Open Geospatial Consortium Inc.

Date: 2013-08-01

Reference number of this OGC® project document: OGC 12-128r8

Version: 0.8.0

Category: OGC® Implementation Specification

Editor: Paul Daisey

OpenGIS® GeoPackage Implementation Specification

Copyright notice

Copyright © 2012. Envitia
Copyright © 2012. Luciad
Copyright © 2012 Sigma Bravo.
Copyright © 2012 The Carbon Project.
Copyright © 2012 U.S. Army Geospatial Center.
Copyright © 2012 U.S. National Geospatial Intelligence Agency.

The companies listed above have granted the Open Geospatial Consortium, Inc. (OGC) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. To obtain additional rights of use, visit http://www.opengeospatial.org/legal/.

Warning

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and MAY not be referred to as an OGC Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.
Introduction

Mobile device users who require map/geospatial application services and operate in disconnected or limited network connectivity environments are challenged by limited storage capacity and the lack of open format geospatial data to support these applications. The current situation is that each map/geospatial application requires its own potentially proprietary geospatial data store. These separate application-specific data stores may contain the same geospatial data, wasting the limited storage available, and requiring custom applications for data translation, replication, and synchronization to enable different map/geospatial applications to share the same world view. In addition, many existing geospatial data stores are platform-specific, which means that users with different platforms must translate data to share it.

An open, standards-based, application-independent, platform-independent, portable, interoperable, self-describing, GeoPackage (GPKG) data container, API and manifest are needed to overcome these challenges and to effectively support multiple map/geospatial applications such as fixed product distribution, local data collection, and geospacially enabled analytics. This standard is intended to facilitate widespread adoption and use of GeoPackages by both COTS and open-source software applications on enterprise production platforms as well as mobile hand-held devices, given that mobile hand held devices do not yet have the processing power or battery life to effectively tackle difficult geospatial product production and analysis tasks. An application that accesses a GPKG will make use of the GPKG capabilities it requires; few if any such applications will make use of all GPKG capabilities.

This OpenGIS Implementation Specification defines GeoPackage files for exchange and GeoPackage SQLite Extensions for direct use of vector geospatial features and/or tile matrix sets of earth images and raster maps at various scales. Direct use means the ability to access and update data in a “native” format without intermediate format translations in an environment (e.g. through an API) that guarantees data model and data set integrity and identical access and update results in response to identical requests from different client applications.

A GeoPackage file is a platform-independent SQLite database file that contains GeoPackage data and metadata tables shown in Figure 1, with specified definitions, integrity assertions, format limitations and content constraints.
A GeoPackage file MAY be “empty” (contain user data table(s) for vector features and/or tile matrix pyramids with no row record content) or contain one or many vector feature type records and/or one or many tile matrix pyramid tile images. GeoPackage file metadata can describe GeoPackage data contents and identify external data synchronization sources and targets. A GeoPackage file MAY contain spatial indexes on feature geometries and SQL triggers to maintain indexes and enforce content constraints.

A GeoPackage SQLite Extension that MAY provide a minimal set of SQL API runtime functions to support spatial indexes and SQL triggers linked to a SQLite library with specified configuration requirements to provide SQL API access to a GeoPackage file. A GeoPackage is a GeoPackage file used with a GeoPackage SQLite Extension. A GeoPackage MAY have extensions that are registered in the gpkg_extensions table.

This specification does not address the issues listed in the B.6 Potential Future Work clause in Annex B, which MAY be addressed in a subsequent version of this specification or by other specifications.
Attention is drawn to the possibility that some of the elements of this document MAY be the subject of patent rights. Open Geospatial Consortium Inc. SHALL not be held responsible for identifying any or all such patent rights. However, to date, no such rights have been claimed or identified.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they MAY be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.
OpenGIS® GeoPackage Implementation Specification

Base

The required capabilities specified in this clause serve as the base for optional options and registered extensions capabilities specified in clauses 1 and 2.

1.1 Core

The mandatory core capabilities defined in sub clauses and requirement statements of this clause SHALL be implemented by every GeoPackage.

1.1.1. SQLite Container

A self-contained, single-file, cross-platform, serverless, transactional, open source RDBMS container is desired to simplify production, distribution and use of GeoPackages and guarantee the integrity of the data they contain. “Self-contained” means that container software requires very minimal support from external libraries or from the operating system. “Single-file” means that a container not currently opened by any software application consists of a single file in a file system supported by a computing platform operating system. “Cross-platform” means that a container file MAY be created and loaded with data on one computing platform, and used and updated on another, even if they use different operating systems, file systems, and byte order (endian) conventions. “Serverless” means that the RDBMS container is implemented without any intermediary server process, and accessed directly by application software. “Transactional” means that RDBMS transactions guarantee that all changes to data in the container are Atomic, Consistent, Isolated, and Durable (ACID) despite program crashes, operating system crashes, and power failures.

1.1.1.1. Data

1.1.1.1.1. File Format

Req 1: The GeoPackage file format SHALL be a SQLite [9] database version 3 format [10][11] file, the first 16 bytes of which contain “SQLite format 3” in ASCII.\(^1\)

It is RECOMMENDED that data providers whose data is intended for use in multiple applications put the value “GPKG” in the SQLite database header application id field. However, applications that put application-specific data in a GeoPackage MAY put their own application acronym in the application id field, so this is NOT a GeoPackage requirement.

The maximum size of a GeoPackage file is about 140TB. In practice a lower size limit MAY be imposed by the filesystem to which the file is written. Many mobile devices require external memory cards to be formatted using the FAT32 file system which imposes a maximum size limit of 4GB.

---

\(^1\) SQLite version 4 [B49], which will be an alternative to version 3, not a replacement thereof, was not available when this specification was written. See Future Work clause in Annex B.

\(^2\) SQLite is in the public domain (see http://www.sqlite.org/copyright.html)
1.1.1.1.2. File Extension Name

**Req 2:** GeoPackage files SHALL have the file extension name “.gpkg”.

1.1.1.1.3. Integrity Check

**Req 3:** The SQLite PRAGMA integrity_check SQL command SHALL return “ok” for a GeoPackage file.

1.1.1.2. API

1.1.1.2.1. Structured Query Language (SQL)

**Req 4:** A GeoPackage SQLite Extension SHALL provide SQL access to GeoPackage file contents via SQLite sqlite3 [12] software APIs.¹

1.1.1.2.2. Every GPKG SQLite Configuration

The SQLite [12] library has many compile time and run time options that MAY be used to configure SQLite for different uses. A GeoPackage application can use the compile options and other SQLite pragmas to get the effective compile and run time option settings and compare them to those required for a particular GeoPackage.

**Req 5:** Every GeoPackage SQLite Extension SHALL have the SQLite library compile and run time options specified in clause 1.1.1.2.2 table 1.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Option</th>
<th>Shall / Not</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_AUTOINCREMENT</td>
<td>Not</td>
<td>A number of tables in GeoPackage are specified to have autoincrement columns.</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_DATETIME_FUNCS</td>
<td>Not</td>
<td>gpkg_contents table last_change column requires timestamps</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_DEPRECATED</td>
<td>Shall</td>
<td>Omit PRAGMAs marked “Do not use this pragma!” in SQLite documentation.</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_FLOATING_POINT</td>
<td>Not</td>
<td>min/max x/y columns in gpkg_contents table are type REAL</td>
</tr>
</tbody>
</table>

¹ New applications should use the latest available SQLite version software [12]
1.1.2. Spatial Reference Systems

1.1.2.1. Data

1.1.2.1.1. Table Definition

**Req 6:** A GeoPackage file SHALL include a gpkg_spatial_ref_sys table or updateable view per clause 1.1.2.1.1, table 2 and Table 19.

A table or updateable view named gpkg_spatial_ref_sys is the first component of the standard SQL schema for simple features described in clause 2.1.1 below. The coordinate reference system definitions it contains are referenced by the GeoPackage gpkg_contents and gpkg_geometry_columns tables to relate the vector and tile data in user tables to locations on the earth.

The gpkg_spatial_ref_sys table includes at a minimum the columns specified in SQL/MM (ISO 13249-3) [16] and shown in table 2 below containing data that defines spatial reference systems. This table or view MAY include additional columns to meet the requirements of implementation software or other specifications. Views of this table or view MAY be used to provide compatibility with the SQL/MM [16] (Table 20) and OGC Simple Features SQL [14][15] (Table 21) specifications.

**Table 2 Spatial Ref Sys Table or View Definition**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>srs_name</td>
<td>text</td>
<td>Human readable name of this SRS</td>
<td></td>
</tr>
<tr>
<td>srs_id</td>
<td>integer</td>
<td>Unique identifier for each Spatial Reference System within a GeoPackage file</td>
<td>PK</td>
</tr>
<tr>
<td>organization</td>
<td>text</td>
<td>Case-insensitive name of the defining organization e.g. EPSG or epsg</td>
<td></td>
</tr>
<tr>
<td>organization_coordsys_id</td>
<td>integer</td>
<td>Numeric ID of the Spatial Reference System assigned by the organization</td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>text</td>
<td>Well-known Text Representation of the Spatial Reference System</td>
<td></td>
</tr>
</tbody>
</table>
See Annex C Table Definition SQL (Normative) C.1 gpkg_spatial_ref_sys.

1.1.2.1.2.  Table Data Values

Req 7: The gpkg_spatial_ref_sys table or updateable view in a GeoPackage SHALL contain a record for organization EPSG or epsg and organization_coordsys_id 4326 [17][18] for WGS-84 [19], a record with an srs_id of -1, an organization of “NONE”, an organization_coordsys_id of -1, and definition “undefined” for undefined Cartesian coordinate reference systems, and a record with an srs_id of 0, an organization of “NONE”, an organization_coordsys_id of 0, and definition “undefined” for undefined geographic coordinate reference systems.

Req 8: The spatial_ref_sys table or updateable view in a GeoPackage file SHALL contain records to define all spatial reference systems used by features and tiles in a GeoPackage.

1.1.3.  Contents

1.1.3.1.  Data

1.1.3.1.1.  Table Definition

Req 9: A GeoPackage file SHALL include a gpkg_contents table or updateable view per clause 1.1.3.1.1, table 3 and table 22.

The purpose of the gpkg_contents table is to provide identifying and descriptive information that an application can display to a user in a menu of geospatial data that is available for access and/or update.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
<th>Null</th>
<th>Default</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>text</td>
<td>The name of the tiles, or feature table</td>
<td>no</td>
<td></td>
<td>PK</td>
</tr>
<tr>
<td>data_type</td>
<td>text</td>
<td>Type of data stored in the table: “features” per clause 2.1.2.1.1, “tiles” per clause 2.2.2.1.1, or an implementer-defined value for other data tables per clause 2.5.</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>identifier</td>
<td>text</td>
<td>A human-readable identifier (e.g. short name) for the table_name content</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>text</td>
<td>A human-readable description for the table_name content</td>
<td>yes</td>
<td>***</td>
<td></td>
</tr>
<tr>
<td>last_change</td>
<td>text</td>
<td>timestamp value in ISO 8601 format as defined by the strftime function '%Y-%m-%dT%H:%M:%fZ' format string</td>
<td>no</td>
<td>strftime('%Y-%m-%dT%H:%M:%fZ', CURRENT_TIMESTAMP)</td>
<td></td>
</tr>
</tbody>
</table>
The gpkg_contents table is intended to provide a list of all geospatial contents in the GeoPackage. The data_type specifies the type of content. The bounding box (min_x, min_y, max_x, max_y) provides an informative bounding box (not necessarily minimum bounding box) of the content. If the srs_id column value references a geographic coordinate reference system (CRS), then the min/max x/y values are in decimal degrees; otherwise, the srs_id references a projected CRS and the min/max x/y values are in the units specified by that CRS.

See Annex C Table Definition SQL (Normative) C.2 gpkg_contents.

1.1.3.1.2. Table Data Values

Req 10: The table_name column value in a gpkg_contents table row SHALL contain the name of a SQLite table or view.

Req 11: Values of the gpkg_contents table last_change column SHALL be in ISO 8601 [41] format containing a complete date plus UTC hours, minutes, seconds and a decimal fraction of a second, with a ‘Z’ (‘zulu’) suffix indicating UTC.¹

Req 12: Values of the gpkg_contents table srs_id column SHALL reference values in the spatial_ref_sys table srs_id column.

Options

The optional capabilities specified in this clause depend on the required capabilities specified in 1 Base above.

Req 13: A Valid GeoPackage SHALL contain features per clause 2.1 and/or tiles per clause 2.2 and row(s) in the gpkg_contents table with data_type column values of “features” and/or “tiles” describing the user data tables.

¹ The following statement selects an ISO 8601 timestamp value using the SQLite strftime function:

SELECT (strftime('%Y-%m-%dT%H:%M:%fZ','now')).
2.1. Features

The optional capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file.

2.1.1. Simple Features SQL Introduction

Vector feature data represents geolocated entities including conceptual ones such as districts, real world objects such as roads and rivers, and observations thereof. International specifications [14][15][16] have standardized practices for the storage, access and use of vector geospatial features and geometries via SQL in relational databases. The first component of the SQL schema for vector features in a GeoPackage is the gpkg.spatial_ref_sys table defined in clause 1.1.2 above. Other components are defined below.

In a GeoPackage file, “simple” features are geolocated using a linear geometry subset of the SQL/MM (ISO 13249-3) [16] geometry model shown in figure 2 below.

![Core Geometry Model](image)

The instantiable (not abstract) geometry types defined in this Standard are restricted to 0, 1 and 2-dimensional geometric objects that exist in 2, 3 or 4-dimensional coordinate space (R2, R3 or R4). Geometry values in R2 have points with coordinate values for x and y. Geometry values in R3 have points with coordinate values for x, y and z or for x, y and m. Geometry values in R4 have points with coordinate values for x, y, z and m. The interpretation of the coordinates is subject to the coordinate reference systems associated to the point. All coordinates within a geometry object should be in the same coordinate reference systems.
Geometries may include z coordinate values. The z coordinate value traditionally represents the third dimension (i.e. 3D). In a Geographic Information System (GIS) this may be height above or below sea level. For example: A map might have a point identifying the position of a mountain peak by its location on the earth, with the x and y coordinate values, and the height of the mountain, with the z coordinate value.

Geometries may include m coordinate values. The m coordinate value allows the application environment to associate some measure with the point values. For example: A stream network may be modeled as multilinestring value with the m coordinate values measuring the distance from the mouth of stream.

All geometry types described in this standard are defined so that instances of Geometry are topologically closed, i.e. all represented geometries include their boundary as point sets. This does not affect their representation, and open version of the same classes may be used in other circumstances, such as topological representations.

A brief description of each geometry type is provided below. A more detailed description can be found in ISO 13249-3[16].

- Geometry: the root of the geometry type hierarchy.
- Point: a single location in space. Each point has an X and Y coordinate. A point may optionally also have a Z and/or an M value.
- Curve: the base type for all 1-dimensional geometry types. A 1-dimensional geometry is a geometry that has a length, but no area. A curve is considered simple if it does not intersect itself (except at the start and end point). A curve is considered closed its start and end point are coincident. A simple, closed curve is called a ring.
- LineString: A Curve that connects two or points in space.
- Surface: the base type for all 2-dimensional geometry types. A 2-dimensional geometry is a geometry that has an area.
- CurvePolygon: A planar surface defined by an exterior ring and zero or more interior ring. Each ring is defined by a Curve instance.
- Polygon: A restricted form of CurvePolygon where each ring is defined as a simple, closed LineString.
- GeometryCollection: A collection of zero or more Geometry instances.
- MultiSurface: A restricted form of GeometryCollection where each Geometry in the collection must be of type Surface.
- MultiPolygon: A restricted form of MultiSurface where each Surface in the collection must be of type Polygon.
- MultiCurve: A restricted form of GeometryCollection where each Geometry in the collection must be of type Curve.
- MultiLineString?: A restricted form of MultiCurve where each Curve in the collection must be of type LineString.
- MultiPoint: A restricted form of GeometryCollection where each Geometry in the collection must be of type Point.
2.1.2. Contents

2.1.2.1. Data

2.1.2.1.1. Contents Table – Features Row

**Req 14:** `gpkg_contents SHALL contain a row with a data_type column value of “features” for each vector features user data table.**

2.1.3. Geometry Encoding

2.1.3.1. Data

2.1.3.1.1. BLOB Format

**Req 15:** A GeoPackage file **SHALL store feature table geometries with or without optional elevation (Z) and/or measure (M) values in SQL BLOBs using the GeoPackageBinary format specified in table 4 and clause 2.1.3.1.1.**

<table>
<thead>
<tr>
<th>Table 4 GeoPackage SQL Geometry Binary Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoPackageBinary {</td>
</tr>
<tr>
<td>byte[2] magic = 0x4750; // 'GP'</td>
</tr>
<tr>
<td>byte version;</td>
</tr>
<tr>
<td>byte flags;</td>
</tr>
<tr>
<td>int32 srs_id;</td>
</tr>
<tr>
<td>double[] envelope;</td>
</tr>
<tr>
<td>WKBGeometry geometry;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

flags layout:

<table>
<thead>
<tr>
<th>bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>use</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Y</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>B</td>
</tr>
</tbody>
</table>

use:

- **R:** reserved for future use; set to 0
- **Y:** empty geometry flag
  - 0: non-empty geometry
  - 1: empty geometry

**E:** **envelope contents indicator code** (3-bit unsigned integer)

<table>
<thead>
<tr>
<th>code</th>
<th>description</th>
<th>envelope</th>
</tr>
</thead>
</table>

---

1 OGC WKB simple feature geometry types specified in [13] are a subset of the ISO WKB geometry types specified in [16]
<table>
<thead>
<tr>
<th>value</th>
<th>length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>5-7</td>
<td>unknown</td>
</tr>
</tbody>
</table>

**B: byte order for header values** (1-bit Boolean)
- 0 = Big Endian (most significant byte first)
- 1 = Little Endian (least significant byte first)

Well-Known Binary as defined in ISO 13249-3 [16] does not provide a standardized encoding for an empty point set (i.e., 'Point Empty' in Well-Known Text). In GeoPackage files these points SHALL be encoded as a Point where each coordinate value is set to an IEEE-754 quiet NaN value. GeoPackage files SHALL use big endian 0x7ff8000000000000 or little endian 0x000000000000f87f as the binary encoding of the NaN values.

When the WKBGeometry in a GeoPackageBinary is empty, either the envelope contents indicator code SHALL be 0 indicating no envelope, or the envelope SHALL have its values set to NaN as defined for an empty point.

### 2.1.3.2. API

#### 2.1.3.2.1. Minimal Runtime SQL Functions

In contrast to functions in application code or a runtime library, triggers are part of the SQLite database file. When an application writes to a GeoPackage file that it did not create itself then there is the possibility that it will invoke a trigger that calls a function that the application’s runtime library does not provide. To avoid this interoperability problem, a small set of functions on the GeoPackageBinary geometry specified in clause 2.1.3.1.1 are defined in Annex D. Every implementation can be sure that triggers that only use these functions in addition to those provided by SQLite will work as intended across implementations. 12

**Req 16:** A GeoPackage SQLite Extension MAY provide SQL function support for triggers in GeoPackage file. One that does so SHALL provide the minimal runtime SQL functions listed in Annex D table 36.

---

1 Functions other than the minimal runtime SQL functions required by triggers in a GeoPackage file SHOULD be documented in the gpkg_extensions table and provided by a GeoPackage SQLite Extension.

2 SQL functions on geometries in addition to those defined in this clause should conform to the SF/SQL [14][15] and SQL/MM [16] specifications cited in clause 2.1.1 above.
2.1.4. Geometry Types

2.1.4.1. Data

2.1.4.1.1. Core Types

Req 17: A GeoPackage file SHALL store feature table geometries with the basic simple feature geometry types (Geometry, Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeomCollection) in Annex G table 46 in the GeoPackageBinary geometry encoding format.

2.1.5. Geometry Columns

2.1.5.1. Data

2.1.5.1.1. Table Definition

Req 18: A GeoPackage file with a gpkg_contents table row with a “features” data_type SHALL contain a gpkg_geometry_columns table or updateable view per clause 2.1.5.1.1, table 5 and Table 24.

The second component of the SQL schema for vector features in a GeoPackage is a gpkg_geometry_columns table that identifies the geometry columns in tables that contain user data representing features. This table or updateable view SHALL contain one row record for each geometry column in each vector feature data table (clause 2.1.6) in a GeoPackage.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>text</td>
<td>Name of the table containing the geometry column</td>
<td>PK, FK</td>
</tr>
<tr>
<td>column_name</td>
<td>text</td>
<td>Name of a column in the feature table that is a Geometry Column</td>
<td>PK</td>
</tr>
<tr>
<td>geometry_type_name</td>
<td>text</td>
<td>Name from table 46 or table 47 in Annex G</td>
<td></td>
</tr>
<tr>
<td>srs_id</td>
<td>integer</td>
<td>Spatial Reference System ID: gpkg.spatial_ref_sys.srs_id</td>
<td>FK</td>
</tr>
</tbody>
</table>
| z           | integer | 0: z values prohibited  
1: z values mandatory  
2: z values optional | |
| m           | integer | 0: m values prohibited  
1: m values mandatory  
2: m values optional | |

The FK on gpkg_geometry_columns.srs_id references the PK on gpkg.spatial_ref_sys.srs_id to ensure that geometry columns are only defined in feature tables for defined spatial reference systems.
The gpkg_geometry_columns table or view MAY include additional columns to meet the requirements of implementation software or other specifications. Views of this table or view MAY be used to provide compatibility with the SQL/MM [16] (Table 25) and OGC Simple Features SQL [14][15] (Table 26) specifications.

See clause C.4 gpkg_geometry_columns.

2.1.5.1.2. Table Data Values

Req 19: Values of the gpkg_geometry_columns table table_name column SHALL reference values in the gpkg_contents table_name column.

Req 20: The column_name column value in a gpkg_geometry_columns table row SHALL be the name of a column in the table specified by the table_name column value for that row.

Req 21: The geometry_type_name value in a gpkg_geometry_columns table row SHALL be one of the geometry type names specified in Annex G.

Req 22: The srs_id value in a gpkg_geometry_columns table row SHALL be an srs_id column value from the gpkg_spatial_ref_sys table.

Req 23: The z value in a gpkg_geometry_columns table row SHALL be one of 0, 1, or 2.

Req 24: The m value in a gpkg_geometry_columns table row SHALL be one of 0, 1, or 2.

2.1.6. Vector Feature User Data Tables

2.1.6.1. Data

2.1.6.1.1. Table Definition

The third component of the SQL schema for vector features in a GeoPackage described in clause 2.1.1 above are tables that contain user data representing features. Feature attributes are columns in a feature table, including geometries.1 Features are rows in a feature table.2

Req 25: A GeoPackage file MAY contain tables or updateable views containing vector features. Every such feature table or view in a GeoPackage file SHALL have a primary key defined on one integer column per table 6 and table 27.

The integer primary key of a feature table allows features to be linked to row level metadata records in the gpkg_metadata table by rowid values in the gpkg_metadata_reference table as described in clause 2.4.3 below.

1 A feature type MAY be defined to have 0..n geometry and/or raster attributes, so the corresponding feature table MAY contain from 0..n geometry and/or raster columns.

2 A GeoPackage is not required to contain any feature data tables. Feature data tables in a GeoPackage MAY be empty.
Req 26: A feature table SHALL have only one geometry column.

Feature data models from non-GeoPackage implementations that have multiple geometry columns per feature table MAY be transformed into GeoPackage implementations with a separate feature table for each geometry type whose rows have matching integer primary key values that allow them to be joined in a view with the same column definitions as the non-GeoPackage feature data model with multiple geometry columns.

Table 6 EXAMPLE: Sample Feature Table or View Definition

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
<th>Null</th>
<th>Default</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>Autoincrement primary key</td>
<td>no</td>
<td></td>
<td>PK</td>
</tr>
<tr>
<td>geometry_one</td>
<td>BLOB</td>
<td>GeoPackage Geometry</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text_attribute</td>
<td>text</td>
<td>Text attribute of feature</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>real_attribute</td>
<td>real</td>
<td>Real attribute of feature</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>numeric_attribute</td>
<td>numeric</td>
<td>Numeric attribute of feature</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>raster_or_photo</td>
<td>BLOB</td>
<td>Photograph of the area</td>
<td>no</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Annex C: Table Definition SQL clause C.5 sample_feature_table

2.1.6.1.2. Table Data Values

A feature geometry is stored in a geometry column specified by the geometry_column value for the feature table in the gpkg_geometries table defined in clause 2.1.5 above.

The geometry type of a feature geometry column specified in the gpkg_geometries table geometry_type_name column is a name from Annex G.

Req 27: Feature table geometry columns SHALL contain geometries of the type or assignable for the type specified for the column by the gpkg_geometries table geometry_type_name column value.

Geometry subtypes are assignable as defined in Annex G and shown in part in Figure 2 – Core Geometry Model. For example, if the geometry_type_name value in the gpkg_geometries table is for a geometry type like POINT that has no subtypes, then the feature table geometry column MAY only contain geometries of that type. If the geometry_type_name value in the gpkg_geometries table is for a geometry type like GEOMCOLLECTION that has subtypes, then the feature table geometry column MAY only contain geometries of that type or any of its direct or indirect subtypes. If the geometry_type_name is GEOMETRY (the root of the geometry type hierarchy) then the feature table geometry column MAY contain geometries of any geometry type.

---

1 GeoPackage applications MAY use SQL triggers or tests in application code to meet Req 27:
The presence or absence of optional elevation (Z) and/or measure (M) values in a geometry does not change its type or assignability.

The spatial reference system type of a feature geometry column specified by a gpkg_geometry_columns table srs_id column value is a code from the gpkg_spatial_ref_sys table srs_id column.

**Req 28**: Feature table geometry columns SHALL contain geometries with the srs_id specified for the column by the gpkg_geometry_columns table srs_id column value.

### 2.2. Tiles

The optional capabilities defined in sub-clauses and requirement statements of this clause MAY be implemented by any GeoPackage file.

#### 2.2.1. Tile Matrix Introduction

There are a wide variety of commercial and open source conventions for storing, indexing, accessing and describing tiles in tile matrix pyramids. Unfortunately, no applicable existing consensus, national or international specifications have standardized practices in this domain. In addition, various image file formats have different representational capabilities, and include different self-descriptive metadata.

The tile store data / metadata model and conventions described below support direct use of tiles in a GeoPackage in two ways. First, they specify how existing applications MAY create SQL Views of the data /metadata model on top of existing application tables that that follow different interface conventions. Second, they include and expose enough metadata information at both the dataset and record levels to allow applications that use GeoPackage data to discover its characteristics without having to parse all of the stored images.

Applications that store GeoPackage tile data, which are presumed to have this information available, SHALL store sufficient metadata to enable its intended use.

The GeoPackage tile store data model MAY be implemented directly as SQL tables in a SQLite database for maximum performance, or as SQL views on top of tables in an existing SQLite tile store for maximum adaptability and loose coupling to enable widespread implementation.

A GeoPackage CAN store multiple raster and tile pyramid data sets in different tables or views in the same container.¹

The tables or views that implement the GeoPackage tile store data / metadata model are described and discussed individually in the following subsections.

---

¹ Images of multiple MIME types MAY be stored in given table. For example, in a tiles table, image/png format tiles without compression could be used for transparency where there is no data on the tile edges, and image/jpeg format tiles with compression could be used for storage efficiency where there is image data for all pixels. Images of multiple bit depths of the same MIME type MAY also be stored in a given table, for example image/png tiles in both 8 and 24 bit depths.
2.2.2. Contents

2.2.2.1 Data

2.2.2.1.1 Contents Table – Tiles Row

Req 29: A gpkg_contents table or updateable view SHALL contain a row with a data_type column value of “tiles” for each tile matrix user data table.

2.2.3. Zoom Levels

2.2.3.1 Data

2.2.3.1.1 Zoom Times Two

Req 30: In a GeoPackage file that contains a tile matrix user data table that contains tile data, by default¹, zoom level pixel sizes for that table SHALL vary by powers of 2 between adjacent zoom levels in the tile matrix metadata table.

2.2.4. Tile Encoding PNG

2.2.4.1 Data

2.2.4.1.1 MIME Type PNG

Req 31: In a GeoPackage file that contains a tile matrix user data table that contains tile data that is not MIME type image/jpeg[21][22][23], by default SHALL store that tile data in MIME type image/png [24][25].²

2.2.5. Tile Encoding JPEG

2.2.5.1 Data

2.2.5.1.1 MIME Type JPEG

Req 32: In a GeoPackage file that contains a tile matrix user data table that contains tile data that is not MIME type image/png [24][25], by default SHALL store that tile data in MIME type image/jpeg[21][22][23],³

2.2.6. Tile Matrix Metadata

2.2.6.1 Data

2.2.6.1.1 Table Definition

Req 33: A GeoPackage that contains a tile matrix user data table SHALL contain a gpkg_tile_matrix_metadata table per clause 2.2.6.1.1, Table 7 and Table 28.

---

¹ See clause 3.2.1.1.1 for use of other zoom levels as a registered extensions.

² See Clauses 3.2.2, 3.2.3, 3.2.4 and 3.2.5 regarding use of alternative tile MIME types as registered extensions.

³ See Clauses 3.2.2, 3.2.3, 3.2.4 and 3.2.5 regarding use of alternative tile MIME types as registered extensions.
Table 7 Tile Matrix Metadata Table or View Definition

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
<th>Null</th>
<th>Default</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>text</td>
<td>[RasterLayerName]_tiles</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zoom_level</td>
<td>integer</td>
<td>0 &lt;= zoom_level &lt;= max_level for t_table_name</td>
<td>no</td>
<td>0</td>
<td>PK, FK</td>
</tr>
<tr>
<td>matrix_width</td>
<td>integer</td>
<td>Number of columns (&gt;= 1) in tile matrix at this zoom level</td>
<td>no</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>matrix_height</td>
<td>integer</td>
<td>Number of rows (&gt;= 1) in tile matrix at this zoom level</td>
<td>no</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>tile_width</td>
<td>integer</td>
<td>Tile width in pixels (&gt;= 1) for this zoom level</td>
<td>no</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>tile_height</td>
<td>integer</td>
<td>Tile height in pixels (&gt;= 1) for this zoom level</td>
<td>no</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>pixel_x_size</td>
<td>double</td>
<td>In t_table_name srs_id units or default meters for srs_id 0 (&gt;0)</td>
<td>no</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>pixel_y_size</td>
<td>double</td>
<td>In t_table_name srs_id units or default meters for srs_id 0 (&gt;0)</td>
<td>no</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The gpkg_tile_matrix_metadata table documents the structure of the tile matrix at each zoom level in each tiles table. It allows GeoPackages to contain rectangular as well as square tiles (e.g. for better representation of polar regions). It allows tile pyramids with zoom levels that differ in resolution by powers of 2, irregular intervals, or regular intervals other than powers of 2.

See Annex C: Table Definition SQL clause C.6 gpkg_tile_matrix_metadata

2.2.6.1.2. Table Data Values

**Req 34:** Values of the tile_matrix_metadata table_name column SHALL reference values in the gpkg_contents table_name column for rows with a data_type of “tiles”.

**Req 35:** A gpkg_tile_matrix_metadata table SHALL contain one row record for each zoom level that contains one or more tiles in each tile matrix user data table.

The gpkg_tile_matrix_metadata table or view MAY contain row records for zoom levels in a tiles table that do not contain tiles.

GeoPackages follow the most frequently used conventions of a tile origin at the upper left and a zoom-out-level of 0 for the smallest map scale “whole world” zoom level view, as

---

1 GeoPackage applications MAY query the gpkg_tile_matrix_metadata table or the tile matrix user data table to determine the minimum and maximum zoom levels for a given tile matrix table.
specified by WMTS [20]. The tile coordinate (0,0) always refers to the tile in the upper left corner of the tile matrix at any zoom level, regardless of the actual availability of that tile.

**Req 36:** The `zoom_level` column value in a `tile_matrix_metadata` table row SHALL not be negative.

**Req 37:** The `matrix_width` column value in a `tile_matrix_metadata` table row SHALL be greater than 0.

**Req 38:** The `matrix_height` column value in a `tile_matrix_metadata` table row SHALL be greater than 0.

**Req 39:** The `tile_width` column value in a `tile_matrix_metadata` table row SHALL be greater than 0.

**Req 40:** The `tile_height` column value in a `tile_matrix_metadata` table row SHALL be greater than 0.

**Req 41:** The `pixel_x_size` column value in a `tile_matrix_metadata` table row SHALL be greater than 0.

**Req 42:** The `pixel_y_size` column value in a `tile_matrix_metadata` table row SHALL be greater than 0.

**Req 43:** The `pixel_x_size` and `pixel_y_size` column values for `zoom_level` column values in a `tile_matrix_metadata` table sorted in ascending order SHALL be sorted in descending order.

Tiles MAY or MAY NOT be provided for level 0 or any other particular zoom level.\(^1\) This means that a tile matrix set can be sparse, i.e. not contain a tile for any particular position at a certain tile zoom level.\(^2\) This does not affect the spatial extent stated by the min/max x/y columns values in the gpkg_contents record for the same table_name, or the tile matrix width and height at that level.\(^3\)

---

\(^1\) GeoPackage applications MAY query the tiles (matrix set) table to determine which tiles are available at each zoom level.

\(^2\) GeoPackage applications that insert, update, or delete tiles (matrix set) table tiles row records are responsible for maintaining the corresponding descriptive contents of the gpkg_tile_matrix_metadata table.

\(^3\) The gpkg_contents table contains coordinates that define a bounding box as the stated spatial extent for all tiles in a tile (matrix set) table. If the geographic extent of the image data contained in these tiles is within but not equal to this bounding box, then the non-image area of matrix edge tiles must be padded with no-data values, preferably transparent ones.
2.2.7. Tile Matrix User Data

2.2.7.1. Data

2.2.7.1.1. Table Definition

Req 44: Each tile matrix set in a GeoPackage file SHALL be stored in a different tiles table or updateable view with a unique name per clause 2.2.7.1.1, Table 8 and Table 31.

Table 8 Tiles Table or View Definition

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
<th>Null</th>
<th>Default</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>Autoincrement primary key</td>
<td>no</td>
<td></td>
<td>PK</td>
</tr>
<tr>
<td>zoom_level</td>
<td>integer</td>
<td>min(zoom_level) &lt;= zoom_level &lt;= max(zoom_level) for t_table_name</td>
<td>no</td>
<td>0</td>
<td>UK</td>
</tr>
<tr>
<td>tile_column</td>
<td>integer</td>
<td>0 to gpkg_tile_matrix_metadata matrix_width – 1</td>
<td>no</td>
<td>0</td>
<td>UK</td>
</tr>
<tr>
<td>tile_row</td>
<td>integer</td>
<td>0 to gpkg_tile_matrix_metadata matrix_height - 1</td>
<td>no</td>
<td>0</td>
<td>UK</td>
</tr>
<tr>
<td>tile_data</td>
<td>BLOB</td>
<td>Of an image MIME type specified in clauses 2.2.4, 2.2.5, 3.2.2, 3.2.3, 3.2.4</td>
<td>no</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Annex C: Table Definition SQL clause C.7 sample_matrix_tiles

2.2.7.1.2. Table Data Values

Each tiles table or view¹ MAY contain tile matrices at zero or more zoom levels of different spatial resolution (map scale).

Req 45: For each distinct table_name from the gpkg_tile_matrix_metadata (tmm) table, the tile matrix set (tms) user data table zoom_level column value in a GeoPackage file SHALL be in the range min(tmm.zoom_level) <= tms.zoom_level <= max(tmm.zoom_level).

Req 46: For each distinct table_name from the tile_matrix_metadata (tmm) table, the tile matrix set (tms) user data table tile_column column value in a GeoPackage file SHALL be in the range 0 <= tms.tile_column <= tmm.matrix_width – 1 where the tmm and tms zoom_level column values are equal.

Req 47: For each distinct table_name from the tile_matrix_metadata (tmm) table, the tile matrix set (tms) user data table tile_row column value in a GeoPackage file SHALL be in the range 0 <= tms.tile_row <= tmm.matrix_height – 1 where the tmm and tms zoom_level column values are equal.

All tiles at a particular zoom level have the same pixel_x_size and pixel_y_size values specified in the gpkg_tile_matrix_metadata row record for that tiles table and zoom level.¹

¹ A GeoPackage is not required to contain any tile matrix data tables. Tile matrix user data tables in a GeoPackage MAY be empty.
2.3. Schema

The optional capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file.

2.3.1. Data Columns

2.3.1.1. Data

2.3.1.1.1. Table Definition

Req 48: A GeoPackage file MAY contain a table or updateable view named gpkg_data_columns. If present it SHALL be defined per clause 2.3.1.1.1, Table 9 and Table 32.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
<th>Null</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>text</td>
<td>Name of the tiles or feature table</td>
<td>no</td>
<td>PK</td>
</tr>
<tr>
<td>column_name</td>
<td>text</td>
<td>Name of the table column</td>
<td>no</td>
<td>PK</td>
</tr>
<tr>
<td>name</td>
<td>text</td>
<td>A human-readable identifier (e.g. short name) for the column_name content</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>text</td>
<td>A human-readable formal title for the column_name content</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>text</td>
<td>A human-readable description for the table_name content</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>mime_type</td>
<td>text</td>
<td>MIME [25] type of column_name if BLOB type, or NULL for other types</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>

GeoPackage applications MAY\(^2\) use the gpkg_data_columns table to store minimal application schema identifying, descriptive and MIME [25] type\(^3\) information about columns in user vector feature and tile matrix data tables that supplements the data available from the SQLite sqlite_master table and pragma table_info(table_name) SQL function. The gpkg_data_columns data CAN be used to provide more specific column data types and value ranges and application specific structural and semantic information to enable more informative user menu displays and more effective user decisions on the suitability of GeoPackage contents for specific purposes.

---

\(^1\) The zoom_level / tile_column / tile_row unique key is automatically indexed, and allows tiles to be selected and accessed by “z, x, y”, a common convention used by some implementations. This table / view definition MAY also allow tiles to be selected based on a spatially indexed bounding box in a separate metadata table.

\(^2\) A GeoPackage is not required to contain a gpkg_data_columns table. The gpkg_data_columns table in a GeoPackage MAY be empty.

\(^3\) GeoPackages MAY contain MIME types other than the raster image types specified in clauses 2.2.4, 2.2.5, 3.2.2, 3.2.3, and 3.2.4 as feature attributes, but they are not required to do so.
See Annex C: Table Definition SQL clause C.8 gpkg_data_columns

2.3.1.1.2. Table Data Values

Req 49: Values of the gpkg_data_columns table table_name column value SHALL reference values in the gpkg_contents table_name column.

Req 50: The column_name column value in a gpkg_data_columns table row SHALL contain the name of a column in the SQLite table or view identified by the table_name column value.

2.4. Metadata

The optional capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file.

2.4.1. Introduction

Two tables in a GeoPackage file provide a means of storing metadata in MIME [25] encodings that are defined in accordance with any authoritative metadata specifications, and relating it to the features, rasters, and tiles data in a GeoPackage. These tables are intended to provide the support necessary to implement the hierarchical metadata model defined in ISO 19115 [40], Annex B B.5.25 MD_ScopeCode, Annex I and Annex J, so that as GeoPackage data is captured and updated, the most local and specific detailed metadata changes associated with the new or modified data MAY be captured separately, and referenced to existing global and general metadata.

The gpkg_metadata table that contains metadata is described in clause 2.4.2, and the gpkg_metadata_reference table that relates gpkg_metadata to GeoPackage data is described in clause 2.4.3. There is no GeoPackage requirement that such metadata be provided or that defined metadata be structured in a hierarchical fashion\(^1\) with more than one level, only that if it is, these tables SHALL be used. Such metadata\(^2\) and data that relates it to GeoPackage contents SHALL NOT be stored in other tables as defined in clause 2.5.

2.4.2. Metadata Table

2.4.2.1. Data

2.4.2.1.1. Table Definition

Req 51: A GeoPackage file MAY contain a table or updateable view named gpkg_metadata. If present it SHALL be defined per clause 2.4.2.1.1, Table 10 and Table 33.

The first component of GeoPackage metadata is the gpkg_metadata table that MAY contain metadata in MIME [25] encodings structured in accordance with any authoritative metadata specification, such as ISO 19115 [40], ISO 19115-2 [B31], ISO 19139 [B32], Dublin Core [B33], CSDGM [B34], DDMS [B35], NMF/NMIS [B36], etc. The GeoPackage interpretation

\(^1\): Informative examples of hierarchical metadata are provided in Annex I.

\(^2\): An informative example of raster image metadata is provided in Annex J.
of what constitutes “metadata” is a broad one that includes UML models [B37] encoded in XMI [B38], GML Application Schemas [B39], ISO 19110 feature catalogues [B40], OWL [B41] and SKOS [B42] taxonomies, etc.

**Table 10 Metadata Table or View Definition**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Description</th>
<th>Null</th>
<th>Default</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>Metadata primary key</td>
<td>no</td>
<td></td>
<td>PK</td>
</tr>
<tr>
<td>md_scope</td>
<td>text</td>
<td>Name of the data scope to which this metadata applies; see table 11 below</td>
<td>no</td>
<td>‘dataset’</td>
<td></td>
</tr>
<tr>
<td>md_standard_uri</td>
<td>text</td>
<td>URI reference to the metadata structure definition authority</td>
<td>no</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td></td>
</tr>
<tr>
<td>mime_type</td>
<td>text</td>
<td>MIME [25] encoding of metadata</td>
<td>no</td>
<td>text/xml</td>
<td></td>
</tr>
<tr>
<td>metadata</td>
<td>text</td>
<td>metadata</td>
<td>no</td>
<td>‘’</td>
<td></td>
</tr>
</tbody>
</table>

See Annex C: Table Definition SQL clause C.9 gpkg_metadata.

2.4.2.1.2. Table Data Values

The md_scope column in the gpkg_metadata table is the name of the applicable scope for the contents of the metadata column for a given row. The list of valid scope names and their definitions is provided in Table 11 below. The initial contents of this table were obtained from the ISO 19115 [40], Annex B B.5.25 MD_ScopeCode code list, which was extended\(^1\) for use in the GeoPackage specification by addition of entries with “NA” as the scope code column in Table 10.

**Table 11 Metadata Scopes**

<table>
<thead>
<tr>
<th>Name (md_scope)</th>
<th>Scope Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>undefined</td>
<td>NA</td>
<td>Metadata information scope is undefined</td>
</tr>
<tr>
<td>fieldSession</td>
<td>012</td>
<td>Information applies to the field session</td>
</tr>
<tr>
<td>collectionSession</td>
<td>004</td>
<td>Information applies to the collection session</td>
</tr>
<tr>
<td>series</td>
<td>006</td>
<td>Information applies to the (dataset) series(^2)</td>
</tr>
<tr>
<td>dataset</td>
<td>005</td>
<td>Information applies to the (geographic feature) dataset</td>
</tr>
<tr>
<td>featureType</td>
<td>010</td>
<td>Information applies to a feature type (class)</td>
</tr>
<tr>
<td>feature</td>
<td>009</td>
<td>Information applies to a feature (instance)</td>
</tr>
<tr>
<td>attributeType</td>
<td>002</td>
<td>Information applies to the attribute class</td>
</tr>
<tr>
<td>attribute</td>
<td>001</td>
<td>Information applies to the characteristic of a feature (instance)</td>
</tr>
</tbody>
</table>

\(^1\) The scope codes in Table 13 include a very wide set of descriptive information types as “metadata” to describe data.

\(^2\) ISO 19139 format metadata [B32] is recommended for general-purpose description of geospatial data at the series and dataset metadata scopes.
OGC 12-128r8

<table>
<thead>
<tr>
<th>Field Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>016</td>
<td>Information applies to a tile, a spatial subset of geographic data</td>
</tr>
<tr>
<td>015</td>
<td>Information applies to a copy or imitation of an existing or hypothetical object</td>
</tr>
<tr>
<td>NA</td>
<td>Metadata applies to a feature catalog</td>
</tr>
<tr>
<td>NA</td>
<td>Metadata applies to an application schema</td>
</tr>
<tr>
<td>NA</td>
<td>Metadata applies to a taxonomy or knowledge system</td>
</tr>
<tr>
<td>013</td>
<td>Information applies to a computer program or routine</td>
</tr>
<tr>
<td>014</td>
<td>Information applies to a capability which a service provider entity makes available to a service user entity through a set of interfaces that define a behaviour, such as a use case</td>
</tr>
<tr>
<td>003</td>
<td>Information applies to the collection hardware class</td>
</tr>
<tr>
<td>007</td>
<td>Information applies to non-geographic data</td>
</tr>
<tr>
<td>008</td>
<td>Information applies to a dimension group</td>
</tr>
</tbody>
</table>

### Req 52: Each md_scope column value in a gpkg_metadata table or updateable view SHALL be one of the name column values from Table 11 in clause 2.4.2.1.2.

### 2.4.3. Metadata Reference Table

#### 2.4.3.1. Data

#### 2.4.3.1.1. Table Definition

### Req 53: A GeoPackage file that contains a gpkg_metadata table SHALL contain a gpkg_metadata_reference table per clause 2.4.3.1.1, Table 12 and Table 34.

The second component of GeoPackage metadata is the gpkg_metadata_reference table that links metadata in the gpkg_metadata table to data in the feature, and tiles tables defined in clauses 2.1.6 and 2.2.7. The gpkg_metadata_reference table is not required to contain any rows.

### Table 12 Metadata Reference Table or View Definition

<table>
<thead>
<tr>
<th>Table or View Name: gpkg_metadata_reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Name</td>
</tr>
<tr>
<td>-------------</td>
</tr>
</tbody>
</table>

1 The “catalog” md_scope MAY be used for Feature Catalog [B40] information stored as XML metadata that is linked to features stored in a GeoPackage.

2 The “schema” md_scope MAY be used for Application Schema [B37][B38][B39][B44] information stored as XML metadata that is linked to features stored in a GeoPackage.

3 The “taxonomy” md_scope MAY be used for taxonomy or knowledge system [B41][B42] “linked data” information stored as XML metadata that is linked to features stored in a GeoPackage.
Every row in gpkg_metadata_reference that has null value as md_parent_id forms the root of a metadata hierarchy².

See Annex C: Table Definition SQL clause C.10 gpkg_metadata_reference.

2.4.3.1.2. Table Data Values


Req 55: Every gpkg_metadata_reference table row with a reference_scope column value of ‘geopackage’ SHALL have a table_name column value that is NULL. Every other

---

¹ In SQLite, the rowid value is always equal to the value of a single-column primary key on an integer column [B30] and is not changed by a database reorganization performed by the VACUUM SQL command.

² Such a metadata hierarchy MAY have only one level of defined metadata.
gpkg_metadata_reference table row SHALL have a table_name column value that references a value in the gpkg_contents table_name column.

Req 56: Every gpkg_metadata_reference table row with a reference_scope column value of 'geopackage', 'table' or 'row' SHALL have a column_name column value that is NULL. Every other gpkg_metadata_reference table row SHALL have a column_name column value that contains the name of a column in the SQLite table or view identified by the table_name column value.

Req 57: Every gpkg_metadata_reference table row with a reference_scope column value of 'geopackage', 'table' or 'column' SHALL have a row_id_value column value that is NULL. Every other gpkg_metadata_reference table row SHALL have a row_id_value column value that contains the ROWID of a row in the SQLite table or view identified by the table_name column value.

Req 58: Every gpkg_metadata_reference table row timestamp column value SHALL be in ISO 8601 [41] format containing a complete date plus UTC hours, minutes, seconds and a decimal fraction of a second, with a ‘Z’ (‘zulu’) suffix indicating UTC.¹

Req 59: Every gpkg_metadata_reference table row md_file_id column value SHALL be an id column value from the gpkg_metadata table.

Req 60: Every gpkg_metadata_reference table row md_parent_id column value that is NOT NULL SHALL be an id column value from the gpkg_metadata table that is not equal to the md_file_id column value for that row.

2.5. Other Data Tables

Application-specific or vendor-specific data tables that do not conform to the table definitions or other requirements for storing feature or tile data or metadata as defined in other sections of this document MAY be defined and used to store supplemental information in a GeoPackage database. Such other data tables MAY be described by rows in the gpkg_spatial_ref_sys, gpkg_contents, gpkg_geometry_columns, gpkg_extensions, gpkg_data_columns, gpkg_tile_matrix_metadata, gpkg_metadata and gpkg_metadata_reference tables. GeoPackage applications MAY ignore such descriptive rows and other data tables they do not recognize

2.6. Extension Mechanism

The GeoPackage extension mechanism is an optional capability that is a required dependency for the optional registered extension capabilities defined in clause 3.

¹ The following statement selects an ISO 8601 timestamp value using the SQLite strftime function:

```
SELECT (strftime('%Y-%m-%dT%H:%M:%fZ','now')).
```
2.6.1. Extensions

2.6.1.1. Data

The optional capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file.

2.6.1.1.1. Table Definition

Req 61: A GeoPackage file MAY contain a table or updateable view named gpkg_extensions. If present this table SHALL be defined per clause 2.6.1.1.1, Table 13 and Table 23.

The gpkg_extensions table in a GeoPackage file registers extension capabilities required to make direct use of its contents. An application that access a GeoPackage file can query the gpkg_extensions table instead of the contents of all of the user data tables to determine if it has the required capabilities, and to “fail fast” and return an error message if it does not.

<table>
<thead>
<tr>
<th>Table 13 GeoPackage Extensions Table or View Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table or View Name</strong>: gpkg_extensions</td>
</tr>
<tr>
<td>Column Name</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>table_name</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>column_name</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>extension_name</td>
</tr>
</tbody>
</table>

See Annex C: Table Definition SQL clause C.3 gpkg_extensions.

2.6.1.1.2. Table Data Values

Req 62: Values of the gpkg_extensions table table_name column SHALL reference values in the gpkg_contents table_name column or be NULL. They SHALL NOT be NULL for rows where the column_name value is not NULL.

Req 63: The column_name column value in a gpkg_extensions table row SHALL be the name of a column in the table specified by the table_name column value for that row, or be NULL.
Req 64: Each extension_name column value in a gpkg_extensions table row SHALL be a unique value of the form &lt;author&gt;_&lt;extension name&gt; where &lt;author&gt; indicates the person or organization that developed and maintains the extension. The valid character set for &lt;author&gt; SHALL be [a-zA-Z0-9]. The valid character set for &lt;extension name&gt; SHALL be [a-zA-Z0-9_]. An extension_name for the “gpkg” author name SHALL be one of those listed in Table 14.

The author value “gpkg” is reserved for GeoPackage extensions that are developed and maintained by OGC. Requirements for extension names for the “gpkg” author name are defined in the clauses listed in Table 14 below. GeoPackage implementers use their own author names to register other extensions.¹

<table>
<thead>
<tr>
<th>Extension Name</th>
<th>Defined in Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpkg_geom_&lt;gname&gt;</td>
<td>3.1.2.1.2</td>
<td>&lt;gname&gt; is name of specific extension geometry type from Annex G Table 47</td>
</tr>
<tr>
<td>gpkg_zoom_other</td>
<td>3.2.1.1.1</td>
<td>Tile matrix set user data table with zoom intervals other than powers of two between adjacent layers</td>
</tr>
<tr>
<td>gpkg_webp</td>
<td>3.2.2.2.1</td>
<td>Raster images in image/webp format</td>
</tr>
<tr>
<td>gpkg_tiff</td>
<td>3.2.3.1.2</td>
<td>Raster images in image/tiff format</td>
</tr>
<tr>
<td>gpkg_nitf</td>
<td>3.2.4.1.2</td>
<td>Raster images in application/vnd.NITF</td>
</tr>
<tr>
<td>gpkg_rtree_index</td>
<td>3.1.3.1.2</td>
<td>Rtree spatial index implemented via SQLite Virtual Table and maintained via triggers that use all minimal runtime SQL functions except ST_SRID.</td>
</tr>
<tr>
<td>gpkg_gtype_trigger</td>
<td>3.1.4.1.2</td>
<td>Trigger to restrict GeoPackage feature table geometry column geometries to geometry types equal to or substitutable for the geometry type specified for the column in the gpkg_geometry_columns table geometry_type_name column.</td>
</tr>
<tr>
<td>gpkg_srs_id_trigger</td>
<td>3.1.5.1.2</td>
<td>Trigger to restrict GeoPackage feature table geometry column geometries to the srs_id specified for the geometry column in the gpkg_geometry_columns table using the ST_SRID SQL function.</td>
</tr>
</tbody>
</table>

¹ See Requirement 67 in Clause 3.
2.6.1.2.  API

The optional capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage SQLite Extension.

2.6.1.2.1.  API GeoPackage SQLite Config

Req 65: A GeoPackage SQLite Extension MAY provide SQL runtime functions or rtree support for a GeoPackage file. One that does so SHALL have the SQLite library compile and run time options specified in clause 2.6.1.2.1 Table 15

Table 15 API GeoPackage SQLite Configuration

<table>
<thead>
<tr>
<th>Setting</th>
<th>Option</th>
<th>Shall / Not (Value)</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_LOAD_EXTENSION</td>
<td>Not</td>
<td>The load_extension() function is required to implement the MinimalRuntimeSQLFunctions</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_VIRTUALTABLE</td>
<td>Not</td>
<td>Virtual tables are required to implement RTrees</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_ENABLE_RTREE</td>
<td>Shall</td>
<td>Rtrees are used for GeoPackage Spatial Indexes. See SpatialIndexRequirements</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_RTREE_INT_ONLY</td>
<td>Not</td>
<td>Rtrees with floating point values are used for GeoPackage Spatial Indexes.</td>
</tr>
</tbody>
</table>

2.6.1.2.2.  Safe GeoPackage SQLite Config

Req 66: A GeoPackage SQLite Extension MAY provide primary/foreign key and trigger support for a GeoPackage file. One that does so SHALL have the SQLite library compile and run time options specified in clause 2.6.1.2.2 Table 16.

Table 16 Safe GeoPackage SQLite Configuration

<table>
<thead>
<tr>
<th>Setting</th>
<th>Option</th>
<th>Shall / Not (Value)</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>compile</td>
<td>SQLITE_DEFAULT_FOREIGN_KEYS</td>
<td>Shall (1)</td>
<td>Foreign key constraints are used to maintain GeoPackage relational integrity.</td>
</tr>
<tr>
<td>Compile</td>
<td>Option</td>
<td>Status</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_FOREIGN_KEY</td>
<td>Not</td>
<td>Foreign key constraints are used to maintain GeoPackage relational integrity.</td>
</tr>
<tr>
<td>run</td>
<td>PRAGMA foreign_keys</td>
<td>Not (OFF)</td>
<td>Foreign key constraints are used to maintain GeoPackage relational integrity.</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_INTEGRITY_CHECK</td>
<td>Not</td>
<td>This option omits support for the <code>integrity_check</code> pragma, which does an integrity check of the entire database. This pragma should be part of GeoPackage conformance validation.</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_SUBQUERY</td>
<td>Not</td>
<td>This option omits support for sub-selects and the IN() operator, both of which are used in GeoPackage? triggers.</td>
</tr>
<tr>
<td>compile</td>
<td>SQLITE_OMIT_TRIGGER</td>
<td>Not</td>
<td>Defining this option omits support for TRIGGER objects. Neither the <code>CREATE TRIGGER</code> or <code>DROP TRIGGER</code> commands are available in this case, and attempting to execute either will result in a parse error. This option also disables enforcement of foreign key constraints, since the code that implements triggers and which is omitted by this option is also used to implement foreign key actions. Foreign keys and triggers are used by Safe GeoPackages. Triggers are used to maintain spatial indexes.</td>
</tr>
</tbody>
</table>

**Registered Extensions**

The optional registered extension capabilities specified in this clause depend on required capabilities specified in clause 1 and optional capabilities specified in clause 2. Any optional
capability in this class that is implemented SHALL be registered using the Extension Mechanism defined in clause 2.6, as specified by the sub clauses below for extension name and extension row for the optional capability.

**Req 67:** Any table in a GeoPackage file subject to a registered extension with an author_name other than “gpkg” SHALL NOT be described by a gpkg_contents table row with a data_type value of “feature” or “tiles”.

Such tables do not count\(^1\) in the determination of GeoPackage file validity per clause 2 Req 13: but otherwise MAY differ from “features” and “tiles” tables only with respect to non-gpkg registered extensions that enable implementers to meet requirements not addressed in the current version of this specification, and may be included in a valid GeoPackage.

3.1. Features

3.1.1. Geometry Encoding

3.1.1.1. Data

The optional registered extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements the requirements of clause 2.1 and clause 2.6.1.1 on which they depend.

3.1.1.1.1. BLOB Format

A GeoPackage file CAN store geometries using BLOB formats other than the GeoPackageBinary format specified in clause 2.1.3.1.1. However, tables with geometry column data encoded in such extension formats are not considered to be GeoPackage feature tables for purposes of determining GeoPackage file validity.

3.1.1.1.2. BLOB format - Extensions Name

**Req 68:** An extension name in the form <author_name>_geometry_encoding SHALL be defined for an author name other than “gpkg” for each geometry BLOB format other than GeoPackageBinary used in a GeoPackage file.

3.1.1.1.3. BLOB format - Extensions Row

**Req 69:** A GeoPackage file that contains geometry column BLOB values encoded in an extension format SHALL contain a gpkg_extensions table that contains row records with table_name and column_name values from the gpkg_geometry_columns table row records that identify extension format uses, and with extension_name column values constructed per clause 3.1.1.1.2.

\(^1\) They MAY be contained in a valid GeoPackage file.
3.1.2. Geometry Types

3.1.2.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements the requirements of clause 2.1 and clause 2.6.1.1 on which they depend, as specified in Annex A.

3.1.2.1.1. Extension Types

Req 70: A GeoPackage file MAY store feature table geometries with the extended non-linear geometry types (CircularString, CompoundCurve, CurvePolygon, MultiCurve, MultiSurface, Curve, Surface) in Annex G. If it does so, they SHALL be encoded in the GeoPackageBinary geometry format.

3.1.2.1.2. Extensions Types - Extensions Name

Req 71: An extension name to specify a feature geometry extension type SHALL be defined for the “gpkg” author name using the “gpkg_geom_<gname>” template where <gname> is the name of the extension geometry type from Annex G used in a GeoPackage file.

3.1.2.1.3. Extensions Types - Extensions Row

Req 72: A GeoPackage file that contains a gpkg_geometry_columns table or updateable view with row records that specify extension geometry_type column values SHALL contain a gpkg_extensions table that contains row records with table_name and column_name values from the geometry_columns row records that identify extension type uses, and extension_name column values for each of those geometry types constructed per clause 3.1.2.1.2.

3.1.3. Spatial Indexes

3.1.3.1. Data

The optional registered extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements requirements of clause 2.1 and clause 2.6 on which they depend.

3.1.3.1.1. Spatial Indexes Implementation

Spatial indexes provide a significant performance advantage for searches with basic envelope spatial criteria that return subsets of the rows in a feature table with a non-trivial number (thousands or more) of rows.¹

Spatial indexes are an optional GeoPackage extension with the following runtime requirements

- The SQLite RTree extension SHALL be available and loaded per clause 2.6.1.2.1.

¹If an application process will make many updates, it is often faster to drop the indexes, do the updates, and then recreate the indexes.
- SQLite trigger support SHALL be present per clause 2.6.1.2.2.
- The ST_MinX, ST_MaxX, ST_MinY, ST_MaxY SQL functions specified in Annex D SHALL be available.

**Req 73:** A GeoPackage file SHALL implement spatial indexes on feature table geometry columns as specified in clause 3.1.3.1.1 using the SQLite Virtual Table RTrees and triggers specified in Annex E.

### 3.1.3.1.2. Spatial Indexes - Extensions Name

**Req 74:** The “gpkg_rtree_index” extension name SHALL be used as a gpkg_extension table extension name column value to specify implementation of spatial indexes on a geometry column.

### 3.1.3.1.3. Spatial Indexes - Extensions Row

**Req 75:** A GeoPackage file that implements spatial indexes SHALL have a gpkg_extensions table that contains a row for each spatially indexed column with extension_name “gpkg_rtree_index”, the table_name of the table with a spatially indexed column, and the column_name of the spatially indexed column.

### 3.1.4. Geometry Type Triggers

#### 3.1.4.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements requirements of clause 2.1 and clause 2.6 on which they depend.

#### 3.1.4.1.1. Geometry Type Triggers – Implementation

**Req 76:** A GeoPackage file SHALL include the SQL insert and update triggers specified in clause 3.1.4.1.1 and Table 17 on every feature table geometry column to enforce the geometry type values specified for those columns in the gpkg_geometry_columns table.

#### Table 17 Geometry Type Trigger Definition SQL Template

```sql
CREATE TRIGGER fgti_<t>_<c> BEFORE INSERT ON '<t>'
FOR EACH ROW
BEGIN
    SELECT RAISE (ROLLBACK, 'insert on <t> violates constraint: ST_GeometryType(<c>) is not assignable from gpkg_geometry_columns.geometry_type_name value')
    WHERE (SELECT geometry_type_name FROM gpkg_geometry_columns
            WHERE Lower(table_name) = Lower('<t>')
            AND Lower(column_name) = Lower('<c>')
            AND gpkg_IsAssignable(geometry_type_name,
                                   ST_GeometryType(NEW.<c>)) = 0;
END
```
CREATE TRIGGER fg tu_<t>_<c> BEFORE UPDATE OF '<c>' ON '<t>'
FOR EACH ROW
BEGIN
SELECT RAISE (ROLLBACK,
'update of <c> on <t> violates constraint: ST_GeometryType(<c>) is not assignable from gpkg_geometry_columns.geometry_type_name value')
WHERE (SELECT geometry_type_name FROM gpkg_geometry_columns
WHERE Lower(table_name) = Lower('<t>')
AND Lower(column_name) = Lower('<c>')
AND gpkg_IsAssignable(geometry_type_name,
ST_GeometryType(NEW.<c>)) = 0;
END

where <t> and <c> are replaced with the names of the feature table and geometry column being inserted or updated.

The triggers to enforce gpkg_geometry_columns geometry_type_name constraints on feature table geometries use Minimal Runtime SQL Functions specified in Annex D.

3.1.4.1.2. Geometry Type Triggers – Extensions Name

Req 77: The “gpkg_geometry_type_trigger” extension name SHALL be used as a geopackage_extension table extension name column value to specify implementation of geometry type triggers.

3.1.4.1.3. Geometry Type Triggers – Extensions Row

Req 78: A GeoPackage file that implements geometry type triggers on feature table geometry columns SHALL contain a gpkg_extensions table that contains a row for each such geometry column with extension_name “gpkg_geometry_type_trigger”, table_name of the feature table with a geometry column, and column_name of the geometry column.

3.1.5. SRS_ID Triggers

3.1.5.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements requirements of clause 2.1 and clause 2.6 on which they depend.

3.1.5.1.1. SRS_ID Triggers – Implementation

Req 79: A GeoPackage file SHALL include the SQL insert and update triggers specified in clause 3.1.5.1.1 and Table 18 on every feature table geometry column to enforce the srs_id values specified for those columns in the gpkg_geometry_columns table.

Table 18 SRS_ID Trigger Definition SQL Templates

CREATE TRIGGER fg si_<t>_<c> BEFORE INSERT ON '<t>'
FOR EACH ROW
BEGIN
SELECT RAISE (ROLLBACK, 'insert on '<t>violates constraint: ST_SRID(<c>) does not match gpkg_geometry_columns.srs_id value')
WHERE (SELECT srs_id FROM gpkg_geometry_columns
    WHERE Lower(table_name) = Lower(''<t>'')
    AND   Lower(column_name) = Lower(''<c>'')
    AND   ST_SRID(NEW.''<c>'') <> srs_id) ;

CREATE TRIGGER fgsu_<t>_<c> BEFORE UPDATE OF '<c>' ON '<t>
FOR EACH ROW
BEGIN
SELECT RAISE (ROLLBACK, 'update of '<c> on '<t> violates constraint: ST_SRID(<c>) does not match gpkg_geometry_columns.srs_id value')
WHERE (SELECT srs_id FROM gpkg_geometry_columns
    WHERE Lower(table_name) = Lower(''<t>'')
    AND   Lower(column_name) = Lower(''<c>'')
    AND   ST_SRID(NEW.''<c>'') <> srs_id) ;
END

where '<t>' and '<c>' are replaced with the names of the feature table and geometry column being inserted or updated.

The triggers to enforce geometry_columns srs_id constraints on feature table geometries use Minimal Runtime SQL Functions specified in Annex F

3.1.5.1.2. SRS_ID Triggers – Extensions Name

Req 80: The “gpkg_srs_id_trigger” extension name SHALL be used as a geopackage_extension table extension name column value to specify implementation of SRS_ID triggers.

3.1.5.1.3. SRS_ID Triggers – Extensions Row

Req 81: A GeoPackage file that implements srs_id triggers on feature table geometry columns SHALL contain a gpkg_extensions table that contains a row for each geometry column with extension_name “gpkg_srs_id_trigger”, table_name of the feature table with a geometry column, and column_name of the geometry column.

3.2. Tiles

3.2.1. Zoom Levels

3.2.1.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements the requirements of clause 2.2 and clause 2.6.1.1 on which they depend.
3.2.1.1.1. Zoom Other Intervals

As a registered extension, a GeoPackage file MAY contain tile matrix set user data tables with pixel sizes that vary by irregular intervals or by regular intervals other than powers of two (the default) between adjacent zoom levels, as described in the gpkg_tile_matrix_metadata table.

3.2.1.1.2. Zoom Other – Extensions Name

**Req 82:** The “gpkg_zoom_other” extension name SHALL be used as a gpkg_extension table extension name column value to specify implementation of other zoom intervals on a tile matrix set user data table.

3.2.1.1.3. Zoom Other – Extensions Row

**Req 83:** A GeoPackage file that implements other zoom intervals SHALL have a gpkg_extensions table that contains a row for each tile matrix set user data table with other zoom intervals with extension_name “gpkg_zoom_other”, the table_name of the table with other zoom intervals, and the “tile_data” column_name.

3.2.2. Tile Encoding WEBP

3.2.2.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements the requirements of clause 2.2 and clause 2.6.1.1 on which they depend.

3.2.2.2. WEBP MIME Type

As a registered extension, a GeoPackage file that contains a tile matrix user data table that contains tile data MAY store tile_data in MIME type image/x-webp[26].

3.2.2.2.1. WEBP -- Extensions Name

**Req 84:** The “gpkg_webp” extension name SHALL be used as a geopackage_extension table extension name column value to specify storage of raster images in WEBP format.

3.2.2.2.2. WEBP -- Extensions Row

**Req 85:** A GeoPackage file that contains tile matrix user data tables with tile_data columns that contain raster images in WEBP format SHALL contain a gpkg_extensions table that contains row records with table_name values for each such table, “tile_data” column_name values and extension_name column values of “gpkg_webp”.
3.2.3. Tiles Encoding TIFF

3.2.3.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements the requirements of clause 2.2 and clause 2.6.1.1 on which they depend.

3.2.3.1.1. TIFF MIME Type

As a registered extension, a GeoPackage file that contains a tile matrix user data table that contains tile data MAY store tile_data in MIME type image/tiff [27] for GeoTIFF images [28][29] that meet the requirements of the NGA Implementation Profile [31] for coordinate transformation case 3 where the position and scale of the data is known exactly, and no rotation of the image is required.

3.2.3.1.2. TIFF -- Extensions Name

Req 86: The “gpkg_tiff” extension name SHALL be used as a geopackage_extension table extension name column value to specify storage of raster images in TIFF format.

3.2.3.1.3. Extensions Row

Req 87: A GeoPackage file that contains tile matrix user data tables with tile_data columns that contain raster images in TIFF format per SHALL contain a gpkg_extensions table that contains row records with table_name values for each such table, “tile_data” column_name values and extension_name column values of “gpkg_tiff”.

3.2.4. Tile Encoding NITF

3.2.4.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements the requirements of clause 2.2 and clause 2.6.1.1 on which they depend.

3.2.4.1.1. NITF MIME Type

As a registered extension, a GeoPackage file that contains a tile matrix user data table that contains tile data MAY store tile_data in MIME type application/vnd.NITF[46] for National Imagery Transmission Format images.

3.2.4.1.2. NITF -- Extensions Name

Req 88: The “gpkg_nitf” extension name SHALL be used as a geopackage_extension table extension name column value to specify storage of raster images in NITF format.
3.2.4.1.3. NITF -- Extensions Row

**Req 89:** A GeoPackage file that contains tile matrix user data tables with tile_data columns that contain raster images in NITF format SHALL contain a gpkg_extensions table that contains row records with table_name values for each such table, “tile_data” column_name values and extension_name column values of “gpkg_nitf”.

3.2.5. Tile Encoding Other

3.2.5.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements the requirements of clause 2.2 and clause 2.6.1.1 on which they depend.

3.2.5.1.1. Other MIME Type

As a registered extension, a GeoPackage file that contains a tile matrix user data table that contains tile data MAY store tile_data in other MIME types. However, a table with such data is not considered to be a “tiles” table for purposes of determining GeoPackage file validity.

3.2.5.1.2. Other Extensions Name

**Req 90:** An extension name in the form `<author_name>_<other>_mime_type` SHALL be defined for an author name other than “gpkg” for each other MIME image format used for tile_data columns in tile matrix set user data tables, where <other> is replaced by the other MIME type abbreviation in uppercase.

3.2.5.1.3. Other Extensions Row

**Req 91:** A GeoPackage file that contains tile matrix user data tables with tile_data columns that contain raster images in a MIME type format other than those defined in this specification SHALL contain a gpkg_extensions table that contains row records with table_name values for each such table, “tile_data” column_name values and extension_name column values of the other format extension name defined per clause 3.2.5.1.2.

3.3. Any Tables

3.3.1. Other Trigger

3.3.1.1. Data

The optional extension capabilities defined in sub clauses and requirement statements of this clause MAY be implemented by any GeoPackage file that implements requirements of clause 2.1 and/or 2.2 and clause 2.6 on which they depend.

3.3.1.1.1. Other Trigger Implementation

As a registered extension, GeoPackage files MAY contain other triggers that require support from GeoPackage SQLite Extension functions other than those provided by SQLite or the
GeoPackage Minimal Runtime SQL Functions to enforce data integrity or application business rule constraints. ¹

3.3.1.1.2. Other Trigger – Extensions Name

**Req 92:** An extension name in the form `<author><extension>` for an author name other than “gpkg” SHALL be defined as a `geopackage_extension` table extension name column value to specify triggers in a GeoPackage file that use SQL functions other than those provided by SQLite or the GeoPackage Minimal Runtime SQL Functions.

3.3.1.1.3. Other Trigger – Extensions Row

**Req 93:** A GeoPackage file that implements triggers that use SQL functions other than those provided by SQLite or the GeoPackage Minimal Runtime SQL Functions SHALL have a `gpkg_extensions` table that contains row records with `table_name` values for each such table, `column_name` values for each such column and `extension_name` column values of the other trigger extension name defined per clause 3.3.1.1.2.

**Security Considerations**

Security considerations for implementations utilizing GeoPackages are in the domain of the implementing application, deployment platform, operating system and networking environment. The GeoPackage specification does not place any constraints on application, platform, operating system level or network security.

¹ Other triggers that rely only on SQLite and Minimal Runtime SQL functions do not need to be registered as an extension.
Annex A  Conformance / Abstract Test Suite (Normative)

A.1 Base
A.1.1 Core
A.1.1.1 SQLite Container
A.1.1.1.1 Data
A.1.1.1.1.1 File Format

Test Case ID: /base/core/container/data/file_format
Test Purpose: Verify that the Geopackage file is an SQLite version_3 database
Test Method: Pass if the first 16 bytes of the file contain “SQLite format 3” in ASCII.
Reference: Clause 1.1.1.1.1 Req 1:
Test Type: Basic

A.1.1.1.1.2 File Extension Name

Test Case ID: /base/core/container/data/file_extension_name
Test Purpose: Verify that the geopackage file extension is ".gpkg"
Test Method: Pass if the geopackage file extension is ".gpkg"
Reference: Clause 1.1.1.1.2 Req 2:
Test Type: Basic

A.1.1.1.1.3 Integrity Check

Test Case ID: /base/core/container/data/file_integrity
Test Purpose: Verify that the geopackage file passes the SQLite integrity check.
Test Method: Pass if PRAGMA integrity_check returns "ok"
Reference: Clause 1.1.1.1.3 Req 3:
Test Type: Capability

A.1.1.2 API
A.1.1.2.1 Structured Query Language

Test Case ID: /base/core/container/api/sql
Test Purpose: Test that the GeoPackage SQLite Extension provides the SQLite SQL API interface.
Test Method:

1) SELECT * FROM sqlite_master
2) Fail if returns an SQL error.
3) Pass otherwise

Reference: Clause 1.1.1.2.1 Req 4:
Test Type: Capability

A.1.1.2.2 Every GPKG SQLite Configuration

Test Case ID: /base/core/container/api/every_gpkg_sqlite_config
Test Purpose: Verify that a GeoPackage SQLite Extension has the Every GeoPackage SQLite Configuration compile and run time options.
Test Method:

1) SELECT sqlite_compileoption_used('SQLITE_OMIT_AUTOINCREMENT')
2) Fail if returns 1
3) SELECT sqlite_compileoption_used('SQLITE_OMIT_DATETIME_FUNCS')
4) Fail if returns 1
5) SELECT sqlite_compileoption_used('SQLITE_OMIT_DEPRECATED ')
6) Fail if returns 0
7) SELECT sqlite_compileoption_used('SQLITE_OMIT_FLOATING_POINT ')
8) Fail if returns 1
9) SELECT sqlite_compileoption_used('SQLITE_OMIT_PRAGMA')
10) Fail if returns 1
11) SELECT sqlite_compileoption_used('SQLITE_OMIT_FLAG_PRAGMAS')
12) Fail if returns 1
13) SELECT sqlite_compileoption_used('SQLITE_OMIT_VIEW')
14) Fail if returns 1
15) Pass otherwise

Reference: Clause 1.1.1.2 Req 5:
Test Type: Basic

A.1.1.2 Spatial Reference Systems

A.1.1.2.1 Data

A.1.1.2.1.1 Table Definition

Test Case ID: /base/core/spatial_ref_sys/data/table_def
Test Purpose: Verify that the spatial_ref_sys table exists and has the correct definition.
Test Method:

1) SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_spatial_ref_sys'
2) Fail if returns an empty result set
3) Pass if column names and column definitions in the returned CREATE TABLE statement in the sql column value, including data type, nullability, default values and primary, foreign and unique key constraints match all of those in the contents of C.1 Table 19. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant.
4) Fail otherwise.

Reference: Clause 1.1.2.1.1 Req 6:
Test Type: Basic

A.1.1.2.1.2 Table Data Values

Test Case ID: /base/core/spatial_ref_sys/data_values_default
Test Purpose: Verify that the spatial_ref_sys table contains the required default contents.
Test Method:

1) SELECT srid, auth_name, auth_srid, srtext FROM spatial_ref_sys WHERE srid = -1 returns -1 “NONE” -1 “Undefined”, AND
2) SELECT srid, auth_name, auth_srid, srtext FROM spatial_ref_sys WHERE srid = 0 returns 0 “NONE” 0 “Undefined”, AND
3) SELECT srid, auth_name, auth_srid, srtext FROM spatial_ref_sys WHERE srid = 4326 returns 4326 epsg 4326 GEOGCS["WGS 84", DATUM["WGS_1984"],
4) Pass if tests 1-3 are met
5) Fail otherwise

Reference: Clause 1.1.2.1.2 Req 7:
Test Type: Capability

Test Case ID: /base/core/spatial_ref_sys/data_values_required
Test Purpose: Verify that the spatial_ref_sys table contains rows to define all srs_id values used by features and tiles in a GeoPackage.
Test Method:

1) SELECT DISTINCT gc.srs_id AS gc_srid, srs.srs_name, srs.srs_id, srs.organization, srs.organization_coordsys_id, srs.definition FROM gpkg_contents AS gc LEFT OUTER JOIN gpkg.spatial_ref_sys AS srs ON srs.srs_id = gc.srs_id
2) Pass if no returned srs values are NULL.
3) Fail otherwise

Reference: Clause 1.1.2.1.2 Req 7:

A.1.1.3 Test Type: CapabilityContents
A.1.1.3.1 Data
A.1.1.3.1.1 Table Definition

Test Case ID: /base/core/contents/data/table_def
Test Purpose: Verify that the gpkg_contents table exists and has the correct definition.
Test Method:

1) SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_contents'
2) Fail if returns an empty result set.
3) Pass if the column names and column definitions in the returned CREATE TABLE statement, including data type, nullability, default values and primary, foreign and unique key constraints match all of those in the contents of C.2 Table 20. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant.
4) Fail Otherwise

Reference: Clause 1.1.3.1.1 Req 9:
Test Type: Basic

A.1.1.3.1.2 Table Data Values

Test Case ID: /base/core/contents/data/data_values_table_name
Test Purpose: Verify that the table_name column values in the gpkg_contents table are valid.
Test Method:
1) SELECT DISTINCT gc.table_name AS gc_table, sm.tbl_name
FROM gpkg_contents AS ge LEFT OUTER JOIN sqlite_master AS sm ON
gc.table_name = sm.tbl_name
2) Not testable if returns an empty result set.
3) Fail if any gpkg_contents.table_name value is NULL
4) Pass otherwise.

Reference: Clause 1.1.3.1.2 Req 10:
Test Type: Capability

Test Case ID: /base/core/contents/data/data_values_last_change
Test Purpose: Verify that the gpkg_contents table last_change column values are in ISO
8601 [41] format containing a complete date plus UTC hours, minutes, seconds and a
decimal fraction of a second, with a ‘Z’ (‘zulu’) suffix indicating UTC.
Test Method:

1) SELECT last_change from gpkg_contents.
2) Not testable if returns an empty result set.
3) For each row from step 1
   a. Fail if format of returned value does not match yyyy-mm-ddThh:mm:ss.hhhZ
   b. Log pass otherwise
4) Pass if logged pass and no fails.

Reference: Clause 1.1.3.1.2 Req 11:
Test Type: Capability

Test Case ID: /base/core/contents/data/data_values_srs_id
Test Purpose: Verify that the gpkg_contents table srs_id column values are defined in the
gpkg_spatial_ref_sys table.
Test Method:

1) SELECT srs_id FROM gpkg_contents
2) Not testable if returns an empty result set
3) SELECT srs_id FROM gpkg_contents WHERE srs_id NOT IN (SELECT srs_id FROM gpkg_spatial_ref_sys)
4) Fail if does not return an empty result set
5) SELECT srs_id FROM gpkg_contents gc WHERE data_type = 'features' AND srs_id
   NOT IN (SELECT srs_id FROM gpkg_geometry_columns WHERE table_name = gc.table_name)
6) Fail if does not return an empty result set
7) Pass otherwise

Reference: Clause 1.1.3.1.2 Req 12:
Test Type: Capability

A.2 Options

Test Case ID: /opt/valid_geopackage
**Test Purpose:** Verify that a GeoPackage contains a features or tiles table and gpkg_contents table row describing it.

**Test Method:**

1) Execute test /opt/features/contents/data/features_row
2) Pass if test passed
3) Execute test /opt/tiles/contents/data/tiles_row
4) Pass if test passed
5) Fail otherwise

**Reference:** Clause 2 Req 13:

**Test Type:** Capability

---

**A.2.1 Features**

**A.2.1.1 Contents**

**A.2.1.1.1 Data**

**Contents Table Feature Row**

**Test Case ID:** /opt/features/contents/data/features_row

**Test Purpose:** Verify that the gpkg_contents table_name value table exists, and is apparently not a tiles table for every row with a data_type column value of “features”

**Test Method:**

1) SELECT table_name FROM gpkg_contents where data_type=”features”
2) Fail if returns empty result set
3) For each row from step 1
   a. PRAGMA table_info(table_name)
   b. Fail if returns an empty result set
   c. Fail if result set contains four rows where the name column values are “zoom_level”, “tile_column”, “tile_row”, and “tile_data”
   d. Fail if result set does not contain one row where the pk column value is 1 and the type column value is “INTEGER”
4) Pass if no fails

**Reference:** Clause 2.1.2.1.1 Req 14:

**Test Type:** Capability

---

**A.2.1.2 Geometry Encoding**

**A.2.1.2.1 Data**

**A.2.1.2.1.1 BLOB Format**

**Test Case ID:** /opt/features/geometry_encoding/data/blob

**Test Purpose:** Verify that geometries stored in feature table geometry columns are encoded in the GeoPackageBinary format.

**Test Method:**

1) SELECT table_name AS tn, column_name AS cn FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type = “features”)
2) Not testable if returns an empty result set
3) For each row from step 1
a. SELECT cn FROM tn
b. Not testable if none found
c. For each cn value from step a
   i. Fail if the first three bytes of each gc are not “GPB”
   ii. Fail if gc.version_number is not 0
   iii. Fail if ST_IsEmpty(cn value) = 1 and gc.flags.envelope != 0 and envelope values are not NaN

4) Pass if no fails

Reference: Clause 2.1.3.1.1 Req 15:
Test Type: Capability

A.2.1.2.2 API
A.2.1.2.2.1 Minimal Runtime SQL Functions

Test Case ID: /opt/features/geometry_encoding/sql_func
Test Purpose: Verify that a GeoPackage SQLite Extension provides the GeoPackage Minimal Runtime SQL functions.
Test Method:

1) Open Geometry Test Data Set GeoPackage with GeoPackage SQLite Extension
2) For each Geometry Test Data Set <gtype_test> data table row for each assignable (gtype, atype) and non-assignable (ntype, atype) combination of geometry type in Annex G, for an assortment of srs_ids, for an assortment of coordinate values, without and with z and / or m values, in both big and little endian encodings:
   a. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_SRID(geom) != srs_id
   b. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_GeometryType(geom) != atype
   c. SELECT ‘Fail’ FROM <gtype_test> WHERE GPKG_IsAssignable(gtype, atype) = 0
   d. SELECT ‘Fail’ FROM <gtype_test> WHERE GPKG_IsAssignable(ntype, atype) = 1
   e. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_Is3D(geom) != is3d
   f. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_IsMeasured(geom) != ism
   g. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_MinX(geom) != minx
   h. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_MaxX(geom) != maxx
   i. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_MinY(geom) != miny
   j. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_MaxY(geom) != maxy
   k. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_MinZ(geom) != minz
   l. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_MaxZ(geom) != maxz
   m. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_MinM(geom) != minm
   n. SELECT ‘Fail’ FROM <gtype_test> WHERE ST_MaxM(geom) != maxm

3) Pass if no ‘Fail’ selected from step 2

Reference: Clause 2.1.3.2.1 Req 16:
Test Type: Capability

A.2.1.3 Geometry Types
A.2.1.3.1 Data
A.2.1.3.1.1 Core Types
**Test Case ID:** /opt/features/geometry_encoding/data/core_types_existing_sparse_data
**Test Purpose:** Verify that existing basic simple feature geometries are stored in valid GeoPackageBinary format encodings.

**Test Method:**

1) SELECT table_name FROM gpkg_geometry_columns
2) Not testable if returns an empty result set
3) SELECT table_name AS tn, column_name AS cn FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type = ‘features’),
4) Fail if returns an empty result set
5) For each row from step 3
   a. SELECT cn FROM tn;
   b. For each row from step a, if bytes 2-5 of cn.wkb as uint32 in endianness of gc.wkb byte 1 of cn from #1 are a geometry type value from Annex G Table 46, then
      i. Log cn.header values, wkb endianness and geometry type
      ii. If cn.wkb is not correctly encoded per ISO 13249-3 clause 5.1.46 then log fail
      iii. If cn.flags.E is 1 - 4 and some cn.wkbx is outside of cn.envelope.minx,maxx then log fail
      iv. If cn.flags.E is 1 - 4 and some gc.wkby is outside of cn.envelope.miny,maxy then log fail
      v. If cn.flags.E is 2,4 and some gc.wkbz is outside of cn.envelope.minz,maxz then log fail
      vi. If cn.flags.E is 3,4 and some gc.wkb.m is outside of cn.envelope.minm,maxm then log fail
      vii. If cn.flags.E is 5-7 then log fail
      viii. Otherwise log pass
6) Pass if log contains pass and no fails

**Reference:** Clause 2.1.4.1.1 Req 17
**Test Type:** Capability

**Test Case ID:** /opt/features/geometry_encoding/data/core_types_all_types_test_data
**Test Purpose:** Verify that all basic simple feature geometry types and options are stored in valid GeoPackageBinary format encodings.

**Test Method:**

1) Open GeoPackage that has feature geometry values of geometry type in Annex G, for an assortment of srs_ids, for an assortment of coordinate values, without and with z and / or m values, in both big and little endian encodings:
2) /opt/features/geometry_encoding/data/core_types_existing_sparse_data
3) Pass if log contains pass record for big and little endian GPB headers containing big and little endian WKBs for 0-1 envelope contents indicator codes for every geometry type value from Annex G Table 46 without and with z and/or m values.
4) Fail otherwise

**Reference:** Clause 2.1.4.1.1 Req 17:

**Test Type:** Capability

### A.2.1.4 Geometry Columns

### A.2.1.4.1 Data

#### A.2.1.4.1.1 Table Definition

**Test Case ID:** /opt/features/geometry_columns/data/table_def

**Test Purpose:** Verify that the gpkg_geometry_columns table exists and has the correct definition.

**Test Method:**

1) SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_geometry_columns'
2) Fail if returns an empty result set.
3) Pass if the column names and column definitions in the returned Create TABLE statement in the sql column value, including data type, nullability, default values and primary, foreign and unique key constraints match all of those in the contents of C.4 Table 22. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant.
4) Fail otherwise.

**Reference:** Clause 2.1.5.1.1 Req 18:

**Test Type:** Basic

#### A.2.1.4.1.2 Table Data Values

**Test Case ID:** /opt/features/geometry_columns/data/data_values_table_name

**Test Purpose:** Verify that the table_name column values in the gpkg_geometry_columns table are valid.

**Test Method:**

1) SELECT DISTINCT table_name FROM gpkg_geometry_columns
2) Not testable if returns an empty result set.
3) For each row from step 1
   a. Fail if table_name value is NULL
4) SELECT DISTINCT ggc.table_name AS ggc_table, gc.table_name AS gc_table_name FROM gpkg_geometry_columns AS ggc LEFT OUTER JOIN geopackage_contents AS gc ON ggc.table_name = gc.table_name
5) For each row from step 4
   a. Fail if ggc.table_name != gc.table_name
6) Pass if no fails.

**Reference:** Clause 2.1.5.1.2 Req 19:

**Test Type:** Capability

**Test Case ID:** /opt/features/geometry_columns/data/data_values_column_name
**Test Purpose:** Verify that the column_name column values in the gpkg_geometry_columns table are valid.

**Test Method:**

1) SELECT table_name, column_name FROM gpkg_geometry_columns
2) Not testable if returns an empty result set
3) For each row from step 1
   a. PRAGMA table_info(table_name)
   b. Fail if gpkg_geometry_columns.column_name value does not equal a name column value returned by PRAGMA table_info.

**Pass if no fails. Reference:** Clause 2.1.5.1.2 Req 20:

**Test Type:** Capability

---

**Test Case ID:** /opt/features/geometry_columns/data/data_values_geometry_type_name

**Test Purpose:** Verify that the geometry_type_name column values in the gpkg_geometry_columns table are valid.

**Test Method:**

1) SELECT DISTINCT geometry_type_name from gpkg_geometry_columns
2) Not testable if returns an empty result set
3) For each row from step 1
   a. Fail if a returned geometry_type value is not in Table 46 or Table 47 in Annex G
4) Pass if no fails.

**Reference:** Clause 2.1.5.1.2 Req 21:

**Test Type:** Capability

---

**Test Case ID:** /opt/features/geometry_columns/data/data_values_srs_id

**Test Purpose:** Verify that the gpkg_geometry_columns table srs_id column values are defined in the gpkg_spatial_ref_sys table.

**Test Method:**

1) SELECT srs_id FROM gpkg_geometry_columns
2) Not testable if returns an empty result set
3) SELECT srs_id FROM gpkg_geometry_columns WHERE srs_id NOT IN (SELECT srs_id FROM gpkg_spatial_ref_sys)
4) Fail if does not return an empty result set
5) Pass otherwise.

**Reference:** Clause 2.1.5.1.2 Req 22:

**Test Type:** Capability

---

**Test Case ID:** /opt/features/geometry_columns/data/data_values_z

**Test Purpose:** Verify that the gpkg_geometry_columns table z column values are valid.

**Test Method:**
1) SELECT z FROM gpkg_geometry_columns
2) Not testable if returns an empty result set
3) SELECT z FROM gpkg_geometry_columns WHERE z NOT IN (1,2,3)
4) Fail if does not return an empty result set
5) Pass otherwise.

**Reference:** Clause 2.1.5.1.2 Req 23:
**Test Type:** Capability

**Test Case ID:** /opt/features/geometry_columns/data/data_values_m
**Test Purpose:** Verify that the gpkg_geometry_columns table m column values are valid.
**Test Method:**
1) SELECT m FROM gpkg_geometry_columns
2) Not testable if returns an empty result set
3) SELECT m FROM gpkg_geometry_columns WHERE m NOT IN (1,2,3)
4) Fail if does not return an empty result set
5) Pass otherwise.

**Reference:** Clause 2.1.5.1.2 Req 24:
**Test Type:** Capability

### A.2.1.5 Vector Features User Data Tables
#### A.2.1.5.1 Data
##### A.2.1.5.1.1 Table Definition

**Test Case ID:** /opt/features/vector_features/data/feature_table_integer_primary_key
**Test Purpose:** Verify that every vector features user data table has an integer primary key.
**Test Method:**
1) SELECT table_name FROM gpkg_contents WHERE data_type = ‘features’
2) Not testable if returns an empty result set
3) For each row from step 1
   a. PRAGMA table_info(table_name)
   b. Fail if returns an empty result set
   c. Fail if result set does not contain one row where the pk column value is 1 and the not null column value is 1 and the type column value is “INTEGER”
4) Pass if no fails.

**Reference:** Clause 2.1.6.1.1 Req 25:
**Test Type:** Basic

**Test Case ID:** /opt/features/vector/features/data/feature_table_one_geometry_column
**Test Purpose:** Verify that every vector features user data table has one geometry column.
**Test Method:**
1) SELECT table_name FROM gpkg_contents WHERE data_type = ‘features’
2) Not testable if returns an empty result set
3) For each row table name from step 1
   a. SELECT column_name from gpkg_geometry_columns where table_name = row table name
   b. Fail if returns more than one column name
4) Pass if no fails

   Reference: Clause 2.1.6.1.1 Req 26:
   Test Type: Capability

A.2.1.5.1.2 Table Data Values

   Test Case ID: /opt/features/vector_features/data/data_values_geometry_type
   Test Purpose: Verify that the geometry type of feature geometries are of the type or are assignable for the geometry type specified by the gpkg_geometry_columns table geometry_type_name column value.
   Test Method:

   1) SELECT table_name AS tn, column_name AS cn, geometry_type_name AS gt_name FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type = 'features')
   2) Not testable if returns an empty result set
   3) For each row from step 1
      a. SELECT DISTINCT ST_GeometryType(cn) FROM tn
      b. For each row actual_type_name from step a
         i. SELECT GPKG_IsAssignable(geometry_type_name, actual_type_name)
         ii. Fail if any returned 0
   4) Pass if no fails

   Reference: Clause 2.1.6.1.2 Req 27:
   Test Type: Capability

   Test Case ID: /opt/features/vector_features/data/tata_value_geometry_srs_id
   Test Purpose: Verify the the srs_id of feature geometries are the srs_id specified for the gpkg_geometry_columns table srs_id column value.
   Test Method:

   1) SELECT table_name AS tn, column_name AS cn, srs_id AS gc_srs_id FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type = 'features')
   2) Not testable if returns an empty result set
   3) For each row from step 1
      a. SELECT DISTINCT st_srid(cn) FROM tn
      b. For each row from step a
         i. Fail if return value not equal to gc_srs_id
   4) Pass if no fails

   Reference: Clause 2.1.6.1.2 Req 28:
   Test Type: Capability
A.2.2 Tiles
A.2.2.1 Contents
A.2.2.1.1 Data
A.2.2.1.1.1 Contents Table –Tiles Row

Test Case ID: /opt/tiles/contents/data/tiles_row

Test Purpose: Verify that the gpkg_contents table_name value table exists and is apparently a tiles table for every row with a data_type column value of “tiles”.

Test Method:

1) SELECT table_name FROM gpkg_contents WHERE data_type = “tiles”
2) Fail if returns empty result set
3) For each row from step 1
   a) PRAGMA table_info(table_name)
   b) Fail if returns an empty result set
   c) Fail if result set does not contain one row where the pk column value is 1 and the not null column value is 1 and the type column value is “INTEGER” and the name column value is “id”
   d) Fail if result set does not contain four other rows where the name column values are “zoom_level”, “tile_column”, “tile_row”, and “tile_data”.
4) Pass if no fails.

Reference: Clause 2.2.2.1.1 Req 29:

Test Type: Capability

A.2.2.2 Zoom Levels
A.2.2.2.1 Data
A.2.2.2.1.1 Zoom Times Two

Test Case ID: /opt/tiles/zoom_levels_data_zoom_times_two

Test Purpose: Verify that by default zoom level pixel sizes for tile matrix user data tables vary by powers of 2 between adjacent zoom levels in the tile matrix metadata table.

Test Method:

1) SELECT CASE
   WHEN (SELECT tbl_name FROM sqlite_master WHERE tbl_name = 'gpkg_extensions') = 'gpkg_extensions' THEN
   (SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles' AND table_name NOT IN
   (SELECT table_name from gpkg_extensions WHERE extension_name = 'gpkg_zoom_other'))
   ELSE (SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles')
   END;
2) Not testable if returns empty result set
3) For each row table_name from step 1
   a. SELECT zoom_level, pixel_x_size, pixel_y_size FROM tile_matrix_metadata
      WHERE table_name = selected table name ORDER BY zoom_level ASC
b. Not testable if returns empty result set, or only one row

c. Not testable if there are not two rows with adjacent zoom levels

d. Fail if any pair of rows for adjacent zoom levels have pixel_x_size or pixel_y_size values that differ by other than powers of two

4) Pass if no fails

Reference: Clause 2.2.3.1.1 Req 30:
Test Type: Capability

A.2.2.3 Tile Encoding PNG
A.2.2.3.1 Data
A.2.2.3.1.1 MIME Type PNG

Test Case ID: /opt/tiles/tiles_encoding/data/mime_type_png
Test Purpose: Verify that a tile matrix user data table that contains tile data that is not MIME type image/jpeg by default contains tile data in MIME type image/png.

Test Method:

1) SELECT CASE
   WHEN (SELECT tbl_name FROM sqlite_master WHERE tbl_name = 'gpkg_extensions') = 'gpkg_extensions' THEN
      (SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles' AND table_name NOT IN
      (SELECT table_name from gpkg_extensions WHERE extension_name IN
      ('gpkg_webp','gpkg_tiff','gpkg_nitf')))
   ELSE (SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles')
   END;

2) Not testable if returns empty result set

3) For each row tbl_name from step 1
   a. SELECT tile_data FROM tbl_name
      b. For each row tile_data from step a
         i. Pass if tile data in MIME type image/png

4) Fail if no passes

Reference: Clause 2.2.4.1.1 Req 31:
Test Type: Capability

A.2.2.4 Tile Encoding JPEG
A.2.2.4.1 Data
A.2.2.4.1.1 MIME Type JPEG

Test Case ID: /opt/tiles/tiles_encoding/data/mime_type_jpeg
Test Purpose: Verify that a tile matrix user data table that contains tile data that is not MIME type image/png by default contains tile data in MIME type image/jpeg.

Test Method:

1) SELECT CASE
WHEN (SELECT tbl_name FROM sqlite_master WHERE tbl_name = 'gpkg_extensions') = 'gpkg_extensions' THEN
(SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles' AND
 table_name NOT IN
 (SELECT table_name from gpkg_extensions WHERE extension_name IN
 ('gpkg_webp','gpkg_tiff','gpkg_nitf')))
ELSE (SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles')
END;

2) Not testable if returns empty result set
3) For each row tbl_name from step 1
   a. SELECT tile_data FROM tbl_name
   b. For each row tile_data from step a
      i. Pass if tile data in MIME type image/jpeg
4) Fail if no passes

Reference: Clause 2.2.5.1.1 Req 32:
Test Type: Capability

A.2.2.5 Tile Matrix Metadata
A.2.2.5.1 Data
A.2.2.5.1.1 Table Definition

Test Case ID: /opt/tiles/tile_matrix_metadata/data/table_def
Test Purpose: Verify that the gpkg_tile_matrix_metadata table exists and has the correct definition.
Test Method:

1) SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_tile_matrix_metadata'
2) Fail if returns an empty result set.
3) Pass if the column names and column definitions in the returned CREATE TABLE statement in the sql column value, including data type, nullability, default values and primary, foreign and unique key constraints match all of those in the contents of C.7 Table 25. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant.
4) Fail otherwise.

Reference: Clause 2.2.5.1.1 Req 32:
Test Type: Capability

A.2.2.5.1.2 Table Data Values

Test Case ID: /opt/tiles/tile_matrix_metadata/data_values_table_name
Test Purpose: Verify that values of the gpkg_tile_matrix_metadata table_name column reference values in the gpkg_contents table_name column for rows with a data type of “tiles”.
Test Method:
1) SELECT table_name FROM gpkg_tile_matrix_metadata
2) Not testable if returns an empty result set
3) SELECT table_name FROM gpkg_tile_matrix_metadata tmm WHERE table_name NOT IN (SELECT table_name FROM gpkg_contents gc WHERE tmm.table_name = gc.table_name)
4) Fail if result set contains any rows
5) Pass otherwise

Reference: Clause 2.2.6.1.2 Req 34:
Test Type: Capability

Test Case ID: /opt/tiles/tile_matrix_metadata/data/data_values_zoom_level_rows
Test Purpose: Verify that the gpkg_tile_matrix_metadata table contains a row record for each zoom level that contains one or more tiles in each tile matrix user data table.

Test Method:
1) SELECT table_name AS <user_data_tiles_table> from gpkg_contents where data_type = ‘tiles’
2) Not testable if returns an empty result set
3) For each row from step 1
   a. SELECT DISTINCT gtmm.zoom_level AS gtmm_zoom, udt.zoom_level AS udt_zoom FROM tile_matrix_metadata AS gtmm
      LEFT OUTER JOIN <user_data_tiles_table> AS udt ON udt.zoom_level = gtmm.zoom_level AND gtmm.t_table_name = '<user_data_tiles_table>'
   b. Fail if any gtmm_zoom column value in the result set is NULL
4) Pass if no fails

Reference: Clause 2.2.6.1.2 Req 35:
Test Type: Capability

Test Case ID: /opt/tiles/tile_matrix_metadata/data/data_values_zoom_level
Test Purpose: Verify that zoom level column values in the gpkg_tile_matrix_metadata table are not negative.

Test Method:
1) SELECT zoom_level FROM gpkg_tile_matrix_metadata
2) Not testable if returns an empty result set
3) SELECT min(zoom_level) FROM gpkg_tile_matrix_metadata.
4) Fail if less than 0.
5) Pass otherwise.

Reference: Clause 2.2.6.1.2 Req 36:
Test Type: Capability

Test Case ID: /opt/tiles/tile_matrix_metadata/data/data_values_matrix_width
**Test Purpose:** Verify that the matrix_width values in the gpkg_tile_matrix_metadata table are valid.

**Test Method:**
1) SELECT matrix_width FROM gpkg_tile_matrix_metadata
2) Not testable if returns an empty result set
3) SELECT min(matrix_width) FROM gpkg_tile_matrix_metadata.
4) Fail if less than 1.
5) Pass otherwise.

**Reference:** Clause 2.2.6.1.2 Req 37:

**Test Type:** Capability

---

**Test Case ID:** /opt/tiles/tile_matrix_metadata/data/data_values_matrix_height

**Test Purpose:** Verify that the matrix_height values in the gpkg_tile_matrix_metadata table are valid.

**Test Method:**
1) SELECT matrix_height FROM gpkg_tile_matrix_metadata
2) Not testable if returns an empty result set
3) SELECT min(matrix_height) FROM gpkg_tile_matrix_metadata.
4) Fail if less than 1.
5) Pass otherwise.

**Reference:** Clause 2.2.6.1.2 Req 38:

**Test Type:** Capability

---

**Test Case ID:** /opt/tiles/tile_matrix_metadata/data/data_values_tile_width

**Test Purpose:** Verify that the tile_width values in the gpkg_tile_matrix_metadata table are valid.

**Test Method:**
1) SELECT tile_width FROM gpkg_tile_matrix_metadata
2) Not testable if returns an empty result set
3) SELECT min(tile_width) FROM gpkg_tile_matrix_metadata.
4) Fail if less than 1.
5) Pass otherwise.

**Reference:** Clause 2.2.6.1.2 Req 39:

**Test Type:** Capability

---

**Test Case ID:** /opt/tiles/tile_matrix_metadata/data/data_values_tile_height

**Test Purpose:** Verify that the tile_height values in the gpkg_tile_matrix_metadata table are valid.

**Test Method:**
1) SELECT tile_height FROM gpkg_tile_matrix_metadata
Test Case ID: /opt/tiles/tile_matrix_metadata/data/data_values_pixel_x_size
Test Purpose: Verify that the pixel_x_size values in the gpkg_tile_matrix_metadata table are valid.
Test Method:
   1) SELECT pixel_x_size FROM gpkg_tile_matrix_metadata
   2) Not testable if returns an empty result set
   3) SELECT min(pixel_x_size) FROM gpkg_tile_matrix_metadata.
   4) Fail if less than 0.
   5) Pass otherwise.
Reference: Clause 2.2.6.1.2 Req 41:
Test Type: Capability

Test Case ID: /opt/tiles/tile_matrix_metadata/data/data_values_pixel_y_size
Test Purpose: Verify that the pixel_y_size values in the gpkg_tile_matrix_metadata table are valid.
Test Method:
   1) SELECT pixel_y_size FROM gpkg_tile_matrix_metadata
   2) Not testable if returns an empty result set
   3) SELECT min(pixel_y_size) FROM gpkg_tile_matrix_metadata.
   4) Fail if less than 0.
   5) Pass otherwise.
Reference: Clause 2.2.6.1.2 Req 42:
Test Type: Capability

Test Case ID: /opt/tiles/tile_matrix_metadata/data/data_values_pixel_size_sort
Test Purpose: Verify that the pixel_x_size and pixel_y_size column values for zoom level column values in a gpkg_tile_matrix_metadata table sorted in ascending order are sorted in descending order, showing that lower zoom levels are zoomed “out”.
Test Method:
   1) SELECT table_name FROM gpkg_contents WHERE data_type = ‘tiles’
   2) Not testable if returns empty result set
   3) For each row table_name from step 1
a. SELECT zoom_level, pixel_x_size, pixel_y_size from gpkg_tile_matrix_metadata WHERE table_name = row table name ORDER BY zoom_level ASC
b. Not testable if returns empty result set
c. Fail if pixel_x_sizes are not sorted in descending order
d. Fail if pixel_y_sizes are not sorted in descending order

4) Pass if testable and no fails

Reference: Clause 2.2.6.1.2 Req 43:
Test Type: Capability

A.2.2.6 Tile Matrix User Data
A.2.2.6.1 Data
A.2.2.6.1.1 Table Definition

Test Case ID: /opt/tiles/tile_matrix/data/table_def
Test Purpose: Verify that multiple tile matrix sets are stored in different tiles tables with unique names containing the required columns.

Test Method:

1) SELECT COUNT(table_name) FROM gpkg_contents WHERE data_type = "tiles"
2) Fail if less than 2
3) SELECT table_name FROM gpkg_contents WHERE data_type = "tiles"
4) For each row from step 3
   a. PRAGMA table_info(table_name)
   b. Fail if returns an empty result set
   c. Fail if result set does not contain one row where the pk column value is 1 and the not null column value is 1 and the type column value is “INTEGER” and the name column value is “id”
   d. Fail if result set does not contain four other rows where the name column values are “zoom_level”, “tile_column”, “tile_row”, and “tile_data”.
5) Pass if no fails

Reference: Clause 2.2.7.1.1 Req 44:
Test Type: Basic

A.2.2.6.1.2 Table Data Values

Test Case ID: /opt/tiles/tile_matrix/data/data_values_zoom_levels
Test Purpose: Verify that the zoom level column values in each tile matrix user data table are within the range of zoom levels defined by rows in the tile_matrix_metadata table.

Test Method:

1) SELECT DISTINCT table_name AS <user_data_tiles_table> FROM gpkg_tile_matrix_metadata
2) Not testable if returns an empty result set
3) For each row <user_data_tiles_table> from step 1
   a. SELECT zoom_level FROM <user_data_tiles_table>
   b. If result set not empty
      i. SELECT MIN(gtmm.zoom_level) AS min_gtmm_zoom,
         MAX(gtmm.zoom_level) AS max_gtmm_zoom FROM
gpkg_tile_matrix_metadata WHERE table_name = 
<user_data_tiles_table>
ii. SELECT id FROM <user_data_tiles_table> WHERE zoom_level <
min_gtmm_zoom
iii. Fail if result set not empty
iv. SELECT id FROM <user_data_tiles_table> WHERE zoom_level >
max_gtmm_zoom
v. Fail if result set not empty
vi. Log pass otherwise
4) Pass if logged pass and no fails

Reference: Clause 2.2.7.1.2 Req 45:
Test Type: Capability

Test Case ID: /opt/tiles/tile_matrix/data/data_values_tile_column
Test Purpose: Verify that the tile_column column values for each zoom level value in each
tile matrix user data table are within the range of columns defined by rows in the
tile_matrix_metadata table.
Test Method:
1) SELECT DISTINCT table_name AS <user_data_tiles_table> FROM
   gpkg_tile_matrix_metadata
2) Not testable if returns an empty result set
3) For each row <user_data_tiles_table> from step 1
   a. SELECT DISTINCT gtmm.zoom_level AS gtmm_zoom, gtmm.matrix_width
      AS gtmm_width, udt.zoom_level AS udt_zoom, udt.tile_column AS
      udt_column FROM tile_matrix_metadata AS gtmm LEFT OUTER JOIN
      <user_data_tiles_table> AS udt ON udt.zoom_level = gtmm.zoom_level AND
      gtmm.t_table_name = '<user_data_tiles_table>' AND (udt_column < 0 OR
      udt_column > (gtmm_width - 1))
   b. Fail if any udt_column value in the result set is not NULL
   c. Log pass otherwise
4) Pass if logged pass and no fails

Reference: Clause 2.2.7.1.2 Req 46:
Test Type: Capability

Test Case ID: /opt/tiles/tile_matrix_data/data_values_tile_row
Test Purpose: Verify that the tile_row column values for each zoom level value in each tile
matrix user data table are within the range of rows defined by rows in the
tile_matrix_metadata table.
Test Method:
1) SELECT DISTINCT table_name AS <user_data_tiles_table> FROM
   gpkg_tile_matrix_metadata
2) Not testable if returns an empty result set
3) For each row <user_data_tiles_table> from step 1
a. SELECT DISTINCT gtmm.zoom_level AS gtmm_zoom, gtmm.matrix_height AS gtmm_height, udt.zoom_level AS udt_zoom, udt.tile_row AS udt_row
FROM tile_matrix_metadata AS gtmm LEFT OUTER JOIN <user_data_tiles_table> AS udt ON udt.zoom_level = gtmm.zoom_level AND gtmm.t_table_name = '<use_r_data_tiles_table>' AND (udt_row < 0 OR udt_row > (gtmm_height - 1))
b. Fail if any udt_row value in the result set is not NULL
c. Log pass otherwise

4) Pass if logged pass and no fails

Reference: Clause 2.2.7.1.2 Req 47:
Test Type: Capability

A.2.3 Schema
A.2.3.1 Data Columns
A.2.3.1.1 Data
A.2.3.1.1.1 Table Definition

Test Case ID: /opt/schema/data_columns/data_table_def
Test Purpose: Verify that the gpkg_data_columns table exists and has the correct definition.
Test Method:

1) SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_data_columns'
2) Fail if returns an empty result set
3) Pass if column names and column definitions in the returned CREATE TABLE statement in the sql column value, including data type, nullability, default values and primary, foreign and unique key constraints match all of those in the contents of C.1 Table 32. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant.
4) Fail otherwise.

Reference: Clause 2.3.1.1.1 Req 48:
Test Type: Basic

A.2.3.1.1.2 Data Values

Test Case ID: /opt/schema/data_columns/data/data_values_table_name
Test Purpose: Verify that values of the gpkg_data_columns table_name column reference values in the gpkg_contents table_name column.
Test Method:

1) SELECT table_name FROM gpkg_data_columns
2) Not testable if returns an empty result set
3) SELECT table_name FROM gpkg_data_columns gdc WHERE table_name NOT IN (SELECT table_name FROM gpkg_contents gc WHERE gdc.table_name = gc.t_table_name)
4) Fail if result set contains any rows
5) Pass otherwise

Reference: Clause 2.3.1.1.2 Req 49:
Test Type: Capability

Test Case ID: /opt/schema/data_columns/data/data_values_column_name
Test Purpose: Verify that for each gpkg_data_columns row, the column_name value is the name of a column in the table_name table.
Test Method:

1) SELECT table_name, column_name FROM gpkg_data_columns
2) Not testable if returns an empty result set
3) For each row from step 1
   a. PRAGMA table_info(table_name)
   b. Fail if gpkg_data_columns.column_name value does not equal a name column value returned by PRAGMA table_info.
4) Pass if no fails.
Reference: Clause 2.3.1.1.2 Req 50:
Test Type: Capability

A.2.4 Metadata
A.2.4.1 Metadata Table
A.2.4.1.1 Data
A.2.4.1.1.1 Table Definition
Test Case ID: /opt/metadata/metadata/data/table_def
Test Purpose: Verify that the gpkg_metadata table exists and has the correct definition.
Test Method:

1) SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_metadata'
2) Fail if returns an empty result set.
3) Pass if the column names and column definitions in the returned Create TABLE statement in the sql column value, including data type, nullability, default values and primary, foreign and unique key constraints match all of those in the contents of Table 33. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant.
4) Fail otherwise.
Reference: Clause 2.4.2.1.1 Req 51:
Test Type: Basic
A.2.4.1.1.2 Table Data Values
Test Case ID: /opt/metadata/metadata/data/data_values_md_scope
Test Purpose: Verify that each of the md_scope column values in a gpkg_metadata table is one of the name column values from Table 11 in clause 2.4.2.1.2.
Test Method:

1) SELECT md_scope FROM gpkg_metadata
2) Not testable if returns an empty result set
3) For each row returned from step 1
   a. Fail if md_scope value not one of the name column values from Table 11 in clause 2.4.2.1.2
4) Pass if no fails

Reference: Clause 2.4.2.1.2 Req 52:
Test Type: Capabilities

A.2.4.2 Metadata Reference Table
A.2.4.2.1 Data
A.2.4.2.1.1 Table Definition

Test Case ID: /opt/metadata/metadata_reference_data_table_def
Test Purpose: Verify that the gpkg_metadata_reference table exists and has the correct definition.
Test Method:
   1) SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_metadata_reference'
   2) Fail if returns an empty result set.
   3) Pass if the column names and column definitions in the returned Create TABLE statement in the sql column value, including data type, nullability, default values and primary, foreign and unique key constraints match all of those in the contents of Table 34. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant.
   4) Fail otherwise.

Reference: Clause 2.4.3.1.1 Req 53:
Test Type: Basic

A.2.4.2.1.2 Data Values

Test Case ID: /opt/metadata/metadata_reference/data/data_values_reference_scope
Test Purpose: Verify that gpkg_metadata_reference table reference_scope column values are valid.
Test Method:
   1) SELECT reference_scope FROM gpkg_metadata_reference
   2) Not testable if returns an empty result set
   3) SELECT reference_scope FROM gpkg_metadata_reference WHERE reference_scope NOT IN ('geopackage','table','column','row','row/col')
   4) Fail if does not return an empty result set
   5) Pass otherwise.

Reference: Clause 2.4.3.1.2 Req 54:
Test Type: Capability

Test Case ID: /opt/metadata/metadata_reference/data/data_values_table_name
**Test Purpose:** Verify that gpkg_metadata_reference table_name column values are NULL for rows with reference_scope values of ‘geopackage’, and reference gpkg_contents table_name values for all other reference_scope values.

**Test Method:**

1) SELECT table_name FROM gpkg_metadata_reference
2) Not testable if returns an empty result set
3) SELECT table_name FROM gpkg_metadata_reference WHERE reference_scope = 'geopackage'
4) Fail if result set contains any non-NULL values
5) SELECT table_name FROM metadata_reference WHERE reference_scope != 'geopackage' AND table_name NOT IN (SELECT table_name FROM gpkg_contents)
6) Fail if result set is not empty
7) Pass otherwise.

**Reference:** Clause 2.4.3.1.2 Req 55:

**Test Purpose:** Verify that gpkg_metadata_reference column_name column values are NULL for rows with reference scope values of ‘geopackage’, ‘table’, or ‘row’, and contain the name of a column in table_name table for other reference scope values.

**Test Method:**

1) SELECT column_name FROM gpkg_metadata_reference
2) Not testable if returns an empty result set
3) SELECT column_name FROM gpkg_metadata_reference WHERE reference_scope IN ('geopackage', 'table', 'row')
4) Fail if result set contains any non-NULL values
5) SELECT <table_name>, <column_name> FROM metadata_reference WHERE reference_scope NOT IN ('geopackage', 'table', 'row')
6) For each row from step 5
   a. SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = '<table_name>'
   b. Fail if returns an empty result set.
   c. Fail if the one of the column names in the returned sql Create TABLE statement is not <column_name>
   d. Log pass otherwise
7) Pass if logged pass and no fails.

**Reference:** Clause 2.4.3.1.2 Req 56:

**Test Case ID:** /opt/metadata/metadata_reference/data/data_values_row_id_value
**Test Purpose:** Verify that gpkg_metadata_reference row_id_value column values are NULL for rows with reference scope values of ‘geopackage’, ‘table’, or ‘row’, and contain the ROWID of a row in the table_name for other reference scope values.

**Test Method:**

1) SELECT row_id_value FROM gpkg_metadata_reference
2) Not testable if returns an empty result set
3) SELECT row_id_value FROM gpkg_metadata_reference WHERE reference_scope IN ('geopackage', 'table', 'row')
4) Fail if result set contains any non-NULL values
5) For each SELECT <table_name>, <row_id_value> FROM gpkg_metadata_reference WHERE reference_scope NOT IN ('geopackage', 'table', 'row')
6) For each row from step 5
   a. SELECT * FROM <table_name> WHERE ROWID = <row_id_value>
   b. Log pass otherwise
7) Pass if logged pass and no fails.

**Reference:** Clause 2.4.3.1.2 Req 57:
**Test Type:** Capability

**Test Case ID:** /opt/metadata/metadata_reference/data/data_values_timestamp
**Test Purpose:** Verify that every gpkg_metadata_reference table row timestamp column value is in ISO 8601 UTC format.

**Test Method:**

1) SELECT timestamp from gpkg_metadata_reference.
2) Not testable if returns an empty result set
3) For each row from step 1
   a. Fail if format of returned value does not match yyyy-mm-ddThh:mm:ss.hhhZ
   b. Log pass otherwise
4) Pass if logged pass and no fails.

**Reference:** Clause 2.4.3.1.2 Req 58:
**Test Type:** Capability

**Test Case ID:** /opt/metadata/metadata_reference/data/data_values_md_file_id
**Test Purpose:** Verify that every gpkg_metadata_reference table row md_file_id column value is an id column value from the gpkg_metadata table.

**Test Method:**

1) SELECT md_file_id FROM gpkg_metadata_reference
2) Not testable if returns an empty result set
3) SELECT gmr.md_file_id, gm.id FROM gpkg_metadata_reference AS gmr
4) LEFT OUTER JOIN gpkg_metadata gm ON gmr.md_file_id = gm.id
5) Fail if result set is empty
6) Fail if any result set gm.id values are NULL
7) Pass otherwise

**Reference:** Clause 2.4.3.1.2 Req 59:
**Test Type:** Capability

**Test Case ID:** /opt/metadata/metadata_reference/data/data_values_md_parent_id
**Test Purpose:** Verify that every gpkg_metadata_reference table row md_parent_id column value that is not null is an id column value from the gpkg_metadata_table that is not equal to the md_file_id column value for that row.

**Test Method:**
1) SELECT md_file_id FROM gpkg_metadata_reference
2) Not testable if returns an empty result set
3) SELECT gmr.md_file_id, gmr.md_parent_id
4) FROM gpkg_metadata_reference AS gmr
5) WHERE gmr.md_file_id == gmr.md_parent_id
6) Fail if result set is not empty
7) SELECT gmr.md_file_id, gmr.md_parent_id, gm.id
8) FROM gpkg_metadata_reference AS gmr
9) LEFT OUTER JOIN gpkg_metadata gm ON gmr.md_parent_id =gm.id
10) Fail if any result set gm.id values are NULL
11) Pass otherwise

**Reference:** Clause 2.4.3.1.2 Req 60:
**Test Type:** Capability

---

**A.2.5 Extension Mechanism**

**A.2.5.1 Extensions**

**A.2.5.1.1 Data**

**A.2.5.1.1.1 Table Definition**

**Test Case ID:** /opt/extension_mechanism/extensions/data/table_def
**Test Purpose:** Verify that a gpkg_extensions table exists and has the correct definition.

**Test Method:**
1) SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_extensions'
2) Fail if returns an empty result set.
3) Pass if the column names and column definitions in the returned Create TABLE statement in the sql column value, including data type, nullability, default values and primary, foreign and unique key constraints match all of those in the contents of Table 23. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant.
4) Fail otherwise.

**Reference:** Clause 2.6.1.1.1 Req 61:
**Test Type:** Basic
A.2.5.1.1.2 Table Data Values

Test Case ID: /opt/extension_mechanism/extensions/data/data_values_table_name
Test Purpose: Verify that the table_name column values in the gpkg_extensions table are valid.

Test Method:

1) SELECT table_name, column_name FROM gpkg_extensions
2) Not testable if returns an empty result set
3) For each row from step one
   a. Fail if table_name value is NULL and column_name value is not NULL.
   b. SELECT DISTINCT ge.table_name AS ge_table, sm.tbl_name
      FROM gpkg_extensions AS ge LEFT OUTER JOIN sqlite_master AS sm ON
      ge.table_name = sm.tbl_name
   c. Log pass if every row ge.table_name = sm.tbl_name (MAY both be NULL).
4) Pass if logged pass and no fails.

Reference: Clause 2.6.1.1.2 Req 63:
Test Type: Capability

Test Case ID: /opt/extension_mechanism/extensions/data/data_values_column_name
Test Purpose: Verify that the column_name column values in the gpkg_extensions table are valid.

Test Method:

1) SELECT table_name, column_name FROM gpkg_extensions
2) Not testable if returns an empty result set
3) SELECT table_name, column_name FROM gpkg_extensions WHERE table_name
    IS NOT NULL AND column_name IS NOT NULL
4) Pass if returns an empty result set
5) For each row from step 3
   a. PRAGMA table_info(table_name)
   b. Fail if gpkg_extensions.column_name value does not equal a name column value
      returned by PRAGMA table_info.
   c. Log pass otherwise
6) Pass if logged pass and no fails.

Reference: Clause 2.6.1.1.2 Req 63:
Test Type: Capability

Test Case ID: /opt/extension_mechanism/extensions/data/data_values_extension_name
Test Purpose: Verify that the extension_name column values in the gpkg_extensions table are valid.

Test Method:

1) SELECT extension_name FROM gpkg_extensions
2) Not testable if returns an empty result set
3) For each row returned from step 1
   a. Log pass if extension_name is one of those listed in Table 14
   b. Separate extension_name into <author> and <extension> at the first “_”
   c. Fail if <author> is “gpkg”
   d. Fail if <author> contains characters other than [a-zA-Z0-9]
   e. Fail if <extension> contains characters other than [a-zA-Z0-9_.]
   f. Log pass otherwise

4) Pass if logged pass and no fails.

Reference: Clause 2.6.1.1.2 Req 64:
Test Type: Capability

A.2.5.1.2 API
A.2.5.1.2.1 API GeoPackage SQLite Config

Test Case ID: /opt/extension_mechanism/extensions/api/api_geopackage_sqlite_config
Test Purpose: Verify that a GeoPackage SQLite Extension has the API GeoPackage SQLite Configuration compile and run time options.
Test Method:

1) SELECT sqlite_compileoption_used('SQLITE_OMIT_LOAD_EXTENSION')
2) Fail if returns 1
3) SELECT sqlite_compileoption_used('SQLITE_OMIT_VIRTUALTABLE')
4) Fail if returns 1
5) SELECT sqlite_compileoption_used('SQLITE_ENABLE_RTREE_')
6) Fail if returns 0
7) SELECT sqlite_compileoption_used('SQLITE_RTREE_INT_ONLY')
8) Fail if returns 1
9) Pass otherwise

Reference: Clause 2.6.1.2.1 Req 65:
Test Type: Basic

A.2.5.1.2.2 Safe GeoPackage SQLite Config

Test Case ID: /opt/extension_mechanism/extensions/api/safe_geopackage_sqlite_config
Test Purpose: Verify that a GeoPackage SQLite Extension has the Safe GeoPackage SQLite Configuration compile and run time options.
Test Method:

1) SELECT sqlite_compileoption_used('SQLITE_DEFAULT_FOREIGN_KEYS')
2) Fail if returns 0
3) SELECT sqlite_compileoption_used('SQLITE_OMIT_FOREIGN_KEY')
4) Fail if returns 1
5) PRAGMA foreign_keys
6) Fail if returns 0
7) SELECT sqlite_compileoption_used('SQLITE_OMIT_INTEGRITY_CHECK')
8) Fail if returns 1
9) SELECT sqlite_compileoption_used('SQLITE_OMIT_SUBQUERY')
10) Fail if returns 1
11) SELECT sqlite_compileoption_used('SQLITE_OMIT_TRIGGER')
12) Fail if returns 1
13) Pass otherwise

Reference: Clause 2.6.1.2.2 Req 66:  
Test Type: Basic

A.3 Registered Extensions

Test Case ID: /reg_ext/all/author_name/not_gpkg/not_features_or_tiles  
Test Purpose: Verify that any table in a GeoPackage file subject to a registered extension with an author_name other than “gpkg” is not described by a gpkg_contents table row with a data_type value of ‘features’ or ‘tiles’.

Test Method:

1) /opt/extension_mechanism/extensions/data/table_def
2) Not testable if failed
3) SELECT table_name FROM geopackage_contents 
   WHERE data_type IN ('features','tiles') AND table_name IN 
   (SELECT table_name FROM gpkg_extensions WHERE 
   substr(lower(extension_name),1,4) != 'gpkg')
4) Fail if result set is not empty
5) Pass otherwise

Reference: Clause 3 Req 67:  
Test Type: Basic

A.3.1 Features
A.3.1.1 Geometry Encoding
A.3.1.1.1 Data
A.3.1.1.1.1 BLOB Format – Extensions Name

Test Case ID: /reg_ext/features/geometry_encoding/data/ext_name
Test Purpose: Verify that an extension name in the form <author_name>_geometry_encoding is defined for an author name other than “gpkg” for each geometry BLOB format other than GeoPackageBinary used in a GeoPackage file.

Test Method:

1) SELECT table_name, column_name FROM gpkg_geometry_columns WHERE 
   table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type NOT 
   IN ('features', 'tiles'))
2) Not testable if returns an empty result set
3) For each row table_name, column_name from step 1 
   a. SELECT result_set_column_name FROM result_set_table_name 
   b. Not testable if returns an empty result set
c. For each geometry column value from step a
   i. If the first three bytes of geometry column value are not “GPB”, then
      1. /opt/extension_mechanism/extensions/data/table_def
      2. Fail if failed
      3. SELECT extension_name FROM gpkg_extensions WHERE
         table_name = result_set_table_name AND column_name =
         result_set_column_name AND
         substr(lower(extension_name),1,4) != 'gpkg'
         a. Fail if returns an empty result set
         b. Separate extension_name into <author> and
            <extension> at the first “_”
         c. Fail if <extension> is not ‘_geometry_encoding’.
         d. Otherwise log pass
   4) Pass if logged pass and no fails

Reference: Clause 3.1.1.1.2 Req 68:
Test Type: Basic

A.3.1.1.1.2 BLOB Format – Extensions Row

Test Case ID: /reg_ext/features/geometry_encoding/data/ext_row
Test Purpose: Verify that the gpkg_extensions table contains a row with an extension_name
in the form <author_name>_geometry_encoding is defined for an author name other than
“gpkg” for each table_name and column_name that contain a geometry BLOB format other
Test Method:
   Same as /reg_ext/features/geometry_encoding/data/ext_name

Reference: Clause 3.1.1.1.3 Req 69:
Test Type: Capability

A.3.1.2 Geometry Types
A.3.1.2.1 Data
A.3.1.2.1.1 Extension Types

Test Case ID:
   /reg_ext/features/geometry_encoding/data/extension_types_existing_sparse_data
Test Purpose: Verify that existing extended non-linear geometry types are stored in valid
GeoPackageBinary format encodings.
Test Method:
   1) SELECT table_name FROM gpkg_geometry_columns
   2) Not testable if returns an empty result set
   3) SELECT table_name AS tn, column_name AS cn FROM gpkg_geometry_columns
      WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE
data_type = ‘features’),
   4) Fail if returns an empty result set
   5) For each row from step 3
a. SELECT cn FROM tn;
b. For each row from step a, if bytes 2-5 of cn.wkb as uint32 in endianness of gc.wkb byte 1 of cn from #1 are a geometry type value from Annex G Table 46, then
   i. Log cn.header values, wkb endianness and geometry type
   ii. If cn.wkb is not correctly encoded per ISO 13249-3 clause 5.1.46 then log fail
   iii. If cn.flags.E is 1 - 4 and some cn.wkbx is outside of cn.envelope.minx,maxx then log fail
   iv. If cn.flags.E is 1 - 4 and some gc.wkby is outside of cn.envelope.miny,maxy then log fail
   v. If cn.flags.E is 2,4 and some gc.wkb.z is outside of cn.envelope.minz,maxz then log fail
   vi. If cn.flags.E is 3,4 and some gc.wkb.m is outside of cn.envelope.minm,maxm then log fail
   vii. If cn.flags.E is 5-7 then log fail
   viii. Otherwise log pass

6) Log pass if log contains pass and no fails

Reference: Clause 3.1.2.1.1 Req 70:
Test Type: Capability

Test Case ID:
/reg_ext/features/geometry_encoding/data/extension_types_all_types_test_data
Test Purpose: Verify that all extended non-linear geometry types and options are stored in valid GeoPackageBinary format encodings.
Test Method:

1) Open GeoPackage that has feature geometry values of geometry type in Annex G, for an assortment of srs_ids, for an assortment of coordinate values, without and with z and / or m values, in both big and little endian encodings:
2) /reg_ext/features/geometry_encoding/data/extension_types_existing_sparse_data
3) Pass if log contains pass record for big and little endian GPB headers containing big and little endian WKBs for 0-1 envelope contents indicator codes for every geometry type value from Annex G Table 47 without and with z and/or m values.
4) Fail otherwise

Reference: Clause 3.1.2.1.1 Req 70:
Test Type: Capability

A.3.1.2.1.2 Geometry Types -- Extensions Name

Test Case ID: /reg_ext/features/geometry_encoding/data/extension_name
Test Purpose: Verify that an extension name in the form gpkg_geom_<gname> is defined for each <gname> extension geometry type from Annex G Table 47 used in a GeoPackage file.
Test Method:
1) SELECT table_name, column_name FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type == 'features'))
2) Not testable if result set is empty
3) For each row result set table_name, column_name from step 3
   a. SELECT result_set_column_name FROM result_set_table_name
   b. For each geometry column value from step a
      i. If the first three bytes of each geometry column value are “GPB”, then
         1. /opt/extension_mechanism/extensions/data/table_def
         2. Fail if failed
         3. SELECT ST_GeometryType(geometry column value) AS <gtype>;
         4. SELECT extension_name FROM gpkg_extensions WHERE table_name = result_set_table_name AND column_name = result_set_column_name AND extension_name = 'gpkg_geom_' || <gtype>
      a. Fail if result set is empty
      b. Log pass otherwise
4) Pass if logged pass and no fails

Reference: Clause 3.1.2.1.2 Req 71:
Test Type: Basic

A.3.1.2.1.3 Geometry Types -- Extensions Row

Test Case ID: /reg_ext/features/geometry_encoding/data/extension_row
Test Purpose: Verify that the gpkg_extensions table contains a row with an extension_name in the form gpkg_geom_<gname> for each table_name and column_name in the gpkg_geometry_columns table with a <gname> geometry_type_name.
Test Method:

/reg_ext/features/geometry_encoding/data/extension_name

Reference: Clause 3.1.2.1.3 Req 72:
Test Type: Capability

A.3.1.3 Spatial Indexes
A.3.1.3.1 Data
A.3.1.3.1.1 Spatial Indexes Implementation

Test Case ID: /reg_ext/features/spatial_indexes/implementation
Test Purpose: Verify the correct implementation of spatial indexes on feature table geometry columns.
Test Method:

1) SELECT table_name, column_name FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type == 'features'))
2) Not testable if result set is empty
3) For each row table_name, column_name from step 1
   a. SELECT sql FROM sqlite_master WHERE tbl_name = 'rtree_' ||
      result_set_table_name || '_' || result_set_column_name
   b. Not testable if result set is empty
   c. Fail if returned sql != ‘CREATE VIRTUAL TABLE rtree_' ||
      result_set_table_name || '_' || result_set_column_name || USING rtree(id, minx, maxx, miny, maxy)
   d. SELECT sql FROM sqlite_master WHERE type = 'trigger' AND tname =
      'rtree_' || result_set_table_name || '_' || result_set_column_name || '_insert'
   e. Fail if returned sql != result of populating insert triggers template in Table 39
      using result_set_table_name for <t> and result_set_column_name for <c>
   f. SELECT sql FROM sqlite_master WHERE type = 'trigger' AND name LIKE
      'rtree_' || result_set_table_name || '_' || result_set_column_name || '_update%'
   g. Fail if returned sql != result of populating 4 update triggers templates in Table 39
      using result_set_table_name for <t> and result_set_column_name for <c>
   h. SELECT sql FROM sqlite_master WHERE type='trigger' AND name =
      'rtree_' || result_set_table_name || '_' || result_set_column_name || '_delete'
   i. Fail if returned sql != result of populating delete trigger template in Table 39
      using result_set_table_name for <t> and result_set_column_name for <c>
   j. Log pass otherwise
4) Pass if logged pass and no fails

Reference: Clause 3.1.3.1.1 Req 73:
Test Type: Capability

A.3.1.3.1.2 Spatial Indexes – Extensions Name

Test Case ID: /reg_ext/features/spatial_indexes/extension_name
Test Purpose: Verify that the “gpkg_rtree_index” extension name is used to register spatial index extensions.
Test Method:

1) SELECT table_name, column_name FROM gpkg_geometry_columns WHERE
   table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type  ==
   'features'))
2) Not testable if result set is empty
3) For each row table_name, column_name from step 3
   a. SELECT sql FROM sqlite_master WHERE tbl_name = 'rtree_' ||
      result_set_table_name || '_' || result_set_column_name
   b. Not testable if returns an empty result set
   c. /opt/extension_mechanism/extensions/data/table_def
   d. Fail if failed
   e. SELECT extension_name from gpkg_extensions WHERE table_name =
      result_set_table_name AND column_name = result_set_column_name
   f. Log pass if result is “gpkg_rtree_index”
   g. Fail otherwise
4) Pass if logged pass and no fails

Reference: Clause 3.1.3.1.2 Req 74:
Test Type: Basic
A.3.1.3.1.3 Spatial Indexes – Extensions Row

Test Case ID: /reg_ext/features/spatial_indexes/extension_row
Test Purpose: Verify that spatial index extensions are registered using the “gpkg_rtree_index” name in the gpkg_extensions table.
Test Method:

/reg_ext/features/spatial_indexes/extension_row

Reference: Clause 3.1.3.1.3 Req 75:
Test Type: Capability

A.3.1.4 Geometry Type Triggers
A.3.1.4.1 Data
A.3.1.4.1.1 Geometry Type Triggers Implementation

Test Case ID: /reg_ext/features/geometry_type_triggers/implementation
Test Purpose: Verify that user feature data table geometry type triggers are implemented correctly.
Test Method:

1) SELECT table_name, column_name FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type == 'features'))
2) Not testable if returns an empty result set
3) For each row table_name, column_name from step 1
   a. SELECT sql FROM sqlite_master WHERE type = 'trigger' AND tbl_name = 'fgti_' || result_set_table_name || '_' || result_set_column_name
   b. Not testable if returns an empty result set
   c. Fail if sql != result of populating the first trigger template in Table 17 with <t> as result_set_table_name and <c> as result_set_column_name
   d. SELECT sql FROM sqlite_master WHERE type = 'trigger' AND tbl_name = 'fgtu_' || result_set_table_name || '_' || result_set_column_name
   e. Fail if sql != result of populating the second trigger template in Table 17 with <t> as result_set_table_name and <c> as result_set_column_name
   f. Log pass otherwise
4) Pass if logged pass and no fails

Reference: Clause 3.1.4.1.1 Req 76:
Test Type: Capability

A.3.1.4.1.2 Geometry Type Triggers – Extensions Name

Test Case ID: /reg_ext/features/geometry_type_triggers/extension_name
Test Purpose: Verify that the “gpkg_geometry_type_trigger” extension name is used to register geometry type triggers.
Test Method:
1) SELECT table_name, column_name FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type == 'features'))
2) Not testable if result set is empty
3) For each row table_name, column_name from step 1
   a. SELECT sql FROM sqlite_master WHERE type = 'trigger' AND tbl_name = 'fgti_' || result_set_table_name || '_' || result_set_column_name
   b. Not testable if result set is empty
   c. /opt/extension_mechanism/extensions/data/table_def
   d. Fail if failed
   e. SELECT extension_name from gpkg_extensions WHERE table_name = result_set_table_name AND column_name = result_set_column_name
   f. Log pass if result is “gpkg_geometry_type_trigger”
   g. Fail otherwise
4) Pass if logged pass and no fails

Reference: Clause 3.1.4.1.2 Req 77:
Test Type: Basic

A.3.1.4.1.3 Geometry Type Triggers – Extensions Row

Test Case ID: /reg_ext/features/geometry_type_triggers/extension_row
Test Purpose: Verify that geometry type triggers are registered using the “gpkg_geometry_type_trigger” extension name.
Test Method:
/regexp/features/geometry_type_triggers/extension_name

Reference: Clause 3.1.4.1.3 Req 78:
Test Type: Capability

A.3.1.5 SRS_ID Triggers
A.3.1.5.1 Data
A.3.1.5.1.1 SRS_ID Triggers – Implementation

Test Case ID: /reg_ext/features/srs_id_triggers/implementation
Test Purpose: Verify that user feature data table srs_id triggers are implemented correctly.
Test Method:

1) SELECT table_name, column_name FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type == 'features'))
2) Not testable if result set is empty
3) For each row table_name, column_name from step 1
   a. SELECT sql FROM sqlite_master WHERE type = 'trigger' AND tbl_name = 'fgsi_' || result_set_table_name || '_' || result_set_column_name
   b. Not testable if result set is empty
c. Fail if sql != result of populating the first trigger template in Table 18 with <t> as result_set_table_name and <c> as result_set_column_name

d. SELECT sql FROM sqlite_master WHERE type = ‘trigger’ AND tbl_name = ‘fgsu_' || result_set_table_name || '__' || result_set_column_name

e. Fail if sql != result of populating the second trigger template in Table 18 with <t> as result_set_table_name and <c> as result_set_column_name

f. Log pass otherwise

4) Pass if logged pass and no fails

References:

Test Type: Capability

A.3.1.5.1.2 SRS_ID Triggers – Extensions Name

Test Case ID: /reg_ext/features/srs_id_triggers/extension_name

Test Purpose: Verify that the “gpkg_srs_id_trigger” extension name is used to register srs_id triggers.

Test Method:

1) SELECT table_name, column_name FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type == ‘features’)

2) Not testable if result set is empty

3) For each row table_name, column_name from step 1

   h. SELECT sql FROM sqlite_master WHERE type = ‘trigger’ AND tbl_name = ‘fgsi_' || result_set_table_name || '__' || result_set_column_name

   i. Not testable if result set is empty

   j. /opt/extension_mechanism/extensions/data/table_def

   k. Fail if failed

   l. SELECT extension_name from gpkg_extensions WHERE table_name = result_set_table_name AND column_name = result_set_column_name

   m. Pass if result is “gpkg_srs_id_trigger”

   n. Fail otherwise

References:

Test Type: Capability

A.3.1.5.1.3 SRS_ID Triggers – Extensions Row

Test Case ID: /reg_ext/features/srs_id_triggers/extension_row

Test Purpose: Verify that srs_id triggers are registered using the “gpkg_srs_id_trigger” extension name.

Test Method:

/reg_ext/features/srs_id_triggers/extension_name

References:

Test Type: Capability

A.3.2 Tiles
A.3.2.1  Zoom Levels
A.3.2.1.1  Data
A.3.2.1.1.1  Zoom Other Intervals—Extensions Name

Test Case ID: /reg_ext/tiles/zoom_levels/data/zoom_other_ext_name
Test Purpose: Verify that the “gpkg_zoom_other” extension name is used to register tiles tables with other than powers of two zoom intervals.
Test Method:

1) SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles'
2) Not testable if empty result set
3) For each row table_name from step 1
   a. SELECT zoom_level, pixel_x_size, pixel_y_size FROM tile_matrix_metadata
      WHERE table_name = selected table name ORDER BY zoom_level ASC
   b. Not testable if returns empty result set
   c. Not testable if there are not two rows with adjacent zoom levels
   d. Not testable if no pair of rows for adjacent zoom levels have pixel_x_size or
      pixel_y_size values that differ by other than powers of two
   e. /opt/extension_mechanism/extensions/data/table_def
   f. Fail if failed
   g. SELECT * FROM gpkg_extensions WHERE table_name = selected table name AND extension_name = 'gpkg_zoom_other'
   h. Fail if returns an empty result set
   i. Log pass otherwise
4) Pass if logged pass and no fails

Reference: Clause 3.2.1.1.2 Req 82:
Test Type: Basic

A.3.2.1.1.2  Zoom Other Intervals – Extensions Row

Test Case ID: /reg_ext/tiles/zoom_levels/data/zoom_other_ext_row
Test Purpose: Verify that tiles tables with other than powers of two zoom intervals are registered using the “gpkg_zoom_other” extension name.
Test Method:

/reg_ext/tiles/zoom_levels/data/zoom_other_ext_name

Reference: Clause 3.2.1.1.3 Req 83:
Test Type: Capability

A.3.2.2  Tile Encoding WEBP
A.3.2.2.1  Data
A.3.2.2.1.1  WEBP – Extensions Name

Test Case ID: /reg_ext/tiles/tile_encoding_webp/data/webp_ext_name
**Test Purpose:** Verify that the “gpkg_webp” extensions name is used to register WEBP tile encoding implementations.

**Test Method:**

1) SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles'
2) Not testable if empty result set
3) For each row table_name from step 1
   a. Select tile_data FROM row table_name
   b. For each row tile_data from step a
      i. Log webp if tile data in MIME type image/webp
   c. Not testable if no logged webps
   d. /opt/extension_mechanism/extensions/data/table_def
   e. Fail if failed
   f. SELECT * FROM gpkg_extensions WHERE table_name = selected table name AND extension_name = 'gpkg_webp'
   g. Fail if returns an empty result set
   h. Log pass otherwise
4) Pass if logged pass and no fails

**Reference:** Clause 3.2.2.2.1 Req 84:

**Test Type:** Basic

**A.3.2.2.1.2 WEBP – Extensions Row**

**Test Case ID:** /reg_ext/tiles/tile_encoding_webp/data/webp_ext_row

**Test Purpose:** Verify that WEBP tile encodings are registered using the “gpkg_webp” extensions name.

**Test Method:**

/reg_ext/tiles/tile_encoding_webp/data/webp_ext_name

**Reference:** Clause 3.2.2.2.2 Req 85:

**Test Type:** Capability

**A.3.2.3 Tile Encoding TIFF**

**A.3.2.3.1 Data**

**A.3.2.3.1.1 TIFF – Extensions Name**

**Test Case ID:** /reg_ext/tiles/tile_encoding_tiff/data/tiff_ext_name

**Test Purpose:** Verify that the “gpkg_tiff” extensions name is used to register TIFF tile encoding implementations.

**Test Method:**

1) SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles'
2) Not testable if empty result set
3) For each row table_name from step 3
   a. Select tile_data FROM row table_name
   b. For each row tile_data from step a
      i. Log tiff if tile data in MIME type image/tiff
   c. Not testable if no logged webps
   d. /opt/extension_mechanism/extensions/data/table_def
A.3.2.3.1.2 TIFF – Extensions Row

Test Case ID: /reg_ext/tiles/tile_encoding_tiff/data/tiff_ext_row
Test Purpose: Verify that TIFF tile encodings are registered using the “gpkg_tiff” extensions name.
Test Method:

1) SELECT * FROM gpkg_extensions WHERE table_name = selected table name AND extension_name = 'gpkg_tiff'
2) Fail if returns an empty result set
3) Log pass otherwise

4) Pass if logged pass and no fails

Reference: Clause 3.2.3.1.2 Req 86:
Test Type: Basic

A.3.2.4 Tile Encoding NITF

A.3.2.4.1 Data

A.3.2.4.1.1 NITF – Extensions Name

Test Case ID: /reg_ext/tiles/tile_encoding_nitf/data/nitf_ext_name
Test Purpose: Verify that the “gpkg_nitf” extensions name is used to register NITF tile encoding implementations.
Test Method:

1) SELECT table_name FROM geopackage_contents WHERE data_type = ‘tiles’
2) Not testable if empty result set
3) For each row table_name from step 3
   a. Select tile_data FROM row table_name
   b. For each row tile_data from step a
      i. Log nitf if tile data in MIME type application/vnd.NITF
   c. Not testable if no logged webps
   d. /opt/extension_mechanism/extensions/data/table_def
   e. Fail if failed
   f. SELECT * FROM gpkg_extensions WHERE table_name = selected table
      name AND extension_name = 'gpkg_nitf'
   g. Fail if returns an empty result set
   h. Log pass otherwise
4) Pass if logged pass and no fails

Reference: Clause 3.2.4.1.2 Req 88:
Test Type: Basic

A.3.2.4.1.2 NITF – Extensions Row
Test Case ID: /reg_ext/tiles/tile_encoding/nitf/data/nitf_ext_row
Test Purpose: Verify that NITF tile encodings are registered in the gpkg_extensions table using the “gpkg_nitf” extensions name.
Test Method:

/req_ext/tiles/tile_encoding_nitf/data/nitf_ext_name

Reference: Clause 3.2.4.1.3 Req 89:
Test Type: Capability

A.3.2.5 Tile Encoding Other
A.3.2.5.1 Data
A.3.2.5.1.1 Other Mime Type – Extensions Name

Test Case ID: /reg_ext/tiles/tile_encoding/other/data/other_ext_name
Test Purpose: Verify that an extension name in the form <author_name>_<other>_mime_type is defined for an author name other than “gpkg” for each other MIME image format used for tile_data columns in tile matrix set user data tables, where <other> is replaced by the other MIME type abbreviation in uppercase.
Test Method:

1) SELECT table_name FROM geopackage_contents WHERE data_type = 'tiles'
2) Not testable if empty result set
3) For each row table_name from step 3
   a. Select tile_data FROM row table_name
   b. For each row tile_data from step a
      i. Log other MIME type name if tile data not in MIME type png, jpeg, webp, tiff or nitf
   c. Not testable if no logged others
   d. /opt/extension_mechanism/extensions/data/table_def
   e. Fail if failed
   f. For each other logged MIME type name for this table_name
      i. SELECT extension_name FROM gpkg_extensions WHERE table_name = result set table name AND column_name = ‘tile_data’ AND substr(lower(extension_name),1,4) != ‘gpkg’) AND instr(extension_name, logged MIME type name) != 0
      ii. Fail if returns an empty result set
      iii. Separate extension_name into <author> and <extension> at the first “_”
      iv. Separate <extension> into <mime> and <ext> at the first “_”
      v. Fail if <mime> not logged MIME type
      vi. Fail if <ext> not “mime_type”
      vii. Log pass otherwise
   4) Pass if logged pass and no fails”

Reference: Clause 3.2.5.1.2 Req 90:
Test Type: Basic

A.3.2.5.1.2 Other Mime Type – Extensions Row
**Test Case ID:** /reg_ext/tiles_tile_encoding/other/data/other_ext_row
**Test Purpose:** Verify that other mime image type tile encodings are registered in the gpkg_extensions table using names of the form `<author_name>_<other>_mime_type`.
**Test Method:**

```
/reg_ext/tiles/tile_encoding/other/data/other_ext_name
```

**Reference:** Clause 3.2.5.1.3 Req 91:

**Test Type:** Capability

### A.3.3 Any Tables

#### A.3.3.1 Other Trigger

##### A.3.3.1.1 Data

#### A.3.3.1.1.1 Other Trigger – Extensions Name

**Test Case ID:** /reg_ext/any/other_triggers/data/ext_name

**Test Purpose:** Verify that an extension name in the form `<author_name>_<extension>` is defined for an author name other than “gpkg” for each other trigger implementation that uses SQL functions other than those provided by SQLite or the GeoPackage Minimal Runtime SQL Functions.

**Test Method:**

1. SELECT sql, tbl_name FROM sqlite_master WHERE type='trigger'
2. For each row sql, tbl_name from step 1
   a. If sql contains an SQL function other than those provided by SQLite or the GeoPackage Minimal Runtime SQL Functions
      i. Log trigger
      ii. /opt/extension_mechanism/extensions/data/table_def
      iii. Fail if failed
      iv. SELECT extension_name FROM gpkg_extensions WHERE table_name = tbl_name AND substr(lower(extension_name),1,4) !- ‘gpkg’
      v. Fail if returns an empty result set
      vi. Log pass otherwise
3. Not testable if no logged triggers
4. Pass if logged pass and no fails

**Reference:** Clause 3.3.1.1.2 Req 92:

**Test Type:** Basic

#### A.3.3.1.1.2 Other Trigger – Extensions Row

**Test Case ID:** /reg_ext/any/other_triggers/data/ext_row

**Test Purpose:** Verify that other trigger implementations that use SQL functions other than those provided by SQLite or the GeoPackage Minimal Runtime SQL Functions are registered with `<author_name>_<extension>` names in the gpkg_extensions table.

**Test Method:**

```
/reg_ext/any/other_triggers/data/ext_name
```

**Reference:** Clause 3.3.1.1.3 Req 93:
Test Type: Capability
Annex B  Background and Context (Informative)

B.1 Preface
An open standard non-proprietary platform-independent GeoPackage container for
distribution and direct use of all kinds of geospatial data will increase the cross-platform
interoperability of geospatial applications and web services. Standard APIs for access and
management of GeoPackage data will provide consistent query and update results across such
applications and services. Increased interoperability and result consistency will enlarge the
potential market for such applications and services, particularly in resource-constrained
mobile computing environments like cell phones and tablets. GeoPackages will become the
standard containers for “MyGeoData” that are used as a transfer format by users and
Geospatial Web Services and a storage format on personal and enterprise devices.

This OpenGIS® GeoPackage Implementation Specification defines a GeoPackage as a self-
contained, single-file, cross-platform, serverless, transactional, open source SQLite data
container with table definitions, relational integrity constraints, an SQL API exposed via a
“C” CLI and JDBC, and manifest tables that together act as an exchange and direct-use
format for multiple types of geospatial data including vector features, features with raster
attributes and tile matrix pyramids, especially on mobile / hand held devices in disconnected
or limited network connectivity environments.

Table formats, definitions of geometry types and metadata tables, relational integrity
constraints, and SQL API are interdependent specification facets of the SF-SQL [13][11][12]
and SQL-MM (Spatial) [14] standards that serve as normative references for the vector
feature portion of this specification.

This specification attempts to support and use relevant raster types, storage table definitions,
and metadata from widely adopted implementations and existing standards such as WMTS
[22] and ISO metadata [42], to integrate use of rasters as attributes of geospatial features, and
to define relational integrity constraints and an SQL API thereon to provide a raster analogy
to the SF-SQL and SF-MM data access and data quality assurance capabilities.

Conformance classes for this specification are classified as core (mandatory) and extension
(optional). The simple core of an Empty GeoPackage contains two SQL tables.

Future versions of this specification may include requirements for elevation data and routes.
Future enhancements to this specification, a future GeoPackage Web Service specification,
and modifications to existing OGC Web Service (OWS) specifications to use GeoPackages as
exchange formats may allow OWS to support provisioning of GeoPackages throughout an
enterprise or information community.

B.2 Submitting Organizations
The following organizations submitted this Implementation Specification to the Open
Geospatial Consortium Inc. as a Request For Comment (RFC).

a) Envitia
b) Luciad
c) Sigma Bravo
d) The Carbon Project
e) U.S. Army Geospatial Center
f) U.S. National Geospatial Intelligence Agency

### B.3 Revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Rel</th>
<th>Editor</th>
<th>Paragraph modified</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-11-15</td>
<td>r1</td>
<td>Paul Daisey</td>
<td>10.3</td>
<td>Remove min/max x/y fields from all tables and text in clause 10.3 Tile Table Metadata per change request 250 / 12-135.</td>
</tr>
<tr>
<td>2012-11-15</td>
<td>r1</td>
<td>Paul Daisey</td>
<td>10.2, Annex B</td>
<td>add compr_qual_factor and georectification columns to raster_columns table create statement and sample insert statement; add triggers for those columns matching those for _rt_metadata per change request 251 / 12-134.</td>
</tr>
<tr>
<td>2013-01-15</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>8.2</td>
<td>Change gpkg_contents description default value per change request 255 / 12-166</td>
</tr>
<tr>
<td>2013-01-15</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>9.2, Annex A</td>
<td>SRS Table Name Change per change request 256 / 12-165</td>
</tr>
<tr>
<td>2013-01-16</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>7, Figure 2</td>
<td>table diagram gpkg_contents min_y REAL instead of BLOB</td>
</tr>
<tr>
<td>2013-01-23</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>11.3, 8.2</td>
<td>Clause reference corrections, change gpkg_contents.identifier default value to “”</td>
</tr>
<tr>
<td>2013-02-01</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>Changes to AS</td>
<td>No changes to AS</td>
</tr>
<tr>
<td>2013-02-01</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>8.2</td>
<td>new last sentence and NOTE1, additional table name triggers</td>
</tr>
<tr>
<td>2013-02-01</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>9.6</td>
<td>drop tables 21, 22 and associated text</td>
</tr>
<tr>
<td>2013-02-01</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>10.5</td>
<td>misc. editorial changes</td>
</tr>
<tr>
<td>2013-02-01</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>11.2</td>
<td>REQ 71 should refer to clause 11.2 and not 11.1</td>
</tr>
<tr>
<td>2013-02-01</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>12</td>
<td>new clause 12 other data</td>
</tr>
<tr>
<td>2013-02-01</td>
<td>r2</td>
<td>Paul Daisey</td>
<td>13.2</td>
<td>rename tables 56,57 remove “ows_” prefix</td>
</tr>
<tr>
<td>2013-02-08</td>
<td>r3</td>
<td>Paul Daisey</td>
<td>10.2, 10.7, 10.8</td>
<td>Use -1 as &quot;magic&quot; value indicating &quot;unknown&quot; for both compr_qual_factor and georectification columns, and make it the default value. Remove NOTE1 in 10.7. Delete the next to last row in Table 46 - Image Routines for gpkgBboxToTiles().</td>
</tr>
<tr>
<td>Date</td>
<td>Author</td>
<td>Task Description</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>2013-02-22</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>Delete the corresponding sub-clause 10.8.10 Renumber sub-clause 10.8.11 to 10.8.10</td>
<td></td>
</tr>
<tr>
<td>2013-03-05</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>The GeoPackage file format and SQL API are provided by SQLite, which is the GeoPackage container implementation, not just a reference implementation.</td>
<td></td>
</tr>
<tr>
<td>2013-03-05</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>Add Security Considerations clause.</td>
<td></td>
</tr>
<tr>
<td>2013-03-30</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>Streaming synchronization</td>
<td></td>
</tr>
<tr>
<td>2013-03-30</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>Move references to geos and proj4 libraries from Normative References to Bibliography, remove references to them from main text.</td>
<td></td>
</tr>
<tr>
<td>2013-03-01</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>New Core / Extension outline.</td>
<td></td>
</tr>
<tr>
<td>2013-03-30</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>auth_name column case-insensitive</td>
<td></td>
</tr>
<tr>
<td>2013-03-30</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>Add feature table layout example</td>
<td></td>
</tr>
<tr>
<td>2013-04-01</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>Move table definition SQL to Annex B</td>
<td></td>
</tr>
<tr>
<td>2013-04-01</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>Remove requirements for SQL/MM functions, REQ 21 – 33.</td>
<td></td>
</tr>
<tr>
<td>2013-04-03</td>
<td>R3</td>
<td>Paul Daisey</td>
<td>Renumber tables, figures, normative references</td>
<td></td>
</tr>
<tr>
<td>2013-04-09</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>Make integer primary keys mandatory for user vector, raster and tile data tables.</td>
<td></td>
</tr>
<tr>
<td>2013-04-09</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>Rewrite clause, remove references to geometry_columns table columns that are superfluous in SQLite implementation.</td>
<td></td>
</tr>
<tr>
<td>2013-04-09</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>Rewrite clause, remove references to SF/SQL gS and gB architectures.</td>
<td></td>
</tr>
<tr>
<td>2013-04-18</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>Remove normative references to RasterLite</td>
<td></td>
</tr>
<tr>
<td>2013-04-29</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>Remove implementation references</td>
<td></td>
</tr>
<tr>
<td>2013-04-29</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>Remove manifest other data entries</td>
<td></td>
</tr>
<tr>
<td>2013-04-29</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>Allow metadata of specified MIME type</td>
<td></td>
</tr>
<tr>
<td>2013-04-29</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>Allow NULLs in metadata_reference table</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Status</td>
<td>Author</td>
<td>Revision</td>
<td>Section/Annex</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------</td>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>2013-04-29</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>6.3.3.2, new Annex F</td>
<td>Geometry type codes</td>
</tr>
<tr>
<td>2013-04-29</td>
<td>R4</td>
<td>Paul Daisey</td>
<td>6.3.2.4, new Annex L</td>
<td>Feature Schema Metadata example</td>
</tr>
<tr>
<td>2013-05-03</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Future Work</td>
<td>Geographic / Geodetic Geometries</td>
</tr>
<tr>
<td>2013-05-07</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.3.4.2, Annex C, E</td>
<td>Remove compr_qual_factor and georectification columns from raster_columns table</td>
</tr>
<tr>
<td>2013-05-07</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.3.2.4, 6.3.4.3, new Annex M</td>
<td>delete _rt_metadata tables add Annex M reference Annex M from note in 6.3.2.4</td>
</tr>
<tr>
<td>2013-05-07</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>7.1.1, Normative References, Bibliography</td>
<td>Add NITF as an extension image format</td>
</tr>
<tr>
<td>2013-05-07</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.3.1</td>
<td>Revise Table Diagram</td>
</tr>
<tr>
<td>2013-05-07</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>7.3.3, Annex C</td>
<td>Remove raster functions</td>
</tr>
<tr>
<td>2013-05-11</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.3.2.4.3</td>
<td>metadata_reference table is not required to contain any rows</td>
</tr>
<tr>
<td>2013-05-11</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.3.2.4.2</td>
<td>Recommend ISO 19139 metadata</td>
</tr>
<tr>
<td>2013-05-11</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.3, Annex B</td>
<td>Default values</td>
</tr>
<tr>
<td>2013-05-11</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>7.3.3, Annex C</td>
<td>Minimal Runtime SQL Functions</td>
</tr>
<tr>
<td>2013-05-11</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>7.3.4, Annex D</td>
<td>Spatial Indexes</td>
</tr>
<tr>
<td>2013-05-13</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6, 7</td>
<td>Reformat requirement tables, unduplicate requirement text</td>
</tr>
<tr>
<td>2013-05-15</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.3.1, 6.3.2.4, 6.3.4.2, 7.3.5.5, Annex B, remove Annex L</td>
<td>Replace raster_columns table, Annex L with gpkg_data_columns table</td>
</tr>
<tr>
<td>2013-05-16</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.3.2.3, 7.4, Annex G,H,I</td>
<td>Drop manifest table, schemas, sample document</td>
</tr>
<tr>
<td>2013-05-16</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Future Work</td>
<td>Add GeoPackage Abstract Object Model</td>
</tr>
<tr>
<td>2013-05-22</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>6.2.1, 6.3.3.1, new 7.1.1, Annex F</td>
<td>Add optional support for non-linear geometry types</td>
</tr>
<tr>
<td>2013-05-22</td>
<td>R5</td>
<td>Paul</td>
<td>7.3.2</td>
<td>Add SQLite configuration requirements</td>
</tr>
<tr>
<td>Date</td>
<td>Reference</td>
<td>Author</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>2013-05-22</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Require only gpkg_contents and spatial_ref_sys tables</td>
<td></td>
</tr>
<tr>
<td>2013-05-24</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Add gpkg_extensions table</td>
<td></td>
</tr>
<tr>
<td>2013-05-24</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Provide spatial index templates instead of examples</td>
<td></td>
</tr>
<tr>
<td>2013-05-25</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Simplify, rewrite, add terms, use terms</td>
<td></td>
</tr>
<tr>
<td>2013-05-26</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Incorporate terms, renumber requirements and tables</td>
<td></td>
</tr>
<tr>
<td>2013-05-28</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Add &quot;GPKG&quot; as SQLite application id</td>
<td></td>
</tr>
<tr>
<td>2013-05-28</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Add SQLITE PRAGMA integrity_check</td>
<td></td>
</tr>
<tr>
<td>2013-05-28</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Geometry format minor changes</td>
<td></td>
</tr>
<tr>
<td>2013-05-28</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Remove references to raster_columns table (removed previously)</td>
<td></td>
</tr>
<tr>
<td>2013-05-28</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Clause number references and text changes required by 5/22 changes</td>
<td></td>
</tr>
<tr>
<td>2013-05-28</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Remove comments on accepted changes</td>
<td></td>
</tr>
<tr>
<td>2013-05-28</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Add non-linear geometry type codes</td>
<td></td>
</tr>
<tr>
<td>2013-05-29</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Change reference from SF/SQL to SQL/MM</td>
<td></td>
</tr>
<tr>
<td>2013-05-29</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Change core and extension requirement names required by 5/22 changes</td>
<td></td>
</tr>
<tr>
<td>2013-05-29</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Change extension to API to avoid overloading extension term</td>
<td></td>
</tr>
<tr>
<td>2013-05-29</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Draft changes to A.2 Conformance Classes</td>
<td></td>
</tr>
<tr>
<td>2013-05-29</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Add gpkg_data_columns table SQL</td>
<td></td>
</tr>
<tr>
<td>2013-05-30</td>
<td>R5</td>
<td>Paul Daisey</td>
<td>Record 5/29 changes</td>
<td></td>
</tr>
<tr>
<td>2013-06-06</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Remove all forward material except title page, submitting orgs, and introduction, and put in annexes.</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Code</td>
<td>Name</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>------------------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>2013-06-07</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Restructure document iaw draft Requirements Dependencies</td>
<td></td>
</tr>
<tr>
<td>2013-06-07</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Revised Requirements Dependencies and Diagram</td>
<td></td>
</tr>
<tr>
<td>2013-06-10</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Fix clause and requirement references based on document restructure</td>
<td></td>
</tr>
<tr>
<td>2013-06-10</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Add Abstract Test Suite (incomplete)</td>
<td></td>
</tr>
<tr>
<td>2013-06-11</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Insert Base and Extension subclauses, renumber more deeply nested subclauses</td>
<td></td>
</tr>
<tr>
<td>2013-06-12</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Remove names and codes for Z and M geometry types, add Figure 5 and geometry subtype definitions</td>
<td></td>
</tr>
<tr>
<td>2013-06-12</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Rewrite clause, add new Requirement 10, 11, renumber existing and subsequent ones.</td>
<td></td>
</tr>
<tr>
<td>2013-06-12</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Add ST_Is3D() and ST_IsMeasured()</td>
<td></td>
</tr>
<tr>
<td>2013-06-12</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Add “gpkg_” prefix to all GeoPackage metadata tables</td>
<td></td>
</tr>
<tr>
<td>2013-06-12</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Update with “gpkg_” prefix</td>
<td></td>
</tr>
<tr>
<td>2013-06-12</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Add Abstract Test Suite (incomplete)</td>
<td></td>
</tr>
<tr>
<td>2013-06-13</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Add sentence to end of first paragraph describing gpkg_other_data_columns content.</td>
<td></td>
</tr>
<tr>
<td>2013-06-13</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Add Abstract Test Suite (incomplete)</td>
<td></td>
</tr>
<tr>
<td>2013-06-17</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Revised notes and turned them into footnotes; moved normative text into requirement statements.</td>
<td></td>
</tr>
<tr>
<td>2013-06-24</td>
<td>R6</td>
<td>Pepijn Van Eeckhoudt</td>
<td>Created and applied Word Styles and Outline List Numbering</td>
<td></td>
</tr>
<tr>
<td>2013-06-26</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>GeoPackage Geometry Encoding Revisions</td>
<td></td>
</tr>
<tr>
<td>2013-06-27</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Add footnote recommendation on Spatial Index drop/add if many updates.</td>
<td></td>
</tr>
<tr>
<td>2013-06-27</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Remove gpkg_tile_table_metadata table</td>
<td></td>
</tr>
<tr>
<td>2013-06-28</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Change requirement statement format to</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Version</td>
<td>Author</td>
<td>Section</td>
<td>Notes</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>--------</td>
<td>--------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>2013-06-28</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Annex B</td>
<td>Update definition of Empty GeoPackage, add definition of Valid GeoPackage</td>
</tr>
<tr>
<td>2013-06-28</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Figure 1, 2.2.7, Annex C, F</td>
<td>Change tile_matrix1_metadata t_table_name column name to table_name iaw changes to gpkg_geometry_columns column name changes.</td>
</tr>
<tr>
<td>2013-06-28</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Figure 1, 2.1.5, 2.2.7, Annex C, F</td>
<td>Add gpkg_geometry_columns and gpkg_tile_matrix_metadata table_name foreign key constraints referencing gpkg_contents table_name now that gpkg_contents rows may describe other data tables.</td>
</tr>
<tr>
<td>2013-06-28</td>
<td>R6</td>
<td>Paul Daisey</td>
<td>Clause 3</td>
<td>Tables with non “gpkg” author registered extensions not data_type “features” or “tiles”</td>
</tr>
<tr>
<td>2013-07-01</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>Annex A</td>
<td>Change ATS format from numbered list to bold heading, add test definitions.</td>
</tr>
<tr>
<td>2013-07-02</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>Annex A</td>
<td>Add test definitions.</td>
</tr>
<tr>
<td>2013-07-03</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>Annex A</td>
<td>Revise, add test definitions.</td>
</tr>
<tr>
<td>2013-07-04</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>1.1.1, Annex A</td>
<td>Change .geopackage to .gpkg</td>
</tr>
<tr>
<td>2013-07-24</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>Annex B</td>
<td>Add support for UTFGrid as a future work item.</td>
</tr>
<tr>
<td>2013-07-24</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>1.1.1.1.1</td>
<td>Add footnote to REQ 1 that SQLite is in the public domain.</td>
</tr>
<tr>
<td>2013-07-24</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>2.1.3.1.1</td>
<td>Add footnote to Table 4 that OGC WKB is subset of ISO WKB</td>
</tr>
<tr>
<td>2013-07-24</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>2.1.3.1.1</td>
<td>Revise definition of geometry type in Table 4 to include is_empty flag; add paragraph on encoding empty point geometries.</td>
</tr>
<tr>
<td>2013-07-24</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>Annex E</td>
<td>Revise spatial index triggers to handle NULL values.</td>
</tr>
<tr>
<td>2013-07-31</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>Annex C, F</td>
<td>Correct SQL errors in tables 13, 32, 43</td>
</tr>
<tr>
<td>2013-07-31</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>Annex E Table 39</td>
<td>Revise spatial index triggers to handle empty geometries, changed ROWID values.</td>
</tr>
<tr>
<td>2013-07-31</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>Annex A A.3.1.3.1.1</td>
<td>Revise test method iaw changes to spatial index triggers</td>
</tr>
<tr>
<td>2013-07-31</td>
<td>R7</td>
<td>Paul Daisey</td>
<td>2.1.3.1.1</td>
<td>Envelope in geopackage geometry binary for empty geometry</td>
</tr>
</tbody>
</table>
### B.4 Changes to the OGC® Abstract Specification

The OGC® Abstract Specification does not require changes to accommodate this OGC® standard.

### B.5 Changes to OpenGIS® Implementation Standards

None at present.

### B.6 Potential Future Work

- MAY investigate GeoPackage implementation on SQLite version 4 [B46].
- Future versions of this specification MAY include requirements for elevation data and routes.
- Future enhancements to this specification, a future GeoPackage Web Service specification and modifications to existing OGC Web Service (OWS) specifications to use GeoPackages as exchange formats MAY allow OWS to support provisioning of GeoPackages throughout an enterprise.
- Future versions of this specification MAY include additional raster / image formats, including fewer restrictions on the image/tiff format.
- Future versions of this specification MAY include additional SQL API routines for interrogation and conversion of raster / image BLOBs.
- Future versions of this specification and/or one for a GeoPackage Web Service MAY address utilities for importing and exporting vector, raster and tile data in various formats.
- Future versions of this specification and/or one for a GeoPackage Web Service MAY address encryption of GeoPackages and/or individual tables or column values.
- Future versions of this specification MAY add infrastructure to the metadata tables such as a temporal_columns table that refers to the time properties of data records.
- MAY specify a streaming synchronization protocol for GeoPackage as part of a future GeoPackage Web Service specification, and/or a future version of the GeoPackage and/or Web Synchronization Service specification(s).
• Future versions of this specification MAY address symbology and styling information.
• Future version of this specification MAY include geographic / geodesic geometry types.
• MAY create a GeoPackage Abstract Object Model to support data encodings other than SQL in a future version of this specification.
• MAY add UTFGrid (https://github.com/mapbox/utfgrid-spec) support in a future version of this specification

B.7 Contributors
The following organizations and individuals have contributed to the preparation of this standard:

• Alessandro Furieri
• Compusult Limited
• Development Seed
• Environmental Systems Research Institute, Inc. (Esri)
• Envitia
• Feng China University
• George Mason University
• Image Matters LLC
• International Geospatial Services Institute (iGSI) GmbH
• LMN Solutions
• Luciad
• MapBox
• OpenGeo
• Open Site Plan
• Sigma Bravo
• The Carbon Project
• Universitat Autònoma de Barcelona (CREAF)
• U.S. Army Geospatial Center (AGC)
• U.S. National Aeronautics and Space Administration (NASA)
• U.S. National Geospatial Intelligence Agency (NGA)

B.8 Terms and Definitions
For the purposes of this document, the following terms and definitions apply.

B.8.1 Empty GeoPackage
A GeoPackage file that contains a spatial_ref_sys table, agpkg_contents table with row record(s) with data_type column values of “features” or “tiles”, and corresponding features tables per clause 2.1 and/or tiles tables per clause 2.2 where the user data tables per clauses 2.1.6 and 2.2.7 exist but contain no rows.

B.8.2 geolocate
identify a real-world geographic location

B.8.3 GeoPackage
A GeoPackage file used with a GeoPackage SQLite Extension

B.8.4 GeoPackage file
a platform-independent SQLite database file that contains GeoPackage data and metadata tables with specified definitions, integrity assertions, format limitations and content constraints.

B.8.5 GeoPackage SQLite Extension
executable software linked to a SQLite library with specified configuration requirements to provide SQL API access to and functional manipulation of GeoPackage file contents.

B.8.6 georectified
raster whose pixels have been regularly spaced in a geographic (i.e., latitude / longitude) or projected map coordinate system using ground control points so that any pixel can be geolocated given its grid coordinate and the grid origin, cell spacing, and orientation.

B.8.7 orthorectified
georectified raster that has also been corrected to remove image perspective (camera angle tilt), camera and lens induced distortions, and terrain induced distortions using camera calibration parameters and DEM elevation data to accurately align with real world coordinates, have constant scale, and support direct measurement of distances, angles, and areas.

B.8.8 Valid GeoPackage
A GeoPackage file that contains features per clause 2.1 and/or tiles per clause 2.2 and row(s) in the gpkg_contents table with data_type column values of “features” and/or “tiles” describing the user data tables.

B.9 Conventions
Symbols (and abbreviated terms)

Some frequently used abbreviated terms:
ACID Atomic, Consistent, Isolated, and Durable
ASCII American Standard Code for Information Interchange
API Application Program Interface
ATOM Atom Syndication Format
BLOB Binary Large OBject
CLICall-Level Interface
COTS Commercial Off The Shelf
DEM Digital Elevation Model
DIGEST Digital Geographic Information Exchange Standard
GeoTIFF  Geographic Tagged Image File Format
GPKG  GeoPackage
GRD  Ground Resolved Distance
EPSG  European Petroleum Survey Group
FK  Foreign Key
IETF  Internet Engineering Task Force
IIRS  Image Interpretability Rating Scale
IRARS  Imagery Resolution Assessments and Reporting Standards (Committee)
ISO  International Organization for Standardization
JDBC  Java Data Base Connectivity
JPEG  Joint Photographics Expert Group (image format)
MIME  Multipurpose Internet Mail Extensions
NATO  North Atlantic Treaty Organization
NITF  National Imagery Transmission Format
OGC  Open Geospatial Consortium
PK  Primary Key
PNG  Portable Network Graphics (image format)
RDBMS  Relational Data Base Management System
RFC  Request For Comments
SQL  Structured Query Language
SRID  Spatial Reference (System) Identifier
TIFF  Tagged Image File Format
TIN  Triangulated Irregular Network
UML  Unified Modeling Language
UTC  Coordinated Universal Time
XML  eXtensible Markup Language
1D  One Dimensional
2D  Two Dimensional
3D  Three Dimensional

B.10  UML Notation
The diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagrams. The UML notations used in this standard for RDBMS tables in the GeoPackage container are described in Figure 1 below.
In this standard, the following two stereotypes of UML classes are used to represent RDBMS tables:

a) <<table>> An instantiation of a UML class as an RDBMS table.
b) <<column>> An instantiation of a UML attribute as an RDBMS table column.

In this standard, the following standard data types are used for RDBMS columns:

a) NULL – The value is a NULL value.
b) INTEGER – A signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value
c) REAL – The value is a floating point value, stored as an 8-byte IEEE floating point number.

d) TEXT – A sequence of characters, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

e) BLOB – The value is a blob of data, stored exactly as it was input.

f) NONE – The value is a Date / Time Timestamp

The UML notations used in this standard for the eXtensible Markup Language (XML) schema for the GeoPackage manifest are described in Figure 2 below.

**Figure 3 UML notation for XML Schema**

In this standard, the following stereotypes of UML classes are used to describe XML schemas:

a) <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). A DataType is a class with no operations whose primary purpose is to hold the information.

b) <<Enumeration>> is a fixed enumeration that uses string values for expressing a list of potential values.
c) \texttt{<<CodeList>>} is an extensible enumeration that uses string values for expressing a list of potential values.

In this standard, the standard data types defined in XML Schema Part 2: Datatypes\cite{41} are used to describe XML Schemas.
Annex C  Table Definition SQL (Normative)

C.1 gpkg_spatial_ref_sys

Table 19 gpkg_spatial_ref_sys Table Definition SQL

| CREATE TABLE gpkg_spatial_ref_sys (    |
| Srs_name TEXT NOT NULL,               |
| Srs_id INTEGER NOT NULL PRIMARY KEY,  |
| organization TEXT NOT NULL,          |
| organization_coordsys_id INTEGER NOT NULL, |
| definition TEXT NOT NULL,            |
| description TEXT                     | ) |

Table 20 SQL/MM View of gpkg_spatial_ref_sys Definition SQL

| CREATE VIEW st_spatial_ref_sys AS SELECT    |
| srs_name,                                  |
| srs_id,                                    |
| organization,                              |
| organization_coordsys_id,                  |
| definition,                                |
| description                               |
| FROM gpkg_spatial_ref_sys;                 |

Table 21 SF/SQL View of gpkg_spatial_ref_sys Definition SQL

| CREATE VIEW spatial_ref_sys AS SELECT    |
| srs_id AS srid,                          |
| organization AS auth_name,               |
| organization_coordsys_id AS auth_srid,   |
| definition AS srtext                     |
| FROM gpkg_spatial_ref_sys;               |

C.2 gpkg_contents

Table 22 gpkg_contents Table Definition SQL

| CREATE TABLE gpkg_contents (    |
| table_name TEXT NOT NULL PRIMARY KEY, |
| data_type TEXT NOT NULL,         |
| identifier TEXT UNIQUE,          |
| description TEXT DEFAULT '',     |
| last_change TEXT NOT NULL DEFAULT |(strftime('%Y-%m-%dT%H:%M:%fZ', CURRENT_TIMESTAMP)) |, |
C.3 gpkg_extension

Table 23 gpkg_extensions Table Definition SQL

```sql
CREATE TABLE gpkg_extensions (  
table_name TEXT,  
column_name TEXT,  
extension_name TEXT NOT NULL,  
CONSTRAINT ge_tce UNIQUE  
    (table_name, column_name, extension_name))
```

C.4 gpkg_geometry_columns

Table 24 gpkg_geometry_columns Table Definition SQL

```sql
CREATE TABLE gpkg_geometry_columns (  
table_name TEXT NOT NULL,  
column_name TEXT NOT NULL,  
geometry_type TEXT NOT NULL,  
srs_id INTEGER NOT NULL,  
z INTEGER NOT NULL,  
m INTEGER NOT NULL,  
CONSTRAINT pk_geom_cols PRIMARY KEY (table_name, column_name),  
CONSTRAINT fk_gc_tn FOREIGN KEY (table_name)  
    REFERENCES gpkg_contents(table_name),  
CONSTRAINT fk_gc_srs FOREIGN KEY (srs_id)  
    REFERENCES gpkg_spatial_ref_sys (srs_id))
```

Table 25 SQL/MM View of gpkg_geometry_columns Definition SQL

```sql
CREATE VIEW st_geometry_columns AS SELECT  
table_name,  
column_name,  
"ST_" || geometry_type_name,  
g.srs_id,  
srs_name  
FROM gpkg_geometry_columns as g JOIN gpkg_spatial_ref_sys AS s  
WHERE g.srs_id = s.srs_id;
```
Table 26 SF/SQL VIEW of gpkg_geometry_columns Definition SQL

```
CREATE VIEW geometry_columns AS SELECT
  table_name AS f_table_name,
  column_name AS f_geometry_column,
  code4name1(geometry_type_name) AS geometry_type,
  2 + (CASE z WHEN 1 THEN 1 WHEN 2 THEN 1 ELSE 0 END) +
  (CASE m WHEN 1 THEN 1 WHEN 2 THEN 1 ELSE 0 END) AS coord_dimension,
  srs_id AS srid
FROM gpkg_geometry_columns;
```

C.5 sample_feature_table (Informative)

Table 27 sample_feature_table Table Definition SQL

```
CREATE TABLE sample_feature_table (id INTEGER AUTOINCREMENT PRIMARY KEY,
  geometry_one BLOB NOT NULL,
  text_attribute TEXT NOT NULL,
  real_attribute REAL NOT NULL,
  numeric_attribute NUMERIC NOT NULL,
  raster_or_photo BLOB NOT NULL)
```

C.6 gpkg_tile_matrix_metadata

Table 28 gpkg_tile_matrix_metadata Table Creation SQL

```
CREATE TABLE gpkg_tile_matrix_metadata (table_name TEXT NOT NULL,
  zoom_level INTEGER NOT NULL,
  matrix_width INTEGER NOT NULL,
  matrix_height INTEGER NOT NULL,
  tile_width INTEGER NOT NULL,
  tile_height INTEGER NOT NULL,
  pixel_x_size DOUBLE NOT NULL,
  pixel_y_size DOUBLE NOT NULL,
  CONSTRAINT pk_ttm PRIMARY KEY (table_name, zoom_level) ON CONFLICT ROLLBACK,
  CONSTRAINT fk_tmm_table_name FOREIGN KEY (table_name) REFERENCES gpkg_contents(table_name))
```

Table 29 EXAMPLE: gpkg_tile_matrix_metadata Insert Statement

```
INSERT INTO gpkg_tile_matrix_metadata VALUES (
  "sample_matrix_tiles",
)
```

1 Implementer must provide code4name(geometry_type_name) SQL function
C.7 sample_matrix_tiles (Informative)

Table 30 EXAMPLE: tiles table Create Table SQL

```
CREATE TABLE sample_matrix_tiles (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    zoom_level INTEGER NOT NULL,  
    tile_column INTEGER NOT NULL,  
    tile_row INTEGER NOT NULL,  
    tile_data BLOB NOT NULL),  
UNIQUE (zoom_level, tile_column, tile_row))
```

Table 31 EXAMPLE: tiles table Insert Statement

```
INSERT INTO sample_matrix_tiles VALUES (  
    1,  
    1,  
    1,  
    1,  
    "BLOB VALUE")
```

C.8 gpkg_data_columns

Table 32 gpkg_data_columns Table Definition SQL

```
CREATE TABLE gpkg_data_columns (  
    table_name TEXT NOT NULL,  
    column_name TEXT NOT NULL,  
    name TEXT,  
    title TEXT,  
    description TEXT,  
    mime_type TEXT,  
    CONSTRAINT pk_gdc PRIMARY KEY (table_name, column_name)  
    ON CONFLICT ROLLBACK,  
    CONSTRAINT fk_gdc_tn FOREIGN KEY (table_name)  
    REFERENCES gpkg_contents(table_name))
```

C.9 metadata

Table 33 gpkg_metadata Table Definition SQL

```
CREATE TABLE gpkg_metadata (  
    id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,  
    ON CONFLICT ROLLBACK AUTOINCREMENT NOT NULL UNIQUE,
C.10 metadata_reference

Table 34 gpkg_metadata_reference Table Definition SQL

CREATE TABLE gpkg_metadata_reference (  
    reference_scope TEXT NOT NULL,  
    table_name TEXT,  
    column_name TEXT,  
    row_id_value INTEGER,  
    timestamp TEXT NOT NULL DEFAULT (strftime('%Y-%m-%dT%H:%M:%fZ', CURRENT_TIMESTAMP)),  
    md_file_id INTEGER NOT NULL,  
    md_parent_id INTEGER,  
    CONSTRAINT crmr_mfi_fk FOREIGN KEY (md_file_id) REFERENCES gpkg_metadata(id),  
    CONSTRAINT crmr_mpi_fk FOREIGN KEY (md_parent_id) REFERENCES gpkg_metadata(id)  
)

Table 35 Example: gpkg_metadata_reference SQL insert statement

INSERT INTO gpkg_metadata_reference VALUES (  
    'table', 'sample_rasters', NULL, NULL, '2012-08-17T14:49:32.932Z', 98, 99  
)
## Annex D  Minimal Runtime SQL Functions (Normative)

### Table 36 Minimal Runtime SQL Functions

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>Description</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_SRID(geom. Geometry) : INTEGER</td>
<td>Returns the spatial reference system id of a Geometry</td>
<td>Check that geometry srid matches what's specified in geometry_columns.srid</td>
</tr>
<tr>
<td>ST_GeometryType(geom. Geometry) : TEXT</td>
<td>Returns the WKB geometry type name of a Geometry</td>
<td>Check that the geometry type matches what’s specified in geometry_columns.geometry_type</td>
</tr>
<tr>
<td>GPKG_IsAssignableFrom(expected_type_name TEXT, actual_type_name TEXT): INTEGER</td>
<td>Returns 1 if a value of type expected_type_name is the same or a super type of type actual_type_name. Returns 0 otherwise.</td>
<td>Determine if the expected geometry type is the same as or a super type of the actual geometry type.</td>
</tr>
<tr>
<td>ST_IsEmpty(geom. Geometry)</td>
<td>Returns 1 if geometry value is empty, 0 if not empty, NULL if geometry value is NULL.</td>
<td>Test if a geometry value corresponds to the empty set</td>
</tr>
<tr>
<td>ST_Is3D(geom. Geometry)</td>
<td>Tests whether a Geometry value has z coordinates</td>
<td>Check that elevation values are present when they are required, and not present when they are prohibited.</td>
</tr>
<tr>
<td>ST_IsMeasured(geom. Geometry)</td>
<td>Tests whether a Geometry value has m coordinates.</td>
<td>Check that measure values are present when they are required, and not present when they are prohibited.</td>
</tr>
<tr>
<td>ST_MinX(geom. Geometry) : REAL</td>
<td>Returns the minimum X value of the bounding Envelope of a Geometry</td>
<td>Update the spatial index on a geometry column in a feature table.</td>
</tr>
<tr>
<td>ST_MaxX(geom. Geometry) : REAL</td>
<td>Returns the maximum X value of the bounding Envelope of a Geometry</td>
<td>Update the spatial index on a geometry column in a feature table.</td>
</tr>
<tr>
<td>ST_MinY(geom. Geometry) : REAL</td>
<td>Returns the minimum Y value of the bounding</td>
<td>Update the spatial index on a geometry column in a feature table.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Update the spatial index on a geometry column in a feature table.</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>ST_MaxY(geom. Geometry) : REAL</td>
<td>Returns the maximum Y value of the bounding Envelope of a Geometry</td>
<td></td>
</tr>
<tr>
<td>ST_MinZ(geom. Geometry) : REAL</td>
<td>Returns the minimum Z value of the bounding Envelope of a Geometry</td>
<td></td>
</tr>
<tr>
<td>ST_MaxZ(geom. Geometry) : REAL</td>
<td>Returns the maximum Z value of the bounding Envelope of a Geometry</td>
<td></td>
</tr>
<tr>
<td>ST_MinM(geom. Geometry) : REAL</td>
<td>Returns the minimum M value of the bounding Envelope of a Geometry</td>
<td></td>
</tr>
<tr>
<td>ST_MaxM(geom. Geometry) : REAL</td>
<td>Returns the maximum M value of the bounding Envelope of a Geometry</td>
<td></td>
</tr>
</tbody>
</table>
Annex E  Spatial Index SQL (Normative)

E.1 Create Virtual Table
GeoPackage files MAY contain a spatial index for each geometry column of a feature table. These spatial indexes SHALL be created using the SQLite Virtual Table RTree extension. An application that creates a spatial index SHALL create it using the following SQL statement template:

```
CREATE VIRTUAL TABLE rtree_<t>_<c>
USING rtree(id, minx, maxx, miny, maxy)
```

where `<t>` and `<c>` are replaced with the names of the feature table and geometry column being indexed. The `rtree` function id parameter becomes the virtual table 64-bit signed integer primary key `id` column, and the min/max x/y parameters are min- and max-value pairs (stored as 32-bit floating point numbers) for each dimension that become the virtual table data columns that are populated to create the spatial `rtree` index.

E.2 Load Spatial Index Values
The indexes provided by the SQLite Virtual Table RTree extension are not automatic indices. This means the index data structure needs to be manually populated, updated and queried. Each newly created spatial index SHALL be populated using the following SQL statement:

```
INSERT OR REPLACE INTO rtree_<t>_<c>
SELECT rowid, st_minx(<c>), st_maxx(<c>),
       st_miny(<c>), st_maxy(<c>)
FROM <t>;
```

where `<t>` and `<c>` are replaced with the names of the feature table and geometry column being indexed.

E.3 Define Triggers to Maintain Spatial Index Values
For each spatial index in a GeoPackage file, corresponding insert, update and delete triggers that update the spatial index SHALL be present on the indexed geometry column. These spatial index triggers SHALL be defined as follows:

```
ncy SQL Triggers Template
/* Conditions: Insertion of non-empty geometry
   Actions   : Insert record into rtree */
CREATE TRIGGER rtree_<t>_<c>_insert AFTER INSERT ON <t>
WHEN (new.<c> NOT NULL AND NOT ST_IsEmpty(NEW.<c>))
BEGIN
  INSERT OR REPLACE INTO rtree_<t>_<c> VALUES (NEW.rowid,
                                               ST_MinX(NEW.<c>), ST_MaxX(NEW.<c>),
```


BEGIN
    INSERT OR REPLACE INTO rtree_<t>_<c> VALUES (NEW.rowid, ST_MinX(NEW.<c>), ST_MaxX(NEW.<c>), ST_MinY(NEW.<c>), ST_MaxY(NEW.<c>));
END;

/* Conditions: Update of geometry column to non-empty geometry No row ID change Actions : Update record in rtree */
CREATE TRIGGER rtree_<t>_<c>_update1 AFTER UPDATE OF <c> ON <t>
WHEN OLD.rowid = NEW.rowid AND (NEW.<c> NOTNULL AND NOT ST_IsEmpty(NEW.<c>))
BEGIN
    INSERT OR REPLACE INTO rtree_<t>_<c> VALUES (NEW.rowid, ST_MinX(NEW.<c>), ST_MaxX(NEW.<c>), ST_MinY(NEW.<c>), ST_MaxY(NEW.<c>));
END;

/* Conditions: Update of geometry column to empty geometry No row ID change Actions : Remove record from rtree for old rowid */
CREATE TRIGGER rtree_<t>_<c>_update2 AFTER UPDATE OF <c> ON <t>
WHEN OLD.rowid = NEW.rowid AND (NEW.<c> ISNULL OR ST_IsEmpty(NEW.<c>))
BEGIN
    DELETE FROM rtree_<t>_<c> WHERE id = OLD.rowid;
END;

/* Conditions: Update of any column Row ID change Non-empty geometry Actions : Remove record from rtree for old rowid Insert record into rtree for new rowid */
CREATE TRIGGER rtree_<t>_<c>_update3 AFTER UPDATE OF <c> ON <t>
WHEN OLD.rowid != NEW.rowid AND (NEW.<c> NOTNULL AND NOT ST_IsEmpty(NEW.<c>))
BEGIN
    DELETE FROM rtree_<t>_<c> WHERE id = OLD.rowid;
    INSERT OR REPLACE INTO rtree_<t>_<c> VALUES (NEW.rowid, ST_MinX(NEW.<c>), ST_MaxX(NEW.<c>), ST_MinY(NEW.<c>), ST_MaxY(NEW.<c>));
END;

/* Conditions: Update of any column Row ID change Empty geometry Actions : Remove record from rtree for old and new rowid */
CREATE TRIGGER rtree_<t>_<c>_update4 AFTER UPDATE ON <t>
WHEN OLD.rowid != NEW.rowid AND
BEGIN
    DELETE FROM rtree_<t>_<c> WHERE id = OLD.rowid;
    DELETE FROM rtree_<t>_<c> WHERE id = NEW.rowid;
    INSERT OR REPLACE INTO rtree_<t>_<c> VALUES (NEW.rowid, ST_MinX(NEW.<c>), ST_MaxX(NEW.<c>), ST_MinY(NEW.<c>), ST_MaxY(NEW.<c>));
END;
### DELETE Trigger Code

```sql
BEGIN
    DELETE FROM rtree_<t>_<c> WHERE id IN (OLD.rowid, NEW.rowid);
END;
```

```sql
/* Conditions: Row deleted
   Actions   : Remove record from rtree for old rowid */
CREATE TRIGGER rtree_<t>_<c>_delete AFTER DELETE ON <t>
WHEN old.<c> NOT NULL
BEGIN
    DELETE FROM rtree_<t>_<c> WHERE id = OLD.rowid;
END;
```

where `<t>` and `<c>` are replaced with the names of the feature table and geometry column being indexed.
Annex F  Trigger Definition SQL (Informative)

F.1 geometry_columns

Table 40 gpkg_geometry_columns Trigger Definitions

```
CREATE TRIGGER 'gpkg_geometry_columns_geometry_type_name_insert'
BEFORE INSERT ON 'gpkg_geometry_columns'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK,'geometry_type name must be one of GEOMETRY, POINT, LINestring, POLYGON, MULTIPoINT, MULTILINESTRING, MULTIPOLYGON, GEMOCOLLECTION, CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE, MULTISURFACE, CURVE, SURFACE')
WHERE NOT(NEW.geometry_type_name IN('GEOMETRY', 'POINT', 'LINESTRING', 'POLYGON', 'MULTIPOINT', 'MULTILINESTRING', 'MULTIPOLYGON', 'GEOMCOLLECTION', 'CIRCULARSTRING', 'COMPOUNDCURVE', 'CURVEPOLYGON', 'MULTICURVE', 'MULTISURFACE', 'CURVE', 'SURFACE'));
END

CREATE TRIGGER 'gpkg_geometry_columns_geometry_type_name_update'
BEFORE UPDATE OF 'geometry_type_name' ON 'gpkg_geometry_columns'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK,'geometry_type name must be one of GEOMETRY, POINT, LINestring, POLYGON, MULTIPoINT, MULTILINESTRING, MULTIPOLYGON, GEMOCOLLECTION, CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE, MULTISURFACE, CURVE, SURFACE')
WHERE NOT(NEW.geometry_type_name IN ('GEOMETRY', 'POINT', 'LINESTRING', 'POLYGON', 'MULTIPOINT', 'MULTILINESTRING', 'MULTIPOLYGON', 'GEOMCOLLECTION', 'CIRCULARSTRING', 'COMPOUNDCURVE', 'CURVEPOLYGON', 'MULTICURVE', 'MULTISURFACE', 'CURVE', 'SURFACE'));
END
```

F.2 gpkg_tile_matrix_metadata

Table 41 gpkg_tile_matrix_metadata Trigger Definition SQL

```
CREATE TRIGGER 'gpkg_tile_matrix_metadata_zoom_level_insert'
BEFORE INSERT ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'insert on table gpkg_tile_matrix_metadata violates constraint: zoom_level cannot be less than 0')
WHERE (NEW.zoom_level < 0);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_zoom_level_update'
BEFORE UPDATE of zoom_level ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'update on table gpkg_tile_matrix_metadata violates constraint: zoom_level cannot be less than 0')
WHERE (NEW.zoom_level < 0);
END
```
be less than 0') WHERE (NEW.zoom_level < 0);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_matrix_width_insert'
BEFORE INSERT ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'insert on table ''gpkg_tile_matrix_metadata'' violates constraint: matrix_width cannot be less than 1')
WHERE (NEW.matrix_width < 1);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_matrix_width_update'
BEFORE UPDATE OF matrix_width ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'update on table ''gpkg_tile_matrix_metadata'' violates constraint: matrix_width cannot be less than 1')
WHERE (NEW.matrix_width < 1);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_matrix_height_insert'
BEFORE INSERT ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'insert on table ''gpkg_tile_matrix_metadata'' violates constraint: matrix_height cannot be less than 1')
WHERE (NEW.matrix_height < 1);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_matrix_height_update'
BEFORE UPDATE OF matrix_height ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'update on table ''gpkg_tile_matrix_metadata'' violates constraint: matrix_height cannot be less than 1')
WHERE (NEW.matrix_height < 1);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_pixel_x_size_insert'
BEFORE INSERT ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'insert on table ''gpkg_tile_matrix_metadata'' violates constraint: pixel_x_size must be greater than 0')
WHERE NOT (NEW.pixel_x_size > 0);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_pixel_x_size_update'
BEFORE UPDATE OF pixel_x_size ON 'gpkg_tile_matrix_metadata'...
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'UPDATE ON TABLE
"gpkg_tile_matrix_metadata" violates constraint: pixel_x_size must
be greater than 0')
WHERE NOT (NEW.pixel_x_size > 0);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_pixel_y_size_insert'
BEFORE INSERT ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'INSERT ON TABLE
"gpkg_tile_matrix_metadata" violates constraint: pixel_y_size must
be greater than 0')
WHERE NOT (NEW.pixel_y_size > 0);
END

CREATE TRIGGER 'gpkg_tile_matrix_metadata_pixel_y_size_update'
BEFORE UPDATE OF pixel_y_size ON 'gpkg_tile_matrix_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'UPDATE ON TABLE
"gpkg_tile_matrix_metadata" violates constraint: pixel_y_size must
be greater than 0')
WHERE NOT (NEW.pixel_y_size > 0);
END

F.3 metadata

Table 42 metadata Trigger Definition SQL

CREATE TRIGGER 'gpkg_metadata_md_scope_insert'
BEFORE INSERT ON 'gpkg_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'INSERT ON TABLE gpkg_metadata violates
constraint: md_scope must be one of undefined | fieldSession |
collectionSession | series | dataset | featureType | feature |
attributeType | attribute | tile | model | catalogue | schema |
taxonomy | software | service | collectionHardware |
nonGeographicDataset | dimensionGroup')
WHERE NOT(NEW.md_scope IN
('undefined', 'fieldSession', 'collectionSession', 'series', 'dataset',
'featureType', 'feature', 'attributeType', 'attribute', 'tile', 'model',
catalogue', 'schema', 'taxonomy', 'software', 'service',
collectionHardware', 'nonGeographicDataset', 'dimensionGroup'));
END

CREATE TRIGGER 'gpkg_metadata_md_scope_update'
BEFORE UPDATE OF 'md_scope' ON 'gpkg_metadata'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'UPDATE ON TABLE gpkg_metadata violates
constraint: md_scope must be one of undefined | fieldSession |
collectionSession | series | dataset | featureType | feature |
attributeType | attribute | tile | model | catalogue | schema |
F.4 metadata_reference

Table 43 gpkg_metadata_reference Trigger Definition SQL

CREATE TRIGGER 'gpkg_metadata_reference_reference_scope_insert'
BEFORE INSERT ON 'gpkg_metadata_reference'
BEGIN
SELECT RAISE(ROLLBACK, 'insert on table gpkg_metadata_reference violates constraint: reference_scope must be one of "geopackage", table", "column", "row", "row/col")
WHERE NOT NEW.reference_scope IN ('geopackage', 'table', 'column', 'row', 'row/col');
END

CREATE TRIGGER 'gpkg_metadata_reference_reference_scope_update'
BEFORE UPDATE OF 'reference_scope' ON 'gpkg_metadata_reference'
BEGIN
SELECT RAISE(ROLLBACK, 'update on table gpkg_metadata_reference violates constraint: reference_scope must be one of "geopackage", table", "column", "row", "row/col")
WHERE NOT NEW.reference_scope IN ('geopackage', 'table', 'column', 'row', 'row/col');
END

CREATE TRIGGER 'gpkg_metadata_reference_column_name_insert'
BEFORE INSERT ON 'gpkg_metadata_reference'
BEGIN
SELECT RAISE(ROLLBACK, 'insert on table gpkg_metadata_reference violates constraint: column name must be defined for the specified table when reference_scope is "column" or "row/col"')
WHERE (NEW.reference_scope IN ('column', 'row/col')
AND NOT NEW.table_name IN (SELECT name FROM SQLITE_MASTER WHERE type = 'table'
AND name = NEW.table_name
AND sql LIKE ('%' || NEW.column_name || '%')));
END

CREATE TRIGGER 'gpkg_metadata_reference_column_name_update'
BEFORE UPDATE OF column_name ON 'gpkg_metadata_reference'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'update on table gpkg_metadata_reference violates constraint: column_name must be NULL when reference_scope is "geopackage", "table" or "row"')
WHERE (NEW.reference_scope IN ('geopackage', 'table', 'row')
AND NEW.column_name IS NOT NULL);
SELECT RAISE(ROLLBACK, 'update on table gpkg_metadata_reference violates constraint: column_name must be defined for the specified table when reference_scope is "column" or "row/column"')
WHERE (NEW.reference_scope IN ('column', 'row/column')
AND NOT NEW.table_name IN (SELECT name FROM SQLITE_MASTER WHERE type = 'table'
AND name = NEW.table_name
AND sql LIKE ('%' || NEW.column_name || '%')));
END

CREATE TRIGGER 'gpkg_metadata_reference_row_id_value_insert'
BEFORE INSERT ON 'gpkg_metadata_reference'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'insert on table gpkg_metadata_reference violates constraint: row_id_value must be NULL when reference_scope is "geopackage", "table" or "column"')
WHERE NEW.reference_scope IN ('geopackage', 'table', 'column')
AND NEW.row_id_value IS NOT NULL;
SELECT RAISE(ROLLBACK, 'insert on table gpkg_metadata_reference violates constraint: row_id_value must exist in specified table when reference_scope is "row" or "row/column"')
WHERE NEW.reference_scope IN ('row', 'row/column')
AND NOT EXISTS (SELECT rowid FROM (SELECT NEW.table_name AS table_name) WHERE rowid = NEW.row_id_value);
END

CREATE TRIGGER 'gpkg_metadata_reference_row_id_value_update'
BEFORE UPDATE OF 'row_id_value' ON 'gpkg_metadata_reference'
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'update on table gpkg_metadata_reference violates constraint: row_id_value must be NULL when reference_scope is "geopackage", "table" or "column"')
WHERE NEW.reference_scope IN ('geopackage', 'table', 'column')
AND NEW.row_id_value IS NOT NULL;
SELECT RAISE(ROLLBACK, 'update on table gpkg_metadata_reference violates constraint: row_id_value must exist in specified table when reference_scope is "row" or "row/column"')
WHERE NEW.reference_scope IN ('row', 'row/column')
AND NOT EXISTS (SELECT rowid FROM (SELECT NEW.table_name AS table_name) WHERE rowid = NEW.row_id_value);
END
CREATE TRIGGER 'gpkg_metadata_reference_timestamp_insert'
BEFORE INSERT ON 'gpkg_metadata_reference'
FOR EACH ROW BEGIN
SELECT RAISE (ROLLBACK, 'insert on table gpkg_metadata_reference
violates constraint: timestamp must be a valid time in ISO 8601
"yyyy-mm-ddThh-mm-ss.sssZ" form')
WHERE NOT (NEW.timestamp GLOB
'[1-2][0-9][0-9][0-9]-[0-1][0-3][0-9]T[0-2][0-9][0-9]:[0-5][0-9]
:[0-5][0-9].[0-9][0-9][0-9]Z'
AND strftime('%s',NEW.timestamp) NOT NULL);
END

CREATE TRIGGER 'gpkg_metadata_reference_timestamp_update'
BEFORE UPDATE OF 'timestamp' ON 'gpkg_metadata_reference'
FOR EACH ROW BEGIN
SELECT RAISE (ROLLBACK, 'update on table gpkg_metadata_reference
violates constraint: timestamp must be a valid time in ISO 8601
"yyyy-mm-ddThh-mm-ss.sssZ" form')
WHERE NOT (NEW.timestamp GLOB
'[1-2][0-9][0-9][0-9]-[0-1][0-3][0-9]T[0-2][0-9][0-9]:[0-5][0-9]
:[0-5][0-9].[0-9][0-9][0-9]Z'
AND strftime('%s',NEW.timestamp) NOT NULL);
END

F.5 sample_feature_table

Table 44 EXAMPLE: features table Trigger Definition SQL

CREATE TRIGGER "sample_feature_table_real_insert"
BEFORE INSERT ON "sample_feature_table"
FOR EACH ROW BEGIN
SELECT RAISE (ROLLBACK, 'insert on table "sample_feature_table"
violates constraint: real_attribute must be greater than 0')
WHERE NOT (NEW.real_attribute > 0);
END

CREATE TRIGGER "sample_feature_table_real_update"
BEFORE UPDATE OF "real_attribute" ON "sample_feature_table"
FOR EACH ROW BEGIN
SELECT RAISE (ROLLBACK, 'update of "real_attribute" on table
"sample_feature_table" violates constraint: real_attribute value
must be > 0')
WHERE NOT (NEW.real_attribute > 0);
END

where <t> and <c> are replaced with the names of the feature table and geometry column
being inserted or updated.

F.6 sample_matrix_tiles
Table 45 EXAMPLE: tiles table Trigger Definition SQL

<table>
<thead>
<tr>
<th>SQL Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TRIGGER &quot;sample_matrix_tiles_zoom_insert&quot;</td>
</tr>
<tr>
<td>BEFORE INSERT ON &quot;sample_matrix_tiles&quot;</td>
</tr>
<tr>
<td>FOR EACH ROW BEGIN</td>
</tr>
<tr>
<td>SELECT RAISE(ROLLBACK, 'insert on table 'sample_matrix_tiles''</td>
</tr>
<tr>
<td>violates constraint: zoom_level not specified for table in</td>
</tr>
<tr>
<td>gpkg_tile_matrix_metadata')</td>
</tr>
<tr>
<td>WHERE NOT (NEW.zoom_level IN (SELECT zoom_level FROM</td>
</tr>
<tr>
<td>gpkg_tile_matrix_metadata WHERE t_table_name = 'sample_matrix_tiles')))</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>CREATE TRIGGER &quot;sample_matrix_tiles_zoom_update&quot;</td>
</tr>
<tr>
<td>BEFORE UPDATE OF zoom_level ON &quot;sample_matrix_tiles&quot;</td>
</tr>
<tr>
<td>FOR EACH ROW BEGIN</td>
</tr>
<tr>
<td>SELECT RAISE(ROLLBACK, 'update on table 'sample_matrix_tiles''</td>
</tr>
<tr>
<td>violates constraint: zoom_level not specified for table in</td>
</tr>
<tr>
<td>gpkg_tile_matrix_metadata')</td>
</tr>
<tr>
<td>WHERE NOT (NEW.zoom_level IN (SELECT zoom_level FROM</td>
</tr>
<tr>
<td>gpkg_tile_matrix_metadata WHERE t_table_name = 'sample_matrix_tiles')))</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>CREATE TRIGGER &quot;sample_matrix_tiles_tile_column_insert&quot;</td>
</tr>
<tr>
<td>BEFORE INSERT ON &quot;sample_matrix_tiles&quot;</td>
</tr>
<tr>
<td>FOR EACH ROW BEGIN</td>
</tr>
<tr>
<td>SELECT RAISE(ROLLBACK, 'insert on table 'sample_matrix_tiles''</td>
</tr>
<tr>
<td>violates constraint: tile_column cannot be &lt; 0')</td>
</tr>
<tr>
<td>WHERE (NEW.tile_column &lt; 0)</td>
</tr>
<tr>
<td>SELECT RAISE(ROLLBACK, 'insert on table 'sample_matrix_tiles''</td>
</tr>
<tr>
<td>violates constraint: tile_column must by &lt; matrix_width specified for</td>
</tr>
<tr>
<td>table and zoom level in gpkg_tile_matrix_metadata')</td>
</tr>
<tr>
<td>WHERE NOT (NEW.tile_column &lt; (SELECT matrix_width FROM gpkg_tile</td>
</tr>
<tr>
<td>matrix_metadata WHERE t_table_name = 'sample_matrix_tiles' AND</td>
</tr>
<tr>
<td>zoom_level = NEW.zoom_level))</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>CREATE TRIGGER &quot;sample_matrix_tiles_tile_column_update&quot;</td>
</tr>
<tr>
<td>BEFORE UPDATE OF tile_column ON &quot;sample_matrix_tiles&quot;</td>
</tr>
<tr>
<td>FOR EACH ROW BEGIN</td>
</tr>
<tr>
<td>SELECT RAISE(ROLLBACK, 'update on table 'sample_matrix_tiles''</td>
</tr>
<tr>
<td>violates constraint: tile_column cannot be &lt; 0')</td>
</tr>
<tr>
<td>WHERE (NEW.tile_column &lt; 0)</td>
</tr>
<tr>
<td>SELECT RAISE(ROLLBACK, 'update on table 'sample_matrix_tiles''</td>
</tr>
<tr>
<td>violates constraint: tile_column must by &lt; matrix_width specified for</td>
</tr>
<tr>
<td>table and zoom level in gpkg_tile_matrix_metadata')</td>
</tr>
<tr>
<td>WHERE NOT (NEW.tile_column &lt; (SELECT matrix_width FROM gpkg_tile</td>
</tr>
<tr>
<td>matrix_metadata WHERE t_table_name = 'sample_matrix_tiles' AND</td>
</tr>
<tr>
<td>zoom_level = NEW.zoom_level))</td>
</tr>
</tbody>
</table>
CREATE TRIGGER "sample_matrix_tiles_tile_row_insert"
BEFORE INSERT ON "sample_matrix_tiles"
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'insert on table ''sample_matrix_tiles''
violates constraint: tile_row cannot be < 0')
WHERE (NEW.tile_row < 0) ;
SELECT RAISE(ROLLBACK, 'insert on table ''sample_matrix_tiles''
violates constraint: tile_row must by < matrix_height specified for
table and zoom level in gpkg_tile_matrix_metadata')
WHERE NOT (NEW.tile_row < (SELECT matrix_height FROM
gpkg_tile_matrix_metadata WHERE t_table_name = 'sample_matrix_tiles'
AND zoom_level = NEW.zoom_level));
END

CREATE TRIGGER "sample_matrix_tiles_tile_row_update"
BEFORE UPDATE OF tile_row ON "sample_matrix_tiles"
FOR EACH ROW BEGIN
SELECT RAISE(ROLLBACK, 'update on table ''sample_matrix_tiles''
violates constraint: tile_row cannot be < 0')
WHERE (NEW.tile_row < 0) ;
SELECT RAISE(ROLLBACK, 'update on table ''sample_matrix_tiles''
violates constraint: tile_row must by < matrix_height specified for
table and zoom level in gpkg_tile_matrix_metadata')
WHERE NOT (NEW.tile_row < (SELECT matrix_height FROM
gpkg_tile_matrix_metadata WHERE t_table_name = 'sample_matrix_tiles'
AND zoom_level = NEW.zoom_level));
END
Annex G  Geometry Types (Normative)

Table 46 Geometry Type Codes (Core)

<table>
<thead>
<tr>
<th>CODE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GEOMETRY</td>
</tr>
<tr>
<td>1</td>
<td>POINT</td>
</tr>
<tr>
<td>2</td>
<td>LINESTRING</td>
</tr>
<tr>
<td>3</td>
<td>POLYGON</td>
</tr>
<tr>
<td>4</td>
<td>MULTIPOLYGON</td>
</tr>
<tr>
<td>5</td>
<td>MULTILINESTRING</td>
</tr>
<tr>
<td>6</td>
<td>MULTIPOLYGON</td>
</tr>
<tr>
<td>7</td>
<td>GEOMCOLLECTION</td>
</tr>
</tbody>
</table>

Table 47 Geometry Type Codes (Extension)

<table>
<thead>
<tr>
<th>CODE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>CIRCULARSTRING</td>
</tr>
<tr>
<td>9</td>
<td>COMPOUND CURVE</td>
</tr>
<tr>
<td>10</td>
<td>CURVE POLYGON</td>
</tr>
<tr>
<td>11</td>
<td>MULTICURVE</td>
</tr>
<tr>
<td>12</td>
<td>MULTISURFACE</td>
</tr>
<tr>
<td>13</td>
<td>CURVE</td>
</tr>
<tr>
<td>14</td>
<td>SURFACE</td>
</tr>
</tbody>
</table>

GEOMETRY subtypes are POINT, CURVE, SURFACE and GEOMCOLLECTION.

CURVE subtypes are LINESTRING, CIRCULARSTRING and COMPOUND CURVE.
SURFACE subtype is CURVEPOLYGON.
CURVEPOLYGON subtype is POLYGON.

GEOMCOLLECTION subtypes are MULTIPOINT, MULTICURVE and MULTISURFACE.
MULTICURVE subtype is MULTILINestring.
MULTISURFACE subtype is MULTIPOLYGON
### Annex H  Other SQL/MM Functions (Informative)

#### H.1 Geometry Routines (Informative)

**Table 48 Geometry Routines**

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddGeometryColumn</td>
<td>adds a geometry column to a feature table</td>
<td>2.3.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DropGeometryColumn</td>
<td>drops a geometry column from a feature table</td>
<td>2.3.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_Dimension</td>
<td>returns the dimension of a Geometry value</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_GeometryType</td>
<td>returns the type of the Geometry value as a String</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_SRID</td>
<td>returns the spatial reference system identifier of an Geometry value</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Transform</td>
<td>returns the Geometry value in the specified spatial reference system</td>
<td>4.1.1.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_IsEmpty</td>
<td>tests if a Geometry value corresponds to the empty set</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_IsSimple</td>
<td>tests if a Geometry value has no anomalous geometric point, such as self intersection or self tangency</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_IsValid</td>
<td>tests if a Geometry value is well formed</td>
<td>4.1.1.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_Is3D</td>
<td>tests whether a Geometry value has z coordinates</td>
<td>4.1.1.1</td>
<td></td>
<td>6.1.2.2</td>
<td></td>
</tr>
<tr>
<td>ST_IsMeasured</td>
<td>tests whether a Geometry value has m coordinate values</td>
<td>4.1.1.1</td>
<td></td>
<td>6.1.2.2</td>
<td></td>
</tr>
<tr>
<td>ST_LocateAlong</td>
<td>returns a derived geometry collection value that matches the specified m coordinate value</td>
<td>4.1.1.1</td>
<td></td>
<td>6.1.2.6</td>
<td></td>
</tr>
<tr>
<td>ST_LocateBetween</td>
<td>returns a derived geometry collection value that matches the specified range of m coordinate values inclusively</td>
<td>4.1.1.1</td>
<td></td>
<td>6.1.2.6</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Section1</td>
<td>Section2</td>
<td>Section3</td>
<td>Section4</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ST_Boundary</td>
<td>returns the boundary of a Geometry value</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Envelope</td>
<td>returns the bounding rectangle of a Geometry value</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_ConvexHull</td>
<td>returns the convex hull of a Geometry value</td>
<td>4.1.1.1</td>
<td>2.1.1.3</td>
<td>6.1.2.4</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Buffer</td>
<td>returns the Geometry value that represents all points whose distance from any point of a Geometry value is less than or equal to a specified distance</td>
<td>4.1.1.1</td>
<td>2.1.1.3</td>
<td>6.1.2.4</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Intersection</td>
<td>returns the Geometry value that represents the point set intersection of two Geometry values</td>
<td>4.1.1.1</td>
<td>2.1.1.3</td>
<td>6.1.2.4</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Union</td>
<td>returns the Geometry value that represents the point set union of two ST_Geometry values</td>
<td>4.1.1.1</td>
<td>2.1.1.3</td>
<td>6.1.2.4</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Difference</td>
<td>returns the Geometry value that represents the point set difference of two Geometry values</td>
<td>4.1.1.1</td>
<td>2.1.1.3</td>
<td>6.1.2.4</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_SymDifference</td>
<td>returns the Geometry value that represents the point set symmetric difference of two Geometry values</td>
<td>4.1.1.1</td>
<td>2.1.1.3</td>
<td>6.1.2.4</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Distance</td>
<td>returns the distance between two geometries</td>
<td>4.1.1.1</td>
<td>2.1.1.3</td>
<td>6.1.2.4</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_AsText</td>
<td>returns the well-known text representation for the specified Geometry value</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_AsBinary</td>
<td>returns the well-known binary representation for the specified Geometry value</td>
<td>4.1.1.1</td>
<td>2.1.1.1</td>
<td>6.1.2.2</td>
<td></td>
</tr>
<tr>
<td>ST_AsGML</td>
<td>returns the GML representation for the specified Geometry value</td>
<td>4.1.1.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_GeomFromText</td>
<td>returns a Geometry value from its well-known text representation</td>
<td>4.1.1.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_GeomFromWKB</td>
<td>returns a Geometry value from its well-known binary representation</td>
<td>4.1.1.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>13249-3 Clause</td>
<td>99-049 Clause</td>
<td>06-103r4 Clause</td>
<td>06-104r4 Clause</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ST_GeomFromGML</td>
<td>returns a Geometry value from its GML representation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 49 Point Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_X</td>
<td>returns the x coordinate value of a Point value</td>
<td>4.1.3.1</td>
<td>3.2.11.2</td>
<td>6.1.4.2</td>
<td>7.2.9.1</td>
</tr>
<tr>
<td>ST_Y</td>
<td>returns the y coordinate value of a Point value</td>
<td>4.1.3.1</td>
<td>3.2.11.2</td>
<td>6.1.4.2</td>
<td>7.2.9.1</td>
</tr>
<tr>
<td>ST_Z</td>
<td>returns the z coordinate value of a Point value</td>
<td>4.1.3.1</td>
<td></td>
<td>6.1.4.2</td>
<td>7.2.9.1</td>
</tr>
<tr>
<td>ST_M</td>
<td>returns the m coordinate value of a Point value</td>
<td>4.1.3.1</td>
<td></td>
<td>6.1.4.2</td>
<td>7.2.9.1</td>
</tr>
<tr>
<td>ST_PointFromText</td>
<td>returns a Point value from the well-known text representation of a Point</td>
<td>4.1.3.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_PointFromWKB</td>
<td>returns a Point value from the well-known binary representation of a Point.</td>
<td>4.1.3.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 50 Curve Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_Length</td>
<td>returns the length of a Curve value</td>
<td>4.1.4.1</td>
<td>2.1.5.1</td>
<td>6.1.6.2</td>
<td>7.2.10.1</td>
</tr>
<tr>
<td>ST_StartPoint</td>
<td>returns the Point value that is the start point of a Curve value</td>
<td>4.1.4.1</td>
<td>2.1.5.1</td>
<td>6.1.6.2</td>
<td>7.2.10.1</td>
</tr>
<tr>
<td>ST_EndPoint</td>
<td>returns the Point value that is the end point of a Curve value</td>
<td>4.1.4.1</td>
<td>2.1.5.1</td>
<td>6.1.6.2</td>
<td>7.2.10.1</td>
</tr>
<tr>
<td>ST_IsClosed</td>
<td>tests if a Curve value is closed</td>
<td>4.1.4.1</td>
<td>2.1.5.1</td>
<td>6.1.6.2</td>
<td>7.2.10.1</td>
</tr>
<tr>
<td>ST_IsRing</td>
<td>tests if an Curve value is a ring</td>
<td>4.1.4.1</td>
<td>2.1.5.1</td>
<td>6.1.6.2</td>
<td>7.2.10.1</td>
</tr>
</tbody>
</table>

### Table 51 LineString Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_NumPoints</td>
<td>returns the cardinality of the Point collection in the LineString value</td>
<td>4.1.5.1</td>
<td>2.1.6.1</td>
<td>6.1.7.2</td>
<td>7.2.11.1</td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>13249-3 Clause</td>
<td>99-049 Clause</td>
<td>06-103r4 Clause</td>
<td>06-104r4 Clause</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>ST_PointN</td>
<td>returns the specified element in the Point collection in the LineString value</td>
<td>4.1.5.1</td>
<td>2.1.6.1</td>
<td>6.1.7.2</td>
<td>7.2.11.1</td>
</tr>
<tr>
<td>ST_LineFromText</td>
<td>returns a LineString value from the well-known text representation of a LineString</td>
<td>4.1.5.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_LineFromWKB</td>
<td>returns a LineString value from the well-known binary representation of a LineString</td>
<td>4.1.5.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 52 Surface Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_Area</td>
<td>returns the area of a Surface value</td>
<td>4.1.8.1</td>
<td>2.1.9.1</td>
<td>6.1.10.2</td>
<td>7.2.12.1</td>
</tr>
<tr>
<td>ST_Centroid</td>
<td>returns the Point value that is the mathematical centroid of the Surface value</td>
<td>4.1.8.1</td>
<td>2.1.9.1</td>
<td>6.1.10.2</td>
<td>7.2.12.1</td>
</tr>
<tr>
<td>ST_PointOnSurface</td>
<td>returns a Point value that is guaranteed to be on the Surface value</td>
<td>4.1.8.1</td>
<td>2.1.9.1</td>
<td>6.1.10.2</td>
<td>7.2.12.1</td>
</tr>
</tbody>
</table>

Table 53 Polygon Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_ExteriorRing</td>
<td>returns the exterior ring of a Polygon value</td>
<td>4.1.9.1</td>
<td>2.1.10.1</td>
<td>6.1.11.2</td>
<td>7.2.13.1</td>
</tr>
<tr>
<td>ST_NumInteriorRing</td>
<td>returns the cardinality of the collection of interior rings of a Polygon value</td>
<td>4.1.9.1</td>
<td>2.1.10.1</td>
<td>6.1.11.2</td>
<td>7.2.13.1</td>
</tr>
<tr>
<td>ST_InteriorRingN</td>
<td>returns the specified element in the collection of interior rings of a Polygon value</td>
<td>4.1.9.1</td>
<td>2.1.10.1</td>
<td>6.1.11.2</td>
<td>7.2.13.1</td>
</tr>
<tr>
<td>ST_PolyFromText</td>
<td>returns a Polygon value from the well-known text representation of a Polygon</td>
<td>4.1.10.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_PolyFromWKB</td>
<td>returns a Polygon value from the well-known binary representation of a Polygon</td>
<td>4.1.10.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 54 MultiGeometry Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_NumGeometries</td>
<td>returns the cardinality of the Geometry collection</td>
<td>4.1.11.1</td>
<td>3.2.16.2</td>
<td>6.1.3.2</td>
<td>7.2.15</td>
</tr>
<tr>
<td>ST_GeometryN</td>
<td>returns the specified element in the Geometry collection</td>
<td>4.1.11.1</td>
<td>3.2.16.2</td>
<td>6.1.3.2</td>
<td>7.2.15</td>
</tr>
<tr>
<td>ST_GeomCollFromTxt</td>
<td>returns a GeomCollection value from the well-known text representation of a GeomCollection</td>
<td>4.1.11.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_GeomCollFromWKB</td>
<td>returns a GeomCollection value from the well-known binary representation of a GeomCollection</td>
<td>4.1.11.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 55 MultiPoint Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_MPointFromText</td>
<td>returns a MultiPoint value from the well-known text representation of a MultiPoint</td>
<td>4.1.12.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_MPointFromWKB</td>
<td>returns a MultiPoint value from the well-known binary representation of a MultiPoint</td>
<td>4.1.12.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 56 MultiCurve Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>13249-3 Clause</td>
<td>99-049 Clause</td>
<td>06-103r4 Clause</td>
<td>06-104r4 Clause</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ST_IsClosed</td>
<td>tests if a MultiCurve value is closed</td>
<td>4.1.13.1</td>
<td>3.2.17.2</td>
<td>6.1.8.2</td>
<td>7.2.17.1</td>
</tr>
<tr>
<td>ST_Length</td>
<td>returns the length of a MultiCurve value</td>
<td>4.1.13.1</td>
<td>3.2.17.2</td>
<td>6.1.8.2</td>
<td>7.2.17.1</td>
</tr>
</tbody>
</table>

Table 57 MultiLineString Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_MLineFromText</td>
<td>returns a MultiLineString value from the well-known text representation of a MultiLineString</td>
<td>4.1.14.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_MLineFromWKB</td>
<td>returns a MultiLineString value from the well-known binary representation of a MultiLineString</td>
<td>4.1.14.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 58 MultiSurface Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_Area</td>
<td>returns the area of a MultiSurface value</td>
<td>4.1.15.1</td>
<td>3.2.18.2</td>
<td>6.1.13.2</td>
<td>7.2.19.1</td>
</tr>
<tr>
<td>ST_Centroid</td>
<td>returns the Point value that is the mathematical centroid of the MultiSurface value</td>
<td>4.1.15.1</td>
<td>3.2.18.2</td>
<td>6.1.13.2</td>
<td>7.2.19.1</td>
</tr>
<tr>
<td>ST_PointOnSurface</td>
<td>returns a Point value that is guaranteed to be on the MultiSurface value</td>
<td>4.1.15.1</td>
<td>3.2.18.2</td>
<td>6.1.13.2</td>
<td>7.2.19.1</td>
</tr>
</tbody>
</table>

Table 59 MultiPolygon Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_MPolyFromText</td>
<td>returns a MultiPolygon value from the well-known text representation of a MultiPolygon</td>
<td>4.1.16.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_MPolyFromWKB</td>
<td>returns a MultiPolygon value from the well-known binary representation of a MultiPolygon</td>
<td>4.1.16.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST_BdMPolyFromText</td>
<td>returns a MultiPolygon value from a well-known text representation of a MultiLineString</td>
<td>4.1.16.2</td>
<td>3.2.6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routine Name</td>
<td>Description</td>
<td>13249-3 Clause</td>
<td>99-049 Clause</td>
<td>06-103r4 Clause</td>
<td>06-104r4 Clause</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>ST_BdMPolyFromWKB</td>
<td>returns a MultiPolygon value from a well-known binary representation of a MultiLineString</td>
<td>4.1.16.2</td>
<td>3.2.7.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 60 Spatial Predicates**

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Description</th>
<th>13249-3 Clause</th>
<th>99-049 Clause</th>
<th>06-103r4 Clause</th>
<th>06-104r4 Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_Equals</td>
<td>tests if a Geometry value is spatially equal to another Geometry value.</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Relate</td>
<td>tests if a Geometry value is spatially related to another Geometry value by testing for intersections between the interior, boundary and exterior of the two Geometry values as specified by the intersection matrix.</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Disjoint</td>
<td>tests if a Geometry value is spatially disjoint from another Geometry value</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Intersects</td>
<td>tests if a Geometry value spatially intersects another Geometry value</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Touches</td>
<td>tests if a Geometry value spatially touches another Geometry value</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Crosses</td>
<td>tests if a Geometry value spatially crosses another Geometry value</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Within</td>
<td>tests if a Geometry value is spatially within another Geometry value</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Contains</td>
<td>tests if a Geometry value spatially contains another Geometry value</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
<tr>
<td>ST_Overlaps</td>
<td>tests if a Geometry value spatially overlaps another Geometry value</td>
<td>4.1.2.3</td>
<td>3.2.19.2</td>
<td>6.1.15.3</td>
<td>7.2.8.1</td>
</tr>
</tbody>
</table>
Annex I  Hierarchical Metadata Example (Informative)

The first example use case is from ISO19115 H.2.

Suppose we have this metadata:

```sql
CREATE TABLE gpkg_metadata (
    id INTEGER NOT NULL PRIMARY KEY,
    md_scope TEXT NOT NULL DEFAULT 'undefined',
    metadata_standard_URI TEXT NOT NULL DEFAULT
    'http://schemas.opengis.net/iso/19139/',
    metadata BLOB NOT NULL DEFAULT (zeroblob(4))
)
```

<table>
<thead>
<tr>
<th>id</th>
<th>md_scope</th>
<th>metadata_standard_uri</th>
<th>metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>undefined</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>3</td>
<td>series</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>4</td>
<td>dataset</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>5</td>
<td>featureType</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>6</td>
<td>feature</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>7</td>
<td>attributeType</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>8</td>
<td>attribute</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
</tbody>
</table>

and this reference table definition:

```sql
CREATE TABLE gpkg_metadata_reference (  
    reference_scope TEXT NOT NULL DEFAULT "table",  
    table_name TEXT NOT NULL DEFAULT "undefined",  
    column_name TEXT NOT NULL DEFAULT "undefined",  
    row_id_value INTEGER NOT NULL DEFAULT 0,  
    timestamp TEXT NOT NULL DEFAULT (strftime('%Y-%m-%dT%H:%M:%fZ', CURRENT_TIMESTAMP)),  
    md_file_id INTEGER NOT NULL DEFAULT 0,  
    md_parent_id INTEGER NOT NULL DEFAULT 0,  
    CONSTRAINT crmr_mfi_fk FOREIGN KEY (md_file_id) REFERENCES gpkg_metadata(id),  
    CONSTRAINT crmr_mpi_fk FOREIGN KEY (md_parent_id) REFERENCES gpkg_metadata(id)
)
```

H.2 1) Consider a geographic data provider generating vector mapping data for three Administrative areas (A, B and C). ... The metadata could be carried exclusively at Dataset Series level.

Then we need a record for each layer table for the three admin areas, like this:

```sql
INSERT INTO gpkg_metadata_reference VALUES (  
    'table', /* reference_type */  
    'roads', /* table_name */  
    'undefined', /* column_name */  
    -1, /* row_id_value */  
    (datetime('now'))
)  
```
H.2 2) After some time alternate vector mapping of Administrative area A becomes available. The metadata would then be extended for Administrative area A, to describe the new quality date values. These values would supersede those given for the Dataset series, but only for Administrative area A. The metadata for B and C would remain unchanged. This new metadata would be recorded at Dataset level.

Then we need a record for each layer table in "A" like this:

```sql
INSERT INTO gpkg_metadata_reference VALUES ('table', /* reference type */ 'roads', /* table name */ 'undefined', /* column_name */ -1, /* row_id_value */ (datetime('now'))),
5, /* md_file_id */ 4 /* md_parent_id */
```

H.2 3) Eventually further data becomes available for Administrative area A, with a complete re-survey of the road network. Again this implies new metadata for the affected feature types. This metadata would be carried at Feature type level for Administrative area A. All other metadata relating to other feature types remains unaffected. Only the metadata for roads in Administrative area A is modified. This road metadata is recorded at Feature type level.

Then we need a record for each layer table for the roads network, like this:

```sql
INSERT INTO gpkg_metadata_reference VALUES ('table', /* reference type */ 'roads', /* table name */ 'undefined', /* column_name */ -1, /* row_id_value */ (datetime('now'))),
5, /* md_file_id */ 4 /* md_parent_id */
```

H.2 4) An anomaly in the road survey is identified, in that all Overhead clearances for the Administrative area A have been surveyed to the nearest metre. These are re-surveyed to the nearest decimetre. This re-survey implies new metadata for the affected attribute type ‘Overhead Clearance’. All other metadata for Administrative area A remains unaffected. This ‘Overhead Clearance’ metadata is recorded at Attribute Type level

Then we need a record for each layer table in the roads network with attribute type 'Overhead Clearance', like this:

```sql
INSERT INTO gpkg_metadata_reference VALUES ('column', /* reference type */ 'roads', /* table name */ 'overhead_clearance', /* column_name */ /* md_file_id */ /* md_parent_id */
```
-1, /* row_id_value */
(datetime('now')),
7, /* md_file_id */
4 /* md_parent_id */
)

H.2 5) A new bridge is constructed in Administrative area A. This new data is reflected in the geographic data for Administrative area A, and new metadata is required to record this new feature. All other metadata for Administrative area A remains unaffected. This new feature metadata is recorded at Feature instance level.

Then we need a record for the bridge layer table row for the new bridge, like this:

```
INSERT INTO gpkg_metadata_reference VALUES ('row', /* reference type */
'bridge', /* table name */
'undefined', /* column_name */
987, /* row_id_value */
(datetime('now')),
6, /* md_file_id */
4 /* md_parent_id */
)
```

H.2 6) The overhead clearance attribute of the new bridge was wrongly recorded, and is modified. Again this new attribute requires new metadata to describe the modification. All other metadata for Administrative area A remains unaffected. This new attribute metadata is recorded at Attribute instance level.

Then we need a record for the clearance attribute value, like this:

```
INSERT INTO gpkg_metadata_reference VALUES ('row/col', /* reference type */
'bridge', /* table name */
'overhead_clearance', /* column_name */
987, /* row_id_value */
(datetime('now')),
8, /* md_file_id */
4 /* md_parent_id */
)
```

The second example use case is for a field data collection session. This use case demonstrates a mechanism to indicate which data in a GeoPackage that was originally loaded with data from one or more services has been collected or updated since the initial load, and therefore MAY need to be uploaded to update the original services (e.g. WFS, WCS, WMTS).

Suppose a user with a mobile handheld device goes out in the field and collects observations of a new "Point of Interest" (POI) feature type, and associated metadata about the field session, the new feature type, some POI instances and some of their attributes (e.g. spatial accuracy, attribute accuracy) that results in the following additional metadata:

<table>
<thead>
<tr>
<th>id</th>
<th>md_scope</th>
<th>metadata_standard_uri</th>
<th>metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>undefined</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>1</td>
<td>fieldSession</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>10</td>
<td>featureType</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
</tbody>
</table>
This example assumes that the field session data is still considered "raw" and won't be considered a data set or part of a data series until it has been verified and cleaned, but if that is wrong then additional series and data set metadata could be added.

Then we need a gpkg_metadata_reference record for the field session for the new POI table, whose md_parent_id is undefined:

```sql
INSERT INTO gpkg_metadata_reference VALUES ('table', /* reference type */ 'poi', /* table name */ 'undefined', /* column_name */ -1, /* row_id_value */ (strftime('%Y-%m-%dT%H:%M:%fZ','now'))), 1, /* md_file_id */ 0 /* md_parent_id */
)
```

Then we need a gpkg_metadata_reference record for the feature type for the new POI table, whose md_parent_id is that of the field session:

```sql
INSERT INTO gpkg_metadata_reference VALUES ('table', /* reference type */ 'poi', /* table name */ 'undefined', /* column_name */ -1, /* row_id_value */ (strftime('%Y-%m-%dT%H:%M:%fZ','now'))), 10, /* md_file_id */ 1 /* md_parent_id */
)
```

Then we need gpkg_metadata_reference records for the poi feature instance rows, whose md_parent_id is that of the field session:

```sql
INSERT INTO gpkg_metadata_reference VALUES ('row', /* reference type */ 'poi', /* table name */ 'undefined', /* column_name */ 1, /* row_id_value */ (strftime('%Y-%m-%dT%H:%M:%fZ','now'))), 11, /* md_file_id */ 1 /* md_parent_id */
)
```
INSERT INTO gpkg_metadata_reference VALUES ('row', /* reference type */
'poi', /* table name */
'undefined', /* column_name */
2, /* row_id_value */
14, /* md_file_id */
1 /* md_parent_id */
)

INSERT INTO gpkg_metadata_reference VALUES ('row', /* reference type */
'poi', /* table name */
'undefined', /* column_name */
3, /* row_id_value */
(strftime('%Y-%m-%dT%H:%M:%fZ','now')),
17, /* md_file_id */
1 /* md_parent_id */
)

And finally we need gpkg_metadata_reference records for the poi attribute instance metadata, whose md_parent_id is that of the field session:

INSERT INTO gpkg_metadata_reference VALUES ('row/col', /* reference type */
'poi', /* table name */
'point', /* column_name */
1, /* row_id_value */
(strftime('%Y-%m-%dT%H:%M:%fZ','now')),
12, /* md_file_id */
1 /* md_parent_id */
)

INSERT INTO gpkg_metadata_reference VALUES ('row/col', /* reference type */
'poi', /* table name */
'point', /* column_name */
2, /* row_id_value */
(strftime('%Y-%m-%dT%H:%M:%fZ','now')),
15, /* md_file_id */
1 /* md_parent_id */
)

INSERT INTO gpkg_metadata_reference VALUES ('row/col', /* reference type */
'poi', /* table name */
'point', /* column_name */
3, /* row_id_value */
(strftime('%Y-%m-%dT%H:%M:%fZ','now')),
18, /* md_file_id */
1 /* md_parent_id */
)

INSERT INTO gpkg_metadata_reference VALUES ('row/col', /* reference type */
'poi', /* table name */
'category', /* column_name */
...
As long as all metadata collected in the field session either directly (as above) or indirectly (suppose there were a data set level metadata_reference record intermediary) refers to the field session metadata via md_parent_id values, then this chain of metadata references identifies the newly collected information, as Joan requested, in addition to the metadata.

So here is the data after both examples:

```
xm1_metadata

<table>
<thead>
<tr>
<th>id</th>
<th>md_scope</th>
<th>metadata_standard_uri</th>
<th>metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>undefined</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>1</td>
<td>fieldSession</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>2</td>
<td>collectionSession</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>3</td>
<td>series</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>4</td>
<td>dataset</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>5</td>
<td>featureType</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>6</td>
<td>feature</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>7</td>
<td>attributeType</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>8</td>
<td>attribute</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>10</td>
<td>featureType</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>11</td>
<td>feature</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>12</td>
<td>attribute</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>13</td>
<td>attribute</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>14</td>
<td>feature</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>15</td>
<td>attribute</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
<tr>
<td>16</td>
<td>attribute</td>
<td><a href="http://schemas.opengis.net/iso/19139/">http://schemas.opengis.net/iso/19139/</a></td>
<td>BLOB</td>
</tr>
</tbody>
</table>
```
| 17 | feature | http://schemas.opengis.net/iso/19139/ | BLOB |
| 18 | attribute | http://schemas.opengis.net/iso/19139/ | BLOB |
| 19 | attribute | http://schemas.opengis.net/iso/19139/ | BLOB |

gpkg_metadata_reference

<table>
<thead>
<tr>
<th>reference_type</th>
<th>table_name</th>
<th>column_name</th>
<th>row_id_value</th>
<th>timestamp</th>
<th>md_file_id</th>
<th>md_parent_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>roads</td>
<td>undefined</td>
<td>0</td>
<td>ts</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>table</td>
<td>roads</td>
<td>undefined</td>
<td>0</td>
<td>ts</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>table</td>
<td>roads</td>
<td>undefined</td>
<td>0</td>
<td>ts</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>column</td>
<td>roads</td>
<td>overhead_clearance</td>
<td>0</td>
<td>ts</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>row</td>
<td>bridge</td>
<td>undefined</td>
<td>987</td>
<td>ts</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>row</td>
<td>bridge</td>
<td>overhead_clearance</td>
<td>987</td>
<td>ts</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>table</td>
<td>poi</td>
<td>undefined</td>
<td>0</td>
<td>ts</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>undefined</td>
<td>0</td>
<td>ts</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>undefined</td>
<td>1</td>
<td>ts</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>undefined</td>
<td>2</td>
<td>ts</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>overhead_clearance</td>
<td>3</td>
<td>ts</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>point</td>
<td>1</td>
<td>ts</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>point</td>
<td>2</td>
<td>ts</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>point</td>
<td>3</td>
<td>ts</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>category</td>
<td>1</td>
<td>ts</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>category</td>
<td>2</td>
<td>ts</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>row</td>
<td>poi</td>
<td>category</td>
<td>3</td>
<td>ts</td>
<td>19</td>
<td>1</td>
</tr>
</tbody>
</table>
Annex J  Raster or Tile Metadata Example (Informative)
A number of raster image processing problems MAY require the support of more metadata that is contained in the image itself. Applications MAY use the gpkg_metadata and gpkg_metadata_reference tables defined in clause 1.2.5 to store raster image metadata defined according to standard authoritative or application or vendor specific metadata models. An example of the data items in such a model is shown in the following table.

- Rational Polynomial Coefficient
- Photometric Interpretation
- No Data Value
- Compression Quality Factor
- Georectification
- NIIRS
- Min X
- Min Y
- Max X
- Max Y
Annex K  Normative References (Normative)
The following normative documents contain provisions which, through reference in this text, constitute provisions of OGC 12-128 For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of OGC 12-128 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.


[19] MIL_STD_2401 DoD World Geodetic System 84 (WGS84), 11 January 1994


[21] ITU-T Recommendation T.81 (09/92) with Corrigendum (JPEG)


[33] W3C Recommendation 26 November 2008 Extensible Markup Language (XML) 1.0 (Fifth Edition) http://www.w3.org/TR/xml/

[34] W3C Recommendation 8 December 2009 Namespaces in XML 1.0 (Third Edition) http://www.w3.org/TR/REC-xml-names/


[40] ISO 19115 Geographic information -- Metadata, 8 May 2003, with Technical Corrigendum 1, 5 July 2006


[42] OGC® 10-100r3 Geography Markup Language (GML) simple features profile (with technical note) http://portal.opengeospatial.org/files/?artifact_id=42729


Annex L  Bibliography (Informative)
[B12] https://github.com/mapbox/mbtiles-spec
[B13] https://www.gaia-gis.it/fossil/librasterlite/index
[B20] NATO IIRS STANAG, NATO Imagery Interpretability Rating Scale (NIIRS) STANAG 7194 Edition 1 2009
[B26] Sample Civil NIIRS Images http://www.fas.org/irp/imint/niirs_c/append.htm
[B30]  https://www.gaia-gis.it/fossil/libspatialite/wiki?name=switching-to-4.0
[B32]  ISO 19115-2 Geographic information - - Metadata - Part 2: Metadata for imagery and gridded data
[B33]  ISO 19139: Geographic information -- Metadata -- XML schema implementation
[B34]  Dublin Core Metadata Initiative  http://dublincore.org/  IETF RFC 5013
[B36]  Content Standard for Digital Geospatial Metadata (CSDGM)
[B38]  Department of Defense Discovery Metadata Specification (DDMS)  
http://metadata.ces.mil/mdr/irs/DDMS/
[B39]  NMF NGA.STND.0012_2.0 / NMIS NGA.STND.0018_1.0
[B43]  ISO 19110 Geographic information – Methodology for feature cataloguing
[B45]  Simple Knowledge Organization System (SKOS)  http://www.w3.org/TR/skos-reference/
[B47]  ISO 19109 Geographic information - Rules for application schema
[B52]  https://www.gaia-gis.it/fossil/libspatialite/index