Open Geospatial Consortium, Inc.

Date: 2011-11-16

Reference number of this document: OGC 11-114

Category: Public Engineering Report

Editor(s): David Danko, Lance Shipman, Paul Ramsey

OGC® OWS 8 Bulk Geodata Transfer with File Geodatabase

Warning

This document is not an OGC Standard. This document presents a discussion of technology issues considered in an initiative of the OGC Interoperability Program. This document does not represent an official position of the OGC. It is subject to change without notice and may not be referred to as an OGC Standard. However, the discussions in this document could very well lead to the definition of an OGC Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	OGC® Public Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
Document language:	English

License Agreement

Preface

This document describes the File Geodatabase API and documents interoperability testing performed during OWS-8 Geosyn-Bulk data transfer testing. It also documents how the File Geodatabase API addresses the requirements specified in the OWS 8-RFQ-CFP.

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

This is the first version without track changes. The changes to be made in this document version should be tracked by Microsoft Word. If you choose to submit suggested changes by editing this document, make your suggested changes with change tracking on.

Contents

Page

OGC® OWS 8 Bulk Geodata Transfer with File Geodatabase

1 Introduction

1.1 Scope

This document provides an overview of the File Geodatabase API and documents the testing performed in the OWS 8 Testbed.

1.2 Document contributor contact points

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

Name	Organization
David Danko	Esri
Lance Shipman	Esri
Paul Ramsey	OpenGeo

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2011-07-28	0.0.1	DD,LS,PR	All-	Initial creation
2011-08--18	0.01	DD,LS,PR	7.	Added: File Geodatabase Interoperability testing and Implementation Clauses
2011-09-06	0.01	DD	6	Updated material

1.4 Future work

This initial IP program tested the interoperability of using the File Geodatabase API to transfer bulk data between to independent platforms. Future work is required to address the remaining requirements in the OWS-8 RFQ-CFP namely the use of topology rules and ISO 19139 metadata.

1.5 Forward

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 Inc. 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.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OGC OWS-8 RFQ/CFP Annex B OW-8 Architecture

ISO 19115:2003
http://www.iso.org/iso/catalogue_detail.htm?csnumber=26020

ISO/TS 19139
http://www.iso.org/iso/catalogue_detail.htm?csnumber=32557

NSG Metadata Foundation (NMF) - Part 1 (v2.0.0)
https://www.gwg.nga.mil/protected/focus_groups/mfg/documents/NMF_%20v1.5.pdf

NSG Metadata Implementation Specification (NMIS) - Part 2 (v2.0.0)
https://www.gwg.nga.mil/protected/focus_groups/asfe/documents/NMIS_Part_2_v1.5.0_draft.pdf

North American Profile of ISO19115:2003 - Geographic information – Metadata (NAP – Metadata, version 1.2.1), http://www.fgdc.gov/standards/projects/incits-l1-standards-projects/NAP-Metadata

DGIWG profile of ISO 19107 – ref

3 Terms and definitions

For the purposes of this report, the definitions specified in OpenGIS® Abstract Specification Topic (w ISO 19107) shall apply. In addition, the following terms and definitions apply.

3.1
Application program interface
set of rules ('code') and specifications that software programs can follow to communicate with each other

4 Conventions

4.1 Abbreviated terms

7zip	an open source file archiver with a high compression ratio
API	Application Program Interface
FGDB	File Geodatabase
ZIP	Windows Zip utility for file compression, file sharing, file encryption, and data backup

5 ER Topic overview

This ER Topic addresses the use of Esri's widely used File Geodatabase and the newly developed, openly available, unencumbered File Geodatabase API for "Bulk Geodata Transfer." The File Geodatabase and the open API seemed a perfect fit with the requirements laid out in the OWS-8 RFQ. It provides a compact and efficient technology for transmitting very large datasets, along with extensive metadata and topology rules (if provided) to guarantee logical consistency and an interoperable understanding of the transmitted data. File Geodatabases can be segmented into multiple datasets and support indexing as called for in the RFQ. The testbed provided an opportunity to test the new open API for interoperability outside of the Esri community.

6 File Geodatabase API for Bulk Transfer

6.1 Introduction

A File Geodatabase is a collection of GIS datasets held in a file system folder. An API is a specified application that a software program uses to access and make use of the resources provided by another application that implements that API. The File Geodatabase API provides a non-proprietary means by which anyone can work with File Geodatabases. The File Geodatabase API is C++ based and provides the ability to perform the following tasks:

- Create, Open and Delete file geodatabases
- Read the schema of the geodatabase
    - All content within a geodatabase can be opened for read access
- Create schema for objects within the simple feature model

- Tables
- Point, Line, and Polygon feature classes
- Feature datasets
- Domains
- Subtypes

- Read the contents of datasets in a geodatabase
  - All dataset content within a geodatabase can be read
- Insert, Delete and Edit the contents of simple datasets:
  - Tables
  - Point, Line, Polygon, Multipoint, and Multipatch feature classes
- Perform attribute and (limited) spatial queries on datasets
  - Spatial queries will be limited to the envelope-intersects operator
- Navigate relationships and work with Attachments

Data in a File Geodatabase can be optionally stored in a read-only compressed format to reduce storage and transmission requirements.

6.1.1 Benefits

The goals of the File Geodatabase are to do the following:

- Provide a widely available, simple, and scalable geodatabase solution for all users.
- Provide a portable geodatabase that works across operating systems.
- Scale up to handle very large datasets.
- Provide excellent performance and scalability, for example, to support individual datasets containing well over 300 million features and datasets that can scale beyond 500 GB per file with very fast performance.
- Use an efficient data structure that is optimized for performance and storage. File geodatabases use about one-third of the feature geometry storage required by shapefiles and personal geodatabases. File geodatabases also allow users to compress vector data to a read-only format to reduce storage requirements even further.

- Outperform shapefiles for operations involving attributes and scale the data size limits way beyond shapefile limits.

6.2 OWS-8 Bulk transfer requirements

The requirements for Bulk Transfer are outlined in the OWS-8 RFP/CFP Annex B and summarized here: *"Evaluate, investigate and demonstrate a method for distributing geospatial data from a source system via a mechanism enabling that data to be efficiently ingested in another system. In this scenario, the geospatial data is typically stored in a native format geospatial database, and the mechanism for transfer will either be a WFS response, a data file, or a file system-like folder or container. The geospatial data may either be transmitted "in bulk" over a communications infrastructure or distributed via hard media using "sneaker-net" methods. The communications infrastructure may be either high-bandwidth or very constrained bandwidth - including frequent unreliable connectivity."*

Key technical requirements include:

1. the ability to represent a maximal amount of the source geospatial content with **no loss in translation** to the content-exchange format/container

2. **segmentation** of the content-exchange format/container to allow different types of content-components to be extracted or ignored

3. effective **file-size minimization** of the content-exchange format/container (including employing compression technology as appropriate)

4. the assurance of **integrity** in content transmission."

The RFQ also addresses investigating the following requirements for the bulk transfer of data:

ISO 19115 conformant metadata describing the data incorporated in the transfer. In particular the NSG Metadata Foundation (NMF) Part 1 and NSG Metadata Implementation Specification (NMIS) Part 2;

The use of check-sum to support data integrity assurance during transfer

Topology- preferable based on the DGIWG Profile of ISO 19107

Indexing capability allowing for direct random access to individual item of content

6.3 File Geodatabase and RFQ requirements

6.3.1 Summary

- The primary goal of the testbed was to show the ability to represent a maximal amount of the source geospatial content with **no loss in translation** to the content-exchange format/container
  - Although on the initial execution of the openGeo utility there were some minor discrepancies working together in the testbed these were fixed and final transfers of data through a round trip were examined and no discrepancies were found. (see 7 below).
- **segmentation** of the content-exchange format/container to allow different types of content-components to be extracted or ignored
  - A File Geodatabase can contain multiple feature, raster, and attribute datasets. It also supports metadata at the database level, the dataset level for each dataset, and the feature class level (see metadata below)
- effective **file-size minimization** of the content-exchange format/container (including employing compression technology as appropriate)
  - File Geodatabases are fairly efficient before additional compression for example fgdb use only 1/3 of the storage for geometry than shapefiles for example. With the File Geodatabase API all coordinate strings are automatically compressed making it extremely efficient for datasets with large numbers of linear and polygonal features.
  - The testbed looked into commonly used compression routines to even further reduce file size with excellent results (see 6.4 below)
- the assurance of **integrity** in content transmission."
  - The use of checksum was discussed. The testbed determined that 7-ZIP provided checksum as well as efficiently compressing the data.
  - The use of topology or topology rules help maintain data integrity if there is any movement of features in the transmission of the data; if the relationship between features and their ranking (which feature should be adjusted) is known they can be adjusted back to their proper relationship.

6.3.2 Topology rules

One of the requirements in the RFQ was the ability to incorporate topology in the transfer files primarily for the purposes of ensuring data integrity. The File Geodatabase API does not provide for the exchange of explicit topology; it does, however, provide for the transfer of the topology rules for the feature classes in the dataset. This provides for huge efficiency in file size for the exchange of data, as well as still providing the topology information and ensuring the integrity of the information. The topology rules for associated feature types are carried within in the File Geodatabase.

During the testbed the following was discussed:

The File Geodatabase API provides access to the topology rules. The function "getRelatedDataSets()" will return all topology information for a feature data set (including topology rules).

Although the round trip in this testbed did not modify/drop any, or corrupt the data, sometimes not all transfers will do as well due to round-off error, system, hardware, and software differences. In this case sometimes a feature might be moved slightly (still within accuracy tolerances) putting their relationship with other features in doubt. In this case the use of topology or topology rules helps; if the relationship between features and their ranking (which feature should be adjusted) is known they can be adjusted back to their proper relationship.

One of the issues that developed during the testbed was that the feature classes to which the topology rules apply were identified by a unique identifier (GUID) and not by feature class name ("transportation curves" for example). A routine was developed which mapped the GUIDs to feature class names and created a separate XML document. A sample XML file for the Topology information for the LTCDS_3_0wTopoRules is provided here https://portal.opengeospatial.org/files/?artifact_id=43811

There is also the capability to provide more information on the topology and feature class relationships; Esri's Defense Mapping product generates an xml file with the topology rules, cluster tolerances, and ranking (which feature class can move vs another in a relationship). In this particular case the tolerances, ranking and topology rules are based on the NGA LTDS extraction rules (and some common sense rules). So if a Vegetation FC overlapped a coastline the vegetation should be adjusted because it is the more ambiguous feature.

Topology rules used by the FileGeodatabase are provided at:
http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//001t000000sp000000.htm and;
http://help.arcgis.com/en/arcgisdesktop/10.0/help/001t/pdf/topology_rules_poster.pdf

6.3.3 Metadata

Geographic information, of course, is only as good as what users know about it; using data without knowing its accuracy, its currency (date of validity), where it came from, who produced it and how, etc can be disastrous. All data used for important decisions should be well documented with metadata. This metadata should be included with the data so that it travels with the data and is always available. Incorporating standardized metadata with the bulk data transfer is one of the requirements in OWS-8. File Geodatabase supports carrying full metadata at the Database level, Dataset Level, and at the Feature Class Level. The metadata is carried as an integral part of the data in ArcGIS Metadata XML. ArcGIS Metadata is a simple XML format incorporating all the combined metadata elements required to support multiple metadata standards and profiles. The File Geodatabase API is designed to read and write whatever xml metadata is provided. This metadata can then be viewed through a style sheet in whatever standard form the user desires as long as the particular metadata elements of interest have been stored in the metadata. The metadata XML in a File Geodatabase does not have to be ArcGIS Metadata – it can be FGDC XML, ISO 19139 XML, or in any XML format – the FileGeodatabase will transfer it. This testbed only looked at the transfer of ArcGIS metadata, and viewing it in an ISO 19115 styling. Future testbeds may want to investigate the adaption of the API, adding an XSLT to "translate" the metadata from ArcGIS

metadata to ISO 19139 metadata or into storing the metadata in ISO 19139 directly in the File Geodatabase.

6.3.4 Physical API vs Document API

The File Geodatabase API is provided as a freely available run-time application that software developers can use to provide their products the ability to read, write, or create a File Geodatabase. Most OGC API specifications are documents which state the requirements and identify objects developers must develop/use to create a proper interface. These written instructive requirements are subject to misinterpretation. The software encoded instructions perform the same no matter who is implementing them.

To help developers meet requirements the written specifications come with conformance tests to help developers understand if they have met the requirements. However as we have found out in OGC just because two products implementing a specification and pass the conformance test doesn't guarantee they well be able to interoperate with each other without a lot of direct interaction to work out the different interpretations of the specification. With a runtime API when two developers meet conformance tests they are more likely to interoperate.

6.4 File Geodatabase Compression

Currently ArcGIS file geodatabase compressed data is not supported by the File Geodatabase API. There are plans to support reading compressed data in a future release. Writing file geodatabase compressed data has never been part of the API design and may be difficult. ArcGIS File geodatabase compression is at the file level. The file geodatabase is still a collection of files on disk, not a zipped folder. You don't write to a compressed file geodatabase table, you compress an existing table. Compressed data can be drawn and queried. It cannot be edited.

However, with the File Geodatabase API all coordinate strings are automatically compressed improving the efficiency for datasets with large numbers of linear and polygonal features. The API uses the Coordinate reference/projection, resolution/scale information to create coordinate off-sets rather than carrying complete coordinates for all vertices.

In the testbed we used 7ZIP to compress and package the file geodatabase. This is a better alternative than using File Geodatabase compression. We found that7ZIP compresses File Geodatabases efficiently. Some results during the testbed:

File name	Original	ZIP	7z
LTDS_3_0wTopoFeotypical.gdb	21.5 MB	3.2 MB (ZIP)	2.32.MB (7z)
MX_TDS_3_Data (four Levels TDS data)	98.1 MB	14.6 MB (ZIP)	8.8 (7z)
*GTDS_3_0.gdb	2.58 MB	0.32 MB ((ZIP)	0.28MB (7z)
**NHDH_21006.gdb	1.7G	568M (ZIP)	

* this dataset had a lot of empty feature classes and didn't benefit as much from coordinate string compression.

**Loading the whole thing into PostGIS took about 15 minutes
And there were some very big layers in there, adding up to about 7.5M features:

NHDFlowlineVAA 1248452
NHDFeatureToMetadata 1887691
NHDReachCode_ComID 864996
NHDReachCrossReference 1205044
NHDFlowline 1248452
NHDWaterbody 1246000

6.5 Check sum investigation

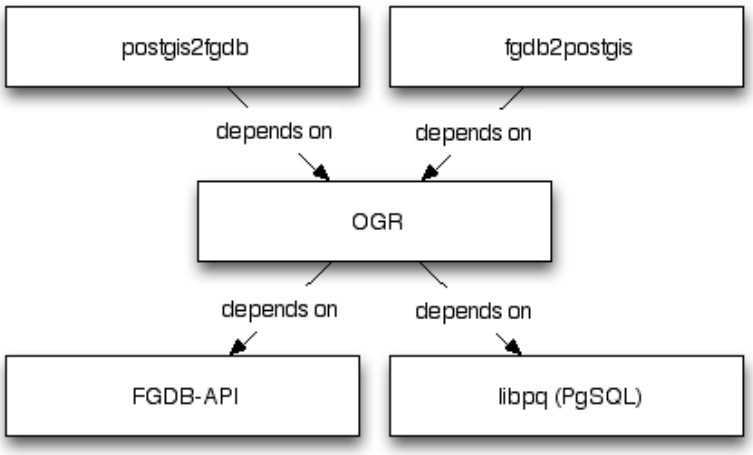
The File Geodatabase API is designed to provide Read/write/query access to simple features in a file geodatabase. It does not currently include any utilities for data validation. Data validation can be achieved through the use of a check sum. We did an investigation and found that none of the archival software packages included a checksum internally in their zip files. 7zip allows the generation of a checksum of a folder. Since File Geodatabases can contain hundreds of files per dataset it was determined in this case it would be best to perform a checksum at the folder level as opposed to, for example, for just the file containing the coordinate information. A checksum at the folder level would be able to identify any corruption in the transmission of the data. Since 7zip has great compression it can be used to generate a check sum then use 7zip to verify that the correct data was received following a transmission. The checksum value would have to travel separately to provide effective security. Alternately a checksum could be generated using a standalone open source program like "FileVerifier++" which can be used to generate a checksum at both ends. Since we are modifying the data as part of our demo, the check sum would not match the data used in the round trip. It could be used to verify that the data that Esri sent to openGeo arrived correctly and that the data sent by openGeo correctly made it to Esri.

7 File Geodatabase Interoperability testing and Implementation

7.1 Introduction

The File Geodatabase API (FGDB-API) was tested for interoperability by building a tool for loading and unloading data from an FGDB file into a PostgreSQL/PostGIS spatial database server. The goal was to produce a new FGDB file, as the result of converting the data from FGDB, to PostGIS, and back again. The exercise required understanding the details of FGDB construction and exercised the FGDB-API in terms of providing sufficient access to low/level read and write controls to produce a perfect copy.

Rather than writing the utility directly using just the FGDB-API and the PostgreSQL client API (libpq), we wrote the utility using an extra abstraction layer, the OGR[1] simple features library. Using the OGR library made some of the development simpler, by providing existing utilities for working with spatial reference systems (SRS) and binary formats, as well as a mature framework for manipulating PostgreSQL/PostGIS data. And it made some of the development more complex, by superimposing an extra abstraction layer between the FGDB data model and the PostGIS data model.



However, the primary benefit of carrying out the development using the OGR library is that the work done to support better FGDB reading and to implement FGDB writing in OGR became part of the widely used OGR library. During development, third parties unconnected with OWS-8 (primary US Army Corps CERL staff and contractors) tested the work in progress and provided valuable feedback. The work completed for OWS-8 will be used by others and improved upon over time, which would be far less likely if the development had been done as a single-purpose FGDB-to-PostGIS toolset

7.2 Implementation

The development effort had two streams: creating the **postgis2fgdb** and **fgdb2postgis** command-line programs; and, upgrading the OGR FGDB driver to the point where it could fulfill the mandate of a perfect round-trip from FGDB to PostGIS to FGDB.

The OGR upgrade was the heart of the work. At the start of the project, the FGDB driver included basic read support, but was extremely slow to start up on a complex file. The OGR development included the implementation of

A new layer reading method to more quickly list the FeatureClasses inside an FGDB file

- A DataSource::CreateLayer method
- A Layer::CreateField method
- A Layer::CreateFeature method

[1] http://www.gdal.org/ogr

- Assorted utility methods (StringToWString, WStringToString, Layer::GetTable, Layer::GetLayerXML, Layer::GetLayerMetadataXML, etc)

The primary work was in the CreateLayer and CreateFeature methods.

CreateLayer required generating an appropriate XML layer definition to feed the FGDB-API, which meant working back from the example XML files to the minimum XML file necessary to successfully create an FGDB table. Finding the correct permutations took some time.

CreateFeature required adding code to the core OGR library to generate the FGDB ShapeBuffer, which is a binary geometry representation common to a number of ESRI formats (shape files, personal geodatabases, file geodatabases). While OGR already had some ShapeBuffer generation code in the shape file handler, it was not generalized, so a new implementation was added to match the existing implementation for reading the shape buffers.

When the core writing code was working sufficiently for testing, the OGR build chain was updated to allow the FGDB driver to be easily built using the standard GNU tools. At this point third parties began testing the driver more heavily. It has now been tested using both command-line tools such as **ogr2ogr** and **ogrinfo** as well as web services like **MapServer**.

The command-line utilities, **postgis2fgdb** and **fgdb2postgis** were straight-forward implementations of an OGR translator, with a few exceptions.

- The PostgreSQL driver is created with extra options to ensure the geometry columns, fid columns, and column name cases are written with fidelity to the original file.

- The **fgdb2postgis** utility, in addition to looping through the OGR layers and features and writing them across, also writes an **fgdb_metadata** table directly into the database, reading the table XML from the FGDB file and storing it in PostgreSQL.

- The **postgis2fgdb** utility, in addition to looping through the OGR layers and features and writing them across, reads from the **fgdb_metadata** table (if it exists) and uses that information to preserve the geometry precision model of the original file (more on that below).

7.3 Implementation Issues

Through development of the round-trip utilities, and pushing the data through the OGR library, some issues received extra attention because they weren't expected or showed up as inconsistencies when testing the output data.

7.3.1 Date Start/Ends

The FGDB-API Row.GetDate and Row.SetDate methods operate on a **struct tm** (from **time.h**) filling out the **tm** fields. As documented in time.h, the tm year is stored as an offset from 1900 and the month number is zero-based. As a result, the OGR driver has to add 1900 to the year, and 1 to the month to fit the time into the OGR model of time before returning (and do the reverse when writing).

7.3.2 UTF16 vs UTF8

The OGR internal model for strings was originally undefined (yes really), but has moved recently to a UTF8 standard. The FGDB-API states that the internal model for FGDB is strictly UTF16 (except for XML documents, which are UTF8). So all column names and string data have to be pushed through a transcoding. This is accomplished with the StringToWString and WStringToString utility methods, which just wrap existing transcoders in the OGR library.

7.3.3 Field Order

Both OGR and FGDB-API will respect the field order generated by repeated calls to their respective field adding methods. In testing the output of the process it was discovered that the output had mis-ordered fields. This was eventually tracked back to the field *reading* code in the OGR PostgreSQL driver, and patched[2].

7.3.4 FeatureClass Extent

When writing to an FGDB file, the extent of the feature class is not necessarily automatically updated. The postgis2fgdb utility works around this problem with a call to the Table.LoadOnlyMode method at the start and end of writing. This seems to cause the extent to be generated (perhaps as a side effect of updating the spatial index).

7.3.5 Long Start-up Times

Users of the OGR FGDB driver will find that the initial time to connect to an FGDB file with a large number of layers seems quite high. That is because the connection not only opens the file handle, but under the covers in OGR reads out what layers are available inside the file. The reading of datasets in the FGDB file is slow at the FGDB-API level. Calling the GeoDatabase.GetChildDatasets method on a file with a large number of FeatureClasses can take several seconds. This effect was only observed on the Linux 32-bit platform, other platforms were not tested.

Other aspects of the FGDB-API performed very well. The **fgdb2postgis** tool was used in on separate OpenGeo project to load a multi-million record USGS NHD file into PostGIS and did so in only a few minutes.

---

[2] http://trac.osgeo.org/gdal/ticket/4194

7.3.6 Geometry Precision

The greatest mystery exposed during development was around coordinate precision. The first prototype could round-trip the data, but the coordinates were being returned with slightly different values than the originals.

Coordinates are stored in PostGIS in full double-precision, and the coordinates in PostGIS exactly matched the coordinates in the source FGDB file. However, the coordinates in the target FGDB file did not match. That implied that either the export from PostGIS was altering them, or that the writing to FGDB was altering them. The answer was the latter.

Though FGBD stores coordinates in double precision, they are not written unaltered. Every coordinate is conditioned by a precision model, which is attached to the spatial reference system of the FeatureClass.

```
<SpatialReference xsi:type="esri:ProjectedCoordinateSystem">
  <XOrigin>-16987000</XOrigin>
  <YOrigin>-8615900</YOrigin>
  <XYScale>10000</XYScale>
  <ZOrigin>-100000</ZOrigin>
  <ZScale>10000</ZScale>
  <XYTolerance>0.001</XYTolerance>
  <ZTolerance>0.001</ZTolerance>
  <HighPrecision>true</HighPrecision>
  <WKID>54030</WKID>
</SpatialReference>
```

The model defines an origin point (XOrigin, YOrigin, ZOrigin), and a rounding factor (XYScale, XScale). The scales are inverses of the precision, so a scale of 100 implies a precision of 0.01. The model also defines values for tolerance, which are not used directly by FGDB, but are used by ESRI tools (ArcMap, etc) to control things like snapping in editing or topology tests.

In order to write an output FGDB file with coordinates that exactly match those of the inputs, the model information needed to be stored in the PostGIS database. Since there is no slot for this information in the existing OGC SFSQL standard that PostGIS implements, we created a table to hold the extra FGDB information.

```
CREATE TABLE fgdb_metadata (
  tablename varchar primary key,
  table_xml varchar,
  metadata_xml varchar
)
```

FGDB expects table names to be unique, so we can use table name as the primary key. There are slots to hold: the table XML definition, which includes all the field information as well as the spatial reference system and precision model; and, the metadata XML, which holds the optional FeatureClass metadata (name, description, provenance, etc).

The information from fgdb_metadata is used to create the new FeatureClasses in the FGDB file to exactly match the ones in the source file. In the event that there aren't rows in the metadata for a table (if, for example, a new table was created in PostGIS that didn't exist in the source file) the new FeatureClass is created with default precision values that try to retain as much of the full double precision as possible.

7.3.7 Spatial Reference Systems

FGDB FeatureClasses are very good about having spatial reference system (SRS) definitions. The SRS definitions use well-known text (WKT) and/or well-known identifiers. The identifiers are from the ESRI collection, which match the EPSG numbers for the lower ones, but include ESRI specific numbers in a higher range. Since PostGIS uses a similar scheme of aping the EPSG numbers, the identifiers are just mapped straight across in our utilities.

The ESRI well-known text deviates from the OGC standard in a few places (some parameter names, some projection names) but the OGR library infrastructure includes a facility for flipping back and forth between the representations (the morphFromESRI() and morphToESRI() methods on the SRS object) which we use in the conversion process.

7.3.8 Type Mappings

The FGDB type model is mapped to the OGR type model in the table below (the preferred ESRI types when converting from OGR are noted with an *). Note that many ESRI types are mapped to the same OGR type. The implication is that in a round-trip, exact type fidelity will be lost as, for example, an esriFieldTypeSmallInteger is mapped to an OFTInteger on the way into OGR and back to an esriFieldTypeInteger on the way out.

ESRI Type	**OGR Type**
esriFieldTypeSmallInteger	OFTInteger
esriFieldTypeInteger*	OFTInteger
esriFieldTypeSingle	OFTReal
esriFieldTypeDouble*	OFTReal
esriFieldTypeGUID	OFTString
esriFieldTypeGlobalID	OFTString
esriFieldTypeXML	OFTString
esriFieldTypeString*	OFTString
esriFieldTypeDate*	OFTDateTime
esriFieldTypeBlob*	OFTBinary
esriFieldTypeOID	OFTInteger
esriFieldTypeGeometry*	OGRGeometry

esriFieldTypeRaster	N/A

In general the ESRI field type definitions are more granular than the OGR types. The OGR types that cannot be converted into ESRI types are arrays of the simple types: OFTIntegerList, OFTRealList, OFTStringList.

7.3.9 Geometry Mappings

The OGR geometry model is a full simple features (SFS) model, with the basic types (POINT, LINESTRING, POLYGON) and the aggregate types (MULTIPOINT, MULTILINESTRING, MULTIPOLYGON) and an anonymous aggregate type (GEOMETRYCOLLECTION).

The FGDB model is the same as the original ESRI shape model[3], with extensions[4] added since for higher dimensionality and patches. There is a Point, a MultiPoint, a PolyLine, and a Polygon (and dimensional Z/M variants of each). There are also new GeneralPolyLine, GeneralPolygon, GeneralPoint, and GeneralMultipoin variants that support curved interpolations as well as simple ones.

The OGR implementation only supports the original types, or general types that do not include curved components.

The ESRI FGDB Polygon is actually, in SFS terms, a MULTIPOLYGON, so when converting from OGR to FGDB, the non-aggregates are fluffed into single-member aggregates. When converting from FGDB to OGR, it is only possible to create SFS aggregate types.

ESRI Type	**SFS Type**
esriGeometryPoint	POINT
esriGeometryMultiPoint	MULTIPOINT
esriGeometryLine	LINESTRING
esriGeometryPolyline	MULTILINESTRING
esriGeometryPolygon	POLYGON
esriGeometryPolygon	MULTIPOLYGON*
N/A	GEOMETRYCOLLECTION

[3] http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[4] http://www.giser.net/wp-content/uploads/2011/01/extended-shapefile-format.pdf

7.3.10 FGDB SubTypes

The FGDB table definitions include the concept “sub-types”, enumerations that can restrict the values in a column. In the PostgreSQL database these would be modeled as either foreign-key tables or possibly (for cases with fewer values) domains. However, since the OGR abstraction has no concept of sub-types or domains, sub-types are not handled in the **fgdb2postgis** or **postgis2fgdb** translation utilities.

7.3.11 FGDB Relationships

Similarly, the FGDB includes the concept of “relationships” between tables, which are similar to the PostgreSQL foreign key constraints. There are probably sufficient differences that not all FGDB relationships could be encoded in the database, but many could. However, since the OGR abstraction has no concept of relationships or foreign keys, so they are not handled in the **fgdb2postgis** or **postgis2fgdb** translation utilities.

7.3.12 Using OGR

In the end, the abstraction layer of OGR was limited to simple features (SFS) so not all the information in the FGDB could be copied across to PostgreSQL. However, the benefit of doing this work in OGR in terms of seeding the community of third party applications that can use FGDB files is well worth it. With this work, FGDB will be usable by MapServer, Mapnik, QGIS, and any other application that uses OGR for GIS file access.

Bibliography

[1] Guidelines for Successful OGC Interface Standards, OGC document 00-014r1