OGC CDB Vector Data in GeoPackage Interoperability Experiment

Table of Contents

1. Summary . . . 4
   1.1. Requirements & Research Motivation . . . 4
   1.2. Prior-After Comparison . . . 4
   1.3. Recommendations for Future Work . . . 4
   1.4. Document contributor contact points . . . 5
   1.5. Foreword . . . 6
2. References . . . 7
3. Terms and definitions . . . 8
   3.1. Abbreviated terms . . . 8
4. Overview . . . 9
   4.1. IE Overview . . . 9
   4.2. Engineering Report Section Overviews. . . . 9
5. Details of the Experiments Performed . . . 10
   5.1. A CDB Data Store - General Overview . . . 10
   5.2. CDB Data and Tools Used By Participants . . . 11
      5.2.1. Overview . . . 11
      5.2.2. Experiment 1: Conversion - The CDB Data Stores and tools used in this IE . . . 11
      5.2.3. Key discussion topics related to CDB data stores . . . 13
   5.3. Details Related to Experiment 2 . . . 13
      5.3.1. Option 1a – 1:1 Conversion of Shapefiles to GeoPackages . . . 13
      5.3.2. Option 1b – Conversion of Shapefiles to GeoPackages using Normalized SQL Data . . . 14
      5.3.3. Option 1c – Flattened Attribution . . . 15
      5.3.4. Option 1d – Flattened Attribution + extensions . . . 15
   5.4. Experiment 3 - Each CDB LoD as a layer in GeoPackage . . . 16
   5.5. Experiment 4: Store each Geocell of Vector Data as a layer in GeoPackage . . . 16
6. IE Experiment Results . . . 18
   6.1. Aechelon Technology IE Report . . . 18
      6.1.1. Use case and experiment focus . . . 18
      6.1.2. Aechelon Experiment Methodology . . . 18
      6.1.3. Metrics . . . 19
      6.1.4. Legend . . . 21
      6.1.5. Notes and observations . . . 21
   6.2. CAE Results for Experiment 2 . . . 22
      6.2.1. Focus of the Experiment . . . 22
      6.2.2. Comparing File Formats . . . 22
      6.2.3. Modifications to the GDAL/OGR Library . . . 23
      6.2.4. Converting a Full CDB . . . 24
      6.2.5. Network Test . . . 24

6.2.6. Real-Time CDB Client Device . . . 25
6.3. Compusult Results from Experiment 2 . . . 26
6.4. FlightSafety Experiment Results . . . 26
6.4.1. FlightSafety International's Use Case for CDB . . . 26
6.4.2. FlightSafety Experiment Focus . . . 26
6.4.3. FlightSafety Experiment Methodology . . . 27
6.4.4. FlightSafety Metrics . . . 28
6.4.5. Shapefile vs. GeoPackage Option 1 (Experiment 2) Testing . . . 29
6.4.6. GeoPackage Option 3 and 4 Testing . . . 32
6.4.7. Further GeoPackage Option 3 & 4 Testing . . . 35
6.5. Hexagon US Federal Technology Experiment Report . . . 36
6.5.1. Experiment Methodology - Dataset Conversion . . . 37
6.5.2. Experiment Methodology - Visualization . . . 37
6.5.3. Metrics . . . 38
6.5.4. Notes on Metrics . . . 38
6.5.5. Notes and Observations . . . 39
6.6. Guidance . . . 39
7. Recommendations, Observations, and Conclusions . . . 41
7.1. Backwards compatibility in the CDB context . . . 41
7.1.1. FlightSafety observations on backwards compatibility . . . 41
7.2. Findings . . . 41
7.2.1. Aechelon Findings . . . 41
7.2.2. CAE Findings . . . 41
7.2.3. FlightSafety Findings . . . 42
7.2.4. Hexagon/Luciad Findings . . . 43
7.3. Recommendations . . . 43
7.3.1. Aechelon . . . 43
7.3.2. CAE Recommendations . . . 43
7.3.3. FlightSafety Recommendations . . . 44
7.3.4. Hexagon/Luciad Recommendation . . . 45
Appendix A: Abstract Test Suite . . . 46
A.1. Test module for conformance level 1 . . . 46
A.1.1. Conformance level 1 . . . 46
A.1.2. Test case for validity of XML response entity . . . 46
A.2. Test module for conformance level 2 . . . 47
A.2.1. Conformance level 2 . . . 47
A.2.2. Test case for validity of XML response entity . . . 47
Appendix B: Revision History . . . 48
Appendix C: Bibliography . . . 49

Publication Date: 2019-MM-DD

Approval Date: 2019-June-27

Submission Date: 2019-June-5

Reference number of this document: OGC 19-007

Reference URL for this document: http://www.opengis.net/doc/PER/CDB-GPKG-IE

Category: OGC Public Engineering Report

Editor: Carl Reed, PhD

Title: OGC CDB Vector Data in GeoPackage Interoperability Experiment

---

OGC Public Engineering Report

COPYRIGHT

Copyright © 2019 Open Geospatial Consortium. To obtain additional rights of use, visit http://www.opengeospatial.org/

WARNING

This document is not an OGC Standard. This document is an Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. This Engineering Report is distributed for review and comment. The content is subject to change without notice and may not be referred to as an OGC Standard. Further, any Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

LICENSE AGREEMENT

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

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.

Chapter 1. Summary

This OGC Engineering Report (ER) documents the results of the CDB Vector Data in GeoPackage Interoperability Experiment (IE). The participants in this IE tested transforming CDB Shapefile vector data into one or more GeoPackage(s) and storing the result in a CDB data store. GeoPackage Version 1.2 and CDB Version 1.1 and related Best Practices were the standards baseline used for this experiment. The IE builds on the work described in the OGC CDB, Leveraging GeoPackage Discussion Paper [https://portal.opengeospatial.org/files/?artifact_id=82553].

A primary objective of this IE was to agree and document possible change requests and/or best practices for storing vector data in a CDB data using encodings and/or containers other than Shapefiles. These suggested changes requests and/or best/practices will be used as the basis for SDB SWG discusions related to possible revisions to the CDB standard.

While there are a number of issues, such as file path naming conventions that need to considered and agreed to by the SWG, the particpant consensus is that the GeoPackage identified and tested in Details Related to Experiment 2.

Please note that a reader of this Engineering Report has a basic understanding of both the CDB storage model and the GeoPackage Standard.

1.1. Requirements & Research Motivation

In the commercial modelling and simulation industry, geospatial data content generation and use is exploding at a rate that is outpacing the innovation and utilization of the traditional Modelling and Simulation (M&S) industry. At the same time, the M&S and geospatial-intelligence (GEOINT) industries are on a path to convergence. Within the OGC, there are two geospatial standards that best enable the unification of the M&S and GEOINT industries: OGC CDB and GeoPackage. OGC CDB and GeoPackage are both standards increasingly used in M&S and GEOINT industry. However, they both contain weaknesses and strengths when it comes to the combined needs of both industries.

1.2. Prior-After Comparison

This IE is based on preliminary work conducted under funding by the U.S. Army Geospatial Center (AGC). The results of that work is publicly released as OGC Document Number 18-077: OGC CDB and GeoPackage Discussion Paper [https://portal.opengeospatial.org/files/?artifact_id=80537&version=1]. The Discussion Paper documents the results of research, design, and prototype efforts to present the OGC with an approach to creating “GeoCDB” — a technology mashing of GeoPackage and OGC CDB — as a deterministic repository of easily read data geospatial datasets suitable for storage, runtime access, and dissemination for live, virtual, constructive, gaming, and mission command (MC) systems.

1.3. Recommendations for Future Work

This Interoperability Experiment addressed the specific issue of how best to and develop guidance for storing GeoPackage containers in a CDB datastore. AS a result of the experiments conducted in this IE activity, the particants have identifed a number of other potential interoperability

experiments that could be conducted in the future.

This current and potential future IEs will provide highly relevant input into the specification and development of CDB Version 2.0.

1.4. Document contributor contact points

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

Contacts

Role	Name	Organization
Initiator/Lead	David Graham	CAE
Technical Lead	Carl Reed	Carl Reed & Associates
editor	Carl Reed	Carl Reed & Associates
Initiator	Kevin Bentley	Cognitics
Initiator	Holly Black	CACI
Initiator	Hermann Brassard	Presagis
Initiator	Brian Ford	FlightSafety
Initiator	Ryan Franz	FlightSafety
Initiator	Jay Freeman	CAE
Initiator	Glen Johnson	VATC
Initiator	Mike Lokuta	CAE
Initiator	Bernard Leclerc	CAE
Initiator	Lance Marrou	Leidos
Initiator	Earl Miller	USSOCOM
Initiator	Ronald Moore	Leidos
Initiator	Susan Raymie	USSOCOM
Initiator	Sara Saeedi	U of Calgary
Participant	Jordan Dauble	Simblocks.io
Participant	Paul Foley	KaDSci
Participant	Priamos Georgiades	Aechelon
Participant	Robert Goff	Hexagon US Federal
Participant	Dave Lajoie	Presagis
Participant	Graham Long	Thales
Participant	Jason McDonald	Compusult
Participant	Nacho Sans-Pastor	Aechelon
Participant	Kathleen Ski	ISPA Technology
Participant	Trent Tinker	Luciad

1.5. Foreword

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.

Chapter 2. References

The following normative documents are referenced in this Engineering Report.

- OGC 15-113r5 [https://portal.opengeospatial.org/files/15-113r5], Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure
- OGC 16-070r3 [https://portal.opengeospatial.org/files/16-070r3], Volume 4: OGC CDB Best Practice use of Shapefiles for Vector Data Storage

The following informative documents are referenced in this Engineering Report.

- OGC 18-077r2, OGC CDB, Leveraging GeoPackage Discussion Paper [https://portal.opengeospatial.org/files/?artifact_id=82553]

Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

3.1. Abbreviated terms

The following abbreviated terms provides a list of the abbreviated terms and the symbols necessary for understanding this document.

```
CRS  Coordinate Reference System
GPKG GeoPackage
ER   Engineering Report
IE   Interoperability Experiment
IG   Image Generation
OGC  Open Geospatial Consortium
OGR  OGR Simple Features Library
OWS  OGC Web Services
SRS  Spatial Reference System
SWG  Standards Working Group
```

Chapter 4. Overview

4.1. IE Overview

The current approved OGC CDB 1.1 Standard provides a Best Practices document describing the encoding of vector data using Esri Shapefiles. All previous versions of CDB (including OGC CDB Standard 1.0 and the Industry-maintained CDB Specification 3.2 and prior) have only described vector data encoding in Shapefiles. All known CDB content and repositories that include vector data are encoded using Shapefiles. As a result, there have been numerous requests to test and document how other vector envodings could be stored in a CDB data store. This requirement is captured in OGC Change Request 545 [http://ogc.standardstracker.org/show_request.cgi?id=545]. Another identified issue is that every use of Shapefiles creates four physical files on storage. This has performance implications for such operations as transferring CDB structured content from one device to another.

The goal of this IE was achieved by replicating the experimentation conducted in the OGC CDB, Leveraging GeoPackage Discussion Paper. IE participants represented CDB content creators, maintainers of CDB repositories, implementers of CDB datasets in run-time and non-run-time CDB use cases, or new CDB 1.1 users who are evaluating CDB implementations.

The IE participants helped produce this OGC Engineering Report that discusses whether the recommended alternate GeoPackage encoding of CDB vector data is fit for use and submission to the OGC for consideration as an approved change to the OGC CDB 1.1 Standard and/or Best Practices.

4.2. Engineering Report Section Overviews.

Section 5 introduces the background and requirements for adding GeoPackages as a storage format in a CDB data store. It describes the situation prior to the Interoperability Experiment and discusses the requirements defined by the CDB user base.

Section 6 discusses the experiments in detail. The concept of operation also provides recommendations on preferred strategies.

Section 7 discusses the results of the various experiments performed as part of this initiative. A clear mapping of requirements for proposed GeoPackages in a CDB datastore candidate standard/best practice is provided.

Section 8 provides a summary of the main findings and discusses proposed change requests.

Annex A provides detailed performance information.

Chapter 5. Details of the Experiments Performed

This section details the experiments and particpant participation in each experiment. The experiments performed in this Interoperability Experiment included:

- Experiment 1: Conversion of Shapefiles in one or more CDB data stores into GeoPackages as required for Experiments 2, 3, and/or 4.
- Experiment 2: One to one conversion of a Shapefile into a GeoPackage. The four sub-options are described below.
- Experiment 3: Store each CDB LOD as a layer in GeoPackage.
- Experiment 4: Store each Geocell of Vector Data as a layer in GeoPackage.

The following table provides an overview of which experiments were executed by each participant.

Participant	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Aechelon	X	X	X	X
CAE	X	X		
Compusult	X	X	X	X
FlightSafety	X	X	X	X
Hexagon/Luciad	X	X	X	X
VATC	X	X		

5.1. A CDB Data Store - General Overview

A CDB structured data store is physically arranged on disk into the following top level directory structures:

- Metadata contains a set of XML metadata and controlled vocabulary files that are global to the data store.
- GTModel contains geotypical models, generic models that are defined once in the CDB and are intended to be rendered in multiple places throughout the data store (contrast with geospecific or GS models). They aren't intended to represent specific objects, but simply a typical representation of an object type such as a tree.
- MModel contains Moving Models, which don't have a fixed location and are intended to be dynamically placed and moved throughout a simulation. An example is an automobile or aircraft.
- Tiles contains tiled datasets. This is the structure in which the majority of vector based Shapefiles are stored.
- Navigation contains global navigation datasets.

A CDB data stored is structured into partitions that organizes the world into 1-degree by 1-degree

cells. Within each cell, the tiled datasets are organized into a Level of Detail (LOD) hierarchy. More specifically, the CDB storage model relies on three important means to organize the data: a) Tiles which organize the data into zones defined by its location with respect to a WGS84 reference system; b) Layers (or datasets) which organize different types of data in a tile; and c) Levels of Detail (LoD) which organize the data in each layer of each tile by its detail.

Most CDB datasets are organized in a tile structure and stored in the \CDB\Tiles\ directory. The tile structure facilitates access to the information in real-time by any runtime client devices. However, for some datasets that require minimal storage, such as, Moving Models or Geotypical Models, there is no significant advantage to be added for such a tile structure. Such datasets are referred to as global datasets and consist of data elements that are global to the earth.

Point, line, and area features are encoded in a CDB store as Shapefiles and are organized into several Vector Datasets and LoDs. For each LoD, the CDB standard specifies the maximum number of points allowed per Tile-LoD and the resulting average Feature Density is defined. All vector data in a CDB store are referenced to the WGS 84 Earth-centered, Earth-fixed terrestrial reference system and geodetic datum.

5.2. CDB Data and Tools Used By Participants

5.2.1. Overview

Three complete CDB data stores were provided for use in this Interoperability Experiment. The provision of CDB data stores and the subsequent conversion of Shapefiles into GeoPackages was defined in the IE Activity Plan in Experiment 1. The participants identified a number of activities associated with executing Experiment 1. A key objective of this experiment was to capture and compare metrics as well as to identify and document any differences in the Shapefile to GeoPackage conversions that resulted from the use of different conversion tools.

5.2.2. Experiment 1: Conversion - The CDB Data Stores and tools used in this IE

This section provides a short summary of the CDB data stores and tools used by each of the participants in this OGC Interoperability Experiment. Greater detail for each particpants work can be found in Section 6 (Results) of this report.

Aechelon

Aechelon used the following CDB data stores in their experiments:

- Yemen (4 geocells), from Presagis.
- Downtown Los Angeles (1 geocell), from VATC.
- Greater Los Angeles (4 geocells), from Cognitics.

Aechelon used the following tools in their experiments. The CDB data stores were used as the source for content to feed a publishing process into the Aechelon Image Generator (IG) runtime format. The publishing software is in python, with invocations of C++ EXEs for performance-critical processing. The feature scan step is entirely in Python, version 3.5. The changes implemented on

the publisher side to support GeoPackage were the minimal necessary to get functional parity with the Shapefile based implementation. In other words, no attempt was made to optimize the code to take advantage of the internals of the GeoPackage files using sqlite, and all data access went through the OGR module.

CAE Montreal

CAE did not use the sample CDB provided by the participants because their goal was to compare the performance of their internal applications when running with a CDB produced by CAE before and after the replacement of Shapefiles with GeoPackage files. However, they believe that our findings may apply equally well to other databases.

CAE used a slightly modified version of GDAL for converting Shapefiles to GeoPackages. See Section 6 Results for more detailed information.

Hexagon/Luciad

Hexagon/Luciad utilized the Yemen CDB data store. The tools used for all experiments are elements of Luciad CDB Studio.

The CDB Studio is a component based modular application developed using the Luciad Platform, a development platform for solving C4ISR needs in the Defense, Aviation and Maritime domains. The Luciad CDB Studio provides the ability to create OGC CDB data stores directly from imagery, elevation, and vector geospatial data products. The application also has strong support for GeoPackages. It can export any source, including CDB into GeoPackage imagery, elevation, or vector data. It also supports a GeoPackage styling extension and the ability to convert CDB 3D Models into OGC 3D Tiles stored in GeoPackage.

Presagis

Presagis provided a CDB data store for the country of Yemen.

VATC

VATC provided a CDB data store created from fully open data sources [1: For the IE, VATC made the data available at https://storage.cloud.google.com/epic_builder/OGC_IE/LosAngeles_CDB.zip?_ga=2.3746352.-1225582785.1543877247]. The data provided in the VATC CDB data store are in one CDB Geocell containing downtown Los Angeles.The data include:

- USGS NAIP 1m background (Entire Geocell) (CDB Lod 7)
- USGS HighRes Ortho Program 1ft Ortho (Southern section of Geocell) (CDB Lod 9)
- USGS 1/3 NED Elevation (CDB Lod 4 for an entire geocell)
- OpenStreetMap Vector Map (Entire geocell)
- Los Angeles County Building footprint information. (Approximately 1,734,043 buildings were extruded from the footprint data)

VATC used opensource lib's that were modified to support CDB (this includes OpenSceneGraph, osgEarth, and GDAL). They did not apply any changes to the GDAL library for this IE and the built

GDAL version from the 3rd Party is expected to suffice for this experiment.

5.2.3. Key discussion topics related to CDB data stores

Given the size of the CDB data stores used in this IE, there was discussion related to how best to provide data stores updated with GeoPackage content. The general concensus was that downloading the entire data store just to get the GeoPackages was non-optimal and time wasteful. Therefore, the particpants discussed using ***Version*** metadata (Volume 1 CDB Standard, Clause 5.1.8). They determined that the original CDB data with Shapefiles would be Version 1 and that CDB enhanced with GeoPackages would be Version 2.

5.3. Details Related to Experiment 2

Experiment 2 focused on approaches to replacing each Shapefile in an existing CDB data store thereby consolidating the three geometry files into a single GeoPackage. The objectives was to determine the best practices for not only replacing Shapefiles for also allowing the storage and use of both Shapefiles and Geopackages in a CDB data store. Part of this experiment was to also evaluate and compare performance using the baseline CDB datasets made available as part of Experiment 1. finally, this experiment also focused on evaluating and analyzing and results from Experiment #2 related to performance, backwards-compatibility and risks to interoperability.

The Participants identifed four possible approaches to converting and/or using GeoPackages in a CDB data store.

5.3.1. Option 1a – 1:1 Conversion of Shapefiles to GeoPackages

This experiment researched the direct 1 to 1 conversion of Shapefiles in a CDB datastore into a corresponding set of Geopackages. GDAL (https://www.gdal.org/drv_geopackage.html) and << >> were used to do the conversion. Charcateristics of the Option 1a approach are:

- There is a 4:1 reduction in the number of files.
- There is one layer (table) per GeoPackage.
- The Feature Class and Extended Attribute files have no geometry.
- “Off the Shelf” GeoPackage Viewers will have no compatibility over the feature class and extended attributes layers.
- This approach under-utilizes the capabilities of GeoPackage.

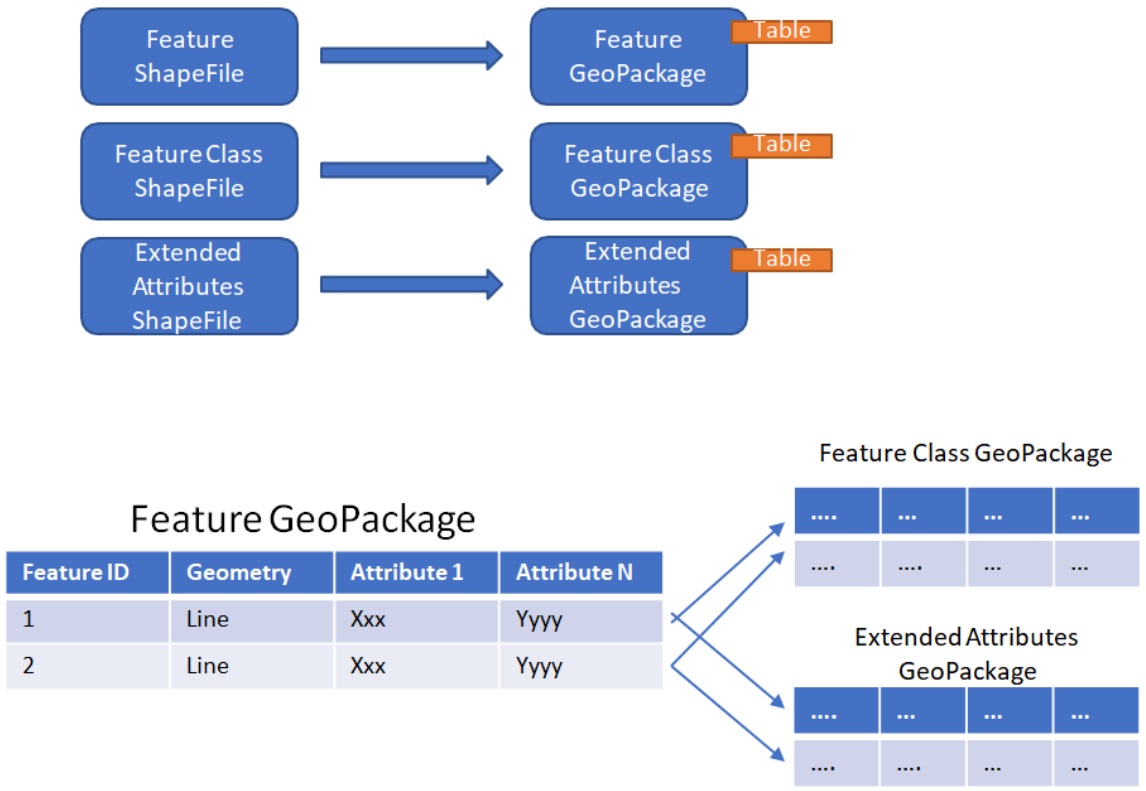

Figure 1. One to one conversion of Shapefiles to GeoPackages

5.3.2. Option 1b – Conversion of Shapefiles to GeoPackages using Normalized SQL Data

This experiment researched the approach of using normalized SQL in the conversion of Shapefiles into GeoPackages. This approach has the following characteristics:

- Utilizes a standard normalized relational database design, utilizing foreign keys.
- There is a 12:1 reduction in the number of files.
- There are three layers per GeoPackage.
- The Feature Class and Extended Attribute tables have no geometry.
- However, “Off the Shelf” GeoPackage software will not be aware of the extended and feature class attributes. This can be somewhat mitigated when a SQL View is used, which gives viewers (clients) read-only visibility over these attributes.

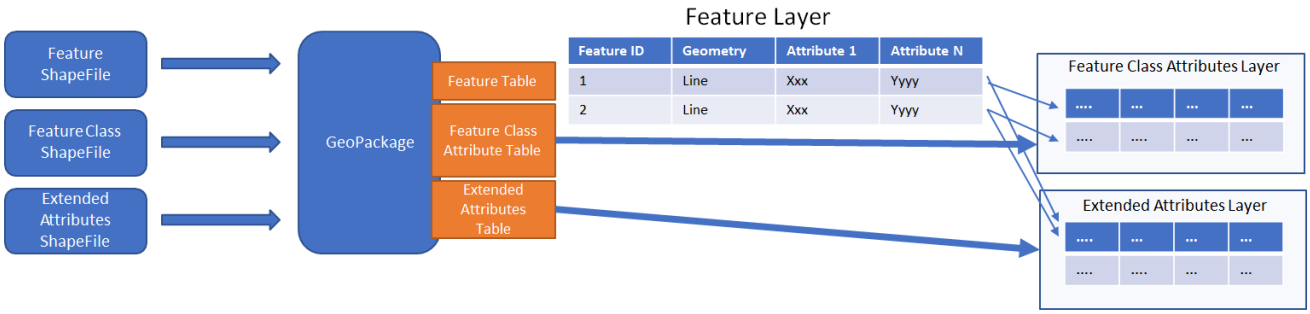

Figure 2. Use of Normalized SQL

5.3.3. Option 1c – Flattened Attribution

This experiment researched the approach of using flattened attribution in the conversion of Shapefiles into GeoPackages. This approach has the following characteristics:

- There is a 12:1 reduction in the number of files.
- Some duplication of data, resulting in larger files.
- There is one layer per GeoPackage.
- The Feature Class and Extended Attributes are populated for each feature.
- This approach utilizes a standard normalized relational database design, utilizing foreign keys.
- Full “Off the Shelf” GeoPackage software compatibility.

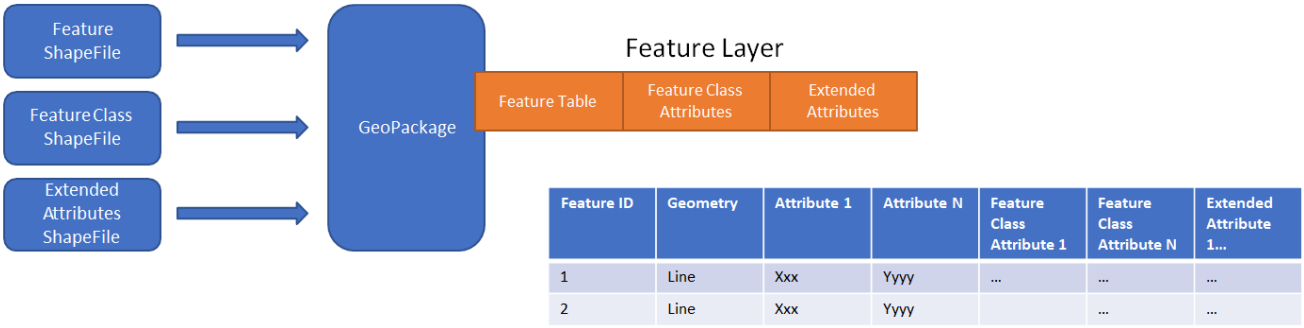

Figure 3. Flattened Attributes Approach

5.3.4. Option 1d – Flattened Attribution + extensions

This experiment researched the approach of using flattened attribution plus the GeoPackage related Tables extension in the conversion of Shapefiles into GeoPackages. The reason for using the extension was to enhance the ease of moving data in both directions (Shapefile to GeoPackage and visa-versa) using existing tools and without any data loss. This approach has the following characteristics:

- Flatten CDB standard instance and class attribute – maximum GIS tools compatibility
- “Off the Shelf” GeoPackage software compatibility for CDB standard attributes.
- Table (related tables) for extended attributes
- This approach utilizes a standard normalized relational database design, utilizing foreign keys.
- Some duplication of data, resulting in larger files (Class attributes).
- There is one layer per GeoPackage.
- The Feature Class and Extended Attributes are populated for each feature.

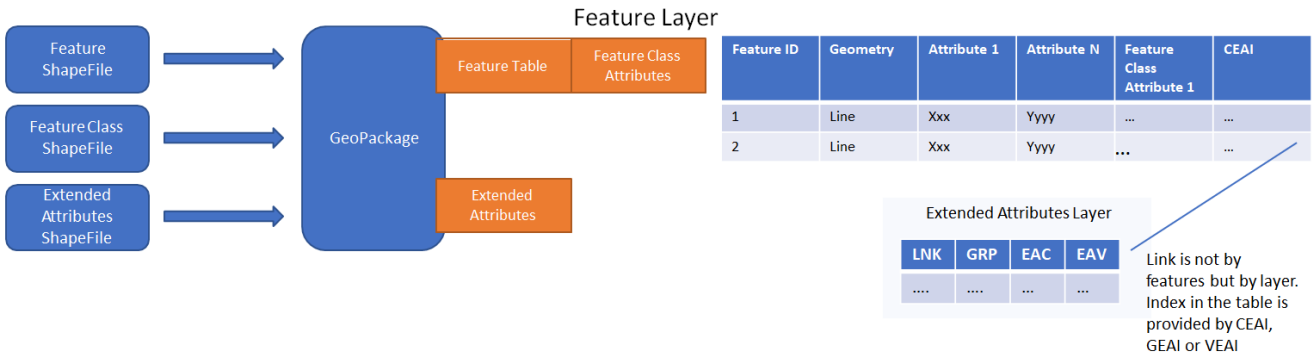

Figure 4. Flattened Attributes Approach

5.4. Experiment 3 - Each CDB LoD as a layer in GeoPackage

The methodology for Experiment 3 involves:

The goal of this experiment is to significantly reduce the number of files in both a CDB datastore and in the resulting GeoPackage. Steps in this experiment include:

- Modify implementation software to support storing an entire CDB Level of Detail (LoD) in a single GeoPackage.
- Evaluate and compare performance using the baseline CDB datasets and the Alternative #2 datasets.
- Evaluate analysis and results from Experiment #3 for performance, backwards-compatibility and risks to interoperability.

In this approach, the tables in the GeoPackage correspond to each LOD of CDB. The GeoPackage would contain 24 tables for each of the CDB LODs. Each CDB geotile would contain a GeoPackage to correspond to the CDB data stores (such road networks, geospecific points, etc.). CDB tiles for a data store combine into a single GeoPackage table within that given LOD where the tile definition (row and column) would be queryable attributes for each feature. In simple language, to find the features in a tile for a particular geotile's road network in LOD 3 of CDB, a consumer would open the road network GeoPackage, open the table that corresponds to LOD, and query for results where the column and row reference matches the CDB tile.

5.5. Experiment 4: Store each Geocell of Vector Data as a layer in GeoPackage

The methodology involves:

This experiment extends Experiment 3 (above) to have a single GeoPackage per Geocell in a CDB datastore. This results in all LODs and all CDB feature layers in a single GeoPackage. The steps in this experiment include:

- Modify implementation software to support storing an entire GeoCell in a GeoPackage.
- Evaluate and compare performance using the baseline CDB datasets and the Alternative #3

datasets.

- Evaluate analysis and results from Experiment #4 for performance, backwards-compatibility and risks to interoperability.

In this approach, the tables in the GeoPackage correspond to each data store of CDB (such road networks, geospecific points, etc.). The GeoPackage would contain eight (8) layers representing each of the CDB data stores (GSFeature, GTFeature, GeoPolitical, VectorMaterial, RoadNetwork, RailRoadNetwork, PowerLineNetwork, and HydrographyNetwork). CDB tiles and LODs for a data store combine into a single GeoPackage table where the tile definition (row and column) and LOD would be queryable attributes for each feature. In simple language, to find the features in a location for a particular geotile's road network in LOD 3 of CDB, a consumer would open the geotile's GeoPackage, open the table that corresponds to data store, and query for results where the LOD column and row reference matches the CDB tile and LOD.

Chapter 6. IE Experiment Results

This section of the Engineering Report provides details of the results of the experiments performed by each of the IE participants.

6.1. Aechelon Technology IE Report

6.1.1. Use case and experiment focus

The following is a description of a key use case in the Aechelon content processing workflow to use in an image generator.

- A CDB data store is used as the source for content to feed a publishing process into the Aechelon Image Generator (IG) runtime format. While reducing CDB storage requirements is a consideration, the primary concern in this workflow is with read access speed. Even so, the time taken by the 'feature scan' step of the publishing process, where the CDB source vector files are scanned to identify the features to import, is 2 orders of magnitude smaller than the rest of the pipeline. However, any improvements in speed and/or storage requirements have a positive impact on the efficiency of the Achelon publication workflow.
- Note: After the feature scan step in the publishing process, all references to all features are in various python data structures, which are then given to the first of multiple processing steps to begin the data transformations. For example, point features with models will have their OpenFlight models converted to an intermediate Aechelon-specific model format, while their instance geographical data are saved in an Aechelon-specific lookup table format.

The publishing software is in python, with invocations of C++ EXEs for performance-critical processing. The feature scan step is entirely in Python, version 3.5. Please note that for the image generation workflow, only metadata fields that affect the appearance of features are considered and remainder of the CDB content is ignored, such as tactical data, or the entire geopolitical dataset. The hydrography network dataset is also ignored since the RMTexture dataset is used to identify areas of water.

For the purposes of this experiment, five feature types were considered and processed: Cultural, Lights, Powerlines, Railroads, and Trees. All of the performance information provided in the tables below are related to these five feature types.

6.1.2. Aechelon Experiment Methodology

The following is a concise description of the methodolocy used for execution of the Aechelon committed tasks in this Interoperability Experiment.

- Following generation of the GeoPackage files for each option, changes were made as needed in the publishing scripts to be able to read and publish the data.
- Each feature type was tested by spot-checking in the image generator using a small reference database with representative data for each of the five feature types.
- The next step was to convert the following three of the four CDB data stores made available for the interoperability experiments. However, the entire publication end-to-end workflow for

image generation was not performed due to the considerable time it would take for each run:

  - Yemen (4 geocells), from Presagis.
  - Downtown Los Angeles (1 geocell), from VATC.
  - Greater Los Angeles (4 geocells), from Cognitics.

- To generate data for Experiment 2, option 1A:
  - Ran Option1.py from the Cognitics conversion scripts in the master branch (https://github.com/Cognitics/GeoCDB/tree/master).
  - Deleted existing .shp, .shx, sidecar .dbf, .dbt & .prj files (i.e. kept .dbf files holding class/extended data.)
- To generate data for Experiment 2, option 1C:
  - Ran Option1.py from the Cognitics conversion scripts in the Presagis branch (https://github.com/Cognitics/GeoCDB/tree/Presagis).
  - Deleted the existing .shp, .shx, .dbf, .dbt & .prj files.
- To generate data for Experiment 2, option 1D:
  - Ran Option1d.py from the Cognitics conversion scripts in the master branch (after update of March 17, 2019, and some local edits to protect against 'None' during conversion of the LA databases.)
  - Deleted the existing .shp, .shx, .dbf, .dbt & .prj files.
- To generate data for Experiment 3:
  - Ran Option3.py from the Cognitics conversion scripts in the master branch (after update of March 17, 2019, and some local edits to uncomment writing the class metadata to the instance tables and to protect against 'None' during conversion of the LA databases.)
  - Deleted the 100_GSFeature, 101_GTFeature, 202_RailroadNetwork and 203_PowerlineNetwork folders from each geocell.
- To generate data for Experiment 4:
  - Ran Option4.py from the Cognitics conversion scripts in the master repository (after update of March 17, 2019, and some local edits to uncomment writing the class metadata to the instance tables and to protect against 'None' during conversion of the LA databases.)
  - Deleted the 100_GSFeature, 101_GTFeature, 202_RailroadNetwork and 203_PowerlineNetwork folders from each geocell.
- Then, for each option, disabled the publishing process beyond the 'feature scan' step and captured the following metrics for the three CDB data stores.

6.1.3. Metrics

The following three tables provide basic performance metrics for the three CDB data stores processed in this experiment. Providing performance metrics is one of the tasks identified in Experiment 1 of the GeoPackage in CDB IE.

In the tables below, "Baseline" refers to metrics based on the source ShapeFiles. Option 1A, Option 1C, and Option 1D refer to the sub-options for Experiment 2 (one to one transformation of

Shapefiles into GeoPackages).

Table 1. Yemen (4 geocells)

			Baseline	**Option 1A**	**Option 1C**	**Option 1D**	**Exper. 3**	**Exper. 4**
Dataset(s)	Feat count	PVF count	time	time	time	time	time	time
tree	64091	440	8	7	7	16	6	2
light	60	13	<1	<1	<1	<1	1	1
cultural	16502	409	12	9	5	7	5	4
powerline	975	20	<1	<1	<1	<1	1	1
railroad	0	0	0	0	0	0	0	0
total time			21	17	13	24	14	8
file count			8224	2056	1023	1023	10	10
size (MB)			34.2	152.5	161.9	165.9	57.6	38.1

Table 2. Downtown Los Angeles (1 geocell)

			Baseline	**Option 1A**	**Option 1C**	**Option 1D**	**Exper. 3**	**Exper. 4**
Dataset(s)	Feat count	PVF count	time	time	time	time	time	time
tree	2	1	<1	<1	<1	<1	<1	<1
light	0	0	0	0	0	0	0	0
cultural	1730622	1948	9:01	7:13	3:06	3:34	3:26	3:41
powerline	1208	56	2	1	1	1	<1	1
railroad	1386	4	1	<1	<1	<1	<1	1
total time			9:04	7:15	3:08	3:36	3:27	3:44
file count			12540	4180	2090	2090	4	4
size (MB)			2185.7	2309.2	958.5	1021.5	791.5	798.0

Table 3. Greater Los Angeles (4 geocells)

			Baseline	**Option 1A**	**Option 1C**	**Option 1D**	**Exper. 3**	**Exper. 4**
Dataset(s)	Feat count	PVF count	time	time	time	time	time	time

			Baseline	Option 1A	Option 1C	Option 1D	Exper. 3	Exper. 4
tree	5	2	<1	1	<1	<1	1	<1
light	0	0	0	0	0	0	1	<1
cultural	3138841	6013	15:02	12:02	6:14	7:25	6:57	7:17
powerline	3932	160	1	1	1	1	1	1
railroad	9367	87	1	1	1	1	1	<1
total time			15:04	12:05	6:16	7:27	7:01	7:19
file count			38961	12986	6493	6493	14	14
size (MB)			3738.2	4275.9	1958.6	2067.0	1335.7	1339.3

6.1.4. Legend

- Feat count: feature count of valid features found of the given type
- PVF count: primary vector file count, after validation, for the given type (i.e. only counting .shp files for Experiment 1 or .gpkg files for Experiment 2.)
- Time: in minute:second notation when over 1 minute, else in seconds
- The cultural feature data set is from both 100_GSFeatures (S001_T001 & S002_T001) and 101_GTFeatures (S001_T001)
- File count: total number of files from 100_GSFeatures, 101_GTFeatures, 202_RailroadNetwork & 203_PowerLineNetwork
- Size: storage, in MB, used by all the files from 100_GSFeatures, 101_GTFeatures, 202_RailroadNetwork & 203_PowerLineNetwork

6.1.5. Notes and observations

- All source CDB files were on a local RAID drive so network traffic did not contribute to the timings.
- In the Greater Los Angeles database, there somehow were more features of some types coming from geopackage files compared to shape files (3140180 instead of 3138841 cultural features, and 4012 instead of 3932 powerline features), but there were also over 1000 warnings from OGR during conversion and while reading of the type "Warning 1: Unable to parse srs_id '100000' well-known text "." After the 1000th such warning, also got "More than 1000 errors or warnings have been reported. No more will be reported from now." Perhaps the conversion from .shp to .gpkg with ogr2ogr.exe generated these excess invalid files. These warnings appeared in the Downtown LA database as well, but the feature counts matched after conversion. Checking any further downstream for discrepencies in the processing pipeline was not performed.
- For the powerline network dataset, statistics include both the tower point features and the wire lineal features.

- There's a slight increase in the file size in the Los Angeles databases when comparing the results of Experiment 3 and Experiment 4. However, there is a significant decrease in the size of the Yemen database. From a quick inspection of the data, this seems to correlate with the fact that almost all the cultural features in Los Angeles come from 100_GSFeatures which require unique records per instance, whereas for Yemen the majority of cultural features come from 101_GTFeatures.
- Experiment 3 has slightly better timings for large-count datasets than Experiment 4 in our use case since we scan each LOD in order, so having LODs in separate layers in the option 3 GeoPackage performs better.

6.2. CAE Results for Experiment 2

6.2.1. Focus of the Experiment

CAE focused on Experiment #2 as described in Section 5 - Details of the Experiments Performed. Specifically, CAE focused on the part of the experiment where each Shapefile is replaced by one GeoPackage file.

CAE performed a series of tests to measure the impact of Shapefile-to-GeoPackage conversion on file size, file count, network transfer, and decode performance. These tests are detailed in the following sections.

CAE conclusion with some observations and recommendations for futher development of the standard can be found in Section 7 Recommendations.

6.2.2. Comparing File Formats

The first step of our experiment was to compare other file formats that would be appropriate to store CDB Vector Data. These formats include GeoJSON and GML in addition to Shapefiles and GeoPackage. The table below lists the observed file sizes.

Table 4. Converted File Sizes

	Shapefile	**GeoPackage**	**GeoJSON**	**GML**
Small input	0.5 KB	112 KB	0.5 KB	3.6 KB
Medium input	32 KB	152 KB	48 KB	106 KB
Large input	336 KB	520 KB	824 KB	928 KB
Largest input	2686 KB	3084 KB	4410 KB	9008 KB

Notes and observations:

- GeoPackages are very space-inefficient at encoding small numbers of features. An amount of data that requires 0.5 KB in Shapefile or GeoJSON encoding was observed to require 112 KB as output by `ogr2ogr`. However, not all database objects generated by `ogr2ogr` appear to be strictly required by the standard, and we were able to construct a GeoPackage that should be minimally standard-compliant in as little as 36 KB (by not including optional tables, indexes, triggers, or sequences).

- The actual space required on disk will increase these numbers to varying degrees; the minimum disk space required for a Shapefile in this test was observed to be 12 KB (3 files times 4 KB allocation unit size).
- GeoJSON and GML are less space-efficient than binary alternatives at larger file sizes. GML is particularly large, being in some cases more than twice the size of GeoJSON and three times the size of the Shapefile. However, GeoJSON and GML remain interesting from the standpoint of interoperability.

Notably, we also observed that every new table added to an sqlite database increases the size by a minimum of 4 KB, which is presumably an internal allocation unit intended to support real-time addition of data rows.

We also did a simple read-performance test on each of these files. For this test, we measured the time taken by GDAL to open the file, iterate through all features, then close the file. All measurements represent the mean of 8 runs.

Table 5. Initial Read Performace Test

	Shapefile	**GeoPackage**	**GeoJSON**	**GML**
Small input	2.3 ms	4.6 ms	0.7 ms	2.3 ms
Medium input	5.5 ms	4.6 ms	4.3 ms	4.7 ms
Large input	14.3 ms	6.7 ms	53.0 ms	18.6 ms
Largest input	278.5 ms	40.4 ms	347.0 ms	183.3 ms

Notes and observations:

- GeoPackage read performance scales extremely well at these file sizes. However, there is a fixed overhead that is rather larger than the other file formats.
- GML read performance scales favorably to Shapefiles, with a fixed overhead comparable to Shapefiles (at least in GDAL).
- GeoJSON has a very low fixed overhead, but scales surprisingly badly. We strongly suspect that this performance problem is due to GDAL's use of the `libjson` library. For future performance tests with this format, we strongly recommend using RapidJSON (http://rapidjson.org/) or other comparably fast JSON parser. At least one benchmark (https://github.com/mloskot/json_benchmark) reports libjson as being 30 times slower than RapidJSON for GeoJSON data, and another benchmark reports that libjson does not correctly handle UTF-8 data (https://github.com/miloyip/nativejson-benchmark), which could be an interoperability issue. With a faster JSON parser, we expect performance to be similar to GML or even faster due to smaller file size.

6.2.3. Modifications to the GDAL/OGR Library

In preparation for more comprehensive tests, CAE then made a few minor modifications to the GDAL library to ensure that the Shapefile-to-GeoPackage conversion was sufficiently lossless for our purposes.

CAE made the following modifications:

- Stopping the library from optimizing away the M dimension in the case of 2D measured geometry. (This optimization saves some space where all M values are `nodata`, but it changes the declared type of the geometry.)
- Mapping the `Logical` DBF field type to the OGR field type `OFTInteger` subtype `OFSTBoolean`. The DBF logical field type was previously handled as a string.
- Mapping the DBF logical values "T" and "F" to "1" and "0", respectively.
- Allowing the DBF reader to correctly read dates with the format `YYYY/MM/DD`.

6.2.4. Converting a Full CDB

CAE did not use the sample CDB provided by the participants because their goal was to compare the performance of our internal applications when running with a CDB produced by CAE before and after the replacement of Shapefiles with GeoPackage files.

However, CAE believe that their findings may apply equally well to other databases.

CAE built a customized script that invoked the `ogr2ogr` executable to convert all vector files in two CDBs. Class DBF files and junction DBF files were converted to standalone GeoPackage files.

CAE found that the conversion to GeoPackages can substantially increase the amount of disk space required for vector data.

Table 6. Disk Space Required

	ESRI Shapefiles	ESRI Shapefiles (disk)	GeoPackages
CDB 1	10.1 GB	10.4 GB	16.4 GB
CDB 2	12.4 GB	20.6 GB	119.1 GB

Notes and observations:

- Some of our CDBs have a very large number of very small vector files. This leads to an increase in disk space usage: CDB 2 in particular requires 20.6 GB to store 12.4 GB of Shapefiles (assuming a 4 KB allocation unit).
- We have not done a complete measurement of file size distributions, but in the case of CDB 1, we do know that over 40% percentage of the shapefiles consist of a single .dbf. The median ESRI Shapefile size (sum of .shp/.shx/.dbf/.dbt) is about 3 KB, lower quartile under 1 KB, upper quartile 38 KB. Only 20% of the shapefiles are larger than 112 KB.
- CDB 2, which we believe appproaches a worst-case scenario in terms of disk usage increase, has a nearly 6 times increase in on-disk vector data-storage requirements (from 20.6 GB to 119.1 GB). This constitutes a non-negligible risk.

6.2.5. Network Test

Our application of CDB involves networked client/server systems, so a key performance factor is the time required to request and transfer files over a network. For this test, we measured the time taken for a client to request and transfer a selection of vector data files from a networked server. Files were loaded "cold": the CDB data volume was freshly mounted immediately before each test to

ensure that the OS file cache was clear. This test loaded files from CDB 2.

Table 7. Network Test

	ESRI Shapefiles	GeoPackages
Request/Open/Transfer Time	5132 ms	3475 ms
Files Transferred	669 files	223 files
Data Transferred	107 MB	161 MB

Notes and observations:

- This test does not load class- or extended-attribute files.
- The file-count reduction ratio is 3:1, not 4:1. We do not store 0-byte files if we can avoid it.
- The amount of data transferred is larger for GeoPackages than for Shapefiles, but the number of files requested is substantially smaller. The largest performance factor in this test seems to be the reduction in the number of files requested, not the I/O volume.
- The data transfer increase was only about 1.5x, compared with a 9.6x increase (12.4 GB to 119.1 GB) in total vector data for this CDB. This test should therefore not be taken an indication of worst-case performance, and suggests that the density of geographic features could vary considerably from location to location. Determination of an accurate worst-case performance profile would require more extensive experiments.

6.2.6. Real-Time CDB Client Device

The final test is to benchmark the loading time for a certain geographical region in a real time system. We measured the decode time, number of files and data transfer volume. The real time system is the client device consuming OGC CDB data over the network. This test loaded files from both CDB 1 and CDB 2.

Table 8. Real-Time CDB Client Test

	ESRI Shapefiles	GeoPackages	difference (+/-)
Decode-only Time	7.37 s	5.65 s (GDAL [*]) 10.81 s (internal)	+9.09 s
Files decoded	5680 files	2838 files	50% fewer files
Data transferred	479 MB	906 MB	89.1% more bytes

[*] Our observed GeoPackge decode time is split into GDAL-related processing and internal format conversion (which is unoptimized). Although our total GeoPackage decode time was measured at 9.09 seconds slower than Shapefile decode time, the GDAL-only portion of the decode time 1.72 seconds faster. If we were to write a well-optimized GeoPackage decoder that decoded directly from sqlite into our internal representation, it would be reasonable to expect a small performance win.

Notes and observations:

- This test loads class- and extended-attribute files where present.
- The file-count reduction ratio is 2:1, not 4:1. We do not store 0-byte files if we can avoid it.

- We did observe a slight overall slowdown in the system, but the total slowdown was less than the 9.09 seconds observed in the decode process. This suggests that the performance gained by halving the file count was greater than the performance lost by doubling the I/O bandwidth requirements.

6.3. Compusult Results from Experiment 2

Approach: One GeoPackage per LOD per dataset

CDB: CDBYemen_4.0.0

Available Datasets:

- 101_GTFeature
- 100_GSFeature
- 401_Navigation
- 201_RoadNetwork

Number of ShapeFiles processed: 358 Number of GeoPackages created: 18 Total byte size of ShapeFiles (bytes): 3,569,324 Total byte size of GeoPackages (bytes): 41,715,712 Elapsed time (seconds): 173

6.4. FlightSafety Experiment Results

6.4.1. FlightSafety International's Use Case for CDB

FlightSafety has developed both a CDB generation tool and a CDB Publisher client. The performance requirements of the CDB Publisher are much greater than CDB generation, so this report will focus on loading and consuming a CDB dataset. The CDB Publisher uses a CDB data store as the source data for building the synthetic environment for FlightSafety's VITAL 1100 image generator system. These systems are used for pilot training on a variety of flight simulator systems. The Publisher does not do any preprocessing of the CDB dataset; all CDB data that it consumes is discovered and loaded during the publishing. This approach was chosen due to the world-wide scope of CDB and unknown quantity of content. The CDB specification's structure makes it easy to find the file(s) containing the data needed for the synthetic environment creation. Based on the flight training system requirements, an appropriate level of detail of vector and model data is discovered and loaded. The Publisher adapts to the available levels of detail of vector data, and the flight characteristics of the training device. The publishing system is primarily in C, and the testing was all performed with C libraries and code. The Shapefile API that is tested is a custom FlightSafety library, optimized for faster performance.

6.4.2. FlightSafety Experiment Focus

The experiments were focused on just-in-time uses of CDB, similar to how a FlightSafety visual system would use the data. Statistics were collected on the original CDB dataset, and the converted GeoPackage CDB datasets. These were used to infer the cost of database configuration management and transmission/deployment to a training device. Testing was done on both currently encoded CDB

shapefiles, and on converted GeoPackage encoded files (covering options 1, 3, and 4). Tests focused on the latency of loading files, processing data, and closing files. Tests were done on different conversion options and settings to come up with optimal recommendations

6.4.3. FlightSafety Experiment Methodology

This is the methodology used to evaluate, convert and test the CDB datasets using GeoPackage vector encoding.

Data Acquisition:

Three CDB datasets were downloaded (from Cognitics, Presagis, and VATC) and loaded on a system. The datasets were then split into two CDBs, one of which contains all vector data and the other contains everything else. They were linked together using CDB's versioning mechanism, so that the FlightSafety publisher sees the data as a single dataset. Further: - Any official or unofficial extension to the CDB was removed for testing purposes. - Any 0 size vector file was deleted from the CDB with vector data. These were 0 size shp and shx files for datasets that should only be dbf, and cases of 0 size dbt files when they weren't needed alongside their dbf parent file.

Data Evaluation:

All three CDB datasets were flown using FlightSafety's VITAL 1100 image generator and CDB publisher. During the fly-through, any data artifacts were noted and recorded.

Data Conversion:

The python conversion scripts developed by Cognitics, Inc. were downloaded from GitHub. The scripts were modified to properly flatten class-level attributes into the feature table, and to properly handle DBase floating point and logical field types. Index tables were also added to aid SQL queries designed to get back data for a specific CDB vector file. Script changes were published to a public GitHub under a FlightSafety account (link). When the scripts were run, they created a new output directory for the CDB vector data. The Metadata folder was copied from the original vector CDB version, which then links this GeoPackage version to the rest of the CDB data. The three main conversion scripts used implemented GeoPackage encoding options 1, 3, and 4.

GeoPackage Testing:

The initial data collection centered on the number of vector files and how much disk space was consumed. All full CDB storage devices used a 4kB block size and recorded sizes include the "dead" space due to the minimum block size. The initial tests were testing shapefiles vs. option 1. All vector files were located, and timed on the file open and accessing the data within the file. Total processing time was recorded and compared between the two encodings. This test accessed the geometry and all the attributes, whether they would have been used by the FlightSafety CDB publisher or not.

The next set of tests involved working with worst case examples and comparing the same file open and access time as before, but for single files. This highlights performance on the largest vector files. The average performance times are reported here.

Further testing was performed to see what the trade offs were between options 1, 3, and 4. These

included loading identical vectors (from a single original shapefile) from each of three GeoPackage files converted in different ways:

- GeoPackage Option 1 (Experiment 2 in the IE Activity Plan) was a straight conversion of the shapefile. The GeoPackage contains a single data table with flattened class-level attributes, with the same number of records as the original shapefile
- GeoPackage Option 3 (Experiment 3 in the IE Activity Plan) was a conversion of each CDB dataset's features into a table for each level of detail (LOD) and component selector set, placed into a single GeoPackage (1 per dataset). It also contained the most tables, and typically had more feature records than option 1 but fewer than option 4.
- GeoPackage Option 4 (Experiment $ in the IE Activity Plan) was a conversion of each CDB dataset's features into a table for each component selector set, placed into a single GeoPackage (1 per dataset). This method placed all levels of detail into the same table, resulting in a handful of tables, but possibly millions of features per table.

Note: *The results for Experiment 1 (Conversion) are provided in the discussions of Option 1, 3, and 4 (IE Experiments 2, 3, and 4).*

6.4.4. FlightSafety Metrics

Original Dataset Statistics

Basic statistics were collected on the original CDB datasets used in the Interoperability Experiment. The CDB storage size and file counts do not include any 0-sized files (they weren't required by the CDB specification) and do not include non-standard extension data. The last two rows represent the proportion of vector data in the CDB, by the percentage of files and storage used. The vector datasets used are: - 100_GSFeature - 101_GTFeature - 102_GeoPolitical - 201_RoadNetwork - 202_RailroadNetwork - 203_PowerlineNetwork - 204_HydrographyNetwork - 401_Navigation

Table 9. Table of Dataset Statistics

	Northwest Pacific	**Yemen**	**Los Angeles**
Provider	Cognitics	Presagis	VATC
CDB Geocell Tiles	27	4	1
CDB Storage Size	214 GB	17.4 GB	59.6 GB
CDB File Count	427,536 files	112,837 files	62,895 files
Vector Storage Size	9,152 MB	53.4 MB	2,381 MB
Vector File Count	109,490 files	4714 files	13,075 files
% of CDB storage as vectors	4.18 %	0.30 %	3.90 %
% of CDB files as vectors	25.6 %	4.18 %	20.8 %

The main takeaway from this table is that vector data does not consume a large amount of storage space, but accounts for a prodigious number of files within a typical CDB. The main driver of file counts are that Shapefiles are a multi-file format, where three (or four with the .prj projection file)

files represent a single Shapefile. In addition to the multifile format, CDB uses extra class-level and extended-level attributes encoded as extra DBF files. So anywhere from 3 to 8 files are used to represent a single logical vector file.

Specific Vector File Test Data

Some of the testing below involved loading specific point/linear/areal vectors that represent a single Shapefile. For these tests, examples were found that represent "worst-case" examples of large vector files. These larger files would take more time to load, and most occurred within higher LODs that would lead to larger tables in options 3 and 4. The following table records the specific shapefile data for individual tests.

	Northwest Pacific	**Yemen**	**Los Angeles**
Point Vector	N46W124_D101_S002_T001_L04_U15_R12	N12E045_D100_S001_T001_L04_U12_R0	N34W119_D100_S001_T001_L05_U8_R20
Linear Vector	N48W123_D201_S002_T003_L01_U0_R0	N12E045_D201_S002_T003_L00_U0_R0	N34W119_D201_S002_T003_L04_U1_R15
Areal Vector	N47W120_D204_S002_T005_L02_U0_R2	N12E044_D100_S002_T005_L02_U3_R3	N34W119_D204_S002_T005_L03_U4_R7

6.4.5. Shapefile vs. GeoPackage Option 1 (Experiment 2) Testing

Option 1 Conversion Statistics

Before the first set of tests, the CDB datasets were converted one-to-one from shapefiles to GeoPackage, using the option 1 conversion. Dataset statistics were then collected on the new datasets and compared with the original datasets.

	Northwest Pacific	**Yemen**	**Los Angeles**
Shapefile Vector Storage Size	9,152 MB	53.4 MB	2,381 MB
Shapefile Vector File Count	109,490 files	4714 files	13,075 files
GeoPackage 1 Storage Size	17,827 MB	157.9 MB	938 MB
GeoPackage 1 File Count	25,083 files	1,146 files	2,615 files
Relative Size (>1 is larger)	1.95	2.96	0.39
% Fewer Vector Files	77 %	76 %	80 %

File counts for the GeoPackage CDB were between a 4:1 and 5:1 reduction in vector files. The size changes varied dramatically, likely due to how efficient the attributes were packed into the original Shapefile's instance and class-level DBF files. In general, an increase in CDB size is expected using option 1.

Option 1 Testing Focus

The testing focused on the latency of loading and processing the vector data files, and traversing all the geometry features and attributes. This approach was used to simulate a flight simulation client's use of CDB.

Test Procedure 1

The first test was to traverse the entire CDB dataset, find all the vector files and collect the time it took to open, process, and close each vector file. For each dataset, every vector file was located by walking the directory structure, and then the file loading and processing was timed. This test was run 30 times on the smaller CDB datasets (Yemen and Los Angeles) and 10 times on the larger Northwest Pacific dataset. The sum of the file load and process steps are recorded below (while ignoring the file search times).

All Vector Files	Northwest Pacific	Yemen	Los Angeles
Shapefile Timing	835 sec	10.2 sec	27.5 sec
GeoPackage Timing	478 sec	4.2 sec	25.7 sec
GeoPackage Speed Comparison	42% faster	58% faster	6.7% faster
Average Shapefile Storage Size	374 kB	48 kB	923 kB

This table shows, on average, that using GeoPackages are faster than using Shapefiles. These results imply that GeoPackage has a better advantage with smaller files. For example, GeoPackage performed best on Yemen with its relatively small shapefile/vector files. However, there is less of an advantage with larger vector files. Therefore, further testing using larger files is recommended.

Test Procedure 2

The next set of tests focused on some of the largest individual vector files. This test was performed to evaluate some of the worst case examples. The exact file names are mentioned above in the Specific Vector File Test Data section. These test datasets were much larger than the average vector file and cover the three basic geometry types: Points, Line Strings and Polygons. This allowed testing of files that have many attributes compared to coordinates (points), and testing of files with many coordinates compared to the number of attributes (polygons).

- The file size for Shapefiles includes both the instance-level files (.shp, .shx, .dbf) and the class-level attributes (.dbf), but no extended attributes or projection information. The GeoPackage file size was the single .gpkg file.
- The timing numbers include opening the file and traversing the geometry and every attribute in each record, including those that would otherwise not be used by the FlightSafety client. The timing test was performed 100 times alternating between loading from the shapefile CDB dataset, and the equivalent GeoPackage CDB dataset.
- The last row represents the relative performance of GeoPackage as compared to Shapefiles, with a number higher than 1.0 representing increased speed.

Point Vectors	Northwest Pacific	Yemen	Los Angeles
Feature Count	16,384	5,552	4,734
Shapefile Size	1.91 MB	1.40 MB	3.63 MB
GeoPackage Size	3.93 MB	1.46 MB	1.18 MB
Shapefile Read	55.8 ms	64.4 ms	17.4 ms
GeoPackage Read	82.3 ms	36.78 ms	39.9 ms
Relative GeoPackage Performance (>1.0 is faster)	0.678	1.751	0.437

GeoPackage performance numbers were mixed for point data. The GeoPackage performance seems linear with the number of features, but the Shapefile API tested was much faster on one case (Los Angeles) and much slower on another (Yemen).

Note: *The Northwest Pacific dataset uses a minimal number of class-level attributes, resulting in a larger flattened GeoPackage size. In contrast, the Los Angeles dataset uses mostly unique class-level attributes, which yields a larger overall Shapefile size, but smaller GeoPackage size because fewer class-level attributes needed to be duplicated.*

Line String Vectors	Northwest Pacific	Yemen	Los Angeles
Feature Count	8,183	2,457	3,343
Shapefile Size	1.96 MB	0.71 MB	2.83 MB
GeoPackage Size	2.65 MB	1.08 MB	1.18 MB
Shapefile Read	62.2 ms	26.3 ms	17.4 ms
GeoPackage Read	49.9 ms	19.0 ms	23.1 ms
* Relative GeoPackage Performance (>1.0 is faster)*	1.246	1.383	1.225

The use of GeoPackage increased performance across the board when linear data (22-38%) is processed and used.

PolygonVectors	Northwest Pacific	Yemen	Los Angeles
Feature Count	94	198	127
Shapefile Size	388 kB	387 kB	126 kB
GeoPackage Size	512 kB	556 kB	188 kB
Shapefile Read	9.3 ms	10.0 ms	7.3 ms
GeoPackage Read	6.3 ms	6.4 ms	4.9 ms
Relative GeoPackage Performance (>1.0 is faster)	1.476	1.569	1.502

Larger performance increases for areal data (47% - 56%), at the cost of relatively larger storage size. However, the sample size (number of polygon features) is quite small.

6.4.6. GeoPackage Option 3 and 4 Testing

Please remember that in the FlightSafety presentation of results:

Option 1 = Experiment 2
Option 3 = Experiment 3
Option 4 = Experiment 4

Option 3 & 4 Conversion Statistics

In addition to the one-to-one shapefile to GeoPackage encoding, we wished to also test the other GeoPackage encodings represented by options 3 and 4. Conversions were performed to create these new CDB datasets using the modified python conversion scripts. These were tested against the option 1 CDB datasets used in the previous tests. The dataset statistics (file sizes and counts) are in the table below. Conversion notes include:

- The parts of the file name (dataset code/component selectors/lod/row/column values) were initially stored as strings. Converting these to integers led to about a 10% improvement over the initial string conversion.
- Index tables were created for the parts of the filename that did not comprise the table name. This led to significant improvements that were up to 90% faster than without the index. — Option 3 table names were of the form: "D100_L04_S001_T001". So indexes were created for the row and column values, assuming that a user might want to generate a SQL query on that table's row and column values. — Option 4 table names were of the form: "D100_S001_T001". So indexes were created for the lod, row and column values, assuming that a user might want to generate a SQL query on that table's lod and row and column values.

Conversion Statistics	**Northwest Pacific**	**Yemen**	**Los Angeles**
GeoPackage 1 File Count	25,083 files	1,146 files	2,615 files
GeoPackage 3/4 File Count	161 files	22 files	7 files
% Fewer Vector Files Options 3/4 vs Option 1	99.4 %	98 %	99.7 %

As expected the number of files is much smaller using options 3 or 4.

Conversion Statistics	**Northwest Pacific**	**Yemen**	**Los Angeles**
Shapefile Storage Size	9,152 MB	53.4 MB	2,381 MB

Conversion Statistics	Northwest Pacific	Yemen	Los Angeles
GeoPackage 1 Storage Size	17,827 MB	157.9 MB	938 MB
GeoPackage 3 Storage Size	16,479 MB	59.3 MB	728 MB
GeoPackage 4 Storage Size	16,729 MB	55.5 MB	740 MB
Options 1 & 3 Relative Size (> 1 is larger)	0.92	0.38	0.78
Options 1 & 4 Relative Size (> 1 is larger)	0.94	0.35	0.79
Shapefile vs Option 3 Size (> 1 is larger)	1.80	1.11	0.31
Shapefile vs Option 4 Size (> 1 is larger)	1.83	1.04	0.31

In all cases, the combined GeoPackage datasets required less storage than the option 1 GeoPackage files. This was true even though the combined datasets have index tables that the Option 1 GeoPackages do not have. Note that in all cases even combining the GeoPackage files into a minimal set does not lead to a smaller vector dataset than Shapefiles.

Testing Procedure

The testing focus for comparing the different GeoPackage encoding options was on the latency of loading the GeoPackage file and using SQL queries to return the records converted from a single Shapefile. This approach is similar to the Shapefiles vs GeoPackage testing done above, but the test was constructed slightly differently.

- The GeoPackage was opened and an SQL query was performed to return the data that represented a single shapefile's vector data. In each query, the number of records in the table varied according to the type of conversion performed, but the amount of data and the values returned by the query were identical.
- The SQL queries used for each GeoPackage option were variations on the following: -- Option 1: SELECT * FROM 'D100_S001_T001_L04_U12_R0' -- Option 3: SELECT * FROM '100_GSFeature_L04_S001_T001' WHERE _UREF='12' AND _RREF='0' -- Option 4: SELECT * FROM '100_GSFeature_S001_T001' WHERE _LOD='4' AND _UREF='12' AND _RREF='0'
- For timing purposes, each GeoPackage's open and query was run 100 times, alternating between each option test in succession.
- The relative speed row shows the performance hit of doing an open on a larger GeoPackage with more tables and more records to search through. For example, a 2.0 represents a test that took twice as long as the option 1 test.

Point Queries	Northwest Pacific	Yemen	Los Angeles
Option 1 Table Size (Count)	16,384	5,552	4,734
Option 1 Read GeoPackage	87.3 ms	39.6 ms	38.6 ms
Option 3 Table Size (Count)	3,375,935	7,766	493,936
Option 3 SQL Query	138.2 ms	59.3 ms	51.5 ms
Speed Relative to Option 1 (>1 is faster)	0.63	0.67	0.75
Option 4 Table Size (Count)	6,865,325	43,122	2,842,150
Option 4 SQL Query	173.9 ms	44.9 ms	79.7 ms
Speed Relative to Option 1 (>1 is faster)	0.50	0.88	0.48

Option 3 GeoPackage file opens are sensitive to the number of tables in the GeoPackage, and tend to dominate the timing of cases with fewer features. Option 4 has fewer tables and faster GeoPackage opens, but is more sensitive to the number of records in the table that need to be searched.

Line String Queries	Northwest Pacific	Yemen	Los Angeles
Option 1 Table Size (Count)	8,183	2,457	3,343
Option 1 Read GeoPackage	53.8 ms	22.3 ms	26.4 ms
Option 3 Table Size (Count)	16,454	2,457	80,697
Option 3 SQL Query	63.6 ms	24.5 ms	28.7 ms
Speed Relative to Option 1 (>1 is faster)	0.85	0.91	0.92
Option 4 Table Size (Count)	79,512	2,457	149,757
Option 4 SQL Query	52.4 ms	23.0 ms	29.7 ms
Speed Relative to Option 1 (>1 is faster)	1.03	0.97	0.89

In both options 3 and 4, GeoPackage files perform slightly worse, but perform better than the point query because of fewer features returned.

Polygon Queries	Northwest Pacific	Yemen	Los Angeles
Option 1 Table Size (Count)	94	198	127
Option 1 Read GeoPackage	5.4 ms	6.1 ms	4.3 ms
Option 3 Table Size (Count)	207	330	1,238
Option 3 SQL Query	11.3 ms	16.2 ms	6.3 ms
Speed Relative to Option 1 (>1 is faster)	0.48	0.37	0.69
Option 4 Table Size (Count)	2,250	1,531	1,480
Option 4 SQL Query	6.1 ms	7.4 ms	5.0 ms
Speed Relative to Option 1 (>1 is faster)	0.88	0.82	0.86

The overhead of opening GeoPackage files with lots of tables in the option 3 encoding is particularly prominent in the polygon queries. The option 4 encoding is close to the single vector file per GeoPackage timing.

6.4.7. Further GeoPackage Option 3 & 4 Testing

Testing Procedure

The initial SQL query testing only performed a single query per GeoPackage open and close. A more typical use case with options 3 and 4 would be to hold a GeoPackage file open for longer periods of time, and perform more queries per file access. In this test, the same query was performed, but 100 queries were performed while the file was open. There are limitations to the results of this test, as the same query was performed over and over. It was likely that the parts of the file being accessed remained in memory the whole time, and this only measures the time to copy the data out of the GeoPackage file. But it is a starting point toward understanding the performance of repeated queries in a large file.

The test results show the time per query, plus a 1/100 portion of the GeoPackage open and close time. It also compares this time with Option 1's performance, where there is little gained by keeping the GeoPackage open.

Test Results

Points - 100 Queries	Northwest Pacific	Yemen	Los Angeles
Option 1 - 1 Query	87.3 ms	46.0 ms	38.6 ms
Option 3 - 100 Queries Average	64.6 ms	25.9 ms	26.4 ms

Points - 100 Queries	Northwest Pacific	Yemen	Los Angeles
Percent Faster than Option 1	26%	35%	32%
Option 4 - 100 Queries Average	68.2 ms	24.2 ms	23.8 ms
Percent Faster than Option 1	22%	39%	38%

Keeping the GeoPackage open between queries improves performance. But note that not all cases are faster than the original Shapefile performance.

Line Strings - 100 Queries	Northwest Pacific	Yemen	Los Angeles
Option 1 - 1 Query	53.8 ms	22.3 ms	26.4 ms
Option 3 - 100 Queries Average	34.1 ms	11.8 ms	13.3 ms
Percent Faster than Option 1	37%	47%	49%
Option 4 - 100 Queries Average	34.9 ms	11.1 ms	13.3 ms
Percent Faster than Option 1	35%	50%	49%

Polygon - 100 Queries	Northwest Pacific	Yemen	Los Angeles
Option 1 - 1 Query	5.4 ms	6.1 ms	4.3 ms
Option 3 - 100 Queries Average	1.1 ms	1.6 ms	0.78 ms
Percent Faster than Option 1	80%	75%	82%
Option 4 - 100 Queries Average	0.79 ms	1.1 ms	0.85 ms
Percent Faster than Option 1	85%	82%	80%

This approach shows that there is some significant overhead in opening large GeoPackage files. Keeping the GeoPackages open can mitigate some of the overhead. We do not believe that a full client would see this level of performance, but there is a good possibility a client would see improved performance over option 1.

6.5. Hexagon US Federal Technology Experiment Report

6.5.1. Experiment Methodology - Dataset Conversion

Experiment 1 - While the initial direction of the Interoperability Experiment involved utilizing the provided open source scripts to facilitate the conversion, there was an interest expressed from a Data Creator role in performing this operation with other software. CDB Studio features the capability to both ingest and generate CDB data stores so this was a natural fit for this application and Hexagon US Federal's participation. The conversion process was developed to align closely with the provided workflow but was slightly altered in specific ways in line with how the application ingests data. Although an exhaustive analysis of the differences between the custom conversion logic and the provided scripts was not performed, where possible any differences are noted in this report. For Experiment 1, the shapefile data was converted by adding the feature geometry and instance-level attributes into a GeoPackage using built-in capabilities in the LuciadLightpseed API. Class attributes were stored in a separate table using the pattern in Option 1b where the CNAM attribute for each feature is a foreign key for the class attributes table. The extended attributes were stored in a separate RTE table as in Option 1d and linked by a mapping table in accordance with the RTE spec.

Experiment 2 – For this experiment the vector feature and attribute information for each tile was converted into a single GeoPackage dataset. The CDB directory structure was maintained with folders under each vector component for the levels of detail and U designation and individual GeoPackages for each R offset.

Experiment 3 – In this experiment all datasets for a specific vector component were combined into a single GeoPackage. The resulting dataset differed slightly from the suggested approach in that the individual tile datasets were not combined into a feature layer in the GeoPackage and instead were kept in separate layers. This facilitated the current architecture of CDB Studio which was built to ingest the data in a tiled manner. It is entirely possible to adjust this pattern to utilize combined features and leverage spatial queries against this larger feature table which might produce an ingest performance increase, but this modification was beyond the scope of the Interoperability Experiment. Class attributes were consolidated for the component into a single table.

Experiment 4 – For the final experiment all the component datasets were further combined to create a single GeoPackage per geocell. As in Experiment 3, the class attributes were combined into a single table per component.

6.5.2. Experiment Methodology - Visualization

The visualization methodology was consistent between all experiments and relied heavily on the existing logic of the CDB Studio application with minor modifications. Standard visualization metrics were recorded and did not differ greatly between the original and the experimental datasets which was expected as the LuciadLightspeed API was designed in the MVC paradigm and keeps the source data and display components independent. Data ingestion was shown to be the more pertinent metric affecting the overall performance of the visualization software. CDB data stores are ingested using a lazy loading strategy to alleviate memory concerns. The current implementation did not utilize the extended attributes so although these were added to the converted GeoPackages they had no impact on the data ingest for visualization. CDB Studio was also designed to match the CDB data stores' tiled architecture and data loads are done as tiles are needed to populate the current display area and scale. Further modifications could be done to improve efficiency with the vector data stored in the combined GeoPackages as in experiments 3

and 4 and produce faster data ingestion but this was out of scope for this Interoperability Experiment.

Visualization metrics were gathered on the initial load of the CDB datasets which involves an animated pan/zoom to the dataset area and an initial display of the data in a view encompassing the entire dataset bounds and at a coarse detail level.

6.5.3. Metrics

Table 10. Yemen

	Shapefile	**Experiment 2**	**Experiment 3**	**Experiment 4**
Time to Convert (s)	N/A	475	392	505
Size on Disk (MB)	52.2	156	61.8	60.2
File Count	9000	1011	24	4
Data Ingest (s)	0.82	0.30	3.07	3.80
Heap Memory Usage (MB)	146	176	192	271
Frames Per Second	170-210			

Table 11. Downtown LA

	Shapefile	**Experiment 2**	**Experiment 3**	**Experiment 4**
Time to Convert (s)	N/A	1626	21436	37113
Size on Disk (MB)	2389.1	1075.2	840.0	839.5
File Count	15198	2533	7	1
Data Ingest (s)	0.26	0.69	134.3	124.3
Heap Memory Usage (MB)	210	144	4368	7061
Frames Per Second	170-240			

6.5.4. Notes on Metrics

Time to Convert – This is the total time to process the conversion of the dataset from the original version containing shapefile data into GeoPackage version. Deletion of the original shapefile data was done as a secondary manual step and was not included in this metric.

Size on Disk – The size on disk was obtained by viewing the Windows Explorer properties window on the Tiles folder of the generated data. It is not inclusive of unmodified data components such as imagery and elevation. The size reported for the original shapefile dataset was a measure of the replaced files.

File Count – Similar to the size on disk, this metric was gathered by viewing the Windows Explorer properties view on the Tiles folder of the generated data and does not include data components that were unaffected by the GeoPackage conversion process.

Data Ingest – Data ingest is the time taken to load all vector data needed the initial display of the CDB data. The visualization logic loads data on demand for the given tiles so this is a subset of the full dataset and could vary greatly depending on the areas and levels of detail being viewed. This metric also only reflects the modified vector datasets.

Memory Usage – Similar to the data ingest metric, the memory usage is a measure of the heap memory footprint of the ingested data repeated across the different sample datasets. The number reported was the delta of general system heap memory utilized before loading the CDB dataset and after. Note that this is not the final memory footprint of the datasets but instead can include temporary data structures used during the ingest process that could later be garbage collected. As such this metric is meant to show the typical memory overhead involved with loading and visualizing a CDB dataset and not the continuing persistent state.

Frames Per Second (FPS) – The frames per second values were gathered by utilizing a diagnostic overlay in the CDB Studio application during the data ingest. This value fluctuates during the tests and a visual inspection of the overlay data was used to determine the typical range of these fluctuations.

6.5.5. Notes and Observations

- The relative times for the dataset conversion differed with the size of the datasets involved. Generating the E3 and E4 datasets for Yemen were roughly on par with or faster than E2's one-to-one conversion, but the large LA dataset showed a vast increase in time for E3 and E4 so the value of the specific method is tied to the expected use case. The E2 dataset with the one-to-one shapefile to GeoPackage conversion was still faster than the original dataset which indicates that this increase involves GeoPackage access and scalability with large datasets.
- The size of the GeoPackage datasets were above that of the original shapefile data. This may be due to overhead and additional metadata in the GeoPackage format as it was much more prevalent on E2's one-to-one conversion.
- Adaptations to the architecture of CDB Studio could further improve efficiency with GeoPackage. Enhancements in the LuciadLightspeeed API could also aid performance, such as built-in RTE handling to minimize JDBC connections.
- Backwards compatibility was built in with minimal effort in the modifications by defining a hierarchy of where to look for vector data. This order was arbitrarily chosen for this Interoperability Experiment, but a suggestion would be to define this order as part of a revision to the CDB standard.

6.6. Guidance

A couple of performance comments (so far):

1. Structure of the data matters. Timing differences in SQL queries on integers rather than strings is enough to matter.

2. As mentioned by others, opening a GeoPackage with lots of tables is slower than having a single table (option 3). .Doing a query to get features out of a very large table is MUCH slower (option 4). I am getting 40x slowdowns for heavily forested areas where I am querying 4700 points out of a table with >2.8M points.
3. The more columns a table has, the larger the slowdown (ie, a query in option 4 vs a query in option3 might take twice as long with 8 columns, but 4 times as long with 30 columns)
    a. Depending on how much time we have left, testing option 1b might be worthwhile. It should yield faster queries to not flatten class-level attributes into the feature table.

Chapter 7. Recommendations, Observations, and Conclusions

This section discusses conclusions and recommendations . There is also a section on the issue of backwards compatability. A number of issues for future discussion and consideration were also identified. These include:

- Forwards compatibility in the CDB context
- How to let an application know what vector data encoding is/are available.
- Versioning and GPKG and CDB 1.2

7.1. Backwards compatibility in the CDB context

7.1.1. FlightSafety observations on backwards compatibility

Our view is that a form of Option 1 is backwards compatible with the existing specification, for the following reasons:

- The CDB concept of a single "logical" file representing a vector file is preserved.
- The CDB conceptual model leans toward mixing the data model and the files on disk. This should be disentangled in a future version of CDB, but it is difficult to do in a minor revision.
- CDB Versioning, where one file from a CDB dataset replaces/hides a file from another CDB dataset, would be difficult to fit together with a GeoPackage that contained "all" the vector files for that dataset. A preferable solution is to evolve the conceptual model to handle this case in CDB 2.0.

7.2. Findings

7.2.1. Aechelon Findings

- For the three Experiment 2 options Aechelon tested, the best outcome in both time and file size came from option 1C.
- For Experiments 3 and 4, speed is slightly improved relative to Experiment 3 sub-option 1D but not sub-option 1C. On the other hand, the resulting storage size is markedly improved when compared against all options in Experiment 2, as would be expected. This is because, by design, these Experiments 3 and 4 go against the spirit of CDB data segmentation by file at the LOD level. This makes it more difficult to remove LODs, if so desired, when copying or exporting the CDB vector data. As such, the approaches used in Experiments 3 and 4 may not be as easy to incorporate and adopt as part of the CDB Standard.

7.2.2. CAE Findings

Although previous experiments with 1-to-1 Shapefile-to-GeoPackage conversion have been very positive, our experiences indicate mixed results. On one hand, we observe substantial benefit in

reducing the number of files stored and loaded, and we also observe the possibility of a comparable (or faster) decode time; but on the other hand, we see potentially-problematic increases in the amount of storage space required.

The negative effects are amplified for CDBs with large numbers of small Shapefiles. The GeoPackage format (especially as outputted by GDAL) is very space-inefficient when it comes to encoding tiles with small quantities of data. (Notably, even if we moved to a 1-tile-per-table instead of a 1-tile-per-file approach, the underlying sqlite format seems to allocate a minimum of 4 KB per table regardless of how little data is in it.)

Please see CAE recommendations

7.2.3. FlightSafety Findings

- Shapefiles are a problematic format for several reasons: -- Shapefiles are a multi-file format to represent a single "logical" file. Management of multi-file formats is always a bit harder than handing a single file. -- The CDB specification's use of extra DBF (DBase III) files for class-level and extended-level attribution is not standard for any current GIS tool, and makes the multi-file format problem worse. -- There are other small-level issues with shapefiles, including limited field name length, limited numbers of attributes, limited data types, and a single geometry type per shapefile. -- In all three test CDB datasets, the number of files greatly exceed the proportion of space on disk. The large number of small files causes problems with efficient disk space usage and slows the distribution of a CDB dataset.
- Replacing a single shapefile with a GeoPackage (Option 1) seems to be a better solution for CDB 1.2 or a future CDB 1.x -- Changes to CDB clients involve less work than other options. -- On average, the performance was better with GeoPackage -- On specific worst case tests, Line Strings and Polygons were faster with GeoPackage. Points were mixed, but likely fast enough for this use case. -- It would also fit within CDB's pre-existing versioning mechanism better than the other options, as it can maintain the single "file" paradigm where a vector file replaces/hides another vector file (even if the "file" is really a multi-file format). -- The class-level attributes would be easy to import into a flattened GeoPackage. Importing extended-level attributes as a related table makes sense as well, although redesigning the odd extended attribute encoding would be recommended. -- The SWG is encouraged to implement a single vector encoding per CDB dataset/version, and a way to discover the vector encoding implemented from a file in the Metadata folder.
- GeoPackage options 3 or 4 would be best suited to a future compatibility-breaking version of CDB 2.0. -- CDB client changes would be more extensive, but the performance gain would likely be worth the effort. -- More detail and additional specification requirements would be necessary to describe how to version CDB datasets when that dataset no longer maintains the "traditional" directory and file format that versioning was designed for. -- The SQL query when loading the vectors that would be contained within a single shapefile can take twice as long as when these vectors are in a single GeoPackage, but the open cost can be amortized across many queries if the GeoPackage files remain open for a longer period of time.
- The OGC Interoperability Experiment was a good testing ground to develop familiarity with GeoPackage and to help the SWG come to a better consensus on the use of GeoPackage. We would recommend having more IE's as other parts of the specification are reviewed and improved.

7.2.4. Hexagon/Luciad Findings

Standard Visualization Metrics - Overall visualization performance was similar between all experiments for each dataset. Converting to GeoPackage has no significant impact either positive or negative on visualization once the data is loaded.

Data Ingest - Data ingest for visualization was highly dependent on the size and structure of the underlying GeoPackage datasets and the architecture of the data ingest code. Both load times and heap memory usage were much greater for the combined GeoPackages on large datasets. While major adjustments to the GeoPackage interactivity was beyond the scope of this IE minor modifications to reduce file I/O on the combined datasets showed potential for improvement in data loading time.

Data Conversion - For data conversion the best results were also obtained from Experiments 3 & 4. The caveat here is that some memory issues were encountered with the provided datasets as more layers and data were added to a single GeoPackage which introduces questions of scalability for these approaches.

7.3. Recommendations

This section describes key recommendations from each of the participating organizations. A short summary of the recommendations is provided is the Summary Section of this Engineering report.

7.3.1. Aechelon

To achieve the improvements in storage while also maintaining the speeds comparable with sub-option 1C and addressing the file-per-LOD issue, Aechelon recommends two additional experiments: (a) where each component selector of each LOD is in its own geopackage file---effectively a variant of sub-option 1C where the U and R references of the same component selectors are combined into one file; and (b) where each dataset's LODs are in a separate geopackage file---effectively a variant of Experiment 3 where instead of storing each LOD in a separate layer in the same geopackage file, each LOD is a separate file.

If Aechelon were to recommend only one processing alternative, among those in this experiment, for inclusion as an alternate primary vector format in a future OCG CDB revision, it would be option 1C in Experiment 2.

7.3.2. CAE Recommendations

Suggestion 1: Storage versus Transport Format

We have essentially been using GeoPackage as a data *transfer* format in this experiment, when its design seems to be much better used as a data *storage* format. The GeoPackage format encodes indexes, triggers, sequences, and metadata as well as feature data, and it also supports real-time addition, removal, and update of records. In this experiment, we make use of none of these features.

We suggest considering GeoJSON as a candidate format for the 1-to-1 Shapefile conversion case (Option 1a). We may, with GeoJSON, obtain the same reduction in file count while simultaneously

obtaining better performance and storage characteristics for CDBs with large numbers of small vector files. To obtain scalable performance characteristics with GeoJSON, we will want to base implementations on RapidJSON or similar parser (rather than libjson, which is currently used by GDAL).

It is worth noting here also that GeoJSON supports other variants of the Option 1 experiment, e.g., attribute flattening (Options 1c/1d).

Suggestion 2: GeoPackage as an Incremental Data Store Version

In this experiement, we have explored the idea of placing GeoPackages *inside* a CDB. We suggest that this may not be the best approach for maximizing compatibility. GeoPackage, like CDB, functions conceptually as an independent data store. We would like to raise the possibility of using GeoPackage as an incremental data store *version*, which would essentially allow a GeoPackage to replace a CDB version at its root (at least to the extent that all data inside the CDB can be converted losslessly into GeoPackage data). The idea is to be able to add a GeoPackage as an incremental version without modifying the underlying CDB, or vice versa. What we would have to do in this case is define a bidirectional equivalency between a CDB directory path and a GeoPackage/sqlite index—this would allow us clearly-defined semantics for mixing and matching GeoPackage and CDB data stores, with minimal impact on existing standards and implementations.

Suggestion 3: The CDB Directory Hierarchy as a Key-Value Store

We would like to raise a particular opportunity for future-proofing the standard. Conceptually, the CDB directory hierarchy functions as an index: any given directory path is essentially a key, and the value accessed by the key is a file. If we introduce a level of abstraction that allows us to discuss the CDB as a type of key-value store, then we open up a range of new possibilities in terms of physical implementation. For example, there are any number of database engines that are able to function as fast key-value stores, from lightweight mobile solutions like sqlite to highly-distributed cloud-capable NoSQL solutions like MongoDB. This would conceptually simplify the idea of a geographic database, allowing implementors more freedom to choose the storage technology that best suits them while simultaneously providing a natural path toward remote/Internet query of CDBs.

7.3.3. FlightSafety Recommendations

GeoPackage Recommendations

- From our experience, GDAL's GeoPackage driver will create non-standard GeoPackage files. The SWG needs to decide if that is allowed, or if a stricter GeoPackage implementation is required.
    - GDAL defaults to using "fid" as the primary key in geometry tables. GeoPackage specifies "id"
    - As the conversion scripts use GDAL to convert geometry from shapefile to GeoPackage, GDAL allows the creation of MultiPolygon and MultiLineString features in Polygon and LineString tables (respectively). When this happens, GDAL emits a warning that it is not creating standard GeoPackage files. We would recommend standardizing with GeoPackage, in only allowing a single geometry type within a table.
    - The conversion from shapefile logical fields to GeoPackage should be standardized. It would be best to convert any CDB logical field (whether it was logical, string, or integer) into a GeoPackage boolean field.

    - The table name should include enough information to be unique, no matter which option is implemented.
- The dataset/component selectors/lod/up/right values must be stored in the table. Integers are recommended for storage for better performance.
- If the SWG decides on using option 3 or 4 for a future version of CDB, then index search tables should be required for better performance when querying data from a specific CDB Tile-LOD.

7.3.4. Hexagon/Luciad Recommendation

Further investigation into the optimal structure of combining multiple sub-datasets (rail, hydrography, etc) and multiple levels of detail into a single GeoPackage could alleviate several encountered performance drawbacks with the CDB datasets in Experiments 3 & 4. At this point without this investigation and guidance Hexagon US Federal recommends the CDB architecture presented in Experiment 2 which provides both a reduction in the dataset's file count and minimizes the impact to existing CDB applications accessing data in the current tiled directory structure.

1. Forwards compatibility in the CDB context
2. How to let an application now what vector data encoding is/are available.
3. Versioning and GPKG and CDB 1.2

Appendix A: Abstract Test Suite

An Abstract Test Suite may be relevant to an Engineering Report.

An Abstract Test Suite is specified in Clause 9 and Annex A of ISO 19105. That Clause and Annex specify the ISO/TC 211 requirements for Abstract Test Suites. Examples of Abstract Test Suites are available in an annex of most ISO 191XX documents, one of the more useful is in ISO 19136. Note that this guidance may be more abstract than needed in an OGC® Implementation Standard.

NOTE	We skip level 2 headers so that asciidoc correctly numbers the subsections in the appendix.

A.1. Test module for conformance level 1

A.1.1. Conformance level 1

Test identifier	/test/case/id
Test purpose:	Confirm that the IUT satisfies all applicable requirements for conformance level 1.
Test method:	Functional testing performed in an automated and/or manual manner. Verify the behaviour of the IUT for the following operations: • GetCapabilities (mandatory) • DescribeRecord (mandatory) • GetRecords (mandatory) • GetRecordById (mandatory) • GetRepositoryItem (mandatory) • GetDomain (optional)
Requirement:	OGC 07-110: cl. 2.2
Test type:	Capability

A.1.2. Test case for validity of XML response entity

Test identifier	http://www.opengis.net/spec/xxx/conf/WRS.General-ValidResponse
Test purpose:	The XML response entity is valid.
Test method:	Validate content of response entity against corresponding element declaration.
Requirement:	OGC 07-006r1: cl. 10.2.5.1, p. 118
Test type:	Capability

A.2. Test module for conformance level 2

A.2.1. Conformance level 2

Test identifier	/test/case/id
Test purpose:	Confirm that the IUT satisfies all applicable requirements for conformance level 1.
Test method:	Functional testing performed in an automated and/or manual manner. Verify the behaviour of the IUT for the following operations: • GetCapabilities (mandatory) • DescribeRecord (mandatory) • GetRecords (mandatory) • GetRecordById (mandatory) • GetRepositoryItem (mandatory) • GetDomain (optional)
Requirement:	OGC 07-110: cl. 2.2
Test type:	Capability

A.2.2. Test case for validity of XML response entity

Test identifier	http://www.opengis.net/spec/xxx/conf/WRS.General-ValidResponse
Test purpose:	The XML response entity is valid.
Test method:	Validate content of response entity against corresponding element declaration.
Requirement:	OGC 07-006r1: cl. 10.2.5.1, p. 118
Test type:	Capability

Appendix B: Revision History

Table 12. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
January 26, 2019	C. Reed	.1	all	Cloned initial version
March 2019	C. Reed	.2	all	Added various overview sections
May 2019	C. Reed	.3	all	Added in experiment results

Appendix C: Bibliography

bibliography::[]