Open Geospatial Consortium

External identifier of this OGC® document: http://www.opengis.net/doc/IS/geopackage/1.2 (http://www.opengis.net/doc/IS/geopackage/1.2)
URL for this OGC® document: http://www.geopackage.org/spec (http://www.geopackage.org/spec)
Internal reference number of this OGC® document: OGC 12-128r13
Version: 1.2-SNAPSHOT
Category: OGC® Encoding Standard
Editor: Jeff Yutzler
Editor Emeritus: Paul Daisey
Previous Version: http://www.opengis.net/doc/IS/geopackage/1.1.0 (http://www.opengis.net/doc/IS/geopackage/1.1.0)
Publication Date: 2016-09-28
Approval Date: 2016-05-08
Submission Date: 2016-02-18
Original Version: http://www.opengis.net/doc/IS/geopackage/1.0 (http://www.opengis.net/doc/IS/geopackage/1.0)

OGC® GeoPackage Encoding Standard

Warning

This document is an OGC Member approved international standard. This document is available on a royalty free, non-discriminatory basis. 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. This is a technical amendment to OGC 12-128r1.

Document type: OGC® Publicly Available Standard
Document subtype: Encoding Standard
Document stage: Approved
Document language: English

License Agreement

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it. None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Patent Call

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

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 standard set forth in this document, and to provide supporting documentation.

Abstract

This OGC® Encoding Standard defines GeoPackages 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" storage 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. GeoPackages are interoperable across all enterprise and personal computing environments, and are particularly useful on mobile devices like cell phones and tablets in communications environments with limited connectivity and bandwidth.

Keywords

ogcdoc, geopackage, sqlite, raster, tiles, vector, feature, data, storage, exchange, mobile, smartphone, tablet

Table of Contents

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 geospatially 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 [B1] [B2], 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 OGC® Encoding Standard defines GeoPackages 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** is a platform-independent SQLite [5] database file that contains GeoPackage data and metadata tables shown in GeoPackage Tables Overview below, with specified definitions, integrity assertions, format limitations and content constraints. The allowable content of a GeoPackage is entirely defined in this standard.

An **Extended GeoPackage** is a **GeoPackage** that contains any additional data elements (tables or columns) or SQL constructs (data types, functions, indexes, constraints or triggers) that are not automatically maintained within the SQLite data file or that result in a change in behavior not specified in this encoding standard.

A **GeoPackage** MAY be “empty” (contain user data table(s) for vector features, non-spatial attributes, 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 metadata CAN describe GeoPackage data contents and identify external data synchronization sources and targets. A GeoPackage MAY contain spatial indexes on feature geometries and SQL triggers to maintain indexes and enforce content constraints.

A **GeoPackage SQLite Configuration** consists of the SQLite 3 software library and a set of compile- and runtime configurations options.

A **GeoPackage SQLite Extension** is a SQLite loadable extension that MAY provide SQL functions [12] to support spatial indexes and SQL triggers linked to a SQLite library with specified configuration requirements to provide SQL API [1] [2] [3] [4] access to a GeoPackage file. This standard does not address the issues listed in the [potential_future_work] clause in Background and Context (Normative), which MAY be addressed in a subsequent version of this standard or by other specifications.

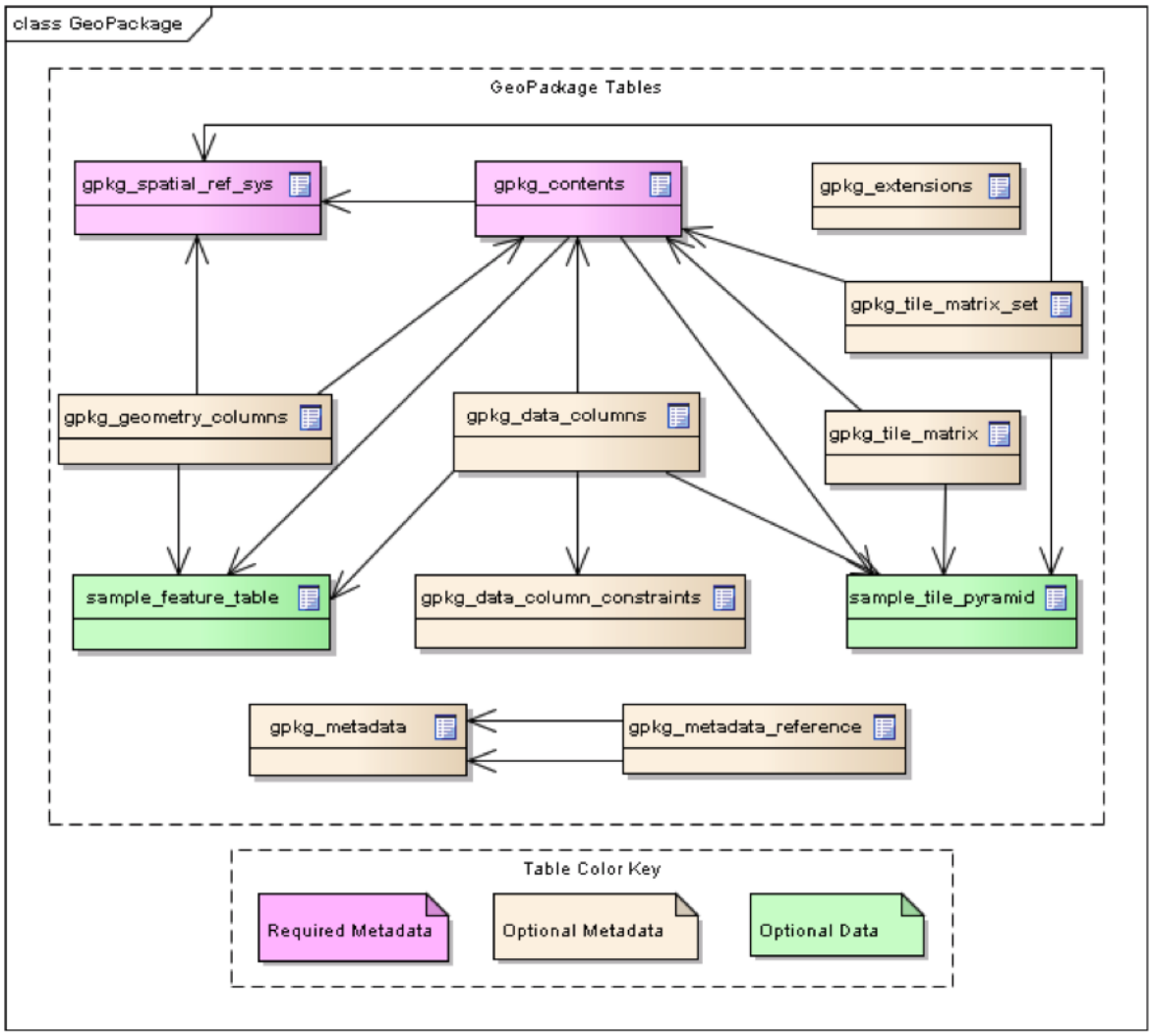

Figure 1. GeoPackage Tables Overview

1. Base

The required capabilities specified in this clause serve as the base for options specified in clause Options and extensions specified in clause Registered Extensions (Normative). All gpkg_* tables and views and all tiles user data tables specified in this standard SHALL have only the specified columns and table constraints. Any features user data tables MAY have columns in addition to those specified. All specified table, view, column, trigger, and constraint name values SHALL be lowercase.

1.1. Core

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

1.1.1. SQLite Container

The SQLite software library provides a self-contained, single-file, cross-platform, serverless, transactional, open source RDBMS container. The GeoPackage standard defines a SQL database schema designed for use with the SQLite software library. Using SQLite as the basis for GeoPackage simplifies production, distribution and use of GeoPackages and assists in guaranteeing 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

> **Requirement 1**
>
> A GeoPackage SHALL be a SQLite (http://www.sqlite.org/) [5] database file using version 3 of the SQLite file format (http://sqlite.org/fileformat2.html) [6] [7]. The first 16 bytes of a GeoPackage SHALL be the null-terminated ASCII [B4] string “SQLite format 3” [1]. [2]

> **Requirement 2**
>
> A GeoPackage SHALL contain a value of 0x47504B47 ("GPKG" in ASCII) in the "application_id" field of the SQLite database header to indicate that it is a GeoPackage. [3] A GeoPackage SHALL contain an appropriate value in "user_version" field of the SQLite database header to indicate its version. The value SHALL be in integer with a major version, two-digit minor version, and two-digit bug-fix. For GeoPackage Version 1.2 this value is 0x000027D8 (the hexadecimal value for 10200). [4]

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.

1.1.1.1.2. File Extension Name

> **Requirement 3**
>
> A GeoPackage SHALL have the file extension name “.gpkg”.

~~It is RECOMMENDED that Extended GeoPackages use the file extension “.gpkx”, but this is NOT a GeoPackage requirement.~~

1.1.1.1.3. File Contents

> **Requirement 4**
>
> A GeoPackage SHALL only contain data elements, SQL constructs and GeoPackage extensions with the “gpkg” author name specified in this encoding standard.

In order to guarantee maximum interoperability between applications, GeoPackages SHALL NOT contain data elements (tables or columns), SQL constructs (data types, indexes, constraints or triggers) or extensions that are not specified in this encoding standard. SQLite databases that use constructs from the GeoPackage standard but extend those constructs to contain elements not specified in the core GeoPackage standard are referred to as Extended GeoPackages throughout this standard.

> **Requirement 5**
>
> The columns of tables in a GeoPackage SHALL only be declared using one of the data types specified in table GeoPackage Data Types.

Table 1. GeoPackage Data Types

Data Type	Size and Description
BOOLEAN	A boolean value representing true or false. Stored as SQLite INTEGER with value 0 for false or 1 for true
TINYINT	8-bit signed two’s complement integer. Stored as SQLite INTEGER with values in the range [-128, 127]
SMALLINT	16-bit signed two’s complement integer. Stored as SQLite INTEGER with values in the range [-32768, 32767]

Data Type	Size and Description
MEDIUMINT	32-bit signed two's complement integer. Stored as SQLite INTEGER with values in the range [-2147483648, 2147483647]
INT, INTEGER	64-bit signed two's complement integer. Stored as SQLite INTEGER with values in the range [-9223372036854775808, 9223372036854775807]
FLOAT	32-bit IEEE floating point number. Stored as SQLite REAL limited to values that can be represented as a 4-byte IEEE floating point number
DOUBLE, REAL	64-bit IEEE floating point number. Stored as SQLite REAL
TEXT{(maxchar_count)}	Variable length string encoded in either UTF-8 or UTF-16, determined by PRAGMA encoding; see http://www.sqlite.org/pragma.html#pragma_encoding (http://www.sqlite.org/pragma.html#pragma_encoding). The optional maxchar_count defines the maximum number of characters in the string. If not specified, the length is unbounded. The count is provided for informational purposes, and applications MAY choose to truncate longer strings if encountered. When present, it is best practice for applications to adhere to the character count. Stored as SQLite TEXT
BLOB{(max_size)}	Variable length binary data. The optional max_size defines the maximum number of bytes in the blob. If not specified, the length is unbounded. The size is provided for informational purposes. When present, it is best practice for applications adhere to the maximum blob size. Stored as SQLite BLOB
<geometry_type_name>	Geometry encoded as per clause Geometry Encoding. <geometry type_name> is one of the geometry types listed in Geometry Types (Normative) encoded per clause 2.1.3 or a user-defined geometry type encoded per clause 3.1.2 and User Defined Geometry Types Extension of GeoPackageBinary Geometry Encoding. Geometry Types XY, XYZ, XYM and XYZM geometries use the same data type. Stored as SQLite BLOB
DATE	ISO-8601 date string in the form YYYY-MM-DD encoded in either UTF-8 or UTF-16. See TEXT. Stored as SQLite TEXT
DATETIME	ISO-8601 date/time string in the form YYYY-MM-DDTHH:MM:SS.SSSZ with T separator character and Z suffix for coordinated universal time (UTC) encoded in either UTF-8 or UTF-16. See TEXT. Stored as SQLite TEXT

1.1.1.1.4. File Integrity

> **Requirement 6**
>
> The SQLite PRAGMA integrity_check SQL command SHALL return "ok" for a GeoPackage file. [5]

> **Requirement 7**
>
> The SQLite PRAGMA foreign_key_check SQL with no parameter value SHALL return an empty result set indicating no invalid foreign key values for a GeoPackage file.

1.1.1.2. API

1.1.1.2.1. Structured Query Language (SQL)

> **Requirement 8**
>
> A GeoPackage SQLite Configuration SHALL provide SQL access to GeoPackage contents via SQLite version 3 (http://www.sqlite.org/download.html) [6] software APIs. [6]

1.1.1.2.2. Every GPKG SQLite Configuration

The SQLite (http://www.sqlite.org/download.html) [8] library has many compile time (http://www.sqlite.org/compile.html) and run time (http://www.sqlite.org/pragma.html) options that MAY be used to configure SQLite for different uses. Certain elements of the GeoPackage specification depend on the availability of SQLite functionality at runtime. This clause specifies the set of compile time options that SHALL or SHALL NOT be used.

Requirement 9

Every GeoPackage SQLite Configuration SHALL have the SQLite library compile time options specified in clause 1.1.1.2.2 table Every GeoPackage SQLite Configuration.

Table 2. Every GeoPackage SQLite Configuration

Setting	Option	Shall / Not	Discussion
compile	SQLITE_OMIT_*	Not	SHALL NOT include any OMIT options from http://www.sqlite.org/compile.html#omitfeatures (http://www.sqlite.org/compile.html#omitfeatures).

1.1.2. Spatial Reference Systems

1.1.2.1. Data

1.1.2.1.1. Table Definition

Requirement 10

A GeoPackage SHALL include a `gpkg_spatial_ref_sys` table per clause 1.1.2.1.1 Table Definition, Table Spatial Ref Sys Table Definition and Table gpkg_spatial_ref_sys Table Definition SQL.

A table named `gpkg_spatial_ref_sys` is the first component of the standard SQL schema for simple features described in clause Simple Features SQL Introduction 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 the columns specified in SQL/MM (ISO 13249-3) [12] and shown in Spatial Ref Sys Table Definition below containing data that defines spatial reference systems. Views of this table MAY be used to provide compatibility with the SQL/MM (http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=53698) [12] (see SQL/MM View of gpkg_spatial_ref_sys Definition SQL (Informative)) and OGC Simple Features SQL (http://portal.opengeospatial.org/files/?artifact_id=25354) [9][10][11] (Table 21) specifications.

Table 3. Spatial Ref Sys Table Definition

Column Name	Column Type	Column Description	Null	Key
`srs_name`	TEXT	Human readable name of this SRS	no	
`srs_id`	INTEGER	Unique identifier for each Spatial Reference System within a GeoPackage	no	PK
`organization`	TEXT	Case-insensitive name of the defining organization e.g. EPSG or epsg	no	
`organization_coordsys_id`	INTEGER	Numeric ID of the Spatial Reference System assigned by the organization	no	
`definition`	TEXT	Well-known Text [32] Representation of the Spatial Reference System	no	
`description`	TEXT	Human readable description of this SRS	yes	

See gpkg_spatial_ref_sys Table Definition SQL.

1.1.2.1.2. Table Data Values

Definition column WKT values in the `gpkg_spatial_ref_sys` table SHALL define the Spatial Reference Systems used by feature geometries and tile images, unless these SRS are unknown and therefore undefined as specified in Requirement 11. Values SHALL be constructed per the EBNF syntax in [32] clause 7. EBNF name and number values MAY be obtained from any specified authority, e.g. [13][14]. For example, see the return value in [spatial_ref_sys_data_values_default] Test Method step (3) used to test the definition for WGS-84 per Requirement 11:

Requirement 11

The `gpkg_spatial_ref_sys` table SHALL contain at a minimum the records listed in Spatial Ref Sys Table Records. The record with an `srs_id` of 4326 SHALL correspond to WGS-84 (http://www.google.com/search?as_q=WGS-84) [15] as defined by EPSG (http://www.epsg.org/Geodetic.html) [B3] in 4326 (http://www.epsg-registry.org/report.htm?type=selection&entity=urn:ogc:def:crs:EPSG::4326&reportDetail=long&title=WGS%2084&style=urn:uuid:report-style:default-with-code&style_name=OGP%20Default%20With%20Code) [13][14]. The record with an `srs_id` of -1 SHALL be used for undefined Cartesian coordinate reference systems. The record with an `srs_id` of 0 SHALL be used for undefined geographic coordinate reference systems.

Table 4. Spatial Ref Sys Table Records

`srs_name`	`srs_id`	`organization`	`organization_coordsys_id`	`definition`	`description`
any	4326	`EPSG` or `epsg`	4326	any	any
any	-1	`NONE`	-1	`undefined`	any
any	0	`NONE`	0	`undefined`	any

Requirement 12

The `gpkg_spatial_ref_sys` table in a GeoPackage SHALL contain records to define all spatial reference systems used by features and tiles in a GeoPackage.

1.1.3. Contents

The `gpkg_contents` table is intended to provide a list of all geospatial contents in a GeoPackage. It provides identifying and descriptive information that an application can display to a user as a menu of geospatial data that is available for access and/or update.

1.1.3.1. Data

1.1.3.1.1. Table Definition

Requirement 13

A GeoPackage file SHALL include a `gpkg_contents` table per table Contents Table or View Definition and gpkg_contents Table Definition SQL.

Table 5. Contents Table or View Definition

Column Name	Type	Description	Null	Default	Key
`table_name`	TEXT	The name of the tiles, or feature table	no		PK
`data_type`	TEXT	Type of data stored in the table	no		
`identifier`	TEXT	A human-readable identifier (e.g. short name) for the table_name content	yes		UNIQUE
`description`	TEXT	A human-readable description for the table_name content	yes	''	
`last_change`	DATETIME	timestamp of last change to content, in ISO 8601 format	no	`strftime('%Y-%m-%dT%H:%M:%fZ', 'now')`	
`min_x`	DOUBLE	Bounding box minimum easting or longitude for all content in table_name	yes		
`min_y`	DOUBLE	Bounding box minimum northing or latitude for all content in table_name	yes		

Column Name	Type	Description	Null	Default	Key
`max_x`	DOUBLE	Bounding box maximum easting or longitude for all content in table_name	yes		
`max_y`	DOUBLE	Bounding box maximum northing or latitude for all content in table_name	yes		
`srs_id`	INTEGER	Spatial Reference System ID: `gpkg_spatial_ref_sys.srs_id`; when `data_type` is features, SHALL also match `gpkg_geometry_columns.srs_id`; When data_type is tiles, SHALL also match `gpkg_tile_matrix_set.srs_id`	yes		FK

See gpkg_contents Table Definition SQL.

1.1.3.1.2. Table Data Values

> **Requirement 14**
>
> The `table_name` column value in a `gpkg_contents` table row SHALL contain the name of a SQLite table or view.

The `data_type` specifies the type of content contained in the table, for example “features” per clause Features, “attributes” per clause Attributes, “tiles” per clause Tiles, or an implementer-defined value for other data tables per clause in an Extended GeoPackage.

The `last_change` SHOULD contain the timestamp of when the content in the referenced table was last updated, in ISO8601 format. Note that since it is not practical to ensure that this value is maintained properly in all cases, this value should be treated as informative.

> **Requirement 15**
>
> Values of the `gpkg_contents` table `last_change` column SHALL be in ISO 8601 (http://www.iso.org/iso/catalogue_detail?csnumber=40874) [29] format containing a complete date plus UTC hours, minutes, seconds and a decimal fraction of a second, with a ‘Z’ (‘zulu’) suffix indicating UTC. The ISO8601 format is as defined by the strftime function *%Y-%m-%dT%H:%M:%fZ* format string applied to the current time. [7]

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.

> **Requirement 16**
>
> Values of the `gpkg_contents` table `srs_id` column SHALL reference values in the `gpkg_spatial_ref_sys` table `srs_id` column.

---

2. Options

The optional capabilities specified in this clause depend on the required capabilities specified in clause Base above. Each subclause of this clause defines an indivisible module of functionality that can be used in GeoPackages. These modules are referred to as options. GeoPackages MAY use one or more options defined in this section. GeoPackages MAY omit the tables for options that are not used. As a minimum, a GeoPackage SHALL contain one user data table as defined by the Features or Tiles options in clauses Features and Tiles respectively.

> **Requirement 17**
>
> A GeoPackage SHALL contain features per clause Features and/or tiles per clause Tiles and row(s) in the `gpkg_contents` table with lowercase `data_type` column values of “features” and/or “tiles” describing the user data tables.

2.1. Features

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 standards [9][10][11][12] 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 Spatial Reference Systems above. Other components are defined below.

In a GeoPackage, “simple” features are geolocated using a linear geometry subset of the SQL/MM (ISO 13249-3) [12] geometry model shown in Core Geometry Model below.

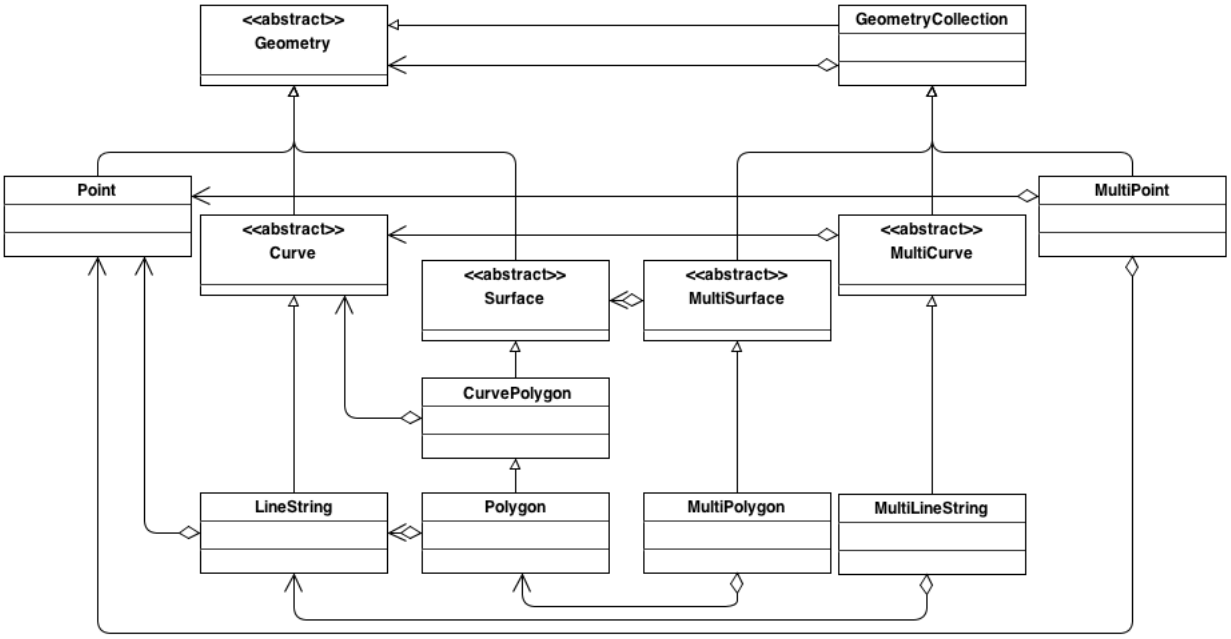

Figure 2. Core Geometry Model

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 [12].

- 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 more 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. [8]
- 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

> **Requirement 18**
>
> The `gpkg_contents` table SHALL contain a row with a lowercase `data_type` column value of “features” for each vector features user data table or view.

2.1.3. Geometry Encoding

2.1.3.1. Data

2.1.3.1.1. BLOB Format

> **Requirement 19**
>
> A GeoPackage SHALL store feature table geometries with or without optional elevation (Z) and/or measure (M) values in SQL BLOBs using the Standard GeoPackageBinary format specified in table GeoPackage SQL Geometry Binary Format and clause BLOB Format.

GeoPackage SQL Geometry Binary Format

```
GeoPackageBinaryHeader {
  byte[2] magic = 0x4750;  1
  byte version;            2
  byte flags;              3
  int32 srs_id;
  double[] envelope;       4
}

StandardGeoPackageBinary {
  GeoPackageBinaryHeader header;  5
  WKBGeometry geometry;           6
}
```

1. *GP* in ASCII
2. 8-bit unsigned integer, 0 = version 1
3. see bit layout of GeoPackageBinary flags byte
4. see flags envelope contents indicator code below
5. The X bit in the header flags field must be set to 0.
6. per OGC 06-103r4 [9] [9][10][11]

Table 6. bit layout of GeoPackageBinary flags byte

bit	7	6	5	4	3	2	1	0
use	R	R	X	Y	E	E	E	B

flag bits use:

- R: reserved for future use; set to 0
- X: GeoPackageBinary type
  - 0: StandardGeoPackageBinary. See below
  - 1: ExtendedGeoPackageBinary. See User Defined Geometry Types Extension of GeoPackageBinary Geometry Encoding.
- Y: empty geometry flag
  - 0: non-empty geometry
  - 1: empty geometry
- E: envelope contents indicator code (3-bit unsigned integer)
  - 0: no envelope (space saving slower indexing option), 0 bytes
  - 1: envelope is [minx, maxx, miny, maxy], 32 bytes
  - 2: envelope is [minx, maxx, miny, maxy, minz, maxz], 48 bytes
  - 3: envelope is [minx, maxx, miny, maxy, minm, maxm], 48 bytes
  - 4: envelope is [minx, maxx, miny, maxy, minz, maxz, minm, maxm], 64 bytes
  - 5-7: invalid
- 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 OGC 06-103r4 [9] does not provide a standardized encoding for an empty point set (i.e., *Point Empty* in Well-Known Text). In GeoPackages these points SHALL be encoded as a Point where each coordinate value is set to an IEEE-754 quiet NaN value. GeoPackages 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.

2.1.4. SQL Geometry Types

2.1.4.1. Data

2.1.4.1.1. Core Types

> **Requirement 20**
>
> A GeoPackage SHALL store feature table geometries with the basic simple feature geometry types (Geometry, Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeomCollection) in Geometry Types (Normative) Geometry Type Codes (Core) in the GeoPackageBinary geometry encoding format.

2.1.5. Geometry Columns

2.1.5.1. Data

2.1.5.1.1. Table Definition

> **Requirement 21**
>
> A GeoPackage with a `gpkg_contents` table row with a “features” `data_type` SHALL contain a `gpkg_geometry_columns` table or updateable view per Geometry Columns Table or View Definition and gpkg_geometry_columns Table Definition SQL.

The second component of the SQL schema for vector features in a GeoPackage is a `gpkg_geometry_columns` table that identifies the geometry columns and geometry types in tables that contain user data representing features.

Table 7. Geometry Columns Table or View Definition

Column Name	Type	Description	Key

`table_name`	TEXT	Name of the table containing the geometry column	PK, FK
`column_name`	TEXT	Name of a column in the feature table that is a Geometry Column	PK
`geometry_type_name`	TEXT	Name from Geometry Type Codes (Core) or Geometry Type Codes (Extension) in Geometry Types (Normative)	
`srs_id`	INTEGER	Spatial Reference System ID: `gpkg_spatial_ref_sys.srs_id`	FK
`z`	TINYINT	0: z values prohibited; 1: z values mandatory; 2: z values optional	
`m`	TINYINT	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.

Views of this table or view MAY be used to provide compatibility with the SQL/MM [12] SQL/MM View of gpkg_geometry_columns Definition SQL (Informative) and OGC Simple Features SQL [9][10][11] SF/SQL VIEW of gpkg_geometry_columns Definition SQL (Informative) specifications.

See gpkg_geometry_columns Table Definition SQL.

2.1.5.1.2. Table Data Values

Requirement 22

The `gpkg_geometry_columns` table or updateable view SHALL contain one row record for the geometry column in each vector feature data table (clause Vector Feature User Data Tables) in a GeoPackage.

Requirement 23

Values of the `gpkg_geometry_columns` `table_name` column SHALL reference values in the `gpkg_contents` `table_name` column for rows with a `data_type` of *features*.

Requirement 24

The `column_name` column value in a `gpkg_geometry_columns` row SHALL be the name of a column in the table or view specified by the `table_name` column value for that row.

Requirement 25

The `geometry_type_name` value in a `gpkg_geometry_columns` row SHALL be one of the uppercase geometry type names specified in Geometry Types (Normative).

Requirement 26

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.

Requirement 27

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

Requirement 28

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 Simple Features SQL Introduction above are tables that contain user data representing features. Feature attributes are columns in a feature table, including geometries. Features are rows in a feature table. [12]

> **Requirement 29**
>
> A GeoPackage MAY contain tables or updateable views containing vector features. Every such feature table or view in a GeoPackage SHALL have a column with column type INTEGER and *PRIMARY KEY AUTOINCREMENT* column constraints per EXAMPLE : Sample Feature Table or View Definition and sample_feature_table Table Definition SQL (Informative).

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 [B5] values in the `gpkg_metadata_reference` table as described in clause Metadata Reference Table below.

> **Requirement 30**
>
> A feature table SHALL have only one geometry column.

Feature data models [B23] 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.

> **Requirement 31**
>
> The declared SQL type of the geometry column in a vector feature user data table SHALL be the uppercase geometry type name from Annex G specified by the geometry_type_name column for that column_name and table_name in the gpkg_geometry_columns table.

Table 8. EXAMPLE : Sample Feature Table or View Definition

Column Name	Type	Description	Null	Default	Key
`id`	INTEGER	Autoincrement primary key	no		PK
`geometry`	GEOMETRY	GeoPackage Geometry	yes		
`text_attribute`	TEXT	Text attribute of feature	yes		
`real_attribute`	REAL	Real attribute of feature	yes		
`boolean_attribute`	BOOLEAN	Boolean attribute of feature	yes		
`raster_or_photo`	BLOB	Photograph of the area	yes		

See sample_feature_table Table Definition SQL (Informative).

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_geometry_columns` table defined in clause Geometry Columns above.

The geometry type of a feature geometry column specified in the `gpkg_geometry_columns` table `geometry_type_name` column is a name from Geometry Types (Normative).

> **Requirement 32**
>
> Feature table geometry columns SHALL contain geometries of the type or assignable for the type specified for the column by the `gpkg_geometry_columns` table `geometry_type_name` uppercase column value [13].

Geometry subtypes are assignable as defined in Geometry Types (Normative) and shown in part in Core Geometry Model. For example, if the `geometry_type_name` value in the `gpkg_geometry_columns` 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_geometry_columns` 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.

The presence or absence of optional elevation (Z) and/or measure (M) values in a geometry does not change its type or assignability. The unit of measure for optional elevation(Z) values is determined by the CRS of the geometry; it is as-defined by a 3D CRS, and undefined for a 2D CRS. The unit of measure for optional measure (M) values is determined by the CRS of the geometry.

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.

> **Requirement 33**
>
> 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

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 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 convention described below support direct use of tiles in a GeoPackage in two ways. First, they specify how existing application 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 level 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. [14] “Tile pyramid” refers to the concept of pyramid structure of tiles of different spatial extent and resolution at different zoom levels, and the tile data itself. “Tile matrix” refers to rows and columns of tiles that all have the same spatial extent and resolution at a particular zoom level. “Tile matrix set” refers to the definition of a tile pyramid’s tiling structure.

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

2.2.2. Contents

2.2.2.1. Data

2.2.2.1.1. Contents Table – Tiles Row

> **Requirement 34**
>
> The `gpkg_contents` table SHALL contain a row with a `data_type` column value of “tiles” for each tile pyramid user data table or view.

2.2.3. Zoom Levels

In a GeoPackage, zoom levels are integers in sequence from 0 to n that identify tile matrix layers in a tile matrix set that contain tiles of decreasing spatial extent and finer spatial resolution. Adjacent zoom levels immediately preceed or follow each other and differ by a value of 1. Pixel sizes are real numbers in the terrain units of the spatial reference system of a tile image specifying the dimensions of the real world area represented by one pixel. Pixel size MAY vary by a constant factor or by different factors or intervals between some or all adjacent zoom levels in a tile matrix set. In the commonly used "zoom times two" convention, pixel sizes vary by a factor of 2 between all adjacent zoom levels, as shown in the example in Tiles Zoom Times Two Example (Informative). Other "zoom other intervals" conventions use different factors or irregular intervals with pixel sizes chosen for intuitive cartographic representation of raster data, or to coincide with the original pixel size of commonly used global image products. See Web Map Tile Service (WMTS) [16] Annex E for additional examples of both conventions.

2.2.3.1. Data

2.2.3.1.1. Zoom Times Two

> **Requirement 35**
>
> In a GeoPackage that contains a tile pyramid user data table that contains tile data, by default [15], zoom level pixel sizes for that table SHALL vary by a factor 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

> **Requirement 36**
>
> In a GeoPackage that contains a tile pyramid user data table that contains tile data that is not MIME type (http://www.ietf.org/rfc/rfc2046.txt) image/jpeg (http://www.jpeg.org/public/jfif.pdf) [17][18][19], by default SHALL store that tile data in MIME type (http://www.iana.org/assignments/media-types/index.html) image/png (http://libpng.org/pub/png/) [20][21]. [16]

2.2.5. Tile Encoding JPEG

2.2.5.1. Data

2.2.5.1.1. MIME Type JPEG

> **Requirement 37**
>
> In a GeoPackage that contains a tile pyramid user data table that contains tile data that is not MIME type (http://www.iana.org/assignments/media-types/index.html) image/png (http://libpng.org/pub/png/) [20][21], by default SHALL store that tile data in MIME type (http://www.ietf.org/rfc/rfc2046.txt) image/jpeg (http://www.jpeg.org/public/jfif.pdf) [17][18][19]. [17]

2.2.6. Tile Matrix Set

2.2.6.1. Data

2.2.6.1.1. Table Definition

> **Requirement 38**
>
> A GeoPackage that contains a tile pyramid user data table SHALL contain `gpkg_tile_matrix_set` table or view per Table Definition, Tile Matrix Set Table or View Definition and gpkg_tile_matrix_set Table Creation SQL.

Table 9. Tile Matrix Set Table or View Definition

Column Name	Column Type	Column Description	Null	Default	Key
`table_name`	TEXT	Tile Pyramid User Data Table Name	no		PK, FK
`srs_id`	INTEGER	Spatial Reference System ID: gpkg_spatial_ref_sys.srs_id	no		FK
`min_x`	DOUBLE	Bounding box minimum easting or longitude for all content in table_name	no		
`min_y`	DOUBLE	Bounding box minimum northing or latitude for all content in table_name	no		
`max_x`	DOUBLE	Bounding box maximum easting or longitude for all content in table_name	no		
`max_y`	DOUBLE	Bounding box maximum northing or latitude for all content in table_name	no		

The gpkg_tile_matrix_set table or updateable view defines the spatial reference system (srs_id) and the maximum bounding box (min_x, min_y, max_x, max_y) for all possible tiles in a tile pyramid user data table. All tiles present in the tile pyramid SHALL fall within this bounding box. However, the bounding box MAY be larger than the minimum bounding rectangle around the actual tiles in that pyramid.

See gpkg_tile_matrix_set Table Creation SQL.

2.2.6.1.2. Table Data Values

The minimum bounding box defined in the gpkg_tile_matrix_set table or view for a tile pyramid user data table SHALL be exact so that the bounding box coordinates for individual tiles in a tile pyramid MAY be calculated based on the column values for the user data table in the gpkg_tile_matrix table or view. For example, because GeoPackages use the upper left tile origin convention defined in clause Table Data Values below, the gpkg_tile_matrix_set (min_x, max_y) ordinate is the upper-left corner of tile (0,0) for all zoom levels in a table_name tile pyramid user data table.

> **Requirement 39**
>
> Values of the `gpkg_tile_matrix_set` `table_name` column SHALL reference values in the gpkg_contents table_name column for rows with a data type of "tiles".

> **Requirement 40**
>
> The gpkg_tile_matrix_set table or view SHALL contain one row record for each tile pyramid user data table.

> **Requirement 41**
>
> Values of the `gpkg_tile_matrix_set` `srs_id` column SHALL reference values in the `gpkg_spatial_ref_sys` `srs_id` column.

2.2.7. Tile Matrix

2.2.7.1. Data

2.2.7.1.1. Table Definition

> **Requirement 42**
>
> A GeoPackage that contains a tile pyramid user data table SHALL contain a `gpkg_tile_matrix` table or view per clause 2.2.7.1.1 Table Definition, Table Tile Matrix Metadata Table or View Definition and Table gpkg_tile_matrix Table Creation SQL.

Table 10. Tile Matrix Metadata Table or View Definition

Column Name	Column Type	Column Description	Null	Key
`table_name`	TEXT	Tile Pyramid User Data Table Name	no	PK, FK
`zoom_level`	INTEGER	0 <= `zoom_level` <= max_level for `table_name`	no	PK
`matrix_width`	INTEGER	Number of columns (>= 1) in tile matrix at this zoom level	no	
`matrix_height`	INTEGER	Number of rows (>= 1) in tile matrix at this zoom level	no	
`tile_width`	INTEGER	Tile width in pixels (>= 1)for this zoom level	no	
`tile_height`	INTEGER	Tile height in pixels (>= 1) for this zoom level	no	
`pixel_x_size`	DOUBLE	In `t_table_name` srid units or default meters for srid 0 (>0)	no	
`pixel_y_size`	DOUBLE	In `t_table_name` srid units or default meters for srid 0 (>0)	no	

The `gpkg_tile_matrix` table or updateable view 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 factors of 2, irregular intervals, or regular intervals other than factors of 2.

See gpkg_tile_matrix Table Creation SQL

2.2.7.1.2. Table Data Values

Requirement 43

Values of the `gpkg_tile_matrix` `table_name` column SHALL reference values in the `gpkg_contents` `table_name` column for rows with a `data_type` of “tiles”.

Requirement 44

The `gpkg_tile_matrix` table or view SHALL contain one row record for each zoom level that contains one or more tiles in each tile pyramid user data table or view.

Requirement 45

The width of a tile matrix (the difference between `min_x` and `max_x` in `gpkg_tile_matrix_set`) SHALL equal the product of `matrix_width`, `tile_width`, and `pixel_x_size` for that zoom level. Similarly, height of a tile matrix (the difference between `min_y` and `max_y` in `gpkg_tile_matrix_set`) SHALL equal the product of `matrix_height`, `tile_height`, and `pixel_y_size` for that zoom level.

The `gpkg_tile_matrix` table or view MAY contain row records for zoom levels in a tile pyramid user data 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 [18], as specified by WMTS (http://portal.opengeospatial.org/files/?artifact_id=35326) [16]. 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.

Requirement 46

The `zoom_level` column value in a `gpkg_tile_matrix` table row SHALL not be negative.

Requirement 47

The `matrix_width` column value in a `gpkg_tile_matrix` table row SHALL be greater than 0.

Requirement 48

The `matrix_height` column value in a `gpkg_tile_matrix` table row SHALL be greater than 0.

Requirement 49

The `tile_width` column value in a `gpkg_tile_matrix` table row SHALL be greater than 0.

Requirement 50

The `tile_height` column value in a `gpkg_tile_matrix` table row SHALL be greater than 0.

Requirement 51

The `pixel_x_size` column value in a `gpkg_tile_matrix` table row SHALL be greater than 0.

Requirement 52

The `pixel_y_size` column value in a `gpkg_tile_matrix` table row SHALL be greater than 0.

Requirement 53

When `zoom_level` column values in the `gpkg_tile_matrix` table are sorted in ascending order, the `pixel_x_size` and `pixel_y_size` column values in the `gpkg_tile_matrix` table SHALL appear sorted in descending order.

Tiles MAY or MAY NOT be provided for level 0 or any other particular zoom level. [19] 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. [20] This does not affect the informative spatial extent stated by the min/max x/y columns values in the `gpkg_contents` record for the same `table_name`, the exact spatial extent stated by the

min/max x/y columns values in the `gpkg_tile_matrix_set` record for the same table name, or the tile matrix width and height at that level. [21]

2.2.8. Tile Pyramid User Data Tables

2.2.8.1. Data

2.2.8.1.1. Table Definition

> **Requirement 54**
>
> Each tile matrix set in a GeoPackage SHALL be stored in a different tile pyramid user data table or updateable view with a unique name that SHALL have a column named "id" with column type INTEGER and *PRIMARY KEY AUTOINCREMENT* column constraints per Clause 2.2.8.1.1 Table Definition, Tiles Table or View Definition and EXAMPLE: tiles table Create Table SQL (Informative).

Table 11. Tiles Table or View Definition

Column Name	Column Type	Column Description	Null	Default	Key
`id`	INTEGER	Autoincrement primary key	no		PK
`zoom_level`	INTEGER	min(zoom_level) <= `zoom_level` <= max(zoom_level) for `t_table_name`	no	0	UK
`tile_column`	INTEGER	0 to `tile_matrix` `matrix_width` – 1	no	0	UK
`tile_row`	INTEGER	0 to `tile_matrix` `matrix_height` - 1	no	0	UK
`tile_data`	BLOB	Of an image MIME type specified in clauses Tile Encoding PNG, Tile Encoding JPEG, [tile_enc_webp]	no		

See EXAMPLE: tiles table Create Table SQL (Informative).

2.2.8.1.2. Table Data Values

Each tile pyramid user data table or view [22] MAY contain tile matrices at zero or more zoom levels of different spatial resolution (map scale).

> **Requirement 55**
>
> For each distinct `table_name` from the `gpkg_tile_matrix` (tm) table, the tile pyramid (tp) user data table `zoom_level` column value in a GeoPackage SHALL be in the range min(tm.zoom_level) <= tp.zoom_level <= max(tm.zoom_level).

> **Requirement 56**
>
> For each distinct `table_name` from the `gpkg_tile_matrix` (tm) table, the tile pyramid (tp) user data table `tile_column` column value in a GeoPackage SHALL be in the range 0 <= tp.tile_column <= tm.matrix_width – 1 where the tm and tp `zoom_level` column values are equal.

> **Requirement 57**
>
> For each distinct `table_name` from the `gpkg_tile_matrix` (tm) table, the tile pyramid (tp) user data table `tile_row` column value in a GeoPackage SHALL be in the range 0 <= tp.tile_row <= tm.matrix_height – 1 where the tm and tp `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` row record for that tiles table and zoom level. [23]

2.3. Extension Mechanism

2.3.1. Introduction

A GeoPackage extension is a set of one or more requirements clauses that are documented by filling out the GeoPackage Extension Template in GeoPackage Extension Template (Informative). A GeoPackage Extension either profiles / extends existing requirements clauses in the GeoPackage standard or adds new requirements clauses. Existing requirement clause extension examples include

additional geometry types, additional SQL geometry functions, and additional tile image formats. New requirement clause extension examples include spatial indexes, triggers, additional tables, other BLOB column encodings, and other SQL functions.

GeoPackage extensions are identified by a name of the form <author>_<extension name> where <author> indicates the person or organization that developed and maintains the extension. The author value “gpkg” is reserved for GeoPackage extensions that are developed and maintained by OGC and used in GeoPackages. Implementers use their own author names to register other extensions[24] used in Extended GeoPackages.

2.3.2. Extensions

2.3.2.1. Data

2.3.2.1.1. Table Definition

> **Requirement 58**
>
> A GeoPackage MAY contain a table or updateable view named gpkg_extensions. If present this table SHALL be defined per clause 2.3.2.1.1 Table Definition, GeoPackage Extensions Table or View Definition (Table or View Name: gpkg_extensions) and gpkg_extensions Table Definition SQL. An extension SHALL NOT modify the definition or semantics of existing columns. An extension MAY define additional tables or columns. An extension MAY allow new values or encodings for existing columns.

The `gpkg_extensions` table or updateable view in a GeoPackage is used to indicate that a particular extension applies to a GeoPackage, a table in a GeoPackage or a column of a table in a GeoPackage. An application that accesses a GeoPackage can query the `gpkg_extensions` table instead of the contents of all the user data tables to determine if it has the required capabilities to read or write to tables with extensions, and to “fail fast” and return an error message if it does not.

Table 12. GeoPackage Extensions Table or View Definition (Table or View Name: gpkg_extensions)

Column Name	Col Type	Column Description	Null	Key
`table_name`	TEXT	Name of the table that requires the extension. When NULL, the extension is required for the entire GeoPackage. SHALL NOT be NULL when the column_name is not NULL.	yes	Unique
`column_name`	TEXT	Name of the column that requires the extension. When NULL, the extension is required for the entire table.	yes	Unique
`extension_name`	TEXT	The case sensitive name of the extension that is required, in the form <author>_<extension_name>.	no	Unique
`definition`	TEXT	Permalink, URI, or reference to a document that defines the extension	no	
`scope`	TEXT	Indicates scope of extension effects on readers / writers: *read-write* or *write-only* in lowercase.	no	

See gpkg_extensions Table Definition SQL.

2.3.2.1.2. Table Data Values

> **Requirement 59**
>
> Every extension of a GeoPackage SHALL be registered in a corresponding row in the gpkg_extensions table. The absence of a gpkg_extensions table or the absence of rows in the gpkg_extensions table SHALL both indicate the absence of extensions to a GeoPackage.

> **Requirement 60**
>
> Values of the `gpkg_extensions` `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.

> **Requirement 61**
>
> The `column_name` column value in a `gpkg_extensions` row SHALL be the name of a column in the table specified by the `table_name` column value for that row, or be NULL.

Requirement 62

Each `extension_name` column value in a `gpkg_extensions` row SHALL be a unique case sensitive value of the form <author>_<extension_name> where <author> indicates the person or organization that developed and maintains the extension. The valid character set for <author> SHALL be [a-zA-Z0-9]. The valid character set for <extension_name> SHALL be [a-zA-Z0-9_]. An `extension_name` for the “gpkg” author name SHALL be one of those defined in this encoding standard or in an OGC document (e.g. Best Practices Document or Encoding Standard) that extends it.

The author value “gpkg” is reserved for GeoPackage extensions that are developed and maintained by OGC. GeoPackage implementers use their own author names to register other extensions.

Requirement 63

The definition column value in a `gpkg_extensions` row SHALL contain a permalink, URI [23], or reference to a document defining the extension as per the GeoPackage Extension Template (Informative).

Examples of how to fill out the GeoPackage Extension Template in GeoPackage Extension Template (Informative) are provided in Annex F. This column is not unique because an extension may define multiple tables.

Requirement 64

The scope column value in a `gpkg_extensions` row SHALL be lowercase "read-write" for an extension that affects both readers and writers, or "write-only" for an extension that affects only writers.

Some extensions do not impose any additional requirements on software that accesses a GeoPackage in a read-only fashion. An example of this is an extension that defines an SQL trigger that uses a non-standard SQL function defined in a GeoPackage SQLite Extension. Triggers are only invoked when data is written to the GeoPackage, so usage of this type of extension can be safely ignored for read-only access. This is indicated by a gpkg_extensions.scope column value of “write_only”.

2.4. Attributes

2.4.1. Introduction

Non-spatial attribute data represents sets (or tuples or rows) of arbitrary relational values that may or may not be relationally linkable to rows in other attribute, feature or tile tables.

2.4.2. Contents

2.4.2.1. Data

2.4.2.1.1. Contents Table - Attributes Row

Requirement 64a

The gpkg_contents table SHALL contain a row with a data_type column value of “attributes” for each attributes data table or view.

2.4.3. Attributes User Data Tables

2.4.3.1. Data

2.4.3.1.1. Table Definition

Non-spatial attribute data is stored in user-defined Attribute tables. Attribute sets are rows in an Attribute table. The attributes are columns in a Attribute table. (A GeoPackage is not required to contain any Attribute data tables. Attribute data tables in a GeoPackage may be empty.)

Requirement 64b

A GeoPackage MAY contain tables or updatable views containing attribute sets. Every such Attribute table or view in a GeoPackage SHALL have a column with column type INTEGER and PRIMARY KEY AUTOINCREMENT column constraints per GeoPackage Attributes Example Table or View Definition and EXAMPLE: Attributes table Create Table SQL (Informative).

The integer primary key of an Attribute table allows attribute sets to be linked to row level metadata records in the gpkg_metadata table by rowid [B5] values in the gpkg_metadata_reference table as described in clause Metadata below.

Table 13. GeoPackage Attributes Example Table or View Definition

Column Name	Col Type	Column Description	Null	Key

`id`	INTEGER	Autoincrement primary key	no	PK
`text_attribute`	TEXT	Text attribute of feature	yes	
`real_attribute`	REAL	Real attribute of feature	yes	
`boolean_attribute`	BOOLEAN	Boolean attribute of feature	yes	
`raster_or_photo`	BLOB	Photo of the area	yes	

---

3. 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 standard does not place any constraints on application, platform, operating system level or network security.

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

File Format

Test Case ID	`/base/core/container/data/file_format`
Test Purpose	Verify that the Geopackage 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

Test Case ID	`/base/core/container/data/file_format/application_id`
Test Purpose	Verify that the SQLite database header application id field indicates GeoPackage version 1.0
Test Method	Pass if the application id field of the SQLite database header starts with “GP” in ASCII and ends with an ASCII value of "11" or greater.
Reference	Clause 1.1.1.1.1 Req 2:
Test Type	Basic

File Extension Name

Test Case ID	`/base/core/container/data/file_extension_name`
Test Purpose	Verify that the geopackage extension is ".gpkg"
Test Method	Pass if the geopackage file extension is ".gpkg"
Reference	Clause 1.1.1.1.2 Req 3:
Test Type	Basic

File Contents

Test Case ID	`/base/core/container/data/file_contents`
Test Purpose	Verify that the Geopackage only contains specified contents
Test Method	1. For each gpkg_* table_name a. PRAGMA table_info(table_name) b. Continue if returns an empty result set c. Fail if column definitions returned by PRAGMA table_info do not match column definitions for the table in Annex C. 2. Do test /opt/features/vector_features/data/feature_table_integer_primary_key 3. Do test /opt/features/vector/features/data/feature_table_one_geometry_column 4. Do test /opt/tiles/contents/data/tiles_row 5. SELECT extension_name FROM gpkg_contents 6. For each row from #4 a. Fail if the substring before the first "_" is not "gpkg" 7. Pass if no fails
Reference	Clause 1.1.1.1.3 Req 4:
Test Type	Basic

Test Case ID	`/base/core/container/data/table_data_types`
Test Purpose	Verify that the data types of GeoPackage columns include only the types specified by [table_column_data_types].
Test Method	1. `SELECT table_name FROM gpkg_contents WHERE data_type = 'features'` 2. Not testable if returns empty set 3. For each row table name from step 1 a. PRAGMA table_info(table_name) b. Fail if returns empty set c. For each row type column value i. Fail if value is not one of the data type names specified by [table_column_data_types] 4. Pass if no fails
Reference	[table_column_data_types] Req 5:
Test Type	Basic

Integrity Check

Test Case ID	`/base/core/container/data/file_integrity`
Test Purpose	Verify that the geopackage passes the SQLite integrity check.
Test Method	Pass if PRAGMA integrity_check returns "ok"
Reference	Clause [_file_integrity] Req 6:
Test Type	Capability

Test Case ID	`/base/core/container/data/foreign_key_integrity`
Test Purpose	Verify that the geopackage passes the SQLite foreign_key_check.
Test Method	Pass if PRAGMA foreign_key_check() with no parameter value returns an empty result set
Reference	Clause [file integrity] Req 7:
Test Type	Capability

A.1.1.1.2. API

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. sqlite3_exec('SELECT * FROM sqlite_master;) 2. Fail if returns an SQL error. 3. Pass otherwise
Reference	Clause 1.1.1.2.1 Req 8:
Test Type	Capability

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. For each "SQLITE_OMIT_*" <option> listed at http://www.sqlite.org/compile.html#omitfeatures (http://www.sqlite.org/compile.html#omitfeatures) a. `SELECT sqlite_compileoption_used(SQLITE_OMIT_<option>)` b. Fail if returns 1 2. PRAGMA foreign_keys 3. Fail if returns 0 4. Pass otherwise
Reference	Clause 1.1.1.2.2 Req 9:
Test Type	Basic

A.1.1.2. Spatial Reference Systems

A.1.1.2.1. Data

Table Definition

Test Case ID	`/base/core/gpkg_spatial_ref_sys/data/table_def`
Test Purpose	Verify that the gpkg_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, and primary key constraints match all of those in the contents of C.1 Table 15. 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 10:

Test Type Basic

Table Data Values

Test Case ID `/base/core/gpkg_spatial_ref_sys/data_values_default`

Test Purpose Verify that the spatial_ref_sys table contains the required default contents.

Test Method

1. `SELECT srs_id, organization, organization_coordsys_id, description FROM gpkg_spatial_ref_sys WHERE srs_`
2. `SELECT srs_id, organization, organization_coordsys_id, description FROM gpkg_spatial_ref_sys WHERE srs_`
3. `SELECT definition FROM gpkg_spatial_ref_sys WHERE organization IN (“epsg”,”EPSG”) AND organization_coor` ["World Geodetic System 1984", SPHEROID["WGS 84", 6378137, 298.257223563 , AUTHORITY["EPSG","7030"]], AUTHORITY["E AUTHORITY["EPSG","8901"]], UNIT["degree", 0.017453292519943278, AUTHORITY["EPSG","9102"]], AUTHORITY["EPSG","432 places, and ignoring any optional EBNF components <twin axes> and <to wgs84> and whitespace differences in the returned text
4. Pass if tests 1-3 are met
5. Fail otherwise

Reference Clause 1.1.2.1.2 Requirement 11:

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 Clause 1.1.2.1.2 Req 12:

Test Type Capability

A.1.1.3. Contents

A.1.1.3.1. Data

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 [gpkg_contents_sql]. 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 13:
Test Type	Basic

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 14:
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 [29]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 15:
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 reference gpkg_spatial_ref_sys srs_id column values.
Test Method	1. PRAGMA foreign_key_check(‘gpkg_contents’) 2. Fail if does not return an empty result set

Reference	Clause 1.1.3.1.2 Req 16:
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 17:
Test Type	Capability

A.2.1. Features

A.2.1.1. Simple Features SQL Introduction

A.2.1.2. Contents

A.2.1.2.1. 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 a feature table for every row with a data_type column value of “features”
Test Method	1. Execute test /opt/features/vector_features/data/feature_table_integer_primary_key
Reference	Clause 2.1.2.1.1 Req 18:
Test Type	Capability

A.2.1.3. Geometry Encoding

A.2.1.3.1. Data

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 StandardGeoPackageBinary 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 two bytes of each gc are not “GP”
      ii. Fail if gc.version_number is not 0
      iii. Fail if gc.flags.GeopackageBinary type != 0
      iv. 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 19:

Test Type Capability

A.2.1.4. SQL Geometry Types

A.2.1.4.1. Data

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 1of cn from #1 are a geometry type value from Annex G Table 42, 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 cnenvelope.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 contanins pass and no fails

Reference Clause 2.1.4.1.1 Req 20:

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 GP headers containing big and little endian WKBs for 0-1 envelope contents indicator codes for every geometry type value from Annex G without and with z and/or m values. 4. Fail otherwise
Reference	Clause 2.1.4.1.1 Req 20:
Test Type	Capability

A.2.1.5. Geometry Columns

A.2.1.5.1. Data

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.4Table 20. 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 21:
Test Type	Basic

Table Data Values

Test Case ID	`/opt/features/geometry_columns/data/data_values_geometry_columns`
Test Purpose	Verify that gpkg_geometry_columns contains one row record for each geometry column in each vector feature user data table.
Test Method	1. SELECT table_name FROM gpkg_contents WHERE data_type = 'features' 2. Not testable if returns an empty result set 3. SELECT table_name FROM gpkg_contents WHERE data_type = 'features' AND table_name NOT IN (SELECT table_name FROM gpkg_geometry_columns) 4. Fail if result set is not empty
Reference	Clause 2.1.5.1.2 Req 22:
Test Type	Capability

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. PRAGMA foreign_key_check('geometry_columns') 2. Fail if returns any rows with a fourth column foreign key index value of 1 (gpkg_contents)
Reference	Clause 2.1.5.1.2 Req 23:
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. 4. Pass if no fails.
Reference	Clause 2.1.5.1.2 Req 24:
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_name value is not in Table 42 or Table 43 in Annex G 4. Pass if no fails.
Reference	Clause 2.1.5.1.2 Req 25:
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 valid.
Test Method	1. PRAGMA foreign_key_check('gpkg_geometry_columns') 2. Fail if returns any rows with a fourth column foreign key index value of 0
Reference	Clause 2.1.5.1.2 Req 26:
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 (0,1,2) 4. Fail if does not return an empty result set 5. Pass otherwise.
Reference	Clause 2.1.5.1.2 Req 27:
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 (0,1,2) 4. Fail if does not return an empty result set 5. Pass otherwise.
Reference	Clause 2.1.5.1.2 Req 28:
Test Type	Capability

A.2.1.6. Vector Features User Data Tables

A.2.1.6.1. Data

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 WERE 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 29:
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 WERE 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 30:
Test Type	Capability

Test Case ID	`/opt/features/vector_features/data/feature_table_geometry_column_type`
Test Purpose	Verify that the declared SQL type of a feature table geometry column is the uppercase geometry type name from Annex G specified by the geometry_type_name column for that column_name and table_name in the gpkg_geometry_columns table.
Test Method	1. SELECT table_name, column_name, geometry_type_name table_name FROM gpkg_geometry_columns WHERE table_name IN (SELECT table_name FROM gpkg_contents WHERE data_type = *features*) 2. For each row selected in (1): a. Fail if selected geometry_type_name value is not a value from the NAME column in Annex G Table 42 or Table 43. b. SELECT sql FROM sqlite_master WHERE type = *table* AND name = ‘{selected table_ name}’ c. Pass if declared type of column_name selected in (1) is the geometry_type_name selected in (1) d. Fail otherwise
Reference	Clause 2.1.6.1.1 Req 31:
Test Type	Capability

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 32:
Test Type	Capability

Test Case ID	`/opt/features/vector_features/data/data_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 returnvalue not equal to gc_srs_id 4. Pass if no fails
Reference	Clause 2.1.6.1.2 Req 33:
Test Type	Capability

A.2.2. Tiles

A.2.2.1. Contents

A.2.2.1.1. Data

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. Not testable 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 34:
Test Type	Capability

A.2.2.2. Zoom Levels

A.2.2.2.1. Data

Zoom Times Two

Test Case ID	`/opt/tiles/zoom_levels/data/zoom_times_two`
Test Purpose	Verify that zoom level pixel sizes for tile matrix user data tables vary by factors of 2 between adjacent zoom levels in the tile matrix metadata table.

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 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 factors of two 4. Pass if no fails
Reference	Clause 2.2.3.1.1 Req 35:
Test Type	Capability

A.2.2.3. Tile Encoding PNG

A.2.2.3.1. Data

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 table_name AS tn FROM gpkg_contents WHERE data_type = 'tiles' 2. For each row tbl_name from step 1 a. WHEN (SELECT tbl_name FROM sqlite_master WHERE tbl_name = 'gpkg_extensions') = 'gpkg_extensions' THEN (SELECT extension_name FROM gpkg_extensions WHERE table_name = 'tn' AND column_name = 'tile_data') END; i. Not testable unless it returns empty result set b. SELECT tile_data FROM tn c. For each row tile_data from step a i. Pass if tile data in MIME type image/jpeg ii. Pass if tile data in MIME type image/png iii. Fail if no passes
Reference	Clause 2.2.4.1.1 Req 36:
Test Type	Capability

A.2.2.4. Tile Encoding JPEG

A.2.2.4.1. Data

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 table_name AS tn FROM gpkg_contents WHERE data_type = 'tiles' 2. For each row tbl_name from step 1 a. WHEN (SELECT tbl_name FROM sqlite_master WHERE tbl_name = 'gpkg_extensions') = 'gpkg_extensions' THEN (SELECT extension_name FROM gpkg_extensions WHERE table_name = 'tn' AND column_name = 'tile_data') END; i. Not testable unless it returns empty result set b. SELECT tile_data FROM tn c. For each row tile_data from step a i. Pass if tile data in MIME type image/jpeg ii. Pass if tile data in MIME type image/png iii. Fail if no passes
Reference	Clause 2.2.5.1.1 Req 37:
Test Type	Capability

A.2.2.5. Tile Matrix Set

A.2.2.5.1. Data

Table Definition

Test Case ID	`/opt/tiles/gpkg_tile_matrix_set/data/table_def`
Test Purpose	Verify that the gpkg_tile_matrix_set table exists and has the correct definition.
Test Method	1. SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_tile_matrix_set' 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 [example_feature_table_sql]. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant. 4. Fail otherwise.
Reference	Clause 2.2.6.1.1 Req 38:
Test Type	Capability

Table Data Values

Test Case ID	`/opt/tiles/gpkg_tile_matrix_set/data/data_values_table_name`
Test Purpose	Verify that values of the gpkg_tile_matrix_set 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_set 2. Not testable if returns an empty result set 3. SELECT table_name FROM gpkg_tile_matrix_set tms WHERE table_name NOT IN (SELECT table_name FROM gpkg_contents gc WHERE tms.table_name = gc.table_name AND gc.data_type != ‘tiles’) 4. Fail if result set contains any rows 5. Pass otherwise
Reference	Clause 2.2.6.1.2 Req 39:

Test Type	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix_set/data/data_values_row_record`
Test Purpose	Verify that the gpkg_tile_matrix_set table contains a row record for each tile pyramid 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 sql FROM sqlite_master WHERE type=’table’ AND tbl_name = <*user_data_tiles_table*> b. Fail if returns an empty result set 4. Pass if no fails
Reference	Clause 2.2.6.1.2 Req 40:
Test Type	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix_set/data/data_values_srs_id`
Test Purpose	Verify that the gpkg_tile_matrix_set table srs_id column values reference gpkg_spatial_ref_sys srs_id column values.
Test Method	1. PRAGMA foreign_key_check(‘gpkg_geometry_columns’) 2. Fail if returns any rows with a fourth column foreign key index value of 1 (gpkg_spatial_ref_sys)
Reference	Clause 2.2.6.1.2 Req 41:
Test Type	Capability

A.2.2.6. Tile Matrix

A.2.2.6.1. Data

Table Definition

Test Case ID	`/opt/tiles/gpkg_tile_matrix/data/table_def`
Test Purpose	Verify that the gpkg_tile_matrix table exists and has the correct definition.
Test Method	1. SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_tile_matrix' 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, primary, and foreign key constraints match all of those in the contents of Annex C Table 23. 4. Fail otherwise.
Reference	Clause 2.2.7.1.1 Req 42:
Test Type	Basic

Table Data Values

Test Case ID	`/opt/tiles/gpkg_tile_matrix/data/data_values_table_name`
Test Purpose	Verify that values of the gpkg_tile_matrix 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
2. Not testable if returns an empty result set
3. SELECT table_name FROM gpkg_tile_matrix tmm WHERE table_name NOT IN (SELECT table_name FROM gpkg_contents gc WHERE tmm.table_name = gc.table_name AND gc.data_type != 'tiles')
4. Fail if result set contains any rows
5. Pass otherwise

Reference Clause 2.2.7.1.2 Req 43:

Test Type Capability

Test Case ID `/opt/tiles/gpkg_tile_matrix/data/data_values_zoom_level_rows`

Test Purpose Verify that the gpkg_tile_matrix table contains a row record for each zoom level that contains one or more tiles in each tile pyramid 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 udtt_zoom FROM gpkg_tile_matrix AS gtmm LEFT OUTER JOIN <user_data_tiles_table> AS udtt ON udtt.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 2.2.7.1.2 Req 44:

Test Type Capability

Test Case ID `/opt/tiles/gpkg_tile_matrix/data/data_values_width_height`

Test Purpose Verify that the tile matrix extents in gpkg_tile_matrix_set match the contents of the gpkg_tile_matrix 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 max_x - min_x from gpkg_tile_matrix_set where table_name = *<user_data_tiles_table>*
   b. SELECT zoom_level, matrix_width * tile_width * pixel_x_size from gpkg_tile_matrix where table_name = *<user_data_tiles_table>*
   c. SELECT max_y - min_y from gpkg_tile_matrix_set where table_name = *<user_data_tiles_table>*
   d. SELECT zoom_level, matrix_height * tile_height * pixel_y_size from gpkg_tile_matrix where table_name = *<user_data_tiles_table>*
   e. Fail if, for any zoom level, the difference for an axis does not equal the product for that axis at that zoom level
4. Pass if no fails

Reference Clause 2.2.7.1.2 Req 45:

Test Type Capability

Test Case ID `/opt/tiles/gpkg_tile_matrix/data/data_values_zoom_level`

Test Purpose	Verify that zoom level column values in the gpkg_tile_matrix table are not negative.
Test Method	1. SELECT zoom_level FROM gpkg_tile_matrix 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 2.2.7.1.2 Req 46:
Test Type	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix/data/data_values_matrix_width`
Test Purpose	Verify that the matrix_width values in the gpkg_tile_matrix table are valid.
Test Method	1. SELECT matrix_width FROM gpkg_tile_matrix 2. Not testable if returns an empty result set 3. SELECT min(matrix_width) FROM gpkg_tile_matrix. 4. Fail if less than 1. 5. Pass otherwise.
Reference:	Clause 2.2.7.1.2 Req 47:
Test Type:	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix/data/data_values_matrix_height`
Test Purpose	Verify that the matrix_height values in the gpkg_tile_matrix table are valid.
Test Method	1. SELECT matrix_height FROM gpkg_tile_matrix 2. Not testable if returns an empty result set 3. SELECT min(matrix_height) FROM gpkg_tile_matrix. 4. Fail if less than 1. 5. Pass otherwise.
Reference	Clause 2.2.7.1.2 Req 48:
Test Type	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix/data/data_values_tile_width`
Test Purpose	Verify that the tile_width values in the gpkg_tile_matrix table are valid.
Test Method	1. SELECT tile_width FROM gpkg_tile_matrix 2. Not testable if returns an empty result set 3. SELECT min(tile_width) FROM gpkg_tile_matrix. 4. Fail if less than 1. 5. Pass otherwise.
Reference	Clause 2.2.7.1.2 Req 49:

Test Type	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix/data/data_values_tile_height`
Test Purpose	Verify that the tile_height values in the gpkg_tile_matrix table are valid.
Test Method	1. SELECT tile_height FROM gpkg_tile_matrix 2. Not testable if returns an empty result set 3. SELECT min(tile_height) FROM gpkg_tile_matrix. 4. Fail if less than 1. 5. Pass otherwise.
Reference	Clause 2.2.7.1.2 Req 50:
Test Type	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix/data/data_values_pixel_x_size`
Test Purpose	Verify that the pixel_x_size values in the gpkg_tile_matrix table are valid.
Test Method	1. SELECT pixel_x_size FROM gpkg_tile_matrix 2. Not testable if returns an empty result set 3. SELECT min(pixel_x_size) FROM gpkg_tile_matrix. 4. Fail if less than 0. 5. Pass otherwise.
Reference	Clause 2.2.7.1.2 Req 51:
Test Type	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix/data/data_values_pixel_y_size`
Test Purpose	Verify that the pixel_y_size values in the gpkg_tile_matrix table are valid.
Test Method	1. SELECT pixel_y_size FROM gpkg_tile_matrix 2. Not testable if returns an empty result set 3. SELECT min(pixel_y_size) FROM gpkg_tile_matrix. 4. Fail if less than 0. 5. Pass otherwise.
Reference	Clause 2.2.7.1.2 Req 52:
Test Type	Capability

Test Case ID	`/opt/tiles/gpkg_tile_matrix/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 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 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 2.2.7.1.2 Req 53:
Test Type	Capability

A.2.2.7. Tile Pyramid User Data

A.2.2.7.1. Data

Table Definition

Test Case ID	`/opt/tiles/tile_pyramid/data/table_def`
Test Purpose	Verify that multiple tile pyramids are stored in different tiles tables with unique names containing the required columns.
Test Method	1. SELECT COUNT(table_name) FROM gpkg_contents WERE data_type = “tiles” 2. Not testable if less than 1 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 2.2.8.1.1 Req 54:
Test Type	Basic

Table Data Values

Test Case ID	`/opt/tiles/tile_pyramid/data/data_values_zoom_levels`
Test Purpose	Verify that the zoom level column values in each tile pyramid user data table are within the range of zoom levels defined by rows in the gpkg_tile_matrix table.

Test Method

1. SELECT DISTINCT table_name AS <user_data_tiles_table> FROM gpkg_tile_matrix
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 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 pas and no fails

Reference Clause 2.2.8.1.2 Req 55:

Test Type Capability

Test Case ID `/opt/tiles/tile_pyramid/data/data_values_tile_column`

Test Purpose Verify that the tile_column column values for each zoom level value in each tile pyramid user data table are within the range of columns defined by rows in the gpkg_tile_matrix table.

Test Method

1. SELECT DISTINCT table_name AS <user_data_tiles_table> FROM gpkg_tile_matrix
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 gpkg_tile_matrix 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 2.2.8.1.2 Req 56:

Test Type Capability

Test Case ID `/opt/tiles/tile_pyramid_data/data_values_tile_row`

Test Purpose Verify that the tile_row column values for each zoom level value in each tile pyramid user data table are within the range of rows defined by rows in the gpkg_tile_matrix table.

Test Method	1. SELECT DISTINCT table_name AS <user_data_tiles_table> FROM gpkg_tile_matrix 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 gpkg_tile_matrix 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_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 2.2.8.1.2 Req 57:
Test Type	Capability

A.2.3. Extension Mechanism

A.2.3.1. Extensions

A.2.3.1.1. Data

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 36. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant. 4. Fail otherwise.
Reference	Clause 2.3.2.1.1 Req 58:
Test Type	Basic

Table Data Values

Test Case ID	`/opt/extension_metchanism/extensions/data/data_values_for_extensions`
Test Purpose	Verify that every extension of a GeoPackage is registered in a row in the gpkg_extensions table

Test Method

1. For each SELECT DISTINCT geometry_type_name FROM geometry_columns
   a. Fail if geometry_type_name IN Annex G Table 43 and gpkg_extensions does not contain a row where extension_name = gpkg_geom_<geometry_type_name>
   b. Fail if geometry_type_name NOT IN Annex G Table 42 or Table 43 and gpkg_extensions does not contain a row where the extension_name does not begin with “gpkg” and the extension_name ends with “*geom*<geometry_type_name>
2. For each SELECT tbl_name FROM sqlite_master WHERE tbl_name LIKE ‘rtree_%’
   a. Fail if gpkg_extensions does not contain a row where extension_name = “gpkg_rtree_index”
3. For each SELECT tbl_name FROM sqlite_master WHERE name LIKE ‘fgti_%’
   a. Fail if gpkg_extensions does not contain a row where extension_name = “gpkg_geometry_type_trigger”
4. For each SELECT tbl_name FROM sqlite_master WHERE name LIKE ‘fgsi_%’
   a. Fail if gpkg_extensions does not contain a row where extension_name = “gpkg_srs_id_trigger”
5. Do test / reg_ext/tiles/zoom_levels/data/zoom_other_ext_row
6. Do test /reg_ext/tiles/tile_encoding_webp/data/webp_ext_row
7. Do test /reg_ext/tiles/tile_encoding_webp/data/tiff_ext_row
8. Do test /reg_ext/tiles/tile_encoding_webp/data/nitf_ext_row
9. Pass if no fails

Reference Clause 2.3.2.1.2 Req 59:

Test Type Capability

Test Case ID `/opt/extension_metchanism/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 2.3.2.1.2 Req 60:

Test Type Capability

Test Case ID `/opt/extension_metchanism/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 2.3.2.1.2 Req 61:

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 Annex F.
   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 2.3.2.1.2 Req 62:

Test Type Capability

Test Case ID `/opt/extension_mechanism/extensions/data/data_values_definition`

Test Purpose Verify that the definition column value contains or references extension documentation

Test Method

1. SELECT definition FROM gpkg_extensions
2. Not testable if returns an empty result set
3. For each row returned from step 1
   a. Inspect if definition value is not like "Annex %", or "http%" or mailto:% or "Extension Title%"
   b. Fail if definition value does not contain or reference extension documentation
4. Pass if no fails

Reference Clause 2.3.2.1.2 Req 63:

Test Type Capability

Test Case ID	`/opt/extension_mechanism/extensions/data/data_values_scope`
Test Purpose	Verify that the scope column value is “read-write” or “write-only”
Test Method	1. SELECT scope FROM gpkg_extensions 2. Not testable if returns an empty result set 3. For each row returned from step 1 a. Fail is value is not “read-write” or “write-only” 4. Pass if no fails
Reference	Clause 2.3.2.1.2 Req 64:
Test Type	Capability

A.2.4. Attributes

A.2.4.1. Contents

A.2.4.1.1. Data

Contents Table – Attributes Row

Test Case ID	`/opt/attributes/contents/data/attributes_row`
Test Purpose	Verify that the gpkg_contents table_name value table exists and is apparently an attributes table for every row with a data_type column value of “attributes”.
Test Method	1. SELECT table_name FROM gpkg_contents WHERE data_type = “attributes” 2. Not testable 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” 4. Pass if no fails.
Reference	Clause 2.4.2.1.1 Req 64a
Reference	Clause 2.4.3.1.1 Req 64b
Test Type	Capability

Annex B: Background and Context (Normative)

B.1. Background

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 OGC® GeoPackage Encoding Standard 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 [9][10][11] and SQL-MM (Spatial) [12] standards that serve as normative references for the vector feature portion of this standard.

This standard attempts to support and use relevant raster types, storage table definitions, and metadata from widely adopted implementations and existing standards such as WMTS [16] and ISO metadata [28], 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 standard are classified as core (mandatory) and extension (optional). The simple core of an Empty GeoPackage contains two SQL tables.

Future versions of this standard may include requirements for elevation data and routes. Future enhancements to this standard, 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. Document terms and definitions

This document uses the standard terms defined in Subclause 5.3 of [OGC 06-121], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word "shall" (not "must") is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

For the purposes of this document, the following terms and definitions apply.

Empty GeoPackage

A GeoPackage that contains a `gpkg_spatial_ref_sys` table, a `gpkg_contents` table with row record(s) with `data_type` column values of "features" or "tiles", and corresponding features tables per clause Features and/or tiles tables per clause Tiles where the user data tables per clauses 2.1.6. and 2.2.8 exist but contain no rows.

Extended GeoPackage

A GeoPackage that contains any additional data elements (tables or columns) or SQL constructs (data types, indexes, constraints or triggers) that are not specified in this encoding standard.

geolocate

identify a real-world geographic location

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.

GeoPackage SQLite Configuration

consists of the SQLite 3 software library and a set of compile- and runtime configurations options.

GeoPackage SQLite Extension

a SQLite loadable extension that MAY provide SQL 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.

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.

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.

tile

a rectangular pictorial representation of geographic data, often part of a set of such elements, covering a spatially contiguous extent and sharing similar information content and graphical styling, which can be uniquely defined by a pair of indices for the column and row along with an identifier for the tile matrix.

tile matrix

a collection of tiles for a fixed scale

tile pyramid

a collection of tile matrices defined at different scales

Valid GeoPackage

A GeoPackage that contains features per clause Features and/or tiles per clause Tiles and row(s) in the `gpkg_contents` table with `data_type` column values of "features" and/or "tiles" describing the user data tables.

B.3. Conventions

Symbols (and abbreviated terms)

ACID

Atomic, Consistent, Isolated, and Durable

ASCII

American Standard Code for Information Interchange

API

Application Program Interface

BLOB

Binary Large OBject

CLI

Call-Level Interface

COTS

Commercial Off The Shelf

DEM

Digital Elevation Model

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

NIIRS

National Imagery Interpretability Rating Scale

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

UML

Unified Modeling Language

UTC

Coordinated Universal Time

XML

eXtensible Markup Language

1D

One Dimensional

2D

Two Dimensional

3D

Three Dimensional

B.4. Submitting Organizations (Informative)

The following organizations submitted this Encoding Standard to the Open Geospatial Consortium as a Request For Comment (RFC).

- Envitia
- Luciad

- Sigma Bravo
- The Carbon Project
- U.S. Army Geospatial Center
- U.S. National Geospatial Intelligence Agency

B.5. Document contributor contact points (Informative)

All questions regarding this document should be directed to the editor or the contributors:

Table 14. Document contributors

Name	Organization	Email
Brett Antonides	LNM Solutions	brett.antonides<at>lmnsolutions.com
Kevin Backe	U.S. Army Geospatial Center GASD	Kevin.Backe<at>usace.army.mil
Roger Brackin	Envitia	roger.brackin<at>envitia.com
Chris Clark	Compusult	chrisc<at>compusult.net
Scott Clark	LNM Solutions	scott.clark<at>lmnsolutions.com
David Cray	U.S. Army Geospatial Center GASD	David.Cray<at>usace.army.mil
Paul Daisey	Image Matters	pauld<at>imagemattersllc.com
Nathan P. Frantz	U.S. Army Geospatial Center ERDC	Nathan.P.Frantz<at>usace.army.mil
Alessandro Furieri	Spatialite	a.furieri<at>lqt.it
Randy Gladish	Image Matters	randyg<at>imagemattersllc.com
Eric Gundersen	MapBox	eric<at>mapbox.com
Brad Hards	Sigma Bravo	bhards<at>sigmabravo.com
Jeff Harrison	The Carbon Project	jharrison<at>thecarbonproject.com
Chris Holmes	OpenGeo	cholmes<at>9eo.org
Frederic Houbie	Luciad	frederic.houbie<at>luciad.com
Sean Hogan	Compusult	sean<at>compusult.net
Kirk Jensen	Image Matters	kirkj<at>imagemattersllc.com
(chinese chars not working) Joshua	Feng China University	joshua<at>gis.tw
Terry A. Idol	U.S. National Geospatial Intelligence Agency	Terry.A.Idol<at>nga.mil
Drew Kurry	Digital Globe	dkurry<at>digitalglobe.com
Steven Lander	Reinventing Geospatial	steven.lander<at>rgi-corp.com
Tom MacWright	MapBox	tom<at>mapbox.com
Joan Maso Pau	Universitat Autònoma de Barcelona (CREAF)	joan.maso<at>uab.es
Kevin S. Mullane	U.S. Army Geospatial Center GASD	Kevin.S.Mullane<at>usace.army.mil
Brian Osborn	CACI	bosborn<at>caci.com
(chinese chars not working) Yi-Min Huang	Feng China University	niner<at>gis.tw

Name	Organization	Email
Andrea Peri	Regione Toscana Italy	andrea.peri<at>regione.toscana.it
Paul Ramsey	OpenGeo	pramsey<at>opengeo.org
Matthew L. Renner	U.S. Army Geospatial Center ERDC	Matthew.L.Renner<at>usace.army.mil
Even Rouault	Mines-Paris	even.rouault<at>mines-paris.org
Keith Ryden	Environmental Systems Research Institute	kryden<at>esri.com
Scott Simmons	CACI	scsimmons<at>caci.com
Ingo Simonis	International Geospatial Services Institute	ingo.simonis<at>igsi.eu
Raj Singh	Open Geospatial Consortium	rsingh<at>opengeospatial.org
Steve Smyth	Open Site Plan	steve<at>opensiteplan.org
Donald V. Sullivan	U.S. National Aeronautics and Space Administration	donald.v.sullivan<at>nasa.gov
Christopher Tucker	Mapstory	tucker<at>mapstory.org
Benjamin T. Tuttle	U.S. National Geospatial Intelligence Agency	Benjamin.T.Tuttle<at>nga.mil
Pepijn Van Eeckhoudt	Luciad	pepijn.vaneeckhoudt<at>luciad.com
David G. Wesloh	U.S. National Geospatial Intelligence Agency	David.G.Wesloh<at>nga.mil
Jeff Yutzler	Image Matters	jeffy<at>imagemattersllc.com
Eric Zimmerman	U.S. Army Geospatial Center ERDC	Eric.Zimmerman<at>usace.army.mil

B.6. Revision History (Informative)

Table 15. Revision History

Date	Rel	Editor	Paragraph modified	Description
2014-07-14	R11	Paul Daisey	1.1.2.1.1	Remove “at a minimum” after “includes” in 2nd paragraph, 1st sentence; conflicts with Clause 1
2014-07-14	R11	Paul Daisey	2.1.4.1.1	Req 20 42in insert space between 42 and in
2014-07-14	R11	Paul Daisey	2.2.8.1.1	Change Table 30 to Table 29
2014-07-14	R11	Paul Daisey	2.5.2.1.1	Change access to accesses in 1st paragraph, 2nd sentence
2014-07-14	R11	Paul Daisey	Annex A, A.3.1.1.1	inAnnex E insert space between in and Annex
2014-07-14	R11	Paul Daisey	Annex A, A.3.1.1.1	43without insert space between 43 and without
2014-07-14	R11	Paul Daisey	Annex B, B.5	Change pepijn.vaneeckhoudt email from gmail to Luciad
2014-07-14	R11	Paul Daisey	Annex B, B.5	Change all "@" to “<at>”
2014-07-14	R11	Paul Daisey	Annex C, C.10	Remove UNIQUE from PK constraint
2014-12-12	R11	Paul Daisey	2.1.1	Add a footnote to "GeometryCollection" description
2014-12-12	R11	Paul Daisey	2.1.6.1.1	Add new Req 30b

Date	Rel	Editor	Paragraph modified	Description
2014-12-12	R11	Paul Daisey	Annex A, A.2.1.6.1.1	Add feature_table_geometry_column_type test
2014-12-12	R11	Paul Daisey	2.1.5.1.1	Add “and geometry types” to 1st paragraph 1st sentence
2014-12-12	R11	Paul Daisey	1.1.1.1.4	Add footnote to Req 6
2014-12-12	R11	Paul Daisey	2.1.3.1.1	replace ISO 13249-3 with OGC 06-103r4
2014-12-12	R11	Paul Daisey	2.1.3.1.1	Correct references in footnote 1: [13] becomes [9] and [16] becomes [12]
2014-12-12	R11	Paul Daisey	1.1.1.2.2	Remove “and run” from clause and Req 9
2014-12-12	R11	Paul Daisey	1.1.1.2.2	Remove PRAGMA foreign_keys runtime option from Table 2
2014-12-12	R11	Paul Daisey	Annex A, A.2.2.7.1.1	Change step 2 to “Not testable if less than 1”
2014-12-12	R11	Paul Daisey	Annex A, A.2.2.6.1.1	Remove obsolete provisions (unique, column order, other columns) from step 3
2014-12-12	R11	Paul Daisey	2.1.6.1.2	Add sentences specifying unit of measure determination for geometry Z and M values.
2014-12-12	R11	Paul Daisey	1.1.3.1.1, Table 4	Change description column default value to ‘’
2014-12-12	R11	Paul Daisey	Annex C, C.7, Table 29	Remove spurious “)” from tile_data column definition
2014-12-12	R11	Paul Daisey	Annex D, D.3, Table 39	Correct ISO 8601 timestamp GLOB expressions
2015-03-16	R11	Scott Simmons	entire document	Minor format corrections
2015-04-27	R11	Joe Brumley	entire document	Minor format corrections for corrigendum
2015-06-10	R12	Brad Hards	2.2.1	Fix typos
2015-06-17	R12	Jeff Yutzler	2.2.6.1.1	Clarify role of bounding box in Tile Matrix Set table https://github.com/opengeospatial/geopackage/issues/102 (https://github.com/opengeospatial/geopackage/issues/102)
2015-06-17	R12	Jeff Yutzler	2.2.7.1.2	Add new Req 44b https://github.com/opengeospatial/geopackage/issues/102 (https://github.com/opengeospatial/geopackage/issues/102)
2015-06-24	R12	Jeff Yutzler	1.1.2.1.2	Clarify Req 11 https://github.com/opengeospatial/geopackage/issues/110 (https://github.com/opengeospatial/geopackage/issues/110)
2015-08-04	R12	Jeff Yutzler	Figures 4, 5, 6	Correct multiplicity https://github.com/opengeospatial/geopackage/issues/117 (https://github.com/opengeospatial/geopackage/issues/117)
2015-08-04	R12	Jeff Yutzler	Annex I	Update Footnote #18 (JFIF) to T.871 https://github.com/opengeospatial/geopackage/issues/104 (https://github.com/opengeospatial/geopackage/issues/104)
2015-08-25	R12	Jeff Yutzler	Whole document	Remove revision markup https://github.com/opengeospatial/geopackage/issues/135 (https://github.com/opengeospatial/geopackage/issues/135)

Date	Rel	Editor	Paragraph modified	Description
2015-08-25	R12	Jeff Yutzler	Annex B, B.6	Remove old changes https://github.com/opengeospatial/geopackage/issues/133 (https://github.com/opengeospatial/geopackage/issues/133)
2015-08-26	R12	Jeff Yutzler	Annex B, B.6	Fix casing of minIsInclusive, maxIsInclusive, and description https://github.com/opengeospatial/geopackage/issues/130 (https://github.com/opengeospatial/geopackage/issues/130)
2015-08-25	R12	Jeff Yutzler	Annex B, B.6	Clarify rules for case sensitivity for views, triggers, constraints https://github.com/opengeospatial/geopackage/issues/131 (https://github.com/opengeospatial/geopackage/issues/131)
2015-09-09	R12	Brad Hards	Intro	Fix typos
2015-09-29	R12	Brad Hards	1.1.3.1.1	Fix typos
2015-09-29	R12	Jeff Yutzler	1.1.3.1.1	User-defined tables do not need to have lowercase column names https://github.com/opengeospatial/geopackage/issues/144 (https://github.com/opengeospatial/geopackage/issues/144)
2015-10-19	R12	Jeff Yutzler	Annex F, F.4	Fix typos https://github.com/opengeospatial/geopackage/issues/152 (https://github.com/opengeospatial/geopackage/issues/152)
2015-10-29	R12	Brad Hards	1.1.3.1.1	Fix typos
2015-11-09	R12	Jeff Yutzler	Multiple	Fix typos
2015-11-19	R12	Jeff Yutzler	Multiple	Numerous administrative edits https://github.com/opengeospatial/geopackage/issues/160 (https://github.com/opengeospatial/geopackage/issues/160)
2015-11-26	R12	Jeff Yutzler	Annex F, F.9	Define GLOB https://github.com/opengeospatial/geopackage/issues/156 (https://github.com/opengeospatial/geopackage/issues/156)
2015-11-26	R12	Jeff Yutzler	Annex F	Collapse all extensions into a single annex https://github.com/opengeospatial/geopackage/issues/132 (https://github.com/opengeospatial/geopackage/issues/132)
2015-11-26	R12	Jeff Yutzler	Annex F, F.8, F.9	Demote Metadata and Schema sections to extensions https://github.com/opengeospatial/geopackage/issues/147 (https://github.com/opengeospatial/geopackage/issues/147)
2015-11-26	R12	Jeff Yutzler	Annex E	Update extension template https://github.com/opengeospatial/geopackage/issues/165 (https://github.com/opengeospatial/geopackage/issues/165)
2015-11-26	R12	Jeff Yutzler	Annex F, F.10	Create new extension for CRS WKT https://github.com/opengeospatial/geopackage/issues/137 (https://github.com/opengeospatial/geopackage/issues/137)
2015-12-02	R12	Jeff Yutzler	Annex A, Annex F	Clean ATS references https://github.com/opengeospatial/geopackage/issues/169 (https://github.com/opengeospatial/geopackage/issues/169)
2015-12-04	R12	Jeff Yutzler	2.3.2.1.2	Update rules for listing extensions https://github.com/opengeospatial/geopackage/issues/175 (https://github.com/opengeospatial/geopackage/issues/175)

Date	Rel	Editor	Paragraph modified	Description
2015-12-04	R12	Jeff Yutzler	All Annexes	Scrub annex references https://github.com/opengeospatial/geopackage/issues/176 (https://github.com/opengeospatial/geopackage/issues/176)
2015-12-26	R12	Brad Hards	2.3.2.1	Fix typos
2015-12-26	R12	Jeff Yutzler	Annex F.9	Make "identifier" unique in Table 42, Table 45 https://github.com/opengeospatial/geopackage/issues/183 (https://github.com/opengeospatial/geopackage/issues/183)
2015-12-28	R12	Jeff Yutzler	Annex A.1.1.2	Fix column names in ATS tests
2016-01-20	R12	Jeff Yutzler	1.1.1.1.1	Bump version number https://github.com/opengeospatial/geopackage/issues/188 (https://github.com/opengeospatial/geopackage/issues/188)
2016-05-08	R13	Jeff Yutzler	1.1.1.1.1	Clarify that GeoPackage header is null terminated https://github.com/opengeospatial/geopackage/pull/209 (https://github.com/opengeospatial/geopackage/pull/209)
2016-05-17	R13	Jeff Yutzler	Annex B.11-13	Update UML diagrams to show DATETIME https://github.com/opengeospatial/geopackage/issues/214 (https://github.com/opengeospatial/geopackage/issues/214)
2016-07-12	R13	Jeff Yutzler	1.1.3	Rewrite description of gpkg_contents https://github.com/opengeospatial/geopackage/issues/221 (https://github.com/opengeospatial/geopackage/issues/221)
2016-07-18	R13	Frederic Houbie	Annex B.2	Add new terms and definitions https://github.com/opengeospatial/geopackage/issues/213 (https://github.com/opengeospatial/geopackage/issues/213)
2016-08-02	R13	Chris Clark	Annex F.9	Correct wording in Table 42 https://github.com/opengeospatial/geopackage/issues/231 (https://github.com/opengeospatial/geopackage/issues/231)
2016-08-02	R13	Chris Clark	Annex F.10	Correct column name for CRS WKT extension https://github.com/opengeospatial/geopackage/issues/258 (https://github.com/opengeospatial/geopackage/issues/258)
2016-08-23	R13	Jeff Yutzler	Annex F.1	Deprecate Requirement 69 https://github.com/opengeospatial/geopackage/issues/234 (https://github.com/opengeospatial/geopackage/issues/234)
2016-08-23	R13	Jeff Yutzler	Annex F.2,4,5	Deprecate extensions F.2, F.4, F.5 https://github.com/opengeospatial/geopackage/issues/235 (https://github.com/opengeospatial/geopackage/issues/235)
2016-08-23	R13	Chris Clark	2.2.7.1.2	Rewrite Requirement 53 https://github.com/opengeospatial/geopackage/issues/237 (https://github.com/opengeospatial/geopackage/issues/237)
2016-08-23	R13	Chris Clark	Multiple	Fix Typos https://github.com/opengeospatial/geopackage/issues/238 (https://github.com/opengeospatial/geopackage/issues/238)
2016-09-06	R13	Chris Clark	Annex F.8	Simplify triggers to avoid failures https://github.com/opengeospatial/geopackage/issues/228 (https://github.com/opengeospatial/geopackage/issues/228)

Date	Rel	Editor	Paragraph modified	Description
2016-09-13	R13	Jeff Yutzler	Annex F.8	Add default to SQL DDL https://github.com/opengeospatial/geopackage/issues/243 (https://github.com/opengeospatial/geopackage/issues/243)
2016-09-20	R13	Jeff Yutzler	2.3.2.1.1	Update description for "definition" column https://github.com/opengeospatial/geopackage/issues/241 (https://github.com/opengeospatial/geopackage/issues/241)
2016-10-04	R13	Jeff Yutzler	2.4	Adding "attributes" section to describe use of non-spatial data https://github.com/opengeospatial/geopackage/issues/221 (https://github.com/opengeospatial/geopackage/issues/221)
2016-10-17	R13	Jeff Yutzler	Multiple	Fix typos https://github.com/opengeospatial/geopackage/issues/249 (https://github.com/opengeospatial/geopackage/issues/249)
2016-10-17	R13	Jeff Yutzler	1.1.1.1.1, Annex A	Allow for version increments in SQLite header https://github.com/opengeospatial/geopackage/issues/255 (https://github.com/opengeospatial/geopackage/issues/255)
2016-10-17	R13	Jeff Yutzler	1.1.1.1.2	Eliminate .gpkx option https://github.com/opengeospatial/geopackage/issues/261 (https://github.com/opengeospatial/geopackage/issues/261)
2016-10-17	R13	Jeff Yutzler	Annex F.9	Correct min/max_is_inclusive in abstract tests https://github.com/opengeospatial/geopackage/issues/260 (https://github.com/opengeospatial/geopackage/issues/260)
2016-10-19	R13	Brian Osborn	1.1.1.1.1	Update application ID
2016-10-19	R13	Jeff Yutzler	Annex F.11	Add elevation extension to standard https://github.com/opengeospatial/geopackage/issues/242 (https://github.com/opengeospatial/geopackage/issues/242)
2016-10-20	R13	Jeff Yutzler	2.3.2.1.1	Add to Requirement 58 to clarify rules for extensions
2016-10-24	R13	Jake Molnar	2.2.6.1.2	Fix typo
2016-12-07	R13	Jeff Yutzler	1.1.1.1.1	Rewrite Requirement 2 to use GPKG application_id + user_version

B.7. Changes to the OGC® Abstract Specification

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

B.8. Changes to OGC® Implementation Standards

None at present.

B.9. Potential Future Work (Informative)

Future versions of this standard MAY do the following: * investigate GeoPackage implementation on SQLite version 4 [B25]. * include requirements for elevation data and routes. * Future enhancements to this standard, 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. * include additional raster / image formats, including fewer restrictions on the image/tiff format. * include additional SQL API routines for interrogation and conversion of raster / image BLOBs. * add infrastructure to the metadata tables such as a `temporal_columns` table that refers to the time properties of data records. * 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). * address symbology and styling information. * include geographic / geodesic geometry types. * create a GeoPackage Abstract Object Model to support data encodings other than SQL. * add UTFGrid (https://github.com/mapbox/utfgrid-spec) support.

Future versions of this standard and/or one for a GeoPackage Web Service MAY do the following: * address utilities for importing and exporting vector, raster and tile data in various formats. * address encryption of GeoPackages and/or individual tables or column values.

B.10. UML Notation

The diagrams that appear in this standard are presented using the Unified Modeling Language (UML) [B14] static structure diagrams. The UML notations used in this standard for RDBMS tables in a GeoPackage are described in UML Notation for RDBMS Tables below.

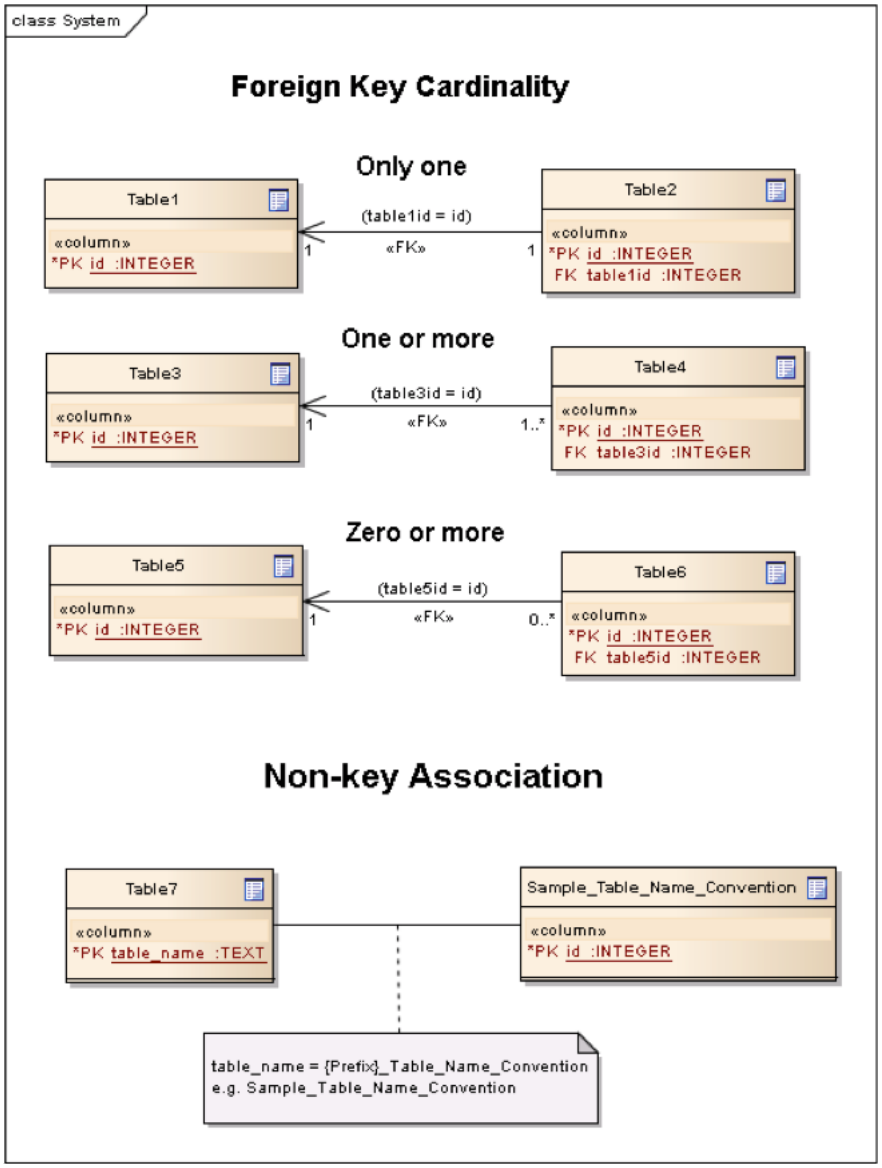

Figure 3. UML Notation for RDBMS Tables

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 RDMBS 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

B.11. GeoPackage Tables Detailed Diagram

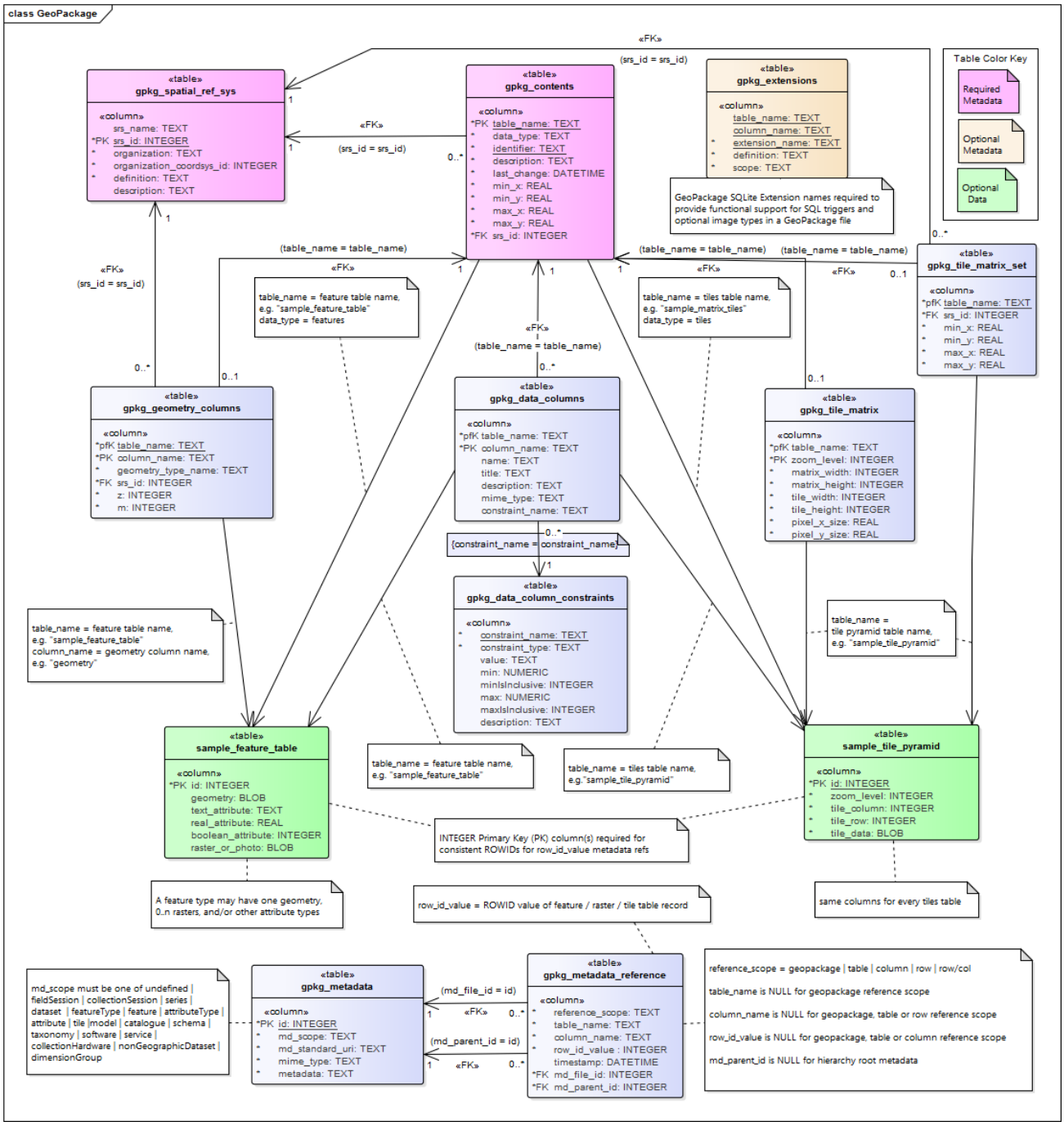

Figure 4. GeoPackage Tables Details

B.12. GeoPackage Minimal Tables for Features Diagram

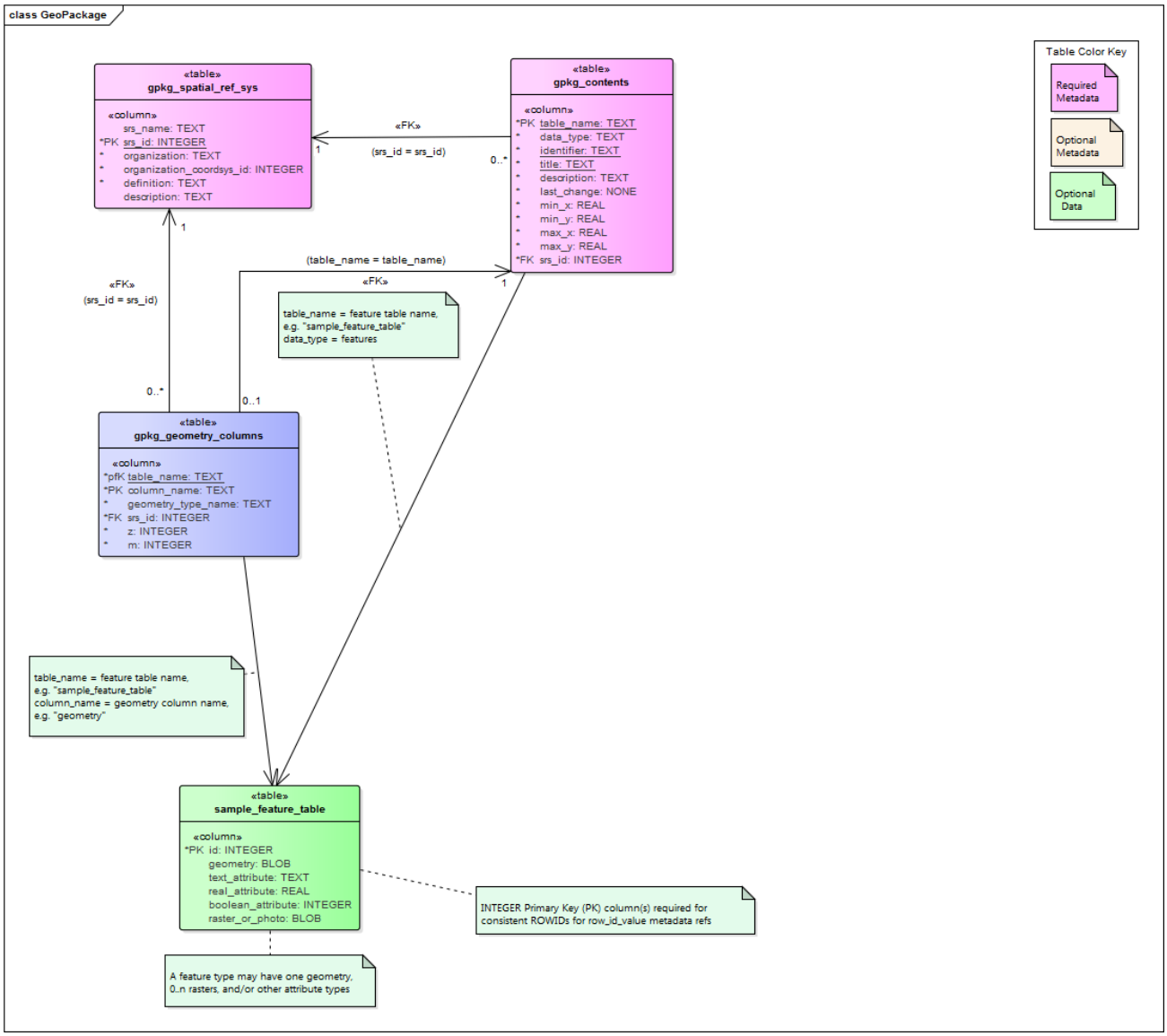

Figure 5. GeoPackage Minimal Tables for Features

B.13. GeoPackage Minimal Tables for Tiles Diagram

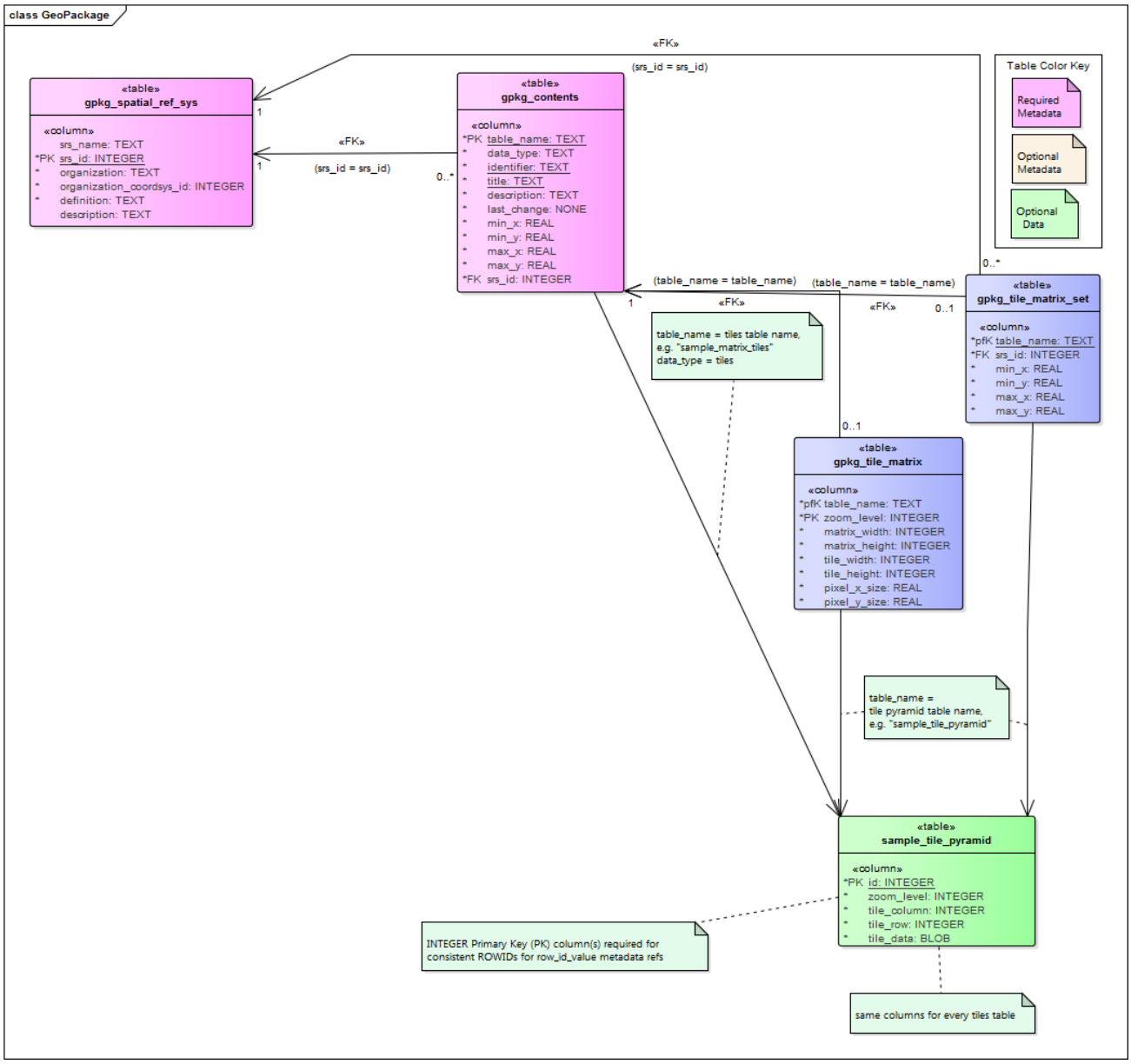

Figure 6. GeoPackage Minimal Tables for Tiles

Annex C: Table Definition SQL (Normative)

C.1. gpkg_spatial_ref_sys

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
);
```

SQL/MM View of gpkg_spatial_ref_sys Definition SQL (Informative)

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

SF/SQL View of gpkg_spatial_ref_sys Definition SQL (Informative)

```
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

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 DATETIME NOT NULL DEFAULT (strftime('%Y-%m-%dT%H:%M:%fZ','now')),
  min_x DOUBLE,
  min_y DOUBLE,
  max_x DOUBLE,
  max_y DOUBLE,
  srs_id INTEGER,
  CONSTRAINT fk_gc_r_srs_id FOREIGN KEY (srs_id) REFERENCES gpkg_spatial_ref_sys(srs_id)
);
```

C.3. gpkg_geometry_columns

gpkg_geometry_columns Table Definition SQL

```
CREATE TABLE gpkg_geometry_columns (
  table_name TEXT NOT NULL,
  column_name TEXT NOT NULL,
  geometry_type_name TEXT NOT NULL,
  srs_id INTEGER NOT NULL,
  z TINYINT NOT NULL,
  m TINYINT NOT NULL,
  CONSTRAINT pk_geom_cols PRIMARY KEY (table_name, column_name),
  CONSTRAINT uk_gc_table_name UNIQUE (table_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)
);
```

SQL/MM View of gpkg_geometry_columns Definition SQL (Informative)

```
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;
```

SF/SQL VIEW of gpkg_geometry_columns Definition SQL (Informative)

```
CREATE VIEW geometry_columns AS
  SELECT
    table_name AS f_table_name,
    column_name AS f_geometry_column,
    code4name (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;
```

Implementer must provide code4name(geometry_type_name) SQL function

C.4. sample_feature_table (Informative)

sample_feature_table Table Definition SQL (Informative)

```
CREATE TABLE sample_feature_table (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  geometry GEOMETRY,
  text_attribute TEXT,
  real_attribute REAL,
  boolean_attribute BOOLEAN,
  raster_or_photo BLOB
);
```

C.5. gpkg_tile_matrix_set

gpkg_tile_matrix_set Table Creation SQL

```
CREATE TABLE gpkg_tile_matrix_set (
  table_name TEXT NOT NULL PRIMARY KEY,
  srs_id INTEGER NOT NULL,
  min_x DOUBLE NOT NULL,
  min_y DOUBLE NOT NULL,
  max_x DOUBLE NOT NULL,
  max_y DOUBLE NOT NULL,
  CONSTRAINT fk_gtms_table_name FOREIGN KEY (table_name) REFERENCES gpkg_contents(table_name),
  CONSTRAINT fk_gtms_srs FOREIGN KEY (srs_id) REFERENCES gpkg_spatial_ref_sys (srs_id)
);
```

C.6. gpkg_tile_matrix

gpkg_tile_matrix Table Creation SQL

```
CREATE TABLE gpkg_tile_matrix (
  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),
  CONSTRAINT fk_tmm_table_name FOREIGN KEY (table_name) REFERENCES gpkg_contents(table_name)
);
```

EXAMPLE: gpkg_tile_matrix Insert Statement (Informative)

```
INSERT INTO gpkg_tile_matrix VALUES (
  "sample_tile_pyramid",
  0,
  1,
  1,
  512,
  512,
  2.0,
  2.0
);
```

C.7. sample_tile_pyramid (Informative)

EXAMPLE: tiles table Create Table SQL (Informative)

```
CREATE TABLE sample_tile_pyramid (
  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)
)
```

EXAMPLE: tiles table Insert Statement (Informative)

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

C.8. gpkg_extensions

gpkg_extensions Table Definition SQL

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

C.9. sample_attributes_table (Informative)

EXAMPLE: Attributes table Create Table SQL (Informative)

```
CREATE TABLE sample_attributes (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  text_attribute TEXT,
  real_attribute REAL,
  boolean_attribute BOOLEAN,
  raster_or_photo BLOB
)
```

EXAMPLE: attributes table Insert Statement (Informative)

```
INSERT INTO sample_attributes(text_attribute, real_attribute, boolean_attribute, raster_or_photo)
VALUES (
  "place",
  1,
  true,
  "BLOB VALUE"
)
```

Annex D: Trigger Definition SQL (Informative)

D.1. gpkg_tile_matrix

Table 16. gpkg_tile_matrix Trigger Definition SQL

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

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

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

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

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

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

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

CREATE TRIGGER 'gpkg_tile_matrix_pixel_x_size_update'
BEFORE UPDATE OF pixel_x_size ON 'gpkg_tile_matrix'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'update on table ''gpkg_tile_matrix'' violates constraint: pixel_x_size must
be greater than 0')
WHERE NOT (NEW.pixel_x_size > 0);
END

CREATE TRIGGER 'gpkg_tile_matrix_pixel_y_size_insert'
BEFORE INSERT ON 'gpkg_tile_matrix'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'insert on table ''gpkg_tile_matrix'' violates constraint: pixel_y_size must
```

```
be greater than 0')
WHERE NOT (NEW.pixel_y_size > 0);
END

CREATE TRIGGER 'gpkg_tile_matrix_pixel_y_size_update'
BEFORE UPDATE OF pixel_y_size ON 'gpkg_tile_matrix'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'update on table ''gpkg_tile_matrix'' violates constraint: pixel_y_size must
be greater than 0')
WHERE NOT (NEW.pixel_y_size > 0);
END
```

D.2. sample_feature_table

Table 17. 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(ABORT, '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 (ABORT, '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.

D.3. sample_tile_pyramid

Table 18. tiles table Trigger Definition SQL

```
CREATE TRIGGER "sample_tile_pyramid_zoom_insert"
BEFORE INSERT ON "sample_tile_pyramid"
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'insert on table ''sample_tile_pyramid'' violates constraint: zoom_level not
specified for table in gpkg_tile_matrix')
WHERE NOT (NEW.zoom_level IN (SELECT zoom_level FROM gpkg_tile_matrix WHERE table_name =
'sample_tile_pyramid')) ;
END

CREATE TRIGGER "sample_tile_pyramid_zoom_update"
BEFORE UPDATE OF zoom_level ON "sample_tile_pyramid"
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'update on table ''sample_tile_pyramid'' violates constraint: zoom_level not
specified for table in gpkg_tile_matrix')
WHERE NOT (NEW.zoom_level IN (SELECT zoom_level FROM gpkg_tile_matrix WHERE table_name =
'sample_tile_pyramid')) ;
END

CREATE TRIGGER "sample_tile_pyramid_tile_column_insert"
BEFORE INSERT ON "sample_tile_pyramid"
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'insert on table ''sample_tile_pyramid'' violates constraint: tile_column
cannot be < 0')
WHERE (NEW.tile_column < 0) ;
SELECT RAISE(ABORT, 'insert on table ''sample_tile_pyramid'' violates constraint: tile_column must
by < matrix_width specified for table and zoom level in gpkg_tile_matrix')
WHERE NOT (NEW.tile_column < (SELECT matrix_width FROM gpkg_tile_matrix WHERE table_name =
'sample_tile_pyramid' AND zoom_level = NEW.zoom_level));
END

CREATE TRIGGER "sample_tile_pyramid_tile_column_update"
BEFORE UPDATE OF tile_column ON "sample_tile_pyramid"
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'update on table ''sample_tile_pyramid'' violates constraint: tile_column
cannot be < 0')
WHERE (NEW.tile_column < 0) ;
SELECT RAISE(ABORT, 'update on table ''sample_tile_pyramid'' violates constraint: tile_column must
by < matrix_width specified for table and zoom level in gpkg_tile_matrix')
WHERE NOT (NEW.tile_column < (SELECT matrix_width FROM gpkg_tile_matrix WHERE table_name =
'sample_tile_pyramid' AND zoom_level = NEW.zoom_level));
END

CREATE TRIGGER "sample_tile_pyramid_tile_row_insert"
BEFORE INSERT ON "sample_tile_pyramid"
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'insert on table ''sample_tile_pyramid'' violates constraint: tile_row cannot
be < 0')
WHERE (NEW.tile_row < 0) ;
SELECT RAISE(ABORT, 'insert on table ''sample_tile_pyramid'' violates constraint: tile_row must by
< matrix_height specified for table and zoom level in gpkg_tile_matrix')
WHERE NOT (NEW.tile_row < (SELECT matrix_height FROM gpkg_tile_matrix WHERE table_name =
'sample_tile_pyramid' AND zoom_level = NEW.zoom_level));
END

CREATE TRIGGER "sample_tile_pyramid_tile_row_update"
BEFORE UPDATE OF tile_row ON "sample_tile_pyramid"
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'update on table ''sample_tile_pyramid'' violates constraint: tile_row cannot
be < 0')
WHERE (NEW.tile_row < 0) ;
SELECT RAISE(ABORT, 'update on table ''sample_tile_pyramid'' violates constraint: tile_row must by
< matrix_height specified for table and zoom level in gpkg_tile_matrix')
WHERE NOT (NEW.tile_row < (SELECT matrix_height FROM gpkg_tile_matrix WHERE table_name =
'sample_tile_pyramid' AND zoom_level = NEW.zoom_level));
END
```

Annex E: GeoPackage Extension Template (Informative)

Extension Title

Title of the Extension

Introduction

Description of extension

Extension Author

Author of extension, author_name.

Extension Name or Template

Name of the extension or definition of the template to create the name of extensions that should be used in gpkg_extensions

Extension Type

"Extension of Existing Requirement in Clause(s) XXX" or "New Requirement Dependent on Clause(s) YYY"

Applicability

Tables and/or columns on which this extension may be applied

Scope

Read-write or write-only with clarification if necessary

Requirements

Definition of extension and interdependencies with other extensions if any.

GeoPackage

Definition of extension data or MIME type(s)

Definition of extension tables or table templates

Definition of triggers or trigger templates

GeoPackage SQLite Configuration

Definition of SQLite configuration settings

Setting compile or runtime	Option	Shall / Not (Value)	Discussion

GeoPackage SQLite Extension

Definition of SQL functions

SQL Function	Description	Use
foo(bar, baz) : datatype	Returns r when w	

Abstract Test Suite

All test cases required to verify conformance to this extension.

Examples (Informative)

Any example or samples demonstrating the extension in use.

Annex F: Registered Extensions (Normative)

This clause specifies requirements for GeoPackage extensions. Definitions of those extensions are in the form specified by the template in GeoPackage Extension Template (Informative).

Extension Name	Content Type
GeoPackage Non-Linear Geometry Types	features
User Defined Geometry Types Extension of GeoPackageBinary Geometry Encoding	features
RTree Spatial Indexes	features
Geometry Type Triggers	features
Geometry SRS ID Triggers	features
Zoom Other Intervals	tiles
Tiles Encoding WebP	tiles
Metadata	general
Schema	features
WKT for Coordinate Reference Systems	spatial reference systems
Tiled Gridded Elevation Data	tiled gridded elevation data

F.1. GeoPackage Non-Linear Geometry Types

Introduction

This extension of clause SQL Geometry Types defines additional geometry types.

Clause 2.1.4 of the GeoPackage Version 1 Encoding Standard specifies support for the Geometry, Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeomCollection geometry types in the GeoPackageBinary geometry encoding format specified in clause 2.1.3. This extension specifies support for the additional CircularString, CompoundCurve, CurvePolygon, MultiCurve, MultiSurface, Curve, and Surface geometry types in the GeoPackage Binary geometry encoding format using the codes from Geometry Type Codes (Extension).

Extension Author

GeoPackage SWG, author_name `gpkg`

Extension Name or Template

Extension names are constructed from the gpkg_geom_<gname> template where <gname> is the uppercase name of the extension geometry type from Geometry Type Codes (Extension).

Extension Type

Extension of Existing Requirement in clause SQL Geometry Types

Applicability

This extension applies to any column specified in the `gpkg_geometry_columns` table.

Scope

Read-write

Requirements

GeoPackage

> **Requirement 65**
>
> A GeoPackage MAY store feature table geometries with the extended non-linear geometry types (CircularString, CompoundCurve, CurvePolygon, MultiCurve, MultiSurface, Curve, Surface) in Geometry Types (Normative).

> **Requirement 66**

The GeoPackageBinary geometry encoding format specified in clause Geometry Encoding SHALL be used to encode non-linear geometry types using the type codes in Geometry Types (Normative) table Geometry Type Codes (Extension).

Requirement 67

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 uppercase name of the extension geometry type from Geometry Types (Normative) used in a GeoPackage.

Requirement 68

A GeoPackage that contains a gpkg_geometry_columns table or updateable view with row records that specify extension geometry_type_name column values SHALL contain a gpkg_extensions table that contains row records with table_name and column_name values from the gpkg_geometry_columns row records that identify extension type uses, and extension_name column values for each of those geometry types constructed per the previous requirement Requirement 67.

GeoPackage SQLite Configuration

None

GeoPackage SQLite Extension

Requirement 69

~~SQL functions that operate on GeoPackageBinary geometries as specified in other extensions SHALL operate correctly on the non-linear geometries specified in this extension~~.

Abstract Test Suite

GeoPackage Extension Types

Test Case ID	`/reg_ext/features/geometry_encoding/data/geopackage_extension_types/existing_sparse_data`
Test Purpose	Verify that existing extended non-linear geometry types are stored in valid StandardGeoPackageBinary 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, log fail if GeoPackageBinary “X” type flag is 1 c. For each row from step a, if bytes 2-5 of cn.wkb as uint32 in endianness of gc.wkb byte 1of cn from #1 are a geometry type value from Annex G Table 43, then d. 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 e. If cn.flags.E is 2,4 and some gc.wkb.z is outside of cnenvelope.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	Annex F.1 Req 65:
Test Type	Capability

Test Case ID	`/reg_ext/features/geometry_encoding/data/geopackage_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 GP headers containing big and little endian WKBs for 0-1 envelope contents indicator codes for every geometry type value from Annex G without and with z and/or m values. 4. Fail otherwise
Reference	Annex F.1 Req 66:
Test Type	Capability

Extensions Name

Test Case ID	`/reg_ext/features/geometry_encoding/data/geopackage_extension_types/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 used in a GeoPackage.
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 two bytes of each geometry column value are “GP”, then A. /opt/extension_mechanism/extensions/data/table_def B. Fail if failed C. SELECT ST_GeometryType(geometry column value) AS <gtype>; D. SELECT extension_name FROM gpkg_extensions WERE table_name = result_set_table_name AND column_name = result_set_column_name AND extension_name = 'gpkg_geom_' || <gtype> I. Fail if result set is empty II. Log pass otherwise 4. Pass if logged pass and no fails
Reference	Annex F.1 Req 67:
Test Type	Basic

Extensions Row

Test Case ID	`/reg_ext/features/geometry_encoding/data/geopackage_extension_types/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	Annex F.1 Req 68:
Test Type	Capability

F.2. User Defined Geometry Types Extension of GeoPackageBinary Geometry Encoding

On August 15, 2016 the GeoPackage SWG voted to remove this extension from the standard due to interoperability concerns. For more information see the release notes.

F.3. RTree Spatial Indexes

Introduction

This extension adds a new capability for spatially indexing columns with geometries encoded per clause Geometry Encoding and User Defined Geometry Types Extension of GeoPackageBinary Geometry Encoding.

The RTree index extension provides a means to encode an RTree index for geometry values in a GeoPackage. An RTree index provides 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.

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.[25]

Extension Author

GeoPackage SWG, author_name `gpkg`.

Extension Name or Template

`gpkg_rtree_index`

Extension Type

New Requirement dependent on clauses Geometry Encoding and User Defined Geometry Types Extension of GeoPackageBinary Geometry Encoding.

Applicability

This extension applies to any column specified in the `gpkg_geometry_columns` table.

Scope

Write-only, because it does not change the result of reads, although it may improve their performance.

Requirements

This extension uses the rtree implementation provided by the SQLite R*Tree Module extension documented at http://www.sqlite.org/rtree.html (http://www.sqlite.org/rtree.html).

GeoPackage

> **Requirement 70**
>
> The “gpkg_rtree_index” extension name SHALL be used as a gpkg_extensions table extension name column value to specify implementation of spatial indexes on a geometry column.

> **Requirement 71**
>
> A GeoPackage 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.

> **Requirement 72**

> A GeoPackage SHALL implement spatial indexes on feature table geometry columns using the SQLite Virtual Table RTrees and triggers specified below. The tables below contain SQL templates with variables. Replace the following template variables with the specified values to create the required SQL statements:
> <t>: The name of the feature table containing the geometry column
> <c>: The name of the geometry column in <t> that is being indexed
> <i>: The name of the integer primary key column in <t> as specified in Requirement 29

Create Virtual Table

RTree spatial indexes on geometry columns 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.

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 <i>, 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 and <i> is replaced with the name of the feature table integer primary key column.

Define Triggers to Maintain Spatial Index Values

For each spatial index in a GeoPackage, 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:

```
/* 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.<i>,
    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.<i> = NEW.<i> AND
       (NEW.<c> NOTNULL AND NOT ST_IsEmpty(NEW.<c>))
BEGIN
  INSERT OR REPLACE INTO rtree_<t>_<c> VALUES (
    NEW.<i>,
    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 */
CREATE TRIGGER rtree_<t>_<c>_update2 AFTER UPDATE OF <c> ON <t>
  WHEN OLD.<i> = NEW.<i> AND
       (NEW.<c> ISNULL OR ST_IsEmpty(NEW.<c>))
BEGIN
  DELETE FROM rtree_<t>_<c> WHERE id = OLD.<i>;
END;

/* Conditions: Update of any column
               Row ID change
               Non-empty geometry
   Actions   : Remove record from rtree for old <i>
               Insert record into rtree for new <i> */
CREATE TRIGGER rtree_<t>_<c>_update3 AFTER UPDATE OF <c> ON <t>
  WHEN OLD.<i> != NEW.<i> AND
       (NEW.<c> NOTNULL AND NOT ST_IsEmpty(NEW.<c>))
BEGIN
  DELETE FROM rtree_<t>_<c> WHERE id = OLD.<i>;
  INSERT OR REPLACE INTO rtree_<t>_<c> VALUES (
    NEW.<i>,
    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 <i> */
CREATE TRIGGER rtree_<t>_<c>_update4 AFTER UPDATE ON <t>
  WHEN OLD.<i> != NEW.<i> AND
       (NEW.<c> ISNULL OR ST_IsEmpty(NEW.<c>))
BEGIN
  DELETE FROM rtree_<t>_<c> WHERE id IN (OLD.<i>, NEW.<i>);
END;

/* Conditions: Row deleted
   Actions   : Remove record from rtree for old <i> */
CREATE TRIGGER rtree_<t>_<c>_delete AFTER DELETE ON <t>
  WHEN old.<c> NOT NULL
```

```
BEGIN
  DELETE FROM rtree_<t>_<c> WHERE id = OLD.<i>;
END;
```

where <t> and <c> are replaced with the names of the feature table and geometry column being indexed and <i> is replaced with the name of the feature table integer primary key column.

GeoPackage SQLite Configuration

Definition of SQLite configuration settings

Setting compile or runtime	Option	Shall / Not (Value)	Discussion
compile	SQLITE_ENABLE_RTREE	Shall	RTrees ares used for GeoPackage Spatial Indexes
compile	SQLITE_RTREE_INT_ONLY	Not	RTrees with floating point values are used for GeoPackage spatial indexes

GeoPackage SQLite Extension

Definition of SQL functions

SQL Function	Description	Use
ST_IsEmpty(geom Geometry): integer	Returns 1 if geometry value is empty, 0 if not empty, NULL if geometry value is NULL	Test if a geometry value corresponds to the empty set
ST_MinX(geom Geometry): real	Returns the minimum X value of the bounding envelope of a geometry	Update the spatial index on a geometry column in a feature table
ST_MaxX(geom Geometry): real	Returns the maximum Y value of the bounding envelope of a geometry	Update the spatial index on a geometry column in a feature table
ST_MinY(geom Geometry): real	Returns the minimum X value of the bounding envelope of a geometry	Update the spatial index on a geometry column in a feature table
ST_MaxY(geom Geometry): real	Returns the maximum Y value of the bounding envelope of a geometry	Update the spatial index on a geometry column in a feature table

> **Requirement 73**
>
> The SQL functions on geometries in this SQLite Extension SHALL operate correctly on extended geometry types specified by User Defined Geometry Types Extension of GeoPackageBinary Geometry Encoding and/or GeoPackage Non-Linear Geometry Types when those extensions are also implemented.

Abstract Test Suite

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 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 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 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 Annex F.3 Req 75

Test Type Capability

Test Case ID `/reg_ext/features/spatial_indexes/implementation/sql_functions`

Test Purpose Verify the correct implementation of sql functions used in spatial indexes on feature table geometry columns.

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 geometry type in Annex G, for an assortment of srs_ids, for an assortment of coordinate values including empty geometries, without and with z and / or m values, in both big and little endian encodings:
   a. SELECT 'Fail' FROM <gtype_test> WHERE ST_IsEmpty(geom.) != empty
   b. SELECT 'Fail' FROM <gtype_test> WHERE ST_MinX(geom) != minx
   c. SELECT 'Fail' FROM <gtype_test> WHERE ST_MaxX(geom) != maxx
   d. SELECT 'Fail' FROM <gtype_test> WHERE ST_MinY(geom) != miny
   e. SELECT 'Fail' FROM <gtype_test> WHERE ST_MaxY(geom) != maxy ..
3. Pass if no 'Fail' selected from step 2

Reference Annex F.3 Req 76

Test Type Capability

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	Annex F.3 Req 77
Test Type	Basic

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_name
Reference	Annex F.3 Req 78
Test Type	Capability

F.4. Geometry Type Triggers

On August 15, 2016 the GeoPackage SWG voted to remote this extension from the standard due to interoperability concerns. For more information see the release notes.

F.5. Geometry SRS ID Triggers

On August 15, 2016 the GeoPackage SWG voted to remote this extension from the standard due to interoperability concerns. For more information see the release notes.

F.6. Zoom Other Intervals

Introduction

This extension of clause Zoom Levels allows zoom level intervals other than a factor of two.

In a GeoPackage, zoom levels are integers in sequence from 0 to n that identify tile matrix layers in a tile matrix set that contain tiles of decreasing spatial extent and finer spatial resolution. Adjacent zoom levels immediately precede or follow each other and differ by a value of 1. Pixel sizes are real numbers in the terrain units of the spatial reference system of a tile image specifying the dimensions of the real world area represented by one pixel. Pixel sizes MAY vary by a constant factor or by different factors or intervals between some or all adjacent zoom levels in a tile matrix set. In the commonly used "zoom times two" convention, pixel sizes vary by a factor of 2 between all adjacent zoom levels, as shown in the example in Tiles Zoom Times Two Example (Informative).

This extension enables use of "zoom other intervals" conventions with different factors or irregular intervals with pixel sizes chosen for intuitive cartographic representation of raster data, or to coincide with the original pixel size of commonly used global image products. See WMTS [16] Annex E for additional examples of both conventions.

Extension Author

GeoPackage SWG, author_name `gpkg`

Extension Name or Template

`gpkg_zoom_other`

Extension Type

Extension of Existing Requirement in clause 2.2.3.

Applicability

This extension applies to any table listed in the `gpkg_contents` table with a data_type of `tiles`.

Scope

Read-write

Requirements

GeoPackage

> **Requirement 74**
>
> The “gpkg_zoom_other” extension name SHALL be used as a gpkg_extensions table extension name column value to specify implementation of other zoom intervals on a tile pyramid user data table as specified in Zoom Other Intervals.

> **Requirement 75**
>
> A GeoPackage that implements other zoom intervals SHALL have a gpkg_extensions table that contains a row for each tile pyramid 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.

> **Requirement 76**
>
> Tile pyramid user data tables MAY have pixel sizes that vary by irregular intervals or by regular intervals other than a factor of two (the default) between adjacent zoom levels. Extends Requirement 35.

The `pixel_x_size` and / or `pixel_y_size` column values in the `gpkg_tile_matrix` table vary by irregular intervals or by regular intervals other than a factor of two (the default) between adjacent zoom levels for a particular tile matrix set pyramid table.

GeoPackage SQLite Configuration

None

GeoPackage SQLite Extension

None

Abstract Test Suite

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 factors of two zoom intervals.

Test Method	1. SELECT table_name FROM gpkg_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 gpkg_tile_matrix 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 factors 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	Annex F.6 Req 87
Test Type	Basic

Extensions Row

Test Case ID	`/reg_ext/tiles/zoom_levels/data/zoom_other_ext_row`
Test Purpose	Verify that tiles tables with other than factors 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	Annex F.6 Req 88
Test Type:	Capability

Zoom Interval

Test Case ID	`/reg_ext/tiles/zoom_levels/data/zoom_intervals`
Test Purpose	Verify that zoom level pixel sizes for tile matrix user data tables vary by factors of 2 between adjacent zoom levels in the tile matrix metadata table only for tile matrix sets that this extension does not apply to.
Test Method	1. Override test `/opt/tiles/zoom_levels/data/zoom_times_two` 2. SELECT table_name AS tn FROM gpkg_contents WHERE data_type = 'tiles' 3. For each row tn from step 2 a. WHEN (SELECT tbl_name FROM sqlite_master WHERE tbl_name = 'gpkg_extensions') = 'gpkg_extensions' THEN (SELECT table_name from gpkg_extensions WHERE extension_name = 'gpkg_zoom_other' AND table_name = 'tn') END; b. If returns empty result set, execute test `/opt/tiles/zoom_levels/data/zoom_times_two` 4. Pass if no fails
Reference	Annex F.6 Req 89
Test Type	Capability

F.7. Tiles Encoding WebP

Introduction

This extension of clauses Tile Encoding PNG and Tile Encoding JPEG allows encoding of tile images in WebP format.

PNG and JPEG are the default MIME types for encoding images in tile pyramid user data tables. This extension allows the use of `image/x-webp` as an additional encoding type.

Extension Author

GeoPackage SWG, author_name `gpkg`.

Extension Name or Template

`gpkg_webp`

Extension Type

Extension of Existing Requirement in clauses Tile Encoding PNG and Tile Encoding JPEG.

Applicability

This extension applies to any table listed in the `gpkg_contents` table with a data_type of `tiles`.

Scope

Read-write

Requirements

GeoPackage

> **Requirement 77**
>
> The "gpkg_webp" extension name SHALL be used as a gpkg_extensions table extension name column value to specify storage of tile pyramid images in WEBP format as specified in Tiles Encoding WebP.

> **Requirement 78**
>
> A GeoPackage that contains tile pyramid user data tables with tile_data columns that contain 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”.

> **Requirement 79**
>
> A GeoPackage that contains a tile pyramid user data table that contains tile data MAY store tile_data in MIME type image/x-webp [22]. The MIME type of values of the `tile_data` column in tile pyramid user data tables SHALL be `image/x-webp`.

GeoPackage SQLite Configuration

None

GeoPackage SQLite Extension

None

Abstract Test Suite

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 gpkg_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	Annex F.7 Req 90
Test Type	Basic

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:	Annex F.7 Req 91
Test Type	Capability

Extensions Mime Type

Test Case ID	`/reg_ext/tiles/tiles_encoding_webp/data/mime_type_webp`
Test Purpose	Verify that a tile matrix user data table that conforms to this extension contains tile data of MIME type image/x-webp.
Test Method	1. SELECT table_name AS tn FROM gpkg_contents WHERE data_type = 'tiles' 2. For each row tn from step 2 a. WHEN (SELECT tbl_name FROM sqlite_master WHERE tbl_name = 'gpkg_extensions') = 'gpkg_extensions' THEN (SELECT extension_name FROM gpkg_extensions WHERE table_name = 'tn') END; b. For each row extension_name from step a i. Not testable if extension_name is not *gpkg_webp* ii. SELECT tile_data from tn A. Pass if tile data in MIME type image/x-webp B. Fail if no passes
Reference	Annex F.7 Req 92
Test Type	Capability

F.8. Metadata

Introduction

Two tables in a GeoPackage provide a means of storing metadata in MIME [21] 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 models as defined in ISO 19115 [28] and illustrated in Hierarchical Metadata Example One - ISO19115. and Raster or Tile Metadata Example. 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 Metadata Table, and the `gpkg_metadata_reference` table that relates `gpkg_metadata` to GeoPackage data is described in clause Metadata Reference Table. There is no GeoPackage requirement that such metadata be provided or that defined metadata be structured in a hierarchical fashion [26] with more than one level, only that if it is, these tables SHALL be used. Such metadata [27] and data that relates it to GeoPackage contents SHALL NOT be stored in other tables.

Extension Author

GeoPackage SWG, author_name `gpkg`

Extension Name or Template

`gpkg_metadata`

Extension Type

New Requirement

Applicability

This extension applies to any content in the GeoPackage.

Scope

Read-write

Requirements

Metadata Table

Table Definition

> **Requirement 80**
>
> A GeoPackage MAY contain a table named gpkg_metadata. If present it SHALL be defined per clause 2.4.2.1.1 Table Definition, Metadata Table Definition and gpkg_metadata Table Definition SQL.

The first component of GeoPackage metadata is the `gpkg_metadata` table that MAY contain metadata in MIME [21] encodings structured in accordance with any authoritative metadata specification, such as ISO 19115 [28], ISO 19115-2 [B6], ISO 19139 [B7], Dublin Core [B8], CSDGM [B10], DDMS [B12], NMF/NMIS [B13], etc. The GeoPackage interpretation of what constitutes “metadata” is a broad one that includes UML models [B14] encoded in XMI [B15], GML Application Schemas [30], ISO 19110 feature catalogues [B18], OWL [B20] and SKOS [B21] taxonomies, etc.

Table 19. Metadata Table Definition

Column Name	Column Type	Column Description	Null	Default	Key
`id`	INTEGER	Metadata primary key	no		PK
`md_scope`	TEXT	Case sensitive name of the data scope to which this metadata applies; see Metadata Scopes below	no	‘dataset’	
`md_standard_uri`	TEXT	URI [23] reference to the metadata structure definition authority [28]	no		
`mime_type`	TEXT	MIME [21] encoding of metadata	no	text/xml [24]	
`metadata`	TEXT	metadata	no	”	

The md_standard_uri data value provides an identifier for the metadata structure (schema) specified by its definition authority. The structure (schema) information could be in whatever encoding is used by the definition authority, e.g. UML [B14], or IDEF1x [B16], or XML/Schema [25][26][27], or RDF/S [B19].

See gpkg_metadata Table Definition SQL.

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 Metadata Scopes 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 [29] for use in the GeoPackage specification by addition of entries with “NA” as the scope code column in Metadata Table Definition.

Table 20. Metadata Scopes

Name (md_scope)	Scope Code	Definition
undefined	NA	Metadata information scope is undefined
fieldSession	012	Information applies to the field session
collectionSession	004	Information applies to the collection session
series	006	Information applies to the (dataset) series [30]
dataset	005	Information applies to the (geographic feature) dataset
featureType	010	Information applies to a feature type (class)
feature	009	Information applies to a feature (instance)
attributeType	002	Information applies to the attribute class
attribute	001	Information applies to the characteristic of a feature (instance)
tile	016	Information applies to a tile, a spatial subset of geographic data
model	015	Information applies to a copy or imitation of an existing or hypothetical object
catalog	NA	Metadata applies to a feature catalog [31]
schema	NA	Metadata applies to an application schema [32]
taxonomy	NA	Metadata applies to a taxonomy or knowledge system [33]
software	013	Information applies to a computer program or routine
service	014	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
collectionHardware	003	Information applies to the collection hardware class
nonGeographicDataset	007	Information applies to non-geographic data
dimensionGroup	008	Information applies to a dimension group

> **Requirement 81**
>
> Each `md_scope` column value in a `gpkg_metadata` table or updateable view SHALL be one of the name column values from Metadata Scopes.

Metadata Reference Table

Table Definition

> **Requirement 82**

A GeoPackage that contains a `gpkg_metadata` table SHALL contain a `gpkg_metadata_reference` table per clause 2.4.3.1.1 Table Definition, Metadata Reference Table Definition (Table Name: gpkg_metadata_reference) and gpkg_metadata_reference Table Definition SQL.

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 21. Metadata Reference Table Definition (Table Name: gpkg_metadata_reference)

Column Name	Col Type	Column Description	Null	Default	Key
`reference_scope`	TEXT	Lowercase metadata reference scope; one of 'geopackage', 'table','column', 'row', 'row/col'	no		
`table_name`	TEXT	Name of the table to which this metadata reference applies, or NULL for reference_scope of 'geopackage'.	yes		
`column_name`	TEXT	Name of the column to which this metadata reference applies; NULL for `reference_scope` of 'geopackage','table' or 'row', or the name of a column in the `table_name` table for `reference_scope` of 'column' or 'row/col'	yes		
`row_id_value` [34]	INTEGER	NULL for `reference_scope` of 'geopackage', 'table' or 'column', or the rowed of a row record in the `table_name` table for `reference_scope` of 'row' or 'row/col'	yes		
`timestamp`	DATETIME	timestamp value in ISO 8601 format as defined by the strftime function '%Y-%m-%dT%H:%M:%fZ' format string applied to the current time	no	strftime('%Y-%m-%dT%H:%M:%fZ', 'now')	
`md_file_id`	INTEGER	`gpkg_metadata` table id column value for the metadata to which this `gpkg_metadata_reference` applies	no		FK
`md_parent_id`	INTEGER	`gpkg_metadata` table id column value for the hierarchical parent `gpkg_metadata` for the `gpkg_metadata` to which this `gpkg_metadata_reference` applies, or NULL if `md_file_id` forms the root of a metadata hierarchy	yes		FK

Every row in `gpkg_metadata_reference` that has null value as `md_parent_id` forms the root of a metadata hierarchy.[35]

See Table Definition SQL (Normative) clause gpkg_metadata_reference Table Definition SQL.

Table Data Values

Requirement 83

Every `gpkg_metadata_reference` table reference scope column value SHALL be one of 'geopackage', 'table', 'column', 'row', 'row/col' in lowercase.

Requirement 84

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 `gpkg_metadata_reference` table row SHALL have a `table_name` column value that references a value in the `gpkg_contents` `table_name` column.

Requirement 85

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.

Requirement 86

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.

Requirement 87

Every `gpkg_metadata_reference` table row timestamp column value SHALL be in ISO 8601 [29] format containing a complete date plus UTC hours, minutes, seconds and a decimal fraction of a second, with a ‘Z’ (‘zulu’) suffix indicating UTC.[36]

Requirement 88

Every `gpkg_metadata_reference` table row `md_file_id` column value SHALL be an id column value from the `gpkg_metadata` table.

Requirement 89

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.

Abstract Test Suite

Metadata Table

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	Annex F.8 Req 93
Test Type	Basic

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 15 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 15 in clause 2.4.2.1.2 4. Pass if no fails
Reference	Annex F.8 Req 94
Test Type:	Capabilities

Metadata Reference Table

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 33. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant. 4. Fail otherwise.
Reference	Annex F.8 Req 95
Test Type	Basic

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	Annex F.8 Req 96
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 Annex F.8 Req 97

Test Type Capability

Test Case ID `/opt/metadata/metadata_reference/data/data_values_column_name`

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 Annex F.8 Req 98

Test Type Capability

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. Fail if result set is empty c. Log pass otherwise 7. Pass if logged pass and no fails.
Reference	Annex F.8 Req 99
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	Annex F.8 Req 100
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 references a gpkg_metadata id column value.
Test Method	1. PRAGMA foreign_key_check('geometry_columns') 2. Fail if returns any rows with a fourth column foreign key index value of 0
Reference	Annex F.8 Req 101
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 FROM gpkg_metadata_reference AS gmr WHERE gmr.md_file_id == gmr.md_parent_id 4. Fail if result set is not empty 5. SELECT gmr.md_file_id, gmr.md_parent_id, gm.id FROM gpkg_metadata_reference AS gmr LEFT OUTER JOIN gpkg_metadata gm ON gmr.md_parent_id =gm.id 6. Fail if any result set gm.id values are NULL 7. Pass otherwise
Reference	Annex F.8 Req 102
Test Type	Capability

Table Definition SQL

gpkg_metadata

gpkg_metadata Table Definition SQL

```
CREATE TABLE gpkg_metadata (
  id INTEGER CONSTRAINT m_pk PRIMARY KEY ASC NOT NULL,
  md_scope TEXT NOT NULL DEFAULT 'dataset',
  md_standard_uri TEXT NOT NULL,
  mime_type TEXT NOT NULL DEFAULT 'text/xml',
  metadata TEXT NOT NULL DEFAULT ''
);
```

gpkg_metadata_reference

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 DATETIME NOT NULL DEFAULT (strftime('%Y-%m-%dT%H:%M:%fZ','now')),
  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)
);
```

Example: gpkg_metadata_reference SQL insert statement (Informative)

```
INSERT INTO gpkg_metadata_reference VALUES (
  'table',
  'sample_rasters',
  NULL,
  NULL,
  '2012-08-17T14:49:32.932Z',
  98,
  99
)
```

Trigger Definition SQL (Informative)

metadata

metadata Trigger Definition SQL

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

metadata_reference

gpkg_metadata_reference Trigger Definition SQL

```
CREATE TRIGGER 'gpkg_metadata_reference_reference_scope_insert'
BEFORE INSERT ON 'gpkg_metadata_reference'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, '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'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'update on table gpkg_metadata_reference
violates constraint: referrence_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'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'insert 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(ABORT, '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(ABORT, '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_nameIS NOT NULL);
SELECT RAISE(ABORT, 'update 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_row_id_value_insert'
BEFORE INSERT ON 'gpkg_metadata_reference'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, '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;
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(ABORT, '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;
END

CREATE TRIGGER 'gpkg_metadata_reference_timestamp_insert'
BEFORE INSERT ON 'gpkg_metadata_reference'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'insert on table gpkg_metadata_reference
violates constraint: timestamp must be a valid time in ISO 8601
"yyyy-mm-ddThh:mm:ss.cccZ" form')
WHERE NOT (NEW.timestamp GLOB
'[1-2][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]T[0-2][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(ABORT, 'update on table gpkg_metadata_reference
violates constraint: timestamp must be a valid time in ISO 8601
"yyyy-mm-ddThh:mm:ss.cccZ" form')
WHERE NOT (NEW.timestamp GLOB
'[1-2][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]T[0-2][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
```

Examples (Informative)

Hierarchical Metadata Example One - ISO19115.

Suppose we have this metadata:

```
CREATE TABLE gpkg_metadata (
  id INTEGER NOT NULL PRIMARY KEY,
  md_scope TEXT NOT NULL DEFAULT 'undefined',
  md_standard_uri TEXT NOT NULL,
  metadata TEXT NOT NULL
)
```

---

id	md_scope	md_standard_uri	metadata
0	undefined	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
3	series	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
4	dataset	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
5	featureType	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
6	feature	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
7	attributeType	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
8	attribute	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT

and this reference table definition:

```
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','now')),
  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)
)
```

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:

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

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:

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

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:

```
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  */
)
```

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;

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

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  */
)
```

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  */
)
```

Hierarchical Metadata Example Two - Field Data Collection

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:

id	md_scope	md_standard_uri	metadata
1	fieldSession	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
10	featureType	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
11	feature	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT

id	md_scope	md_standard_uri	metadata
12	attribute	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
13	attribute	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
14	feature	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
15	attribute	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
16	attribute	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
17	feature	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
18	attribute	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT
19	attribute	http://schemas.opengis.net/iso/19139/ (http://schemas.opengis.net/iso/19139/)	TEXT

(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:

```
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:

```
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:

```
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 */
1, /* row_id_value  */
(strftime('%Y-%m-%dT%H:%M:%fZ','now')),
13,  /* md_file_id  */
1  /* md_parent_id  */
)

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

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

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:

Table 22. xml_metadata

id	md_scope	md_standard_uri	metadata
0	undefined	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
1	fieldSession	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
2	collectionSession	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
3	series	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
4	dataset	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
5	featureType	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
6	feature	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
7	attributeType	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
8	attribute	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
10	featureType	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
11	feature	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
12	attribute	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
13	attribute	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
14	feature	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
15	attribute	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
16	attribute	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
17	feature	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
18	attribute	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT
19	attribute	http://www.isotc211.org/2005/gmd (http://www.isotc211.org/2005/gmd)	TEXT

Table 23. gpkg_metadata_reference

reference_type	table_name	column_name	row_id_value	timestamp	md_file_id	md_parent_id
table	roads	undefined	0	ts	3	0
table	roads	undefined	0	ts	4	3
table	roads	undefined	0	ts	5	4
column	roads	overhead_clearance	0	ts	7	4
row	bridge	undefined	987	ts	6	4
row/col	bridge	overhead_clearance	987	ts	8	4
table	poi	undefined	0	ts	1	0
row	poi	undefined	0	ts	10	1
row	poi	undefined	1	ts	11	1
row	poi	undefined	2	ts	14	1
row/col	poi	undefined	3	ts	17	1
row/col	poi	point	1	ts	12	1

reference_type	table_name	column_name	row_id_value	timestamp	md_file_id	md_parent_id
row/col	poi	point	2	ts	15	1
row/col	poi	point	3	ts	18	1
row/col	poi	category	1	ts	13	1
row/col	poi	category	2	ts	16	1
row/col	poi	category	3	ts	19	1

Raster or Tile Metadata Example

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 metadata 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

F.9. Schema

Introduction

The schema option provides a means to describe the columns of tables in a GeoPackage with more detail than can be captured by the SQL table definition directly. The information provided by this option can be used by applications to, for instance, present data contained in a GeoPackage in a more user-friendly fashion or implement data validation logic.

Extension Author

GeoPackage SWG, author_name `gpkg`

Extension Name

`gpkg_schema`

Extension Type

New Requirement Dependent on Clause Features

Applicability

This extension may apply to any Vector Feature User Data Tables.

Scope

Read-write

Requirements

Data Columns

Table Definition

> **Requirement 90**

A GeoPackage MAY contain a table or updateable view named `gpkg_data_columns`. If present it SHALL be defined per clause 2.3.2.1.1 Table Definition, Data Columns Table or View Definition and gpkg_data_columns Table Definition SQL.

Table 24. Data Columns Table or View Definition

Column Name	Column Type	Column Description	Null	Key
`table_name`	TEXT	Name of the tiles or feature table	no	PK
`column_name`	TEXT	Name of the table column	no	PK
`name`	TEXT	A human-readable identifier (e.g. short name) for the column_name content	yes	UNIQUE
`title`	TEXT	A human-readable formal title for the column_name content	yes	
`description`	TEXT	A human-readable description for the column_name content	yes	
`mime_type`	TEXT	MIME (http://www.iana.org/assignments/media-types/index.html) [21] type of column_name if BLOB type, or NULL for other types	yes	
`constraint_name`	TEXT	Column value constraint name (lowercase) specified by reference to `gpkg_data_column_constraints.constraint` name	yes	

GeoPackage applications MAY [37] use the `gpkg_data_columns` table to store minimal application schema identifying, descriptive and MIME (http://www.iana.org/assignments/media-types/index.html) [21] type [38] 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.

See gpkg_data_columns Table Definition SQL.

Table Data Values

Requirement 91

Values of the `gpkg_data_columns` table `table_name` column value SHALL reference values in the `gpkg_contents` `table_name` column.

Requirement 92

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.

Requirement 93

The constraint_name column value in a gpkg_data_columns table MAY be NULL. If it is not NULL, it SHALL contain a constraint_name column value (which SHALL be lowercase) from the gpkg_data_column_constraints table.

Data Column Constraints

Table Definition

Requirement 94

A GeoPackage MAY contain a table or updateable view named gpkg_data_column_constraints. If present it SHALL be defined per clause 2.3.3.1.1 Table Definition, Data Column Constraints Table or View Definition and gpkg_data_columns Table Definition SQL.

The `gpkg_data_column_constraints` table contains data to specify restrictions on basic data type column values. The constraint_name column is referenced by the `constraint_name` column in the `gpkg_data_columns` table defined in Data Columns Table or View Definition.

Table 25. Data Column Constraints Table or View Definition

Column Name	Column Type	Column Description	Null	Key
`constraint_name`	TEXT	Name of constraint (lowercase)	no	Unique
`constraint_type`	TEXT	Type name of constraint: *range* | *enum* | *glob*	no	Unique
`value`	TEXT	Specified case sensitive value for *enum* or *glob* or NULL for *range* constraint_type	yes	Unique
`min`	NUMERIC	Minimum value for ‘range’ or NULL for ‘enum’ or ‘glob’ constraint_type	yes	
`min_is_inclusive`	BOOLEAN	0 (false) if min value is exclusive, or 1 (true) if min value is inclusive	yes	
`max`	NUMERIC	Maximum value for ‘range’ or NULL for ‘enum’ or ‘glob’ constraint_type	yes	
`max_is_inclusive`	BOOLEAN	0 (false) if max value is exclusive, or 1 (true) if max value is inclusive	yes	
`description`	TEXT	For ranges and globs, describes the constraint; for enums, describes the enum value.	yes	

The min and max columns are defined as NUMERIC to be able to contain range values for any numeric data column defined with a data type from Table 1. These are the only exceptions to the data type rule stated in Req 5.

See gpkg_data_columns Table Definition SQL.

Table Data Values

The lowercase gpkg_data_column_constraints constraint_type column value specifies the type of constraint: "range", "enum", or "glob" (GLOB is a text pattern match - see [33]). The case sensitive value column contains an enumerated legal value for constraint_type "enum", a pattern match string for constraint_type "glob", or NULL for constraint_type "range". The set of value column values in rows of constraint_type "enum" with the same constraint_name contains all possible enumerated values for the constraint name. The min and max column values specify the minim and maximum valid values for constraint_type "range", or are NULL for constraint_type "enum" or "glob". The min_is_inclusive and max_is_inclusive column values contain 1 if the min and max values are inclusive, 0 if they are exclusive, or are NULL for constraint_type "enum" or "glob". These restrictions MAY be enforced by SQL triggers or by code in applications that update GeoPackage data values.

Table 26. Sample Data Column Constraints

constraint_name	constraint_type	value	min	min_is_inclusive	max	max_is_inclusive
sampleRange	range	NULL	1	true	10	true
sampleEnum	enum	1	NULL	NULL	NULL	NULL
sampleEnum	enum	3	NULL	NULL	NULL	NULL
sampleEnum	enum	5	NULL	NULL	NULL	NULL
sampleEnum	enum	7	NULL	NULL	NULL	NULL
sampleEnum	enum	9	NULL	NULL	NULL	NULL
sampleGlob	glob	[1-2][0-9][0-9][0-9]	NULL	NULL	NULL	NULL

Requirement 95

The `gpkg_data_column_constraints` table MAY be empty. If it contains data, the lowercase `constraint_type` column values SHALL be one of "range", "enum", or "glob".

Requirement 96

gpkg_data_column_constraint constraint_name values for rows with constraint_type values of *range* and *glob* SHALL be unique.

Requirement 97

The `gpkg_data_column_constraints` table MAY be empty. If it contains rows with constraint_type column values of "range", the `value` column values for those rows SHALL be NULL.

Requirement 98

The `gpkg_data_column_constraints` table MAY be empty. If it contains rows with `constraint_type` column values of "range", the `min` column values for those rows SHALL be NOT NULL and less than the `max` column value which shall be NOT NULL.

Requirement 99

The `gpkg_data_column_constraints` table MAY be empty. If it contains rows with `constraint_type` column values of "range", the `min_is_inclusive` and `max_is_inclusive` column values for those rows SHALL be 0 or 1.

Requirement 100

The `gpkg_data_column_constraints` table MAY be empty. If it contains rows with `constraint_type` column values of "enum" or "glob", the `min`, `max`, `min_is_inclusive` and `max_is_inclusive` column values for those rows SHALL be NULL.

Requirement 101

The `gpkg_data_column_constraints` table MAY be empty. If it contains rows with `constraint_type` column values of "enum" or "glob", the `value` column SHALL NOT be NULL.

Abstract Test Suite

Data Columns

Table Definition

Test Case ID	`/opt/schema/data_columns/data/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 Table 42. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant. 4. Fail otherwise.
Reference	Annex F.9 Req 103
Test Type	Basic

Data Values

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	Annex F.9 Req 104
Test Type	Capability

Test Case ID	`/opt/schema/data_columns/data/data_values_constraint_name`
Test Purpose	Verify that for each gpkg_data_columns row, the constraint_name value is either NULL or a constraint_name column value from the gpkg_data_column_constraints table.
Test Method	1. SELECT constraint_name AS cn FROM gpkg_data_columns 2. Not testable if returns an empty result set 3. For each NOT NULL cn value from step 1 a. SELECT constraint_name FROM gpkg_data_column_constraints WHERE constraint_name = cn b. Fail if returns an empty result set 4. Pass if no fails
Reference	Annex F.9 Req 105
Test Type	Capability

Test Case ID	`/opt/schema/data_columns/data/data_values_constraint_type`
Test Purpose	Verify that for each gpkg_data_columns row, if the constraint_name value is NOT NULL then the constraint_type column value contains a constraint_type column value from the gpkg_data_column_constraints table for a row with a matching constraint_name value.
Test Method	1. SELECT constraint_name AS cn, constraint_type AS ct FROM gpkg_data_columns 2. Not testable if returns an empty result set 3. For each NOT NULL cn value from step 1 a. Fail if ct is NULL b. If ct NOT NULL, SELECT constraint_type FROM gpkg_data_column_constraints WHERE constraint_name = cn AND constraint_type = ct c. Fail if returns an empty result set 4. Pass if no fails
Reference	Annex F.9 Req 106
Test Type	Capability

Data Column Constraints

Table Definition

Test Case ID	`/opt/schema/data_column_constraints/data/table_def`

Test Purpose	Verify that the gpkg_data_column_constraints table exists and has the correct definition.
Test Method	1. SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name = 'gpkg_data_column_constraints' 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 Table 43. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant. 4. Fail otherwise.
Reference	Annex F.9 Req 107
Test Type	Basic

Data Values

Test Case ID	`/opt/schema/data_column_constraints/data/data_values_constraint_type`
Test Purpose	Verify that the gpkg_data_column_constraints constraint_type column values are one of "range", "enum", or "glob".
Test Method	1. SELECT constraint_type AS ct FROM gpkg_data column_constraints 2. Not testable if returns an empty result set 3. For each ct value returned by step 1 a. Fail if ct NOT IN ("range", "enum", "glob"). 4. Pass if no fails.
Reference	Annex F.9 Req 108
Test Type	Capability

Test Case ID	`/opt/schema/data_column_constraints/data/data_values_constraint_names_unique`
Test Purpose	Verify that the gpkg_data_column_constraints constraint_name column values for constraint_type values of "range", or "glob" are unique.
Test Method	1. For each SELECT DISTINCT constraint_name AS cn FROM gpkg_data_column_constraints WHERE constraint_type IN (‘range’, ‘glob’) a. SELECT count(*) FROM gpkg_data column_constraints WHERE constraint_name = cn b. Fail if count > 1 2. Pass if no fails.
Reference	Annex F.9 Req 109
Test Type	Capability

Test Case ID	`/opt/schema/data_column_constraints/data/data_values_value_for_range`
Test Purpose	Verify that the gpkg_data_column_constraints value column values are NULL for rows with a constraint_type value of "range".

Test Method	1. SELECT constraint_type AS ct, value AS v FROM gpkg_data column_constraints WHERE constraint_type = ‘range’ 2. Not testable if returns an empty result set 3. For each v value returned by step 1 a. Fail if v IS NOT NULL 4. Pass if no fails.
Reference	Annex F.9 Req 110
Test Type	Capability

Test Case ID	`/opt/schema/data_column_constraints/data/data_values_min_max_for_range`
Test Purpose	Verify that the gpkg_data_column_constraints min column values are NOT NULL and less than the max column values for rows with a constraint_type value of "range".
Test Method	1. SELECT min, max FROM gpkg_data column_constraints WHERE constraint_type = ‘range’ 2. Not testable if returns an empty result set 3. For each set of min and max values returned by step 1 a. Fail if min IS NULL b. Fail if max IS NULL c. Fail if min >= max 4. Pass if no fails.
Reference	Annex F.9 Req 111
Test Type	Capability

Test Case ID	`/opt/schema/data_column_constraints/data/data_values_inclusive_for_range`
Test Purpose	Verify that the gpkg_data_column_constraints min_is_inclusive and max_is_inclusive column values are NOT NULL and either 0 or 1 for rows with a constraint_type value of "range".
Test Method	1. SELECT min_is_inclusive, max_is_inclusive FROM gpkg_data column_constraints WHERE constraint_type = ‘range’ 2. Not testable if returns an empty result set 3. For each set of values returned by step 1 a. Fail if min_is_inclusive IS NULL b. Fail if max_is_inclusive IS NULL c. Fail if min_is_inclusive is NOT IN (0,1) d. Fail if max_is_inclusive is NOT IN (0,1) 4. Pass if no fails.
Reference	Annex F.9 Req 112
Test Type	Capability

Test Case ID:	`/opt/schema/data_column_constraints/data/data_values_min_max_inclusive_for_enum_glob`
Test Purpose:	Verify that the gpkg_data_column_constraints min, max, min_is_inclusive and max_is_inclusive column values are NULL for rows with a constraint_type value of "enum" or “glob”.

Test Method:	1. SELECT min, max, min_is_inclusive, max_is_inclusive FROM gpkg_data column_constraints WHERE constraint_type IN (‘enum’,’glob) 2. Not testable if returns an empty result set 3. For each set of values returned by step 1 a. Fail if min IS NOT NULL b. Fail if max IS NOT NULL c. Fail if min_is_inclusive IS NOT NULL d. Fail if max_is_inclusive IS NOT NULL 4. Pass if no fails.
Reference	Annex F.9 Req 113
Test Type	Capability

Test Case ID:	`/opt/schema/data_column_constraints/data/data_values_value_for_enum_glob`
Test Purpose:	Verify that the gpkg_data_column_constraints value column values are NOT NULL for rows with a constraint_type value of "enum" or “glob”.
Test Method:	1. SELECT value FROM gpkg_data column_constraints WHERE constraint_type IN (‘enum’,’glob) 2. Not testable if returns an empty result set 3. For each value returned by step 1 a. Fail if value IS NULL 4. Pass if no fails.
Reference	Annex F.9 Req 114
Test Type	Capability

Table Definition SQL

gpkg_data_columns

gpkg_data_columns Table Definition SQL

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

gpkg_data_column_constraints

gpkg_data_columns Table Definition SQL

```
CREATE TABLE gpkg_data_column_constraints (
  constraint_name TEXT NOT NULL,
  constraint_type TEXT NOT NULL, // 'range' | 'enum' | 'glob'
  value TEXT,
  min NUMERIC,
  min_is_inclusive BOOLEAN, // 0 = false, 1 = true
  max NUMERIC,
  max_is_inclusive BOOLEAN, // 0 = false, 1 = true
  description TEXT,
  CONSTRAINT gdcc_ntv UNIQUE (constraint_name, constraint_type, value)
)
```

F.10. WKT for Coordinate Reference Systems

Introduction

The OGC GeoPackage standard was adopted prior to the adoption of "OGC Well known text representation of Coordinate Reference Systems" [34], in 13 August, 2014. As a result, the OGC GeoPackage standard references an older document [32] which has known ambiguities that are being encountered in the field. This extension establishes a new column to contain values that conform to the new standard.

Extension Author

GeoPackage SWG, author_name `gpkg`.

Extension Name or Template

`gpkg_crs_wkt`

Extension Type

Extension of Existing Requirement in clause Table Definition.

Applicability

Applies to the `gpkg_spatial_ref_sys` table.

Scope

Read-write

Requirements

Table

Table Definition

> **Requirement 102**
>
> The `gpkg_spatial_ref_sys` table in a GeoPackage SHALL have an additional column called `definition_12_063` as per Spatial Ref Sys Table Definition and gpkg_spatial_ref_sys Table Definition SQL (CRS WKT Extension).

Table 27. Spatial Ref Sys Table Definition

Column Name	Column Type	Column Description	Null	Key
`srs_name`	TEXT	Human readable name of this SRS	no	
`srs_id`	INTEGER	Unique identifier for each Spatial Reference System within a GeoPackage	no	PK
`organization`	TEXT	Case-insensitive name of the defining organization e.g. EPSG or epsg	no	
`organization_coordsys_id`	INTEGER	Numeric ID of the Spatial Reference System assigned by the organization	no	
`definition`	TEXT	Well-known Text [32] Representation of the Spatial Reference System	no	

Column Name	Column Type	Column Description	Null	Key
`description`	TEXT	Human readable description of this SRS	yes	
`definition_12_063`	TEXT	Well-known Text [34] Representation of the Spatial Reference System	no	

Table Data Values

Requirement 103

Values of the `definition_12_063` column SHALL be constructed per the WKT syntax in [34].

Requirement 104

At least one definition column SHALL be defined with a valid definition unless the value of the `srs_id` column is `0` or `-1`. Both columns SHOULD be defined. If it is not possible to produce a valid [32] definition then the value of the `definition` column MAY be `undefined`. If it is not possible to produce a valid [34] definition then the value of the `definition_12_063` column MAY be `undefined`.

Abstract Test Suite

Table Definition

Table Definition

Test Case ID	`/extension_crs_wkt/table_def`
Test Purpose	Verify that the gpkg_spatial_ref_sys table exists and has the correct definition. Extends `/base/core/gpkg_spatial_ref_sys/data/table_def`.
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, and primary key constraints match all of those in the contents of Annex F.10 Table 47. Column order, check constraint and trigger definitions, and other column definitions in the returned sql are irrelevant. 4. Fail otherwise.
Reference	Annex F.10 Req 115
Test Type	Basic

Table Data Values

Test Case ID	`/extension_crs_wkt/data_values_default`
Test Purpose	Verify that the gpkg_spatial_ref_sys table contains the required default contents. Extends `/base/core/gpkg_spatial_ref_sys/dat`

Test Method	1. `SELECT srs_id, organization, organization_coordsys_id, definition, definition_12_063 FROM gpkg_spatial_` 2. `SELECT srs_id, organization, organization_coordsys_id, definition, definition_12_063 FROM gpkg_spatial_` 3. `SELECT definition FROM gpkg_spatial_ref_sys WHERE organization IN (“epsg”,”EPSG”) AND organization_coor` SPHEROID["WGS 84", 6378137, 298.257223563 , AUTHORITY["EPSG","7030"]], AUTHORITY["EPSG","6326"]], PRIMEM["Greenv AUTHORITY["EPSG","9102"]], AUTHORITY["EPSG","4326"] (rounding the UNIT conversion factors to 16 decimal places, and igno in the returned text) 4. `SELECT definition_12_063 FROM gpkg_spatial_ref_sys WHERE organization IN (“epsg”,”EPSG”) AND organizati` 1984", ELLIPSOID["WGS 84",6378137, 298.257223563, LENGTHUNIT["metre", 1.0]]], PRIMEM["Greenwich", 0.0, ANGLEUNIT["de AXIS["longitude", east, ORDER[2]], ANGLEUNIT["degree", 0.0174532925199433], ID["EPSG", 4326]] 5. Pass if tests 1-4 are met 6. Fail otherwise
Reference	Annex F.10 Req 116
Test Type	Capability

Test Case ID	`/extension_crs_wkt/data_values_undefined`
Test Purpose	Verify that the gpkg_spatial_ref_sys table contains the required default contents. Replaces `/base/core/gpkg_spatial_ref_sys/data_values_default`.
Test Method	1. `SELECT definition, definition_12_063 FROM gpkg_spatial_ref_sys WHERE srs_id NOT IN (0, -1)` 2. Fail if both definition values are `undefined` 3. Pass otherwise
Reference	Annex F.10 Req 117
Test Type	Capability

Table Definition SQL

gpkg_spatial_ref_sys

gpkg_spatial_ref_sys Table Definition SQL (CRS WKT Extension)

```
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,
  definition_12_063 TEXT NOT NULL
);
```

F.11. Tiled Gridded Elevation Data

Extension Title

Tiled Gridded Elevation Data

Introduction

The GeoPackage Standards Working Group (SWG) has developed the ability to store 16-bit and 32-bit tiled gridded elevation data in a GeoPackage. This capability will be used to support use cases such as the following:

- Visualization
  - 2D (hillshade, color relief, slope)
  - 3D (supporting changing view angles and level of detail)

- Analysis
  - Viewshed and line-of-sight
  - Cross-country mobility (off-road routing)
  - Site suitability and planning (slope analysis such as helicopter landing zones)
  - 3D geometry representations of features (ground-based, airspace)
  - Terrain association (associating images to mapped locations)
  - Augmented reality training

This capability was designed to be relatively easy to implement and to be suitable for a wide variety of computing environments including the mobile/handheld computing environment[39].

This extension to the OGC GeoPackage Encoding Standard leverages the existing structure for raster tiles using PNG (16-bit) and TIFF (32-bit) files as the container for the elevation values themselves. It is the data producer's responsibility to ensure that the intended recipient is able to read GeoPackages with the elevation encoding (PNG or TIFF) being used.

Extension Author

GeoPackage SWG, author_name `gpkg`.

Extension Name or Template

`gpkg_elevation_tiles`

Extension Type

New requirement dependent on Clause 2.2 (http://www.geopackage.org/spec/#tiles).

Applicability

This extension applies to tile pyramid user data tables (http://www.geopackage.org/spec/#tiles_user_tables) that are used to hold tiled, gridded elevation data.

Scope

read-write

Requirements

Table Definitions

Coverage Ancillary

> **Requirement 105**
>
> A GeoPackage that contains tiled gridded elevation data SHALL contain a `gpkg_2d_gridded_coverage_ancillary` table or view as per Coverage Ancillary Table Definition. Subsequent extensions or custom implementations MAY add additional columns to this table. Clients SHALL ignore additional columns that are unrecognized.

Table 28. Coverage Ancillary Table Definition

Column Name	Column Type	Column Description	Null	Default	Key	Cons
`id`	INTEGER	Autoincrement primary key	no		PK	
`tile_matrix_set_name`	TEXT	Foreign key to `table_name` in `gpkg_tile_matrix_set` (http://www.geopackage.org/spec/#tile_matrix_set_data_table_definition)	no		FK	UNIQ
`datatype`	TEXT	*integer* or *float*	no	*integer*		
`scale`	REAL	Scale as a multiple relative to the unit of measure	yes	1		
`offset`	REAL	The offset to the 0 value	yes	0		
`precision`	REAL[40]	The smallest value that has meaning for this dataset	yes	1		
`data_null`	REAL[41]	The value that indicates NULL	yes			

Tile Ancillary

> **Requirement 106**
>
> A GeoPackage that contains tiled gridded elevation data SHALL contain a `gpkg_2d_gridded_tile_ancillary` table or view as per Tile Ancillary Table Definition. Subsequent extensions or custom implementations MAY add additional columns to this table. Clients SHALL ignore additional columns that are unrecognized.

Table 29. Tile Ancillary Table Definition

Column Name	Column Type	Column Description	Null	Default	Key
`id`	INTEGER	Autoincrement primary key	no		PK
`tpudt_name`	TEXT	Name of tile pyramid user data table (http://www.geopackage.org/spec/#tiles_user_tables)	no		UNIQUE[42]
`tpudt_id`	INTEGER	Foreign key to `id` in tile pyramid user data table (http://www.geopackage.org/spec/#tiles_user_tables)	no		UNIQUE[43]
`scale`	REAL	Scale as a multiple relative to the unit of measure	yes	1	
`offset`	REAL	The offset to the 0 value	yes	0	
`min`	REAL[44]	Minimum value of this tile	yes		
`max`	REAL[45]	Maximum value of this tile	yes		
`mean`	REAL	The arithmetic mean of values in this tile	yes		
`std_dev`	REAL	The standard deviation of values in this tile	yes		

Table Values

gpkg_spatial_ref_sys

> **Requirement 107**
>
> GeoPackages complying with this extension SHALL have a row in the `gpkg_spatial_ref_sys` table as described in Spatial Ref Sys Table Record:

Table 30. Spatial Ref Sys Table Record

`srs_name`	`srs_id`	`organization`	`organization_coordsys_id`	`definition`	`description`
any	`4979`	`EPSG` or `epsg`	`4979`	any	any

> **Requirement 108**
>
> The `gpkg_spatial_ref_sys` table in a GeoPackage SHALL contain records to define all spatial reference systems used by tiled gridded elevation data in a GeoPackage. The spatial reference system SHALL be used to define the vertical datum, reference geoid, and units of measure for the tiled gridded elevation data.

gpkg_contents

> **Requirement 109**
>
> (extends GPKG-34 (http://www.geopackage.org/spec/#_requirement-34)) The `gpkg_contents` (http://www.geopackage.org/spec/#_contents) table SHALL contain a row with a `data_type` column value of *2d-gridded-coverage* for each tile pyramid containing tiled gridded elevation data. The `srs_id` column value for that row SHOULD reference an SRS that has a vertical datum[46].

gpkg_extensions

> **Requirement 110**

GeoPackages complying with this extension SHALL have rows in the `gpkg_extensions` table as described in Extensions Table Record.

Table 31. Extensions Table Record

`table_name`	`column_name`	`extension_name`	`definition`
gpkg_2d_gridded_coverage_ancillary	null	`gpkg_elevation_tiles`	http://www.geopackage.org/spec/#extensio (http://www.geopackage.org/spec/#extensic
gpkg_2d_gridded_tile_ancillary	null	`gpkg_elevation_tiles`	http://www.geopackage.org/spec/#extensio (http://www.geopackage.org/spec/#extensic
name of actual tile pyramid user data table (http://www.geopackage.org/spec/#tiles_user_tables) containing elevation data	`tile_data`	`gpkg_elevation_tiles`	http://www.geopackage.org/spec/#extensio (http://www.geopackage.org/spec/#extensic

gpkg_2d_gridded_coverage_ancillary

The following requirements refer to the `gpkg_2d_gridded_coverage_ancillary` table as per Coverage Ancillary Table Definition.

> **Requirement 111**
>
> Values of the `tile_matrix_set_name` column SHALL reference values in the `gpkg_tile_matrix_set` `table_name` column.

> **Requirement 112**
>
> Values of the `datatype` column MAY be *integer* or *float*.

gpkg_2d_gridded_tile_ancillary

The following requirements refer to the `gpkg_2d_gridded_tile_ancillary` table as per Tile Ancillary Table Definition.

> **Requirement 113**
>
> Values of the `tpudt_name` column SHALL reference existing tile pyramid user data tables (http://www.geopackage.org/spec/#tiles_user_tables).

> **Requirement 114**
>
> Values of the `tpudt_id` column SHALL reference values in `id` column of the table referenced in `tpudt_name`.

> **Requirement 115**
>
> The `min`, `max`, and `mean` values SHALL be natural, i.e., not scaled or offset. Similarly, the `std_dev` SHALL be calculated based on the natural values.

Using the Scale and Offset Values

> **Requirement 116**
>
> Integer elevation values MAY be scaled and offset in order to make more efficient use of 16-bit integer space available in PNG files. The scale and offset MAY be applied to the entire coverage and/or the individual tile. The scale and offset do not apply to the `data_null` value as defined in Coverage Ancillary Table Definition.

Actual elevation values SHALL be calculated by:

- first multiplying the stored value by the `gpkg_2d_gridded_tile_ancillary_table.scale` value and then adding the `gpkg_2d_gridded_tile_ancillary_table.offset`,
- followed by multiplying that value by the `gpkg_2d_gridded_coverage_ancillary.scale` value and then adding the `gpkg_2d_gridded_coverage_ancillary.offset`.

In pseudo-code, this conversion would look like:

```
elevationInUnitOfMeasure = (SomeElevationCoverage.tile_data->pngpixels[i] *
gpkg_2d_gridded_tile_ancillary.scale + gpkg_2d_gridded_tile_ancillary.offset) *
gpkg_2d_gridded_coverage_ancillary.scale + gpkg_2d_gridded_coverage_ancillary.offset;
```

Requirement 117

Floating point elevation values SHALL NOT have a scale and offset applied. The `scale` and `offset` values SHALL be null.

Tile Pyramid User Data Tables

Requirement 118

For data where the `datatype` column of the corresponding row in the `gpkg_2d_gridded_coverage_ancillary` table is *integer*, the `tile_data` BLOB in the tile pyramid user data table (http://www.geopackage.org/spec/#tiles_user_tables) containing tiled, gridded elevation data SHALL be of MIME type `image/png` and the data SHALL be 16-bit unsigned integer (single channel - "greyscale").

Requirement 119

For data where the `datatype` column of the corresponding row in the `gpkg_2d_gridded_coverage_ancillary` table is *float*, the `tile_data` BLOB in the tile pyramid user data table (http://www.geopackage.org/spec/#tiles_user_tables) containing tiled, gridded elevation data SHALL be of MIME type `image/tiff` and the data SHALL be 32-bit floating point as described by the TIFF Encoding (Requirement 120).

Table Definition SQL

Coverage Ancillary Table Definition SQL

```
CREATE TABLE 'gpkg_2d_gridded_coverage_ancillary' (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  tile_matrix_set_name TEXT NOT NULL UNIQUE,
  datatype TEXT NOT NULL DEFAULT 'integer',
  scale REAL DEFAULT 1.0,
  offset REAL DEFAULT 0.0,
  precision REAL DEFAULT 1.0,
  data_null REAL,
  CONSTRAINT fk_g2dgtct_name FOREIGN KEY('tile_matrix_set_name') REFERENCES gpkg_tile_matrix_set (
table_name )
  CHECK (datatype in ('integer','float')));
```

Tile Ancillary Table Definition SQL

```
CREATE TABLE gpkg_2d_gridded_tile_ancillary (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  tpudt_name TEXT NOT NULL,
  tpudt_id INTEGER NOT NULL,
  scale REAL DEFAULT 1.0,
  offset REAL DEFAULT 0.0,
  min REAL DEFAULT NULL,
  max REAL DEFAULT NULL,
  mean REAL DEFAULT NULL,
  std_dev REAL DEFAULT NULL,
  CONSTRAINT fk_g2dgtat_name FOREIGN KEY (tpudt_name) REFERENCES gpkg_contents(table_name),
  UNIQUE (tpudt_name, tpudt_id));
```

TIFF Encoding

Requirement 120

A TIFF file used for storing tiled gridded elevation data SHALL conform to the TIFF specification [35].

Requirement 121

(constrains TIFF[35] Section 2) A TIFF file storing tiled gridded elevation data SHALL have one sample per pixel.

Requirement 122

(constrains TIFF[35] Section 2) A TIFF file storing tiled gridded elevation data SHALL have the 32-bit floating (FLOAT – 11) data type.

Requirement 123

A TIFF file storing tiled gridded elevation data MAY use the LZW compression option as per TIFF[35] Section 13.

Client applications that support the TIFF encoding are expected to support this option.

Requirement 124

(constrains TIFF[35] Section 2) A TIFF file storing tiled gridded elevation data SHALL NOT contain multiple images per TIFF file.

Requirement 125

(constrains TIFF[35] Section 15) A TIFF file storing tiled gridded elevation data SHALL NOT contain internal tiles as per TIFF Section 15.

Annex G: Geometry Types (Normative)

Table 32. Geometry Type Codes (Core)

Code	Name
0	GEOMETRY
1	POINT
2	LINESTRING
3	POLYGON
4	MULTIPOINT
5	MULTILINESTRING
6	MULTIPOLYGON
7	GEOMETRYCOLLECTION

Table 33. Geometry Type Codes (Extension)

Code	Name
8	CIRCULARSTRING
9	COMPOUNDCURVE
10	CURVEPOLYGON
11	MULTICURVE
12	MULTISURFACE
13	CURVE
14	SURFACE

GEOMETRY subtypes are POINT, CURVE, SURFACE and GEOMCOLLECTION.

CURVE subtypes are LINESTRING, CIRCULARSTRING and COMPOUNDCURVE.

SURFACE subtype is CURVEPOLYGON.

CURVEPOLYGON subtype is POLYGON.

GEOMETRYCOLLECTION subtypes are MULTIPOINT, MULTICURVE and MULTISURFACE.

MULTICURVE subtype is MULTILINESTRING.

MULTISURFACE subtype is MULTIPOLYGON.

Annex H: Tiles Zoom Times Two Example (Informative)

Table 34. Zoom Times Two Example

table_name	zoom_level	matrix_width	matrix_height	tile_width	tile_height	pixel_x_size	pixel_y_size
MyTiles	0	8	8	512	512	69237.2	68412.1
MyTiles	1	16	16	512	512	34618.6	34206.0
MyTiles	2	32	32	512	512	17309.3	17103.0
MyTiles	3	64	64	512	512	8654.64	8654.64
MyTiles	4	128	128	512	512	4327.32	4275.75
MyTiles	5	256	256	512	512	2163.66	2137.87
MyTiles	6	512	512	512	512	1081.83	1068.93
MyTiles	7	1024	1024	512	512	540.915	543.469
MyTiles	8	2048	2048	512	512	270.457	267.234

Annex I: 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.

- [1] ISO/IEC 9075:1992 Information Technology - Database Language SQL (SQL92)
- [2] ISO/IEC 9075-1:2011 Information Technology - Database Language SQL - Part 1: Framework
- [3] ISO/IEC 9075-2:2011 Information Technology - Database Language SQL - Part 2: Foundation
- [4] ISO/IEC 9075-3:2008 Information Technology - Database Language SQL - Part 3: Call- Level Interface (SQL/CLI)
- [5] SQLite (all parts) http://www.sqlite.org/ (http://www.sqlite.org/) (online) http://www.sqlite.org/sqlite-doc-3071300.zip (http://www.sqlite.org/sqlite-doc-3071300.zip) (offline)
- [6] http://sqlite.org/fileformat2.html (http://sqlite.org/fileformat2.html)
- [7] http://www.sqlite.org/formatchng.html (http://www.sqlite.org/formatchng.html)
- [8] http://www.sqlite.org/download.html (http://www.sqlite.org/download.html)
- [9] OGC 06-103r4 OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture Version: 1.2.1 2011-05-28 http://portal.opengeospatial.org/files/?artifact_id=25355 (http://portal.opengeospatial.org/files/?artifact_id=25355) (also ISO/TC211 19125 Part 1)
- [10] OGC 06-104r4 OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option Version: 1.2.1 2010-08-04 http://portal.opengeospatial.org/files/?artifact_id=25354 (http://portal.opengeospatial.org/files/?artifact_id=25354) (also ISO/TC211 19125 Part 2)
- [11] OGC 99-049 OpenGIS® Simple Features Specification for SQL Revision 1.1 5, 1999, Clause 2.3.8 http://portal.opengeospatial.org/files/?artifact_id=829 (http://portal.opengeospatial.org/files/?artifact_id=829) May