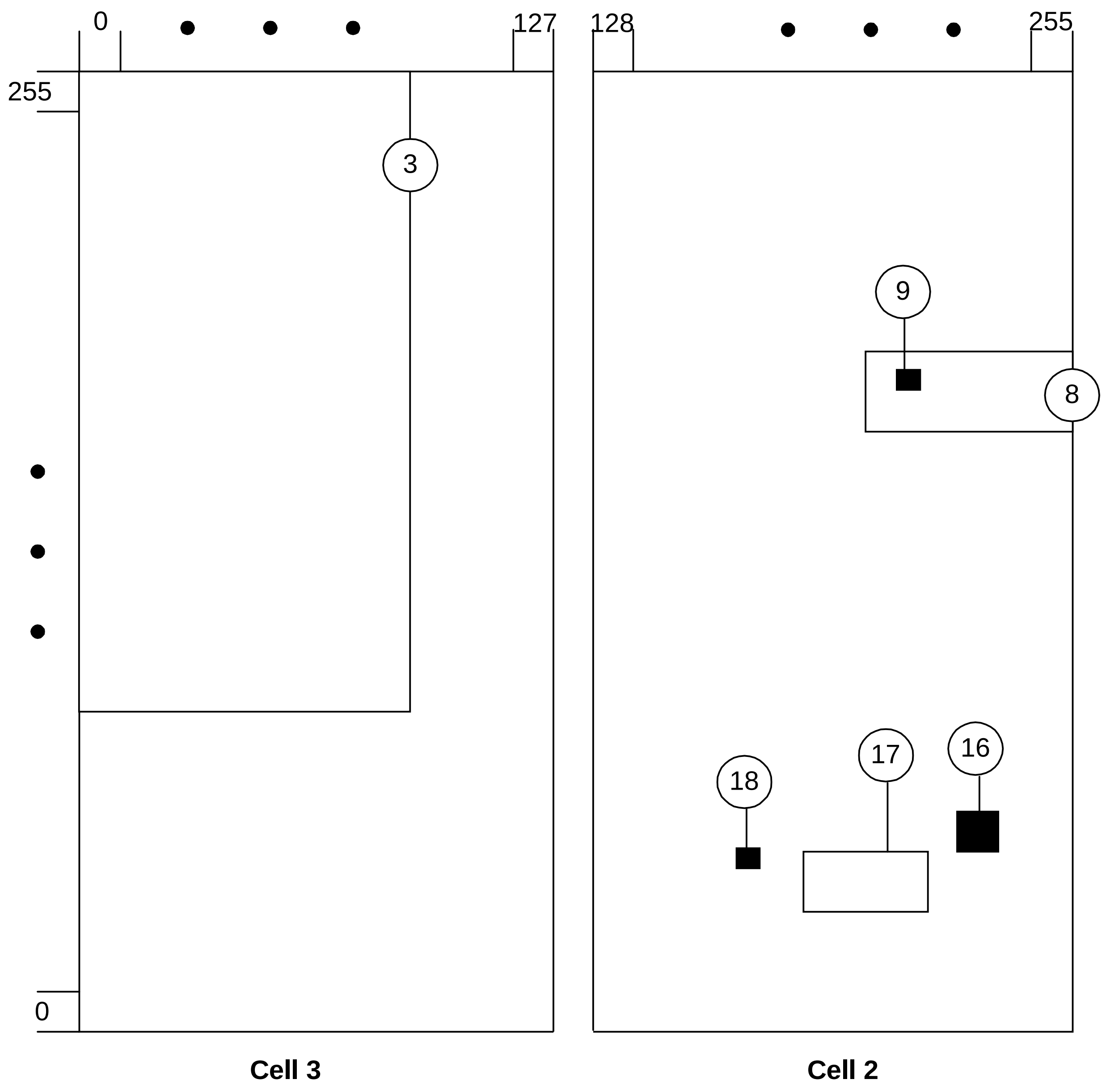


Figure C6-4 First Cell Split

Since this exceeds the defined bucket size, cell 3 is split (Figure C6-4). The MBR of primitive 3 will cross the boundary of cells 6 and 7. Therefore it must remain in cell 3. Eleven primitives remain. Based on each primitive's MBR, the contents of the new cells are: primitives 2, 4, 5, 6, 7, 10, 11, and 12 in cell 6 and primitives 14, 15, and 19 in cell 7. Note that no primitives are allocated to cells 4 and 5, since all five primitives on the right half of the tile could be held in cell 2. The five primitives do not 'over-flow' the defined bucket size. All of the primitives in this example have now been allocated to the tree.

In theory, the process of splitting cells and distributing primitives based on each primitive's MBR and the bucket size continues until no cell requires splitting or the subdivision process reaches the bottom level of the grid-based binary tree (where each cell is 1x1 in normalized coordinates). This implies a grid-based binary tree with 17 levels and 131,071 cells (2^{17} - 1). As noted in C6.4.3, however, Product Specifications may limit the number of cells allowed in the grid-based binary tree for performance reasons.

The spatial index that results for this example is shown in Table C6-3.

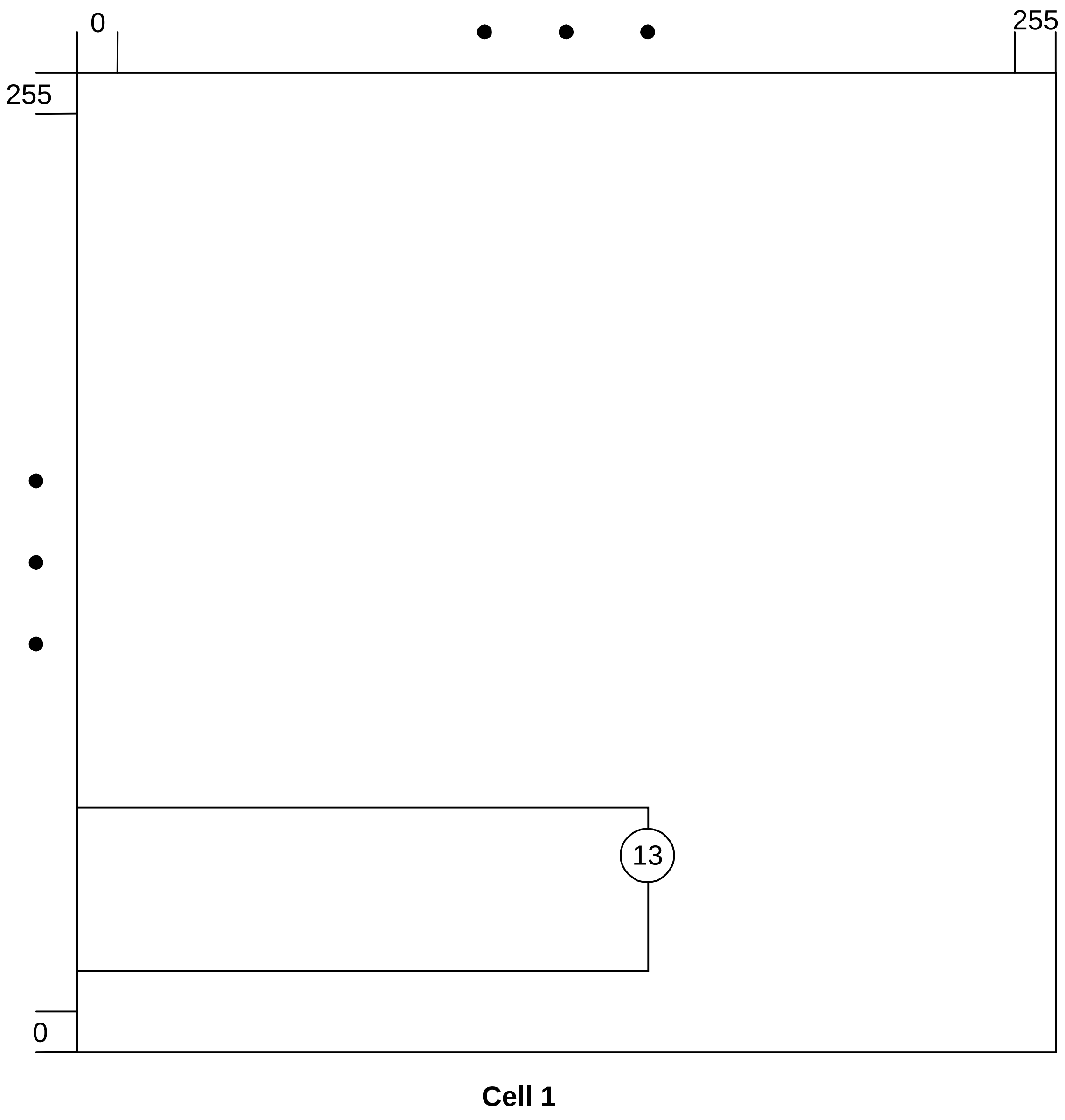


Figure C6-5 Content of Cell 1

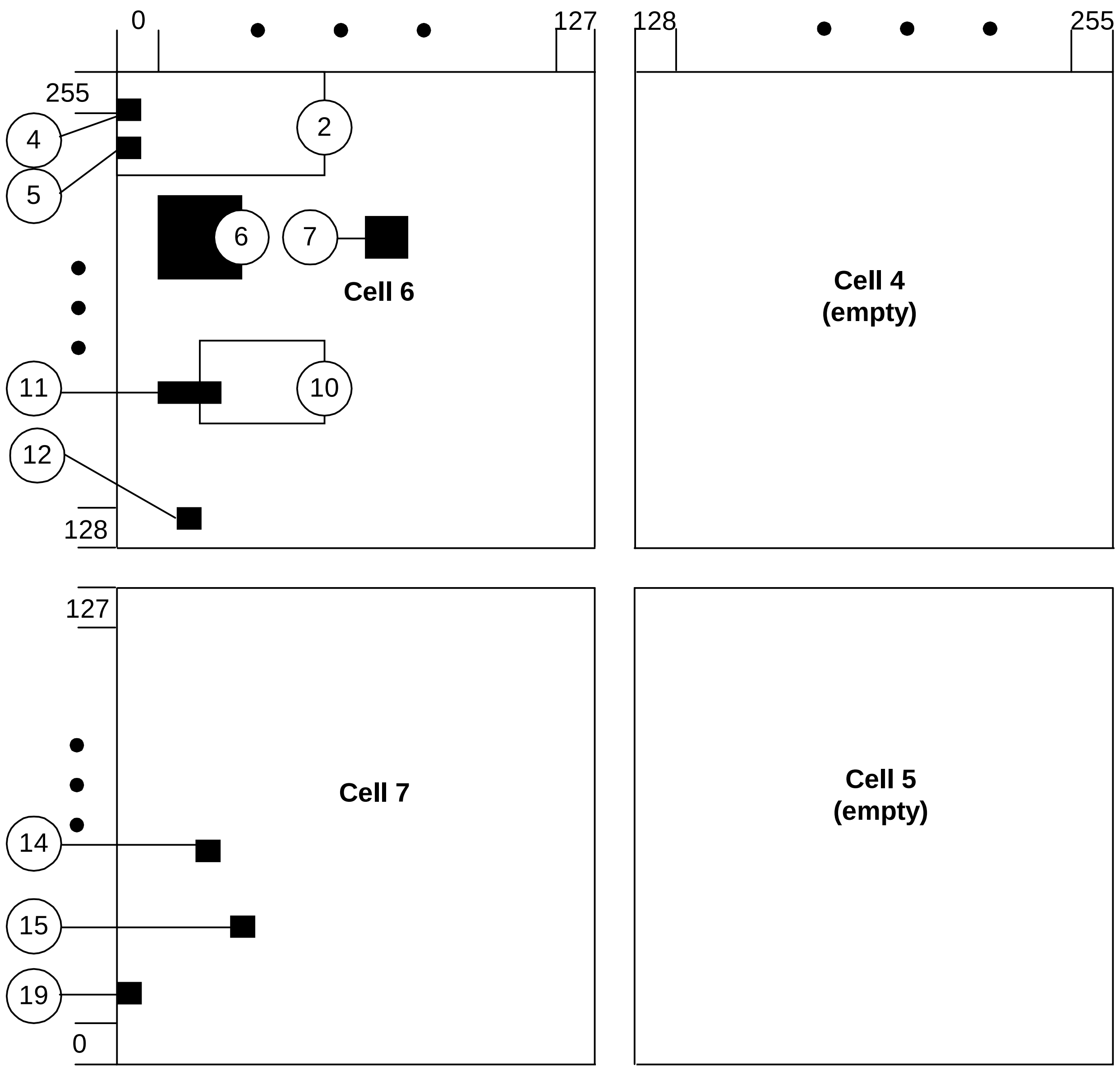


Figure C6-6 Second Cell Split

Table C6-3 Example of Spatial Index

Header:

Number of primitives: 18
MBR: -5.00, 50.00, 0.00, 55.00
Number of cells: 7

Bin Array Record:

cell (information not in actual spatial index)	offset	primitive count
1	0	1
2	8	5
3	48	1
4	0	0
5	0	0
6	56	8
7	120	3

Bin Data Record:

Record Number	Offset Address	x_1	y_1	x_2	y_2	Primitive ids
Cell 1						
1	0	0	26	135	93	13
Cell 2						
2	8	153	35	155	35	18
3	16	173	29	199	39	17
4	24	202	39	206	42	16
5	32	226	187	227	188	9
6	40	218	180	255	190	8
Cell 3						
7	48	0	102	115	255	3
Cell 6						
8	56	87	206	93	211	7
9	64	10	206	35	225	6
10	72	0	242	0	243	5
11	80	0	250	0	252	4
12	88	0	236	72	255	2
13	96	20	159	48	175	10
14	104	14	165	22	169	11
15	112	9	140	11	141	12
Cell 7						
16	120	0	8	0	8	19
17	128	16	59	17	61	15
18	136	14	83	16	84	14

C6.4.5 Spatial Query

A spatial index can support a spatial query in several ways. For node primitives, software may require the point to be within a specified distance of a pixel designated during the query process, or within a box generated during the query process. For an edge primitive, the software may specify that candidate edges are to be within some specified perpendicular distance of the query pixel or have MBRs that intersect a query box. For a face primitive, the software may define the candidate edges as having MBRs that intersect a query box, be fully contained within the query MBR, or be determined by solving the "point-in-polygon" puzzle. In any case, the spatial index forms the starting point for the database search. It works as follows:

a. The user designates a query point (pixel).

b. The software converts the query point into the spatial index coordinate system ((0,0) to (255,255)).

c. The software tests all features within the top level cell, if any, to determine if these features qualify for the query response.

d. The software calculates the next smaller cell containing the query point.

e. The software repeats the test (if necessary) and continues to decrease cell size until no more records exist.

C6.4.6 Spatial Query Using the Sample Tree

a. The user designates a query point and the software determines the normalized coordinates at (192, 32).

b. Beginning at the root of the tree, the software goes to cell 1 and finds face 13.

c. The software checks the MBR of face 13 to determine if it includes the query point.

d. Since face 13 does not include the query point the software calculates the children of cell 1, which are cells 2 and 3. Cell 3 cannot contain the query point, so cell 2 is examined. Faces 8, 9, 16, 17, and 18 are found.

e. The software checks the MBR of these faces to determine if they include the query point.

f. The software reports the query result which is face 17