Open Geospatial Consortium

Date: 2011-12-19

http://www.opengis.net/doc/ows8-aixm-compression

Reference number of this document: OGC 11-097

Category: Public Engineering Report

Editor(s): Jérôme JANSOU (AtoS) / Thibault DACLA (Atmosphere)

OGC® OWS-8 AIXM 5.1 Compression Benchmarking

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

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

Preface

This OGC document presents the results of a compression benchmarking campaign of various algorithms applied on AIXM 5.1 files, with a special focus on D-NOTAM 1.0 and today Datalink capacities

This document is a deliverable for the OGC Web Services 8 (OWS-8) testbed activity. OWS testbeds are part of OGC's Interoperability Program, a global, hands-on and collaborative prototyping program designed to rapidly develop, test and deliver proven candidate standards or revisions to existing standards into OGC's Standards Program, where they are formalized for public release. In OGC's Interoperability Initiatives, international teams of technology providers work together to solve specific geoprocessing interoperability problems posed by the Initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments and interoperability support services - all designed to encourage rapid development, testing, validation and adoption of OGC standards.

The OWS-8 sponsors are organizations seeking open standards for their interoperability requirements. After analyzing their requirements, the OGC Interoperability Team recommend to the sponsors that the content of the OWS-8 initiative be organized around the following threads:

* Observation Fusion

* Geosynchronization (Gsync)

* Cross-Community Interoperability (CCI)

* Aviation

More information about the OWS-8 testbed can be found at:

http://www.opengeospatial.org/standards/requests/74

OGC Document [11-139] “OWS-8 Summary Report” provides a summary of the OWS-8 testbed and is available for download:

https://portal.opengeospatial.org/files/?artifact_id=46176

Contents

OGC® OWS-8 AIXM 5.1 Compression Benchmarking

1 Introduction

1.1 Scope

AIXM stands today for the de-facto standard for Aeronautical Information Publication, used by air control service providers from Europe, USA and Australia. With version 5.1, it reaches a level of maturity allowing the support of Digital NOTAMs, as the first official version of these messages was published this year.

In a near future, AIXM will be carried inside WFS requests but also into notification messages along WS event services. This last channel will be the one dedicated to D-NOTAMs. As D-NOTAM is aimed at aircrafts pilots, their transmission to the aircraft will use air/ground data link. Today, datalink communications lack bandwidth and future datalink will still have a limited capacity.

Uploading D-NOTAM aboard raises the question of the pertinence of using XML voluble message through the narrow datalink channel. The viability of AIXM through datalink relies on how good a compression can be applied on these messages. If proof can be made compressed AIXM doesn't weight much more than a handmade binary representation, AIXM should make its way onboard.

This OGC document presents the results of a compression benchmarking campaign of various algorithms applied on AIXM 5.1 files, with a special focus on D-NOTAM 1.0 and today Datalink capacities.

The compression candidates, and input files made the object of a thorough selection and classification in coordination with the aviation thread of OWS-8 team. AIXM inner characteristics are studied to put light on the benchmark results and provide explanation on the outcomes.

This OGC document also gives recommendation on how to implement compression on client/server communication using AIXM.

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.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editors:

Name	Organization
J. Jansou	AtoS
T. Dacla	Atmosphere gbmh

We also want to thank the following contributors to this benchmarking campaign, or report writing:

Name	Organization
J. Arquey	AtoS
M. Aguidi	AtoS
R. Jarzmik	AtoS
S Vingataramin	Atmosphere gbmh

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2011-06-24	V0R1	J. Jansou		First draft
2011-08-26	V0R2	J. Jansou		Second draft
2011-09-30	V1R0	J. Jansou		First release

1.4 Guidelines for the reader

□ Chapter §2 presents the reference material used to write this report

□ Chapter §3 lists frequently used terms and definitions

□ Chapter §4 gives conventions used along this document, in 4.1 you will find abbreviated terms

- Chapter §5 introduces the platform used for the benchmarking, as well as the various candidates under test with their possible configurations

- Chapter §6 enumerates the AIXM files used as input data for the benchmark

- Chapter §7 treats results obtain and proposes explanation

- Chapter §8 shows how whose results could be used for actual applications

- Chapter §9 opens ways to further work on the AIXM compression topic

- Chapter §10 finally concludes, summing up achievements and facts from this study

1.5 Overview / results of the study

The following lines resume the work done in the study and browse conclusions for the hurried reader.

- 3 families of AIXM files were identified and populated with AIXM files

- AIXM selected files were analyzed to show general aspects of AIXM and specificities due to each feature available.

- Measurements were made on compaction performance, CPU consumption and memory footprint (for both setup, encoding and decoding phases) for various algorithms (deflate alone, FI, FI with deflate, CWXML, deflate with dictionary, deflate with several levels of compression, EXI without schema knowledge neither deflate, EXI without schema knowledge but with deflate, EXI with schema knowledge but without deflate, EXI with both schema knowledge and deflate) using the exi-ttfms platform, modified for the purpose.

Main conclusions regarding results obtain are:

- EXI is very efficient to compress small AIXM files without too much coordinates inside. Exificient with both AIXM schema knowledge and deflate post-compression allow to produce compressed D-NOTAMs around 700 bytes (only 13% of the original file size). The dark spot of using exificient for such data is the huge memory footprint necessary (~100MB) and the time needed to perform deflate post compression. This is not much a problem for a 2011 server, but will certainly raise some issue for an EFB.

- For compaction of bigger AIXM files (>100KB), the difference of performance between EXI and other compression algorithms decrease, and thus as you can reach a comparable level of compaction with Fast Info Set with a CPU and memory consumption so much lower, we strongly suggest to stick on Fast Info Set with deflate post-compression.

1.6 Future work

Maintaining the platform:

Improvements in this document are desirable to keep the benchmark results in line with the current state of the art in compression algorithms, specially the ones fully dedicated to XML data. Additionally, as we brought the exi-ttfms up to date with 2011's versions of components, we encourage maintaining the platform in phase with latest versions of libraries for future usage.

Improving CPU / Memory measurements for unbiased comparisons:

Also, some axis of comparison between algorithm in different languages (mostly C,C++ and Java) are not so obvious (e.g. memory foot print) and will need a fresh perspective to be totally fair (ask the operating system an unbiased measurement). The same remark applies for SAX, as all algorithms do not offer a native SAX interface (XML reader / SAX parser).

Continuing following advances of EXI and benefits for AIXM or GML data:

EXI is quite a new techno, especially for open source implementations. Right now if EXI seems the more promising XML compression techno around, it still lacks features to be fully usable for GML data, which are by nature voluminous:

- A new way to handle floating point numbers, which is compatible with deflate post-compression.

- Make some drastic improvement of memory consumption and CPU needed to handle deflate post compression.

1.7 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.

[AIXM_XSD] AIXM 5.1 Schemas 2010-02-01
http://www.aixm.aero/gallery/content/public/AIXM51/AIXM-5-1-20100201-xsd.zip

[AIXM_UML] AIXM 5.1 Conceptual Model 2010-02-01
http://www.aixm.aero/gallery/content/public/AIXM51/AIXM-5-1-20100201-webview.zip

[D_NOTAM_ES] Digital NOTAM Event Specification - version 1.0 – 2011-06-08
http://www.aixm.aero/gallery/content/public/digital_notam/Specifications/Digital%20NOTAM%20Event%20Specification%201.0.doc

[D_NOTAM_FILES] Digital NOTAM samples – version 1.0 – 2011-06-10
http://www.aixm.aero/gallery/content/public/digital_notam/Specifications/Digital%20NOTAM%20samples%20-%20including%20Event%20Schema.zip

[DEF] RFC 1951 - DEFLATE Compressed Data Format Specification - version 1.3 – 1996-06
http://www.ietf.org/rfc/rfc1951.txt

[FIS] X.891: Information technology - Generic applications of ASN.1: Fast infoset – 2007-01-30
http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.891-200505-I!!PDF-E&type=items

[EXI] Efficient XML Interchange (EXI) Format - W3C Recommendation – version 1.0 – 2011-03-10
http://www.w3.org/TR/2011/REC-exi-20110310/

[BXML] OGC 03-002r8 - Binary-XML Encoding Specification - version 0.0.8 – 2003-05
http://www.opengis.org/techno/discussions/03-002r8.pdf

[JAPEX] JAPEX Manual - version 1.1.3 - 2007-12
http://japex.java.net/docs/manual.html

[TTFMS] Efficient XML Interchange Measurements Note
http://www.w3.org/TR/2007/WD-exi-measurements-20070725/

[TTFMS_CODE] W3C EXI Measurement Test Framework
http://www.w3.org/XML/EXI/framework/exi-ttfms.zip

[IRIDIUM_SBD] Iridium's SBD Service explained to developers
http://www.deltawavecomm.com/prices/Iridium/Technical%20Documentation/Data%20Notes/Iridium_SBDS_Developers_Guide.pdf

[OGC] OGC 06-121r3, *OpenGIS® Web Services Common Standard*

NOTE This OWS Common Specification contains a list of normative references that are also applicable to this Implementation Specification.

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] shall apply. In addition, the following terms and definitions apply:

3.1
Candidate
In the JAPEX context, a candidate consists of an autonomous jar archive, used by the test sequencer of JAPEX to perform benchmarking. A candidate can be seen as a couple of 2 software components linked together:

- A library used to perform compression / decompression of data (This library can be coded using C/C++ or Java, offer a SAX interface or not, be able to work on a stream or just on a full memory buffer)

- An adaptation layer between Japex driver and the raw library. This layer will be in charge of converting the high level Japex solicitations (like: initialize for testing, run a test on such a topic (cpu, memory,…)) to concrete calls to the library, using its own internal API. The adaptation layer use some code from the platform that facilitate the declination of generic Japex driver into more compression focused specific drivers (JNI, SAX,…).

3.2
Platform
The Efficient XML Interchange Working Group lead by W3C, develop an EXI test platform to compare the performance of EXI against other algorithms. This platform [TTFMS_CODE] was last modified in 2007 and was reused for the present Benchmarking of AIXM. In this context, and on for the rest of this document, Platform stands for the W3C EXI platform modified for this present AIXM Benchmark.

3.3
Compaction
Measure, expressed in percentage, of the size of data output by a compression algorithm compared to the original size of an input data.

3.4

Data Link

Any way to exchange data messages between an aircraft and the ground when aircraft is moving on the ground or flying.

4 Conventions

4.1 Abbreviated terms

ACARS	Aircraft Communications Addressing and Reporting System
Aero-MAX	WiMAX with special profile for Aeronautical usage
AIP	Aeronautical Information Publication
AIS	Aeronautical Information Services
AIXM	Aeronautical Information Exchange Model
ANSP	Air Navigation Service Provider
AOA	ACARS Over AVLC
AOC	Airline Operational Communication
ASN.1	Abstract Syntax Notation 1
ATC	Air Traffic Control
ATM	Air Traffic Management
ATN	Aeronautical Telecommunications Network
BER	Basic Encoding Rules
BGAN	Broadband Global Area Network
COTS	Commercial Off The Shelf
CPDLC	Controller Pilot Datalink Communications
CPU	Central Processing Unit
D-NOTAM	Digital NOTAM
DVB	Digital Video Broadcasting
DVB-S	DVB for Satellite
DVB-RCS	DVB with Return Channel via Satellite
Eurocontrol	European Organisation for the Safety of Air Navigation
EFB	Electronic Flight Bag
EXI	Efficient XML Interchange
FAA	Federal Aviation Administration
FI/FIS	FastInfoSet
FLAC	Free Lossless Audio Codec

FSB	Front Side Bus
GEO	Geostationary Earth Orbit
GML	Geography Markup Language
GZIP	GNU ZIP
HTTP	HyperText Transfert Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ESA	European Space Agency
ISO	International Standards Organisation
JDK	Java Development Kit
JDSL	Japex Driver Standard Library
JNI	Java Native Interface
JRE	Java RunTime Environment
LDACS	L-band Datalink Aeronautical Communication System
LEO	Low Earth Orbit
MTF	Move To Front
NEWSKY	NEtWorking the SKY
NextGen	Next Generation Air Traffic System
NOTAM	Notice To Airmen
NS	Name Space
OFDM	Orthogonal Frequency Division Multiplexing
PC	Personal Computer
PER	Packed Encoding Rules
POA	Plain Old ACARS
QoS	Quality of Service
RAM	Random-Access Memory
RFC	Request For Comments
SatCom	Satellite Communication
SBB	SwiftBroadBand
SBD	Short Burst Data (Service)
SESAR	Single European Sky ATM Research program

SOAP	Simple Object Access Protocol
SWIM	System Wide Information Management
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
VDL	VHF Data Link
VDL2	VDL Mode 2
VGS	VHF Ground Station
VHF	Very high frequency
VM	Virtual Machine
W3C	World Wide Web Consortium
WiMAX	Worldwide Interoperability for Microwave Access
WP	Work Package
XML	Extensible Markup Language

4.2 Used parts of other documents

This document use parts of other documents. To indicate to readers those quoted parts of such a document, the copied parts are shown with a light grey background (15%). The referenced document will be mentioned through its shortcut reference [XXX] as described in 2

5 Benchmarking platform settings

This chapter focuses on describing the test bed used for this benchmarking campaign we run in both AtoS Toulouse and Atmosphere Munich offices during the spring and summer 2011. As its contains is mostly technical, and do not treat AIXM input files neither results, maybe it will bring interest mostly to people aiming at running this test bed by their own, or enthusiast of compression algorithms.

5.1 Test system configuration

The following lines explain the hardware and operating system of the PC used for the test bed, with the versions of products.

5.1.1 Hardware

There is no special adherence to a specific computer or OS to run the platform. Basically any computer able to run Java 1.6 and Netbeans 7.0 with a C compiler should be enough. Our concern was more to propose a PC architecture as close as possible to an EFB or a standard server.

5.1.1.1 Client side (EFB)

Consumption constraints, heat dissipation, and passive cooling oriented us to choose a recent, but very low consumption PC.

That's why we chose an ASUS Eee PC 1001HA Seashell with the following characteristics:

- Intel Atom N270 with
  - 1.6 GHz single core CPU
  - 512KB L2 Cache
  - Only 32 bits instruction set
  - 2.5W maximum power dissipation
- 533 MHz FSB
- 1 GB of SO DIMM PC2-5300 RAM, single channel
- 1 Sata-300 2.5 inches HDD of 160 GB (5400 rpm)

The idea is that any PC brought today will perform better than this one, in any area (CPU frequency, nb of cores, L2 cache, Memory/FSB Speed,…). So the results we'll give will be lower bound numbers.

5.1.1.2 Server side (WFS server)

With the same idea to use a server hardware still up to date, but which can be beat by any new 2011 server, we chose a model from 2006, based on a P4/D architecture (Dell Precision 470):

- A dual socket motherboard with 2x Intel Xeon processor (Paxville/Netburst DP) with
  - 2.8 GHz double-core
  - 2MB L2 cache on each core
  - 32 bits instruction set

- 800 MHz FSB
- 4GB DDR2 RAM
- 2x73GB SCSI disk (10krpm) on SCSI ultra 320 AIC7902 controllers.

5.1.2 Operating System

Once again, the choice of the operating system is not crucial here, any OS with Java and a C compiler will do for the benchmark.

We choose a x86 32 bits debian 5.0 “lenny” distribution mainly for stability on the client system and a RHEL 4 on the server system

5.1.3 Versions of software / COTS

Table 1 — Version of software components used for the test bed

Software component	Version of component (client)
GCC	3.4.4 (3.4)
Glibc	2.3.4 (2.7.1)
Jdk	1.6.0_24
Kernel	2.6.9-22 for 686 (2.6.26 for 486)
Make	3.80 (3.81-5)
Netbeans	7.0

Table 2 — Version of JAVA software components used for the test bed

Software component	Version of component
Ant	1.7.0
japex	1.2.2 modified
jaxb-api	2.0
Jdsl (part of japex)	1.2.2
jfreechart	1.0.12

Software component	Version of component
Jsr	1.7.3
Xerces	2.9.1

5.2 Operating mode

The following chapters describe the test bed structure, how it uses Japex and the W3C declination of Japex to produce compression benchmark for XML files.

5.2.1 Japex flow

5.2.1.1 Japex quick presentation

JAPEX is a micro benchmarking framework, allowing programmers to rapidly mount small benchmark, without having to concentrate in developing again:

□ Repetitive run of the same measurement on different drivers on different samples

□ Sequencing of tests

□ Loading and initializing drivers

□ Run some dry-run of the test to “warm up” (mobilize the maxium amount of cache) before running the real test

□ Measure duration of a test, CPU consumption, and some memory consumption stats

□ Producing reports, including histograms, means, tables,…

Japex is support by Oracle, through the glassfish project, and is mostly the work of Santiago Pericas-Geertsen. The documentation is scarce and not necessarily up to date (http://japex.java.net/docs/manual.html), as the last update dates from 2007 and the last version of the product dates from 2011.

So put apart the manual, the best way to discover Japex consist in digging into the code, what we did.

In the beginning, Japex was created to evaluate compression performance on Fast Info Set, then reuse for the W3C EXI performance evaluation. We don't know any public use of Japex apart of these 2 previous usages. Probably Japex was designed to be able to make many things, but as it was only used for compression benchmarking, so we will stay on this track and present only the topics already used and tested and which are used for the present benchmark. As we modified Japex for the purpose of this benchmark, to use some "experimental" functions we will address these modification later in this document.

5.2.1.2 Drivers

Japex uses "drivers". A driver is an individual java program (a jar archive) which implements the Japex driver interface:

```
public interface JapexDriver extends Runnable {
            public void initializeDriver();
            public void prepare(TestCase testCase);
            public void warmup(TestCase testCase);
            public void run(TestCase testCase);
            public void finish(TestCase testCase);
            public void terminateDriver();
    }
```

In most cases (at least in the case of the W3C EXI test bed), the JapexDriver interface is implemented through the inheritance of the JapexDriverBase class which implements both the JapexDriver and Params interface.

InitializeDriver is called by Japex when the driver is loaded and terminateDriver before the driver is destroyed. Prepare is used to let the driver prepare the consecutive runs that will follow, for instance loading into memory, files content from disk, or doing any pre-calculation.

Warmup is called when Japex want to proceed to a sequence of dry runs (runs without any measurement) to warmup the VM (warmup can play on prediction schemes on the processor, cache hits success ratio for processor L2 / L3 instruction / data caches or disk buffering into memory for instance, even if it also "pollutes" the 3 java heaps (eden, survivor, old), and therefore implies a random garbage collection). Some would tell warmup could be considered as cheating, depending on the test case, but in our case, it makes some sense if we consider that a WFS server will be processing only WFS requests in a kind of infinite loop…

There is no real difference on the driver side between warmup and run, both methods will trigger a "run". Japex will make some real measurements in the "run" case, like measuring how long the call to "run" takes.

Finish is called back by Japex once all the runs were made, in case the driver wants to compute things by its own.

5.2.1.3 Test configuration and scheduling

Once Drivers have been developed, Japex is ready to run some tests on these drivers. A test campaign is described through XML configuration files. The common way to use Japex, and run a campaign, is to provide 2 files:

- A “driver group” file, gathering the list of drivers “driver name” to run and the parameters to give to these drivers and some “global” japex parameters like “unity” for measurements,…

- A “test suite” file, grouping “test case” items grouping a list of parameters defining a test case. These parameters depend of the nature of the test. For our compression test, they are mostly input files

As the number of XML files used to describe a campaign can change due to the possibility of using Xinclude, it’s important to understand that an individual test is characterized by:

- The selection of a driver, with parameters (let’s say EXI with schema knowledge and post-compression with an interest on “cpu” consumption when compressing)

- The selection of a test case, in our case, an input file (for instance a notam)

When defining a driver group, with a test suite containing multiple test cases, you will have a run corresponding for each test case by each driver. That is the main reason to define one file for drivers (you will run them every time), and multiple files for test suites.

5.2.1.4 Report generation

Japex generates reports in both HTML and XML. HTML pages include pictures of histograms (in our case, but other graphs are possible). When a campaign uses multiples drivers on multiples files, you get several graphs, including:

- One for each test case (or input file) showing the compared performance of each driver (on histogram bar by driver)

- One global representing the average performance of each driver for all the files, ranging for 3 different means calculation (Arithmetic, Geometric and Harmonic)

Result data are also shown in table, grouped by driver, one line for each test case (or input file in our situation).

For % graphs (compaction for instance), the first “driver” sets the reference (100%) and other are expressed related to the performance of the first one.

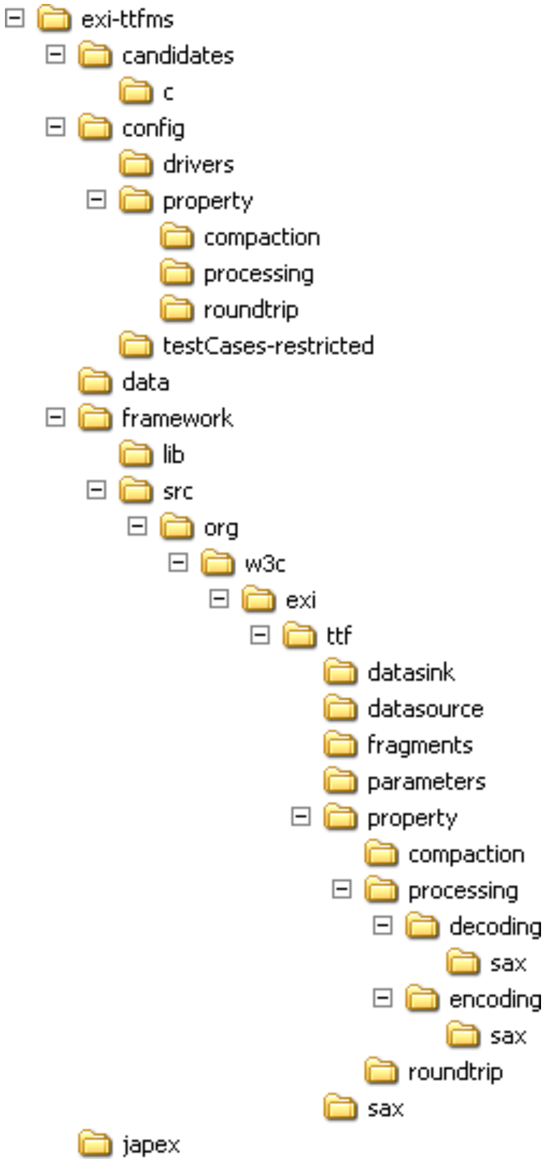
XML output is easy to post process if you plan to generate your own graphs (excel,…).

5.2.1.5 W3C EXI test bed

As said earlier, the W3C EXI test bed is built on Japex. If you download the W3C EXI test bed, you won’t find Japex, but a special empty directory where you’re supposed to put Japex. As the version of Japex used by the original Framework was not downloadable anymore, we used the latest version of Japex 1.2.2 (in mars 2011).

5.2.1.5.1 Structure

The W3C EXI test bed file structure is as following:



```
exi-ttfms
  candidates                  <= japex drivers for java
    c                            or c
  config
    drivers                   <= drivers config
    property                  <= kind of test to run
      compaction
      processing
      roundtrip
    testCases-restricted      <= test cases
  data                        <= XML sample data (some of them with
                              schema)
  framework
    lib
    src                       <= sources of the framework
      org
        w3c
          exi
            ttf               <= here are the “drivers” extensions
              datasink        <= datasinks (memory or network)
              datasource      <= datasource (can be memory or file)
              fragments
              parameters      <= new params (added to japex)
              property        <= properties depending of the topic
                compaction
                processing
                  decoding
                    sax
                  encoding
                    sax
                roundtrip
              sax             <= sax driver
  japex                       <= japex jar to place here
```

5.2.1.5.2 New parameters

Basically, and to sum up, the test bed offers the following new parameters:

- **driver.candidateName**
  - the name of the candidate, e.g. my_killer_compression_lib
- **measurementProperty**, to choose between:
  - ***compactness*** (to perform measurement on compaction ratio (compression alone))
  - ***encode*** (to perform measurement on CPU, memory on encoding)
  - ***decode*** (to perform measurement on CPU, memory on decoding)
- **applicationClass** which takes values between:
  - ***neither*** (raw use of candidate, without additional compression)
  - ***document*** (in this case a gzip compression is added on the datasink stream, sometimes the candidate (e.g. EXI) manages itself the post-compression)
  - ***schema*** (we give the schema to the candidate to be used to improve compression)
  - ***both*** (we use both schema knowledge and post-compression, see document)
- **applicationClass.documentAnalysing.GZIP**
  - if set to ***true***, GZIP is used to compress the datasink
- **dataSourceSink.URI**
  - Supposed to be "memory:/" as the input file is loaded into memory before feeding the datasource stream and as datasink is supposed to feed a buffer in memory.

5.2.1.5.3 Japex drivers specialisation

Japex offers 2 classes to extend its own drivers:

- One is for java drivers (cf. 5.2.1.2): **JapexDriverBase**
- One is for C (and C++) drivers: **JapexNativeDriver**, who is brought through the JDSL (Japex Driver Standard Library) project inside Japex (this project brings also the latest version of FastInfoSet).

The W3C EXI test bed offers some extension of these “classes” (inside ttfs sub-directory):

- **BaseDriver** extends **JapexDriverBase**
  - **SAXDriver** extends **BaseDriver**
  - **CustomDriver** extends **BaseDriver**
- **BaseNativeDriver** extends **JapexNativeDriver**

For Java drivers, we used only SAXDriver. To describe briefly what is brought by both Base and Sax Driver to Japex:

BaseDriver:

- “Warmup” calls “Run”, making no difference between the 2 methods
- “prepare” manages parameters for driver and testcase separately, creates datasink and datasource depending on the “measurement” choosen by the testcase configuration. For “compactness” measurement it will perform compression / decompression directly on the “prepare” phase by calling a transcodeTestCase method.
- “finish” return “compaction” measure as % data to Japex when selected.

SAXDriver:

- The “run” method manages both “encode” and “decode” measurements butnothing for “compaction” run (see prepare method of BaseDriver)
- For “encode”, SAX events read from XML input file will be passed to the SAX handler interface instantiated by the “candidate” (SAX parser)
- For “decode”, the encoded input datasink will be given to the XML Reader interface offered by the candidate which will generate SAX events.

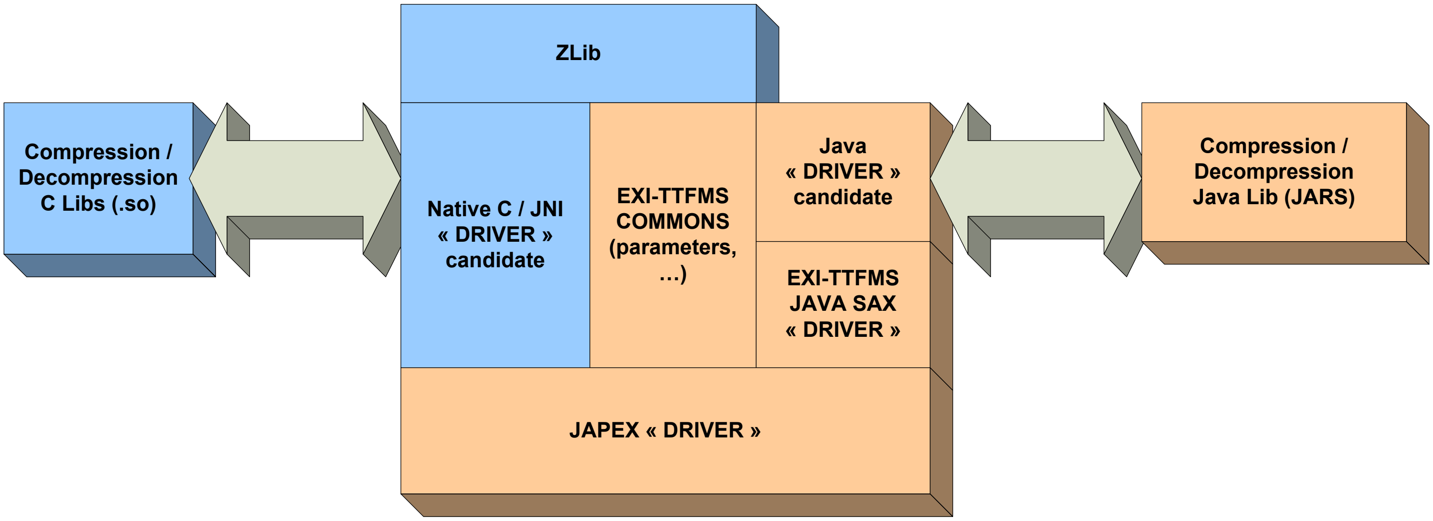
It is important to understand that a candidate, using “SAXDriver” will process its operations by directly handling “SAX events”, without seeing the raw XML at all, at least for encode and decode measurements. For the compaction measurement, it depends on the candidate. Some implement a “transcodeTestCase” method which directly works on raw XML without using SAX (e.g. Jaxp), some re-use SAX (e.g. EXI to benefit from the schema grammar).

5.2.1.5.4 To the “candidate”

As the W3C EXI test bed only reference “candidates” and no “drivers”, we need to specify what is making a candidate. A candidate is a Japex Driver:

- extending the SAXDriver or the BaseNativeDriver, so some of the Japex calls are directly handled by the “platform” (see below)
- using some additional parameters brought by the platform (see below)
- linked with a compression/decompression algorithm (that could be in java or native)
- which eventually supports its own post-compression by itself (see “document” applicationClass with applicationClass.documentAnalysing.GZIP unset)
- which handles eventually the parsing of schema
- which simply handles SAX parser for compression and SAX reader for decompression if using the SAXDriver (this is very very simplified but gives the idea)

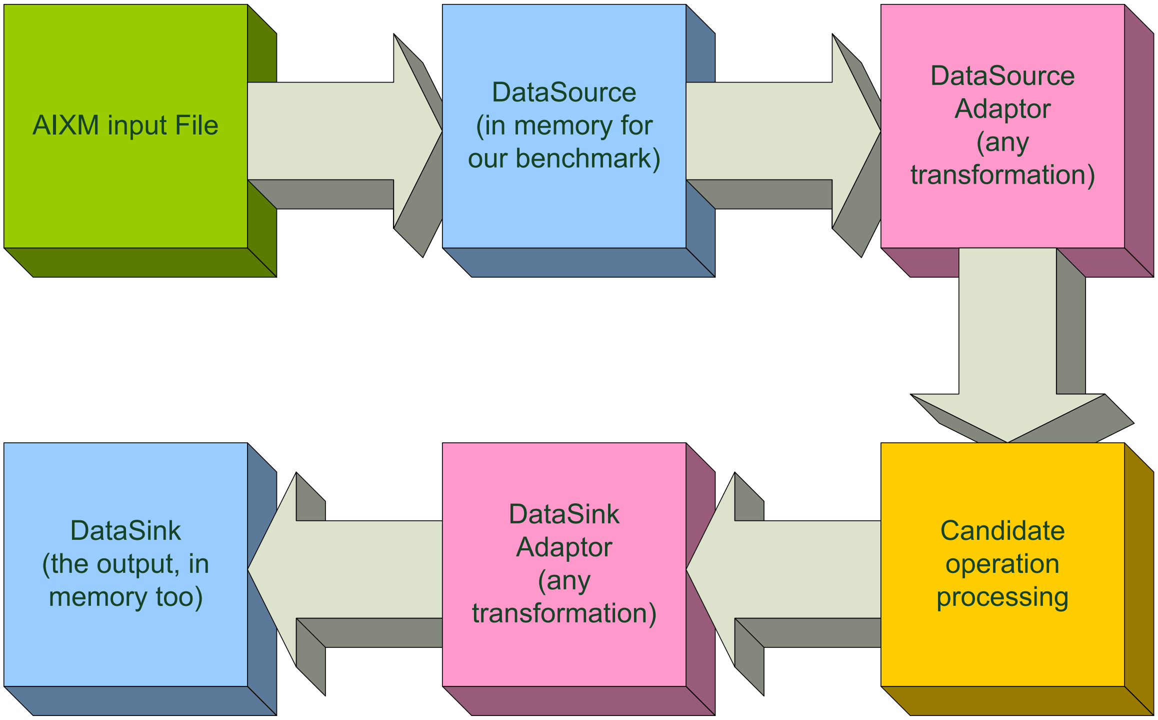
The following schema show the generic candidate structure for both JNI and java bases algorithms used:


ZLib
Compression / Decompression C Libs (.so)
Native C / JNI « DRIVER » candidate
EXI-TTFMS COMMONS (parameters, …)
Java « DRIVER » candidate
EXI-TTFMS JAVA SAX « DRIVER »
JAPEX « DRIVER »
Compression / Decompression Java Lib (JARS)

5.2.1.5.5 Sinks and Sources

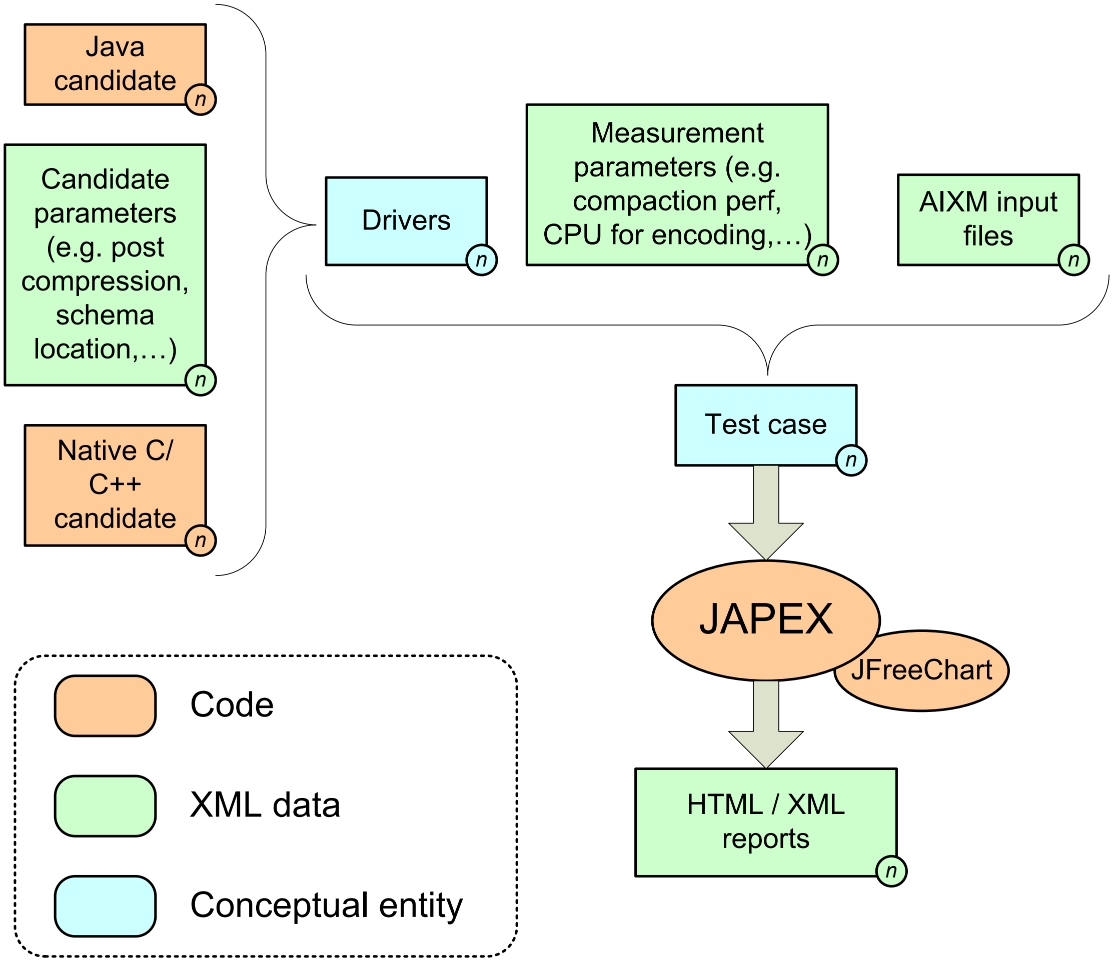
Even if the platform loads input files into memory, and store output also in memory, it will manage input and output of encoding / decoding operations directly from/to streams. The input stream is called the datasource and the output stream the datasink. The usage of adaptors on these streams allows to perform additional gzip compression, and to manage it without soliciting the candidate which still sees streams on both sides.

For instance the gzip compression alone on a file (XML neither candidate) is obtained by using xerces to decode SAX events produced by the platform reading input file, then applying a gzip adaptor before the datasink. The following schema explains those transitions:


AIXM input File
DataSource (in memory for our benchmark)
DataSource Adaptor (any transformation)
Candidate operation processing
DataSink Adaptor (any transformation)
DataSink (the output, in memory too)

5.2.1.6 Flow overview

The following schema presents the exi-ttfms flow, to sum up the precedents chapters:



5.2.2 Measurements

The followings paragraphs give detail information about the various possible measurements, already referenced in 5.2.1.5 through the Measurement property values.

5.2.2.1 Compaction

Compaction produces data about how much the algorithm interfaced by the client can compress an input file. The result is in bytes or expressed in % of the raw file. The configuration Japex lines to produce such a campaign are:

```
<param name="japex.warmupIterations" value="0"/>
<param name="japex.runIterations" value="0"/>
<param name="japex.resultUnit" value="bytes"/>
```

```
    <param name="org.w3c.exi.ttf.measurementProperty"
value="compactness"/>
    <param name="org.w3c.exi.ttf.dataSourceSink.URI" value="memory:/"/>
```

Some examples are present in exi-ttfms/config/property/compaction.

5.2.2.2 CPU Consumption

CPU consumption can be monitored only on “encode” or “decode” measurements. The measure is made on total real time (elapsed) spent inside the “run” method. As we use a one core / no hyper-threading computer, we are not polluted by concurrent runs of other threads (Japex allows that).

The configuration Japex lines to produce such a campaign are:

```
    <param name="japex.warmupTime" value="5"/>
    <param name="japex.runTime" value="5"/>
    <param name="japex.resultUnit" value="ms"/>
    <param name="org.w3c.exi.ttf.measurementProperty" value="encode"/>
    <param name="org.w3c.exi.ttf.dataSourceSink.URI" value="memory:/"/>
    <param name="org.w3c.exi.ttf.recordDecodedEvents" value="no"/>
```

Encode examples are in exi-ttfms/config/property/processing/encoding/java and decode examples in exi-ttfms/config/property/processing/decoding/java

Japex allow 2 unites to be used for CPU consumption. If you are interested in duration choose “ms” for resultUnit, else choose the default “tps” (transactions par seconds). You can also select how many warm up iterations you want and how many real runs before drawing a mean. We use 5 and 5, but tests can be long, so maybe shorter values (1/2) can ease some long campaign and still give the same results as long as you don’t touch the computer.

5.2.2.3 Memory consumption

Memory consumption was an experimental measurement in Japex, as it was not possible to show it on a graph. We modified Japex to be able to give 2 figures:

- First: the maximum amount of memory consumed by the VM (this give a fair idea of the pre-calculations made by some sophisticated algorithm (EXI to build its grammar for instance)). We call it the global memory.

- Second: the dynamic maximum amount of memory taken by the candidate on a run. It is a delta between how much memory is consumed once compressing is at its peak and the memory used before calling the run method. As garbage collecting can be tricky many runs are necessary if you want to get a fair figure. Garbage collecting is called before the run, so only the “top-up” value can be a minor figure is GC was called during the run. As GC takes some CPU to run, this comment also works for CPU consumption.

The memory consumption is based on the sum of all banks of heap memory, so no BSS or TXT data is taken into consideration in these memory measures.

5.3 Candidates

This chapter focused on describing the different candidates used in the benchmark and theirs various configurations.

5.3.1 Java based

5.3.1.1 Java Sax parser

This is a SAX parser providing a SAX parser and reader, doing no compression by its own.

5.3.1.1.1 Without compression (XMLNeither)

With no specific option, this candidate will handle SAX events provided by the platform (a SAX parser, same for all candidates), then regenerate XML.

This particular configuration serves as a reference for other candidates as it will generate “no compression” on any file, so it will stand for the 100% mark. Generally it’s the worst candidate regarding “compaction” but the best for “CPU” in “encode”.

This configuration is in:

- exi-ttfms/config/drivers/sax/xml-neither.xml

```
…
    <param name="japex.driverClass"
value="org.w3c.exi.ttf.candidate.xml.jaxp.JAXPSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName" value="XML"/>
    <param name="org.w3c.exi.ttf.driver.isXmlProcessor" value="true"/>
    <driver name="XMLNeither" normal="true">
        <param name="description" value="XML"/>
        <param name="org.w3c.exi.ttf.applicationClass"
value="neither"/>
    </driver>
…
```

5.3.1.1.2 With GZIP default compression (XMLDocument)

Same as XMLNeither except a gzip compressor is connected between the output of the SAX reader and the output stream (or datasink). This could be seen as a basic GZIP on the raw XML file, except SAX is used.

This configuration is in:

- exi-ttfms/config/drivers/sax/xml-document.xml

```
…
    <param name="japex.driverClass"
value="org.w3c.exi.ttf.candidate.xml.jaxp.JAXPSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName" value="XML"/>
    <param name="org.w3c.exi.ttf.driver.isXmlProcessor" value="true"/>
    <driver name="XMLDocument" normal="true">
        <param name="description" value="XML using document analysis"/>
        <param name="org.w3c.exi.ttf.applicationClass"
value="document"/>
        <param
name="org.w3c.exi.ttf.applicationClass.documentAnalysing.GZIP"
value="true"/>
    </driver>
…
```

5.3.1.1.3 With GZIP compression for a given compression level (1-9)

As we modified the platform to be able to integrate a new parameter (org.w3c.exi.ttf.applicationClass.documentAnalysing.level) to set the level for gzip compression (default is 4 between 1 and 9), you can use it on the XML candidate also.

```
…
    <param name="japex.driverClass"
value="org.w3c.exi.ttf.candidate.xml.jaxp.JAXPSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName" value="XML"/>
    <param name="org.w3c.exi.ttf.driver.isXmlProcessor" value="true"/>
    <driver name="XMLDocument" normal="true">
        <param name="description" value="XML using document analysis
level 9"/>
        <param name="org.w3c.exi.ttf.applicationClass"
value="document"/>
        <param
name="org.w3c.exi.ttf.applicationClass.documentAnalysing.GZIP"
value="true"/>
        <param
name="org.w3c.exi.ttf.applicationClass.documentAnalysing.level"
value="9"/>
        </driver>
…
```

We didn’t want to disable gzip by just using a level 4 compression performance, so we choose to use 9 (maximum) for all XML Document tests.

5.3.1.1.4 With GZIP compression, level and pre-loaded dictionnary

We also modified the platform for gzip to be able to use a pre-loaded dictionary (maximum 32KB) for encode and decode. We also developed a tool to parse a specific schema (or a set of schema files when referencing) and put the best data possible in this dictionary to boost gzip compression. More detail about this tool is given in 9.3

As this use case is not standard, and asking some twisting of the platform and is not so really easy to put in place in an industrial use (to uncompress, you have to read 0 bytes before giving deflate its dictionary, then resume the reading from gzip, so it won’t match all server environments), we give some example of performance of this method without adding it to all campaigns.

5.3.1.2 Fast Info Set

We used Fast Info Set 1.2.9 (brought by Japex 1.2 (JSDL)), and had to modify the candidate given with the W3C platform to be able to deal with this new version (the original one was 1.2.2), because some method / parameters changed between these versions.

5.3.1.2.1 Without post-compression (FastinfosetNeither)

Without gzip post compression, FIS is mostly working in locating elements, namespaces, attributes,… and converting them into ASN.1 using PER (a token might not be aligned on a byte but on fewer bits (or more)). FIS comes with its own way to deal with SAX giving it an edge other a std SAX parser because its input is more compact.

This configuration is in:

- `exi-ttfms/config/drivers/sax/fastinfoset-neither.xml`

```
…
    <param name="japex.driverClass"
value="org.w3c.exi.ttf.candidate.fastinfoset.FastInfosetSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName"
value="FastInfoset"/>
    <param
name="org.w3c.exi.ttf.driver.candidate.fastinfoset.characterContentChun
kSizeLimit" value="32"/>
    <param
name="org.w3c.exi.ttf.driver.candidate.fastinfoset.attributeValueSizeLi
mit" value="32"/>
```

```
    <driver name="FastInfosetNeitherSAX">
        <param name="description" value="Fast Infoset"/>
        <param name="org.w3c.exi.ttf.applicationClass"
value="neither"/>
    </driver>
…
```

5.3.1.2.2 With GZIP post-compression (FastinfosetDocument)

When we add a GZIP post-compression to the output of FIS encoding, the FIS candidate modifies the behavior or the FIS algorithm to generate byte aligned tokens, and uses BER and not PER. The raw FIS stream is less compressed this way but can benefit from the GZIP post compression which only works on byte aligned data (else gives random results).

This configuration is in:

- exi-ttfms/config/drivers/sax/fastinfoset-document.xml

```
…
    <param name="japex.driverClass"
value="org.w3c.exi.ttf.candidate.fastinfoset.FastInfosetSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName"
value="FastInfoset"/>
    <param
name="org.w3c.exi.ttf.driver.candidate.fastinfoset.characterContentChun
kSizeLimit" value="32"/>
    <param
name="org.w3c.exi.ttf.driver.candidate.fastinfoset.attributeValueSizeLi
mit" value="32"/>
    <driver name="FastInfosetDocumentSAX">
        <param name="description" value="Fast Infoset using document
analysis"/>
        <param name="org.w3c.exi.ttf.applicationClass"
value="document"/>
        <param
name="org.w3c.exi.ttf.applicationClass.documentAnalysing.GZIP"
value="true"/>
    </driver>
…
```

5.3.1.3 Exificient (EXI)

The original EXI candidate from the W3C EXI test bed was the commercial product from AgileDelta, who was a main contributor to the W3C EXI Performance campaign. We couldn’t reuse this candidate, because the library we use (the open source Exificient 0.7 from Siemens) was quite different in its API. We didn’t buy the AgileDelta Software product, but we had some email exchanges with them and decided to go for the open-source product to check if the performance was compatible with the commercial product.

As results showed, the compaction performance is (we believe) similar, but maybe some tests could be made regarding CPU and memory consumption where the AgileDelta product seems to perform better.

As we face some slight bugs on the processing of list of floats and integers, we submitted a bug report to the Exificient team, and got a fast fix (thanks Daniel). So the version we used is a patch on release 0.7 and corresponds to the revision 361 from the sourceforge svn repository (so it’s not a strait 0.7)

5.3.1.3.1 Without previous schema knowledge and “deflate” post-compression

Like FIS and its default PER encoding, EXI uses a bit stream and not a byte stream. On this configuration without previous schema knowledge nor post-compression, the compaction is mainly brought by XML parsing and structure recognition. It is not so different from FIS in this mode.

This configuration is in:

- `exi-ttfms/config/drivers/sax/exificient-neither.xml`

```
…
    <param name="japex.driverClass"
value="org.w3c.exi.ttf.candidate.exificient.ExificientSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName"
value="Exificient 0.7 (Siemens EXI Java open source impl.)"/>

    <driver name="ExificientNeitherSAX">
        <param name="description" value="EXI without document analysis
or schema optimizations"/>
        <param name="org.w3c.exi.ttf.applicationClass"
value="neither"/>
    </driver>
…
```

5.3.1.3.2 Without previous schema knowledge but with “deflate” post-compression

As for FIS, the post-compression needs to output byte aligned symbols, and this choice is made by the candidate when it sees the “document” application-class parameter activated. But, contrary to FIS or JAXP, EXI manages its own “deflate” algorithm and does not use zlib’s API. That’s why the GZIP parameter is not set in the config file.

This configuration is in:

- `exi-ttfms/config/drivers/sax/exificient-document.xml`

```

…
    <param name="japex.driverClass"
value="org.w3c.exi.ttf.candidate.exificient.ExificientSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName"
value="Exificient 0.7 (Siemens EXI Java open source impl.)"/>
```

```
    <driver name="ExificientDocumentSAX">
        <param name="description" value="EXI with deflate"/>
        <param name="org.w3c.exi.ttf.applicationClass" 
value="document"/>
    </driver>
…
```

5.3.1.3.3 With previous schema knowledge but without “deflate” post-compression

This mode is very close to the first one, except that we give the location of schema (the first xsd including others) to the algorithm. In the preparation phase, EXI will parse .xsd files beginning by AIXM_BasicMessage.xsd, then will compute its grammar to apply for encoding or decoding. The output is bit aligned.

The configuration is in:

- `exi-ttfms/config/drivers/sax/exificient-schema.xml`

```
…
    <param name="japex.driverClass" 
value="org.w3c.exi.ttf.candidate.exificient.ExificientSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName" 
value="Exificient 0.7 (Siemens EXI Java open source impl.)"/>

    <driver name="ExificientSchemaSAX">
        <param name="description" value="EXI with schema optimizations 
but without deflate"/>
        <param name="org.w3c.exi.ttf.applicationClass" value="schema"/>
    </driver>
…
```

5.3.1.3.4 With previous schema knowledge and “deflate” post-compression

This mode is the merge of mode 2 (deflate) and 3 (schema knowledge), the output given by mode 3 is byte aligned, then deflate plays its part. The overall output is bit aligned.

The configuration is in:

- `exi-ttfms/config/drivers/sax/exificient-both.xml`

```
…
    <param name="japex.driverClass" 
value="org.w3c.exi.ttf.candidate.exificient.ExificientSAXDriver"/>
    <param name="org.w3c.exi.ttf.driver.candidateName" 
value="Exificient 0.7 (Siemens EXI Java open source impl.)"/>

    <driver name="ExificientBothSAX">
        <param name="description" value="EXI with schema optimizations 
and deflate"/>
        <param name="org.w3c.exi.ttf.applicationClass" value="both"/>
    </driver>
```

…

5.3.2 C/C++ based

5.3.2.1 CWXML

CubeWerck’s binary XML is very like Fast Info Set, except it doesn’t offer a PER encoding allowing a bit stream as output. The output is byte aligned and then offers a good entry point for deflate. CWXML use its own linking with Zlib to invoke deflate.

CWXML uses dictionary to store strings like elements name and has a special feature to guess what is a floating point number when it encounters one and convert it to IEEE 754.

5.3.2.1.1 Without post-compression

Without deflate post-compression, CWXML can only take advantage of element name repetition and IEEE 754 serialization of floats / doubles.

The configuration is in:

```
     exi-ttfms/config/property/compaction/compaction-native.xml
…
    <driver name="CWXML neither">
        <param name="libraryPath"
value="${japex.exi.ttfms.candidatesDir}/c/cwxml"/>
        <param name="libraryName" value="cwxml"/>
        <param name="org.w3c.exi.ttf.applicationClass"
value="neither"/>
        <param name="description" value="CWXML neither"/>
    </driver>
…
```

5.3.2.1.2 With GZIP post-compression

With deflate used as post-compression, the output is processed through a second pass trying to reduce redundancy.

The configuration is in:

```
     exi-ttfms/config/property/compaction/compaction-native.xml
…
   <driver name="CWXML document">
        <param name="libraryPath"
value="${japex.exi.ttfms.candidatesDir}/c/cwxml"/>
        <param name="libraryName" value="cwxml"/>
        <param name="org.w3c.exi.ttf.applicationClass"
value="document"/>
        <param name="description" value="CWXML document"/>
    </driver>
```

…

6 AIXM input files

All the test cases, used for this benchmarking campaign use AIXM 5.1 input files, except some “technical files” which can be simple XML and are just here to compare algorithm when dealing with a specific aspect (formatting, autoclosing tags…)

In order to get a relevant sample set of AIXM data, we processed data from various sources (mostly Snowflake, Comsoft, Luciad WFS servers, but also D-NOTAMs from Eurocontrol) to identify characteristics from AIXM (feature by feature)

6.1 Families of files

As we collected too much data (almost 1 GB), we extracted only portions of full databases and put these parts into files, then gathered those files into families. The files described below could have been the result from various WFS request, mostly based on BBOX. In fact we worked offline using full exports of features and sorted them using tools (scripts detailed in 6.2.1)

Four families of AIXM files were identified and populated:

- A first family composed of small files (<10kB): tree D-NOTAMs from Eurocontrol.
- A second family composed of medium sized files (between 10kB and 1MB), each of it made from a single AIXM feature, bringing its own characteristics (for instance airspaces, geo borders, runways and taxiways elements contains much more coordinates than others features. Routes have simple structure (only 28 different elements taking 65% of the file) compared to airspaces (60 different elements taking 30% of the file))
- A third family made of bigger files (>1MB), alternating both mixed features (as the whole Estonian database or the sum of all features from family 2) or single features to see how the performance of compression algorithms evolve along with volume
- A fourth family made of technical files, useful to check a specific aspect against all algorithms (influence of order, handling of autoclosing tags, drops of comments, formatting …).

Original AIXM files (readable) are stored into exittfms/data. Normalization script works from this path when displaying file names.

6.1.1 Small files (<10kB)

This family is very short (only 3 files), and contains the D-NOTAMs provided by Eurocontrol and based on fake DONLON AIXM demo database:

- restricted_area_event.xml: corresponding to an Airspace Reservation (or special activity to use FAA vocabulary) “TEMPORARY RESTRICTED AREA NORTH OF SJAELLANDS ODDE”
- tra_ear23_active.xml: activation of EAR23 from DONLON
- tra_ear23_active_cancel.xml: cancellation, previous activation being based on TEMPDELTA timeslice.

When put through our cleaning script, we reduced the size of original files:

```
./notams/restricted_area_event.xml (normalization) (8817 -> 6986 bytes)
./notams/tra_ear23_active.xml (normalization) (6019 -> 4537 bytes)
./notams/tra_ear23_active_cancel.xml (normalization) (5629 -> 4441
bytes)
```

6.1.2 Medium files (between 10kB and 1MB)

This family contains 10 files, all extracted from the snowflake WFS server. Each file references only AIXM data from the same feature type.

```
./from_snowflake/files_between_10k_and_1m/airspaces_all_85k.xml
(normalization) (85493 -> 84360 bytes)
./from_snowflake/files_between_10k_and_1m/geo_border_florida_775k.xml
(normalization) (776717 -> 774238 bytes)
./from_snowflake/files_between_10k_and_1m/geo_border_puerto_rico_107k.x
ml (normalization) (107309 -> 106846 bytes)
./from_snowflake/files_between_10k_and_1m/navaids_alaska_310k.xml
(normalization) (314978 -> 313548 bytes)
./from_snowflake/files_between_10k_and_1m/route_alaska_50k.xml
(normalization) (50019 -> 50284 bytes)
./from_snowflake/files_between_10k_and_1m/runway_elements_alaska_40k.xm
l (normalization) (39193 -> 39024 bytes)
./from_snowflake/files_between_10k_and_1m/runways_alaska_22k.xml
(normalization) (21853 -> 21988 bytes)
./from_snowflake/files_between_10k_and_1m/taxiway_elements_alaska_730k.
xml (normalization) (730052 -> 737057 bytes)
./from_snowflake/files_between_10k_and_1m/taxiways_alaska_375k.xml
(normalization) (377145 -> 388601 bytes)
./from_snowflake/files_between_10k_and_1m/vertical_structure_alaska_230
k.xml (normalization) (233144 -> 232073 bytes)
```

6.1.3 Large files (>1MB)

This family contains 15 files, 13 from snowflake and remain mono-feature, 1 (Estonian database) mixing features from Comsoft and one from Luciad (FAA airspaces).

```
./from_snowflake/files_over_1m/airports_from_florida_2_2m.xml
(normalization) (2170633 -> 2146231 bytes)
./from_snowflake/files_over_1m/geo_borders_calif_nev_1_5m.xml
(normalization) (1567728 -> 1563628 bytes)
./from_snowflake/files_over_1m/geo_borders_megalop_20m.xml
(normalization) (19846237 -> 19802453 bytes)
./from_snowflake/files_over_1m/navaids_megalop_3_6m.xml (normalization)
(3591108 -> 3594804 bytes)
./from_snowflake/files_over_1m/route_segment_florida_1_2m.xml
(normalization) (1207640 -> 1204582 bytes)
./from_snowflake/files_over_1m/route_segment_megalop_15m.xml
(normalization) (14485225 -> 14452017 bytes)
./from_snowflake/files_over_1m/runway_elements_all_4_4m.xml
(normalization) (4419641 -> 4434489 bytes)
./from_snowflake/files_over_1m/runways_all_2_5m.xml (normalization)
(2532075 -> 2581599 bytes)
./from_snowflake/files_over_1m/taxiway_elements_calif_nev_5_4m.xml
(normalization) (5351531 -> 5410260 bytes)
./from_snowflake/files_over_1m/taxiways_calif_nev_3m.xml
(normalization) (3048904 -> 3143078 bytes)
./from_snowflake/files_over_1m/taxiways_megalop_12m.xml (normalization)
(11704191 -> 12041044 bytes)
./from_snowflake/files_over_1m/vertical_structure_florida_1_1m.xml
(normalization) (1154738 -> 1150632 bytes)
./from_snowflake/files_over_1m/vertical_structure_megalop_8m.xml
(normalization) (7828675 -> 7802553 bytes)
./from_comsoft/estonia-ows8.xml (normalization) (4845567 -> 3761036
bytes)
/from_luciad/airspaces.xml (normalization) (10513313 -> 7036816 bytes)
```

6.1.4 Technical files

Technical files come from snowflake, or are generated by hand.

- config/testCases-restricted/technical_family.xml contains tests related to check the influence of:
    - indenting (as all our sample files from family 1 to 3 are cleaned, so without indenting)
    - autoclosing elements (is <a></a> making a difference with <a/>)
    - namespace alias (does referencing a namespace “n1” makes a difference with referencing it “my_long_namespace_name”)

```
./technical/auto_closing_elements_off.xml ( 440 )
```

```
./technical/auto_closing_elements_on.xml ( 386 )
./technical/explicit_name_spaces_off.xml ( 354 )
./technical/explicit_name_spaces_on.xml ( 564 )
./technical/indenting_off.xml ( 325 )
./technical/indenting_on.xml ( 405 )
```

- config/testCases-restricted/family_4.xml contains a test related to the order of features for the same set of data.

```
./from_snowflake/technical/all_features_from_family_2_in_disorder.xml
(normalization) (2726759 -> 2741692 bytes)
./from_snowflake/technical/all_features_from_family_2_sorted_by_feature
.xml (normalization) (2726759 -> 2741692 bytes)
```

- config/testCases-restricted/technical_doubles.xml contains a test related to the influence of coordinates on candidates. The AIXM file used contains a fake geoborder whose coordinates are in fact all coordinates from all airports in the Snowflake WFS server (or a gml:poslist of 22320 coordinates). So we can say this file is made of merely only double precision data. Coordinates are unique and sorted along their longitude (X).

```
./technical/doubles.xml (744963)
```

6.2 Characterization of families / files

The followings paragraphs detail characteristics of sample files selected for test cases. Statistical analysis figures are provided, with (when possible) two graphs showing the geographical distribution of features on a map (on a global scale, then zoomed on the targeted area). The graphs are only given for family 2 and 3 and only for the data coming from snowflake (because using BBOXs) and luciad (only FAA airspaces).

6.2.1 Statistical analysis

Here is a short list of scripts used for data selection, edition and analysis, mostly in perl or bourne shell. They are located into the folder exi-ttfms/data/scripts:

- bbox_filter.pl: allows output only a subset of features whose BBOX are inside the one given as parameters to the script. Allows you to dump a large set of features from a WFS server, then sort them offline.
- bboxs_to_png.pl: generates a geographical footprint of the feature you use and place them on a map (in our case the north east quarter of the globe based on a satellite shot). The BBOX of each feature is added to the graph, using a gradient color map to show the stacking level of BBOXes.
- bboxs_to_png_luciad.pl: same script as previously, but calculates BBOXes from individual coordinates of each airspace. Works only with LUCIAD WFS server data.
- gen_japex.pl: automatically create configuration files for japex testcase providing AIXM input files.
- concat.pl: join AIXM files into a single one, keeping only one header

- mixer.pl: takes multiples AIXM as input and generates a merged AIXM resulting file, mixing features from all the files in an interleaved way (opposite of concat.pl)
- populate_families.sh: Shell script which manages the cleaning of all files from different families, and the copy to the right directory.
- post.pl: posts a SOAP message or a form from a previously loaded page from a WFS server. This script is useful to automate some requests on a WFS server.
- stax.pl: Stax parser which provides statistical data of AIXM parsed file (gives figures about coordinates / date usage proper to AIXM).
- xml_cleaner.pl: Clean an XML file (unnecessary spaces, tabs, indenting including carriage returns, DOS end of line formatting, replace autoclosing tags (<tag/>) by a pair or tags (<tag></tag>, remove comments,…). This filter was applied to all our AIXM files before benching.

6.2.2 Small files (<10kB)

6.2.2.1 f1_restricted_area_event.xml

```
total size: 6986
composition: %elems = 64.4, %att(names) = 6.8, %att(values) = 11.2,
%text/c_data = 15.5, %other = 2.0
elements: total count = 128, differents = 90, min. occ. = 1, max. occ.
= 3, avg. occ. = 1.4, avg. size = 35.2 bytes
attributes: total count = 50, avg. size = 9.5 bytes (names) & 15.6
bytes (values)
text / c_data: %coords = 11.6, %dates = 12.9, %other = 75.5
```

6.2.2.2 f1_tra_ear23_active.xml

```
total size: 4537
composition: %elems = 64.4, %att(names) = 7.9, %att(values) = 13.6,
%text/c_data = 11.7, %other = 2.4
elements: total count = 83, differents = 69, min. occ. = 1, max. occ. =
3, avg. occ. = 1.2, avg. size = 35.2 bytes
attributes: total count = 34, avg. size = 10.6 bytes (names) & 18.1
bytes (values)
text / c_data: %coords = 0.0, %dates = 15.0, %other = 85.0
```

6.2.2.3 f1_tra_ear23_active_cancel.xml

```
total size: 4441
composition: %elems = 65.9, %att(names) = 7.1, %att(values) = 13.6,
%text/c_data = 11.1, %other = 2.3
elements: total count = 82, differents = 68, min. occ. = 1, max. occ. =
3, avg. occ. = 1.2, avg. size = 35.7 bytes
attributes: total count = 32, avg. size = 9.8 bytes (names) & 18.8
bytes (values)
text / c_data: %coords = 0.0, %dates = 8.1, %other = 91.9
```

6.2.2.4 Analysis

From figures:

D-NOTAMs present a high number of elements (65% of file), few attributes, and few c_data (circa 15%). As they are small, and concern generally only one event, we do not find a lot of repetition (average of 1.2 element redundancy, let's say all elements are unique or near). This configuration is a favorable ground for a schema based algorithm.

From content:

There is too much redundancy on D-NOTAM, in terms of meaning. The same info is given from different ways (Textual, field by field, and then projected on AIP database). As this redundancy cannot be guessed by algorithm, the compression won't be very good.

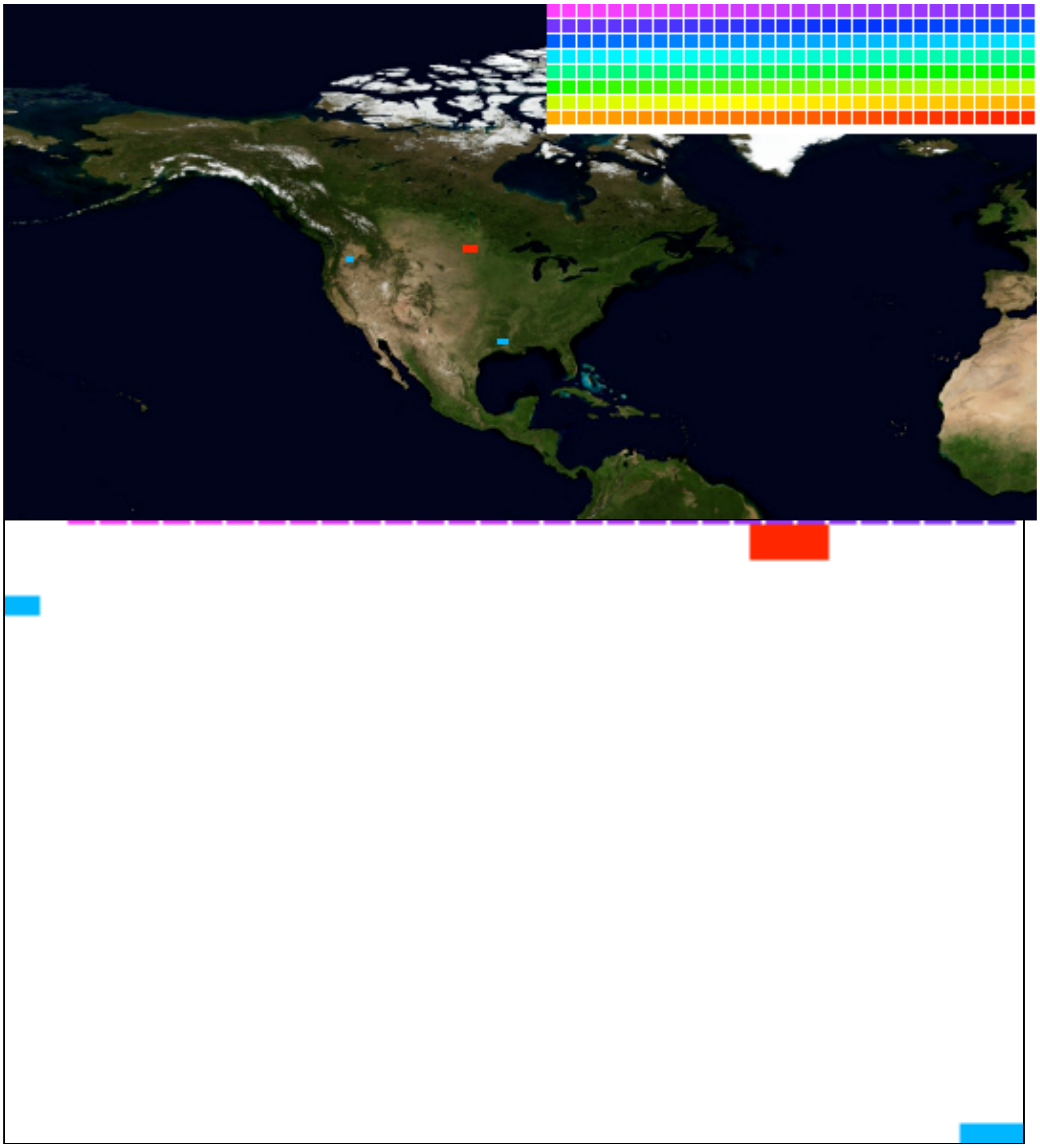
Also, the c_data is mostly free text, it's too bad, because enumerate usage could benefit to compression.

6.2.3 Medium files (between 10kB and 1MB)

Each file of the second family is only composed of elements from the same feature type. This composition allows to see for each feature how performs every algorithm. From this principle, we focus on what makes each feature unique and how the inner structure of a feature is willing to offer a good profile for compression methods used in ours algorithms.

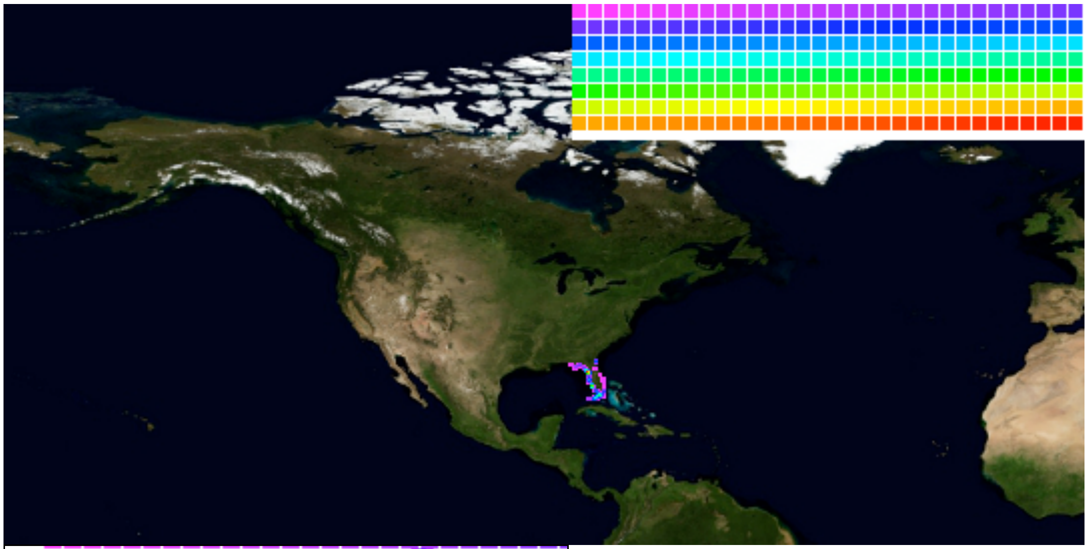
6.2.3.1 f2_airspaces_all_85k.xml

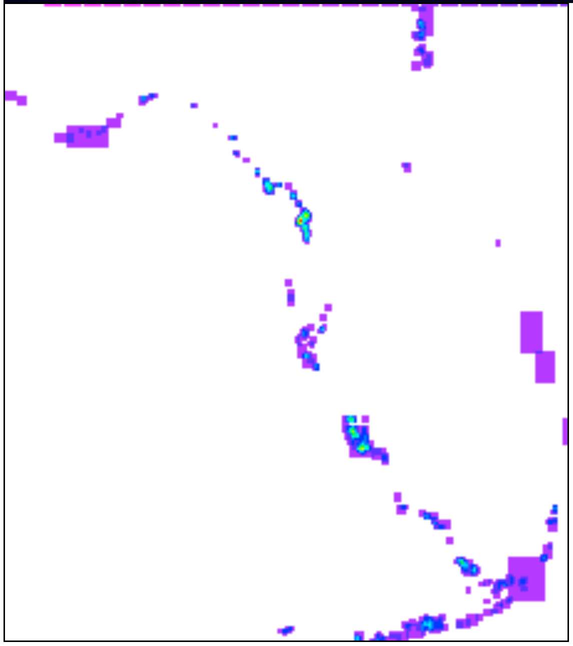
```
total size: 84360
composition: %elems = 29.9, %att(names) = 2.5, %att(values) = 8.2,
%text/c_data = 58.7, %other = 0.7
elements: total count = 657, differents = 60, min. occ. = 1, max. occ.
= 27, avg. occ. = 10.9, avg. size = 38.4 bytes
attributes: total count = 249, avg. size = 8.3 bytes (names) & 27.8
bytes (values)
text / c_data: %coords = 93.5, %dates = 0.7, %other = 5.8
```



6.2.3.2 f2_geo_border_florida_775k.xml

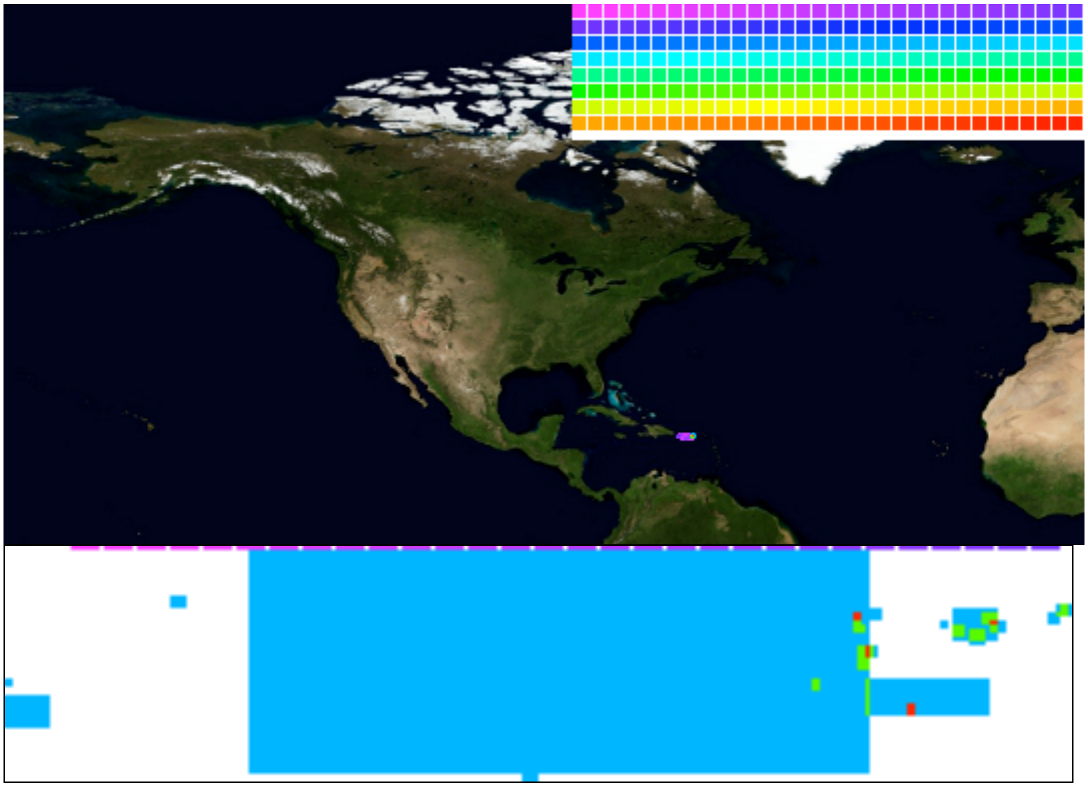
```
total size: 774238
composition: %elems = 45.4, %att(names) = 4.7, %att(values) = 10.4,
%text/c_data = 38.5, %other = 0.9
elements: total count = 10109, differents = 29, min. occ. = 1, max.
occ. = 361, avg. occ. = 348.6, avg. size = 34.8 bytes
attributes: total count = 3623, avg. size = 10.1 bytes (names) & 22.3
bytes (values)
text / c_data: %coords = 89.0, %dates = 9.1, %other = 1.9
```





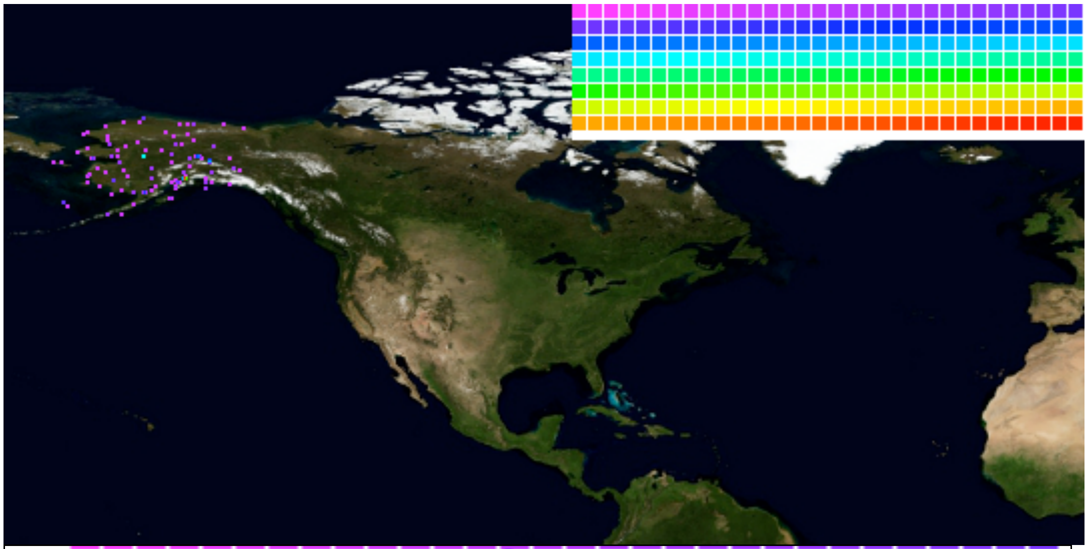
6.2.3.3 f2_geo_border_puerto_rico_107k.xml

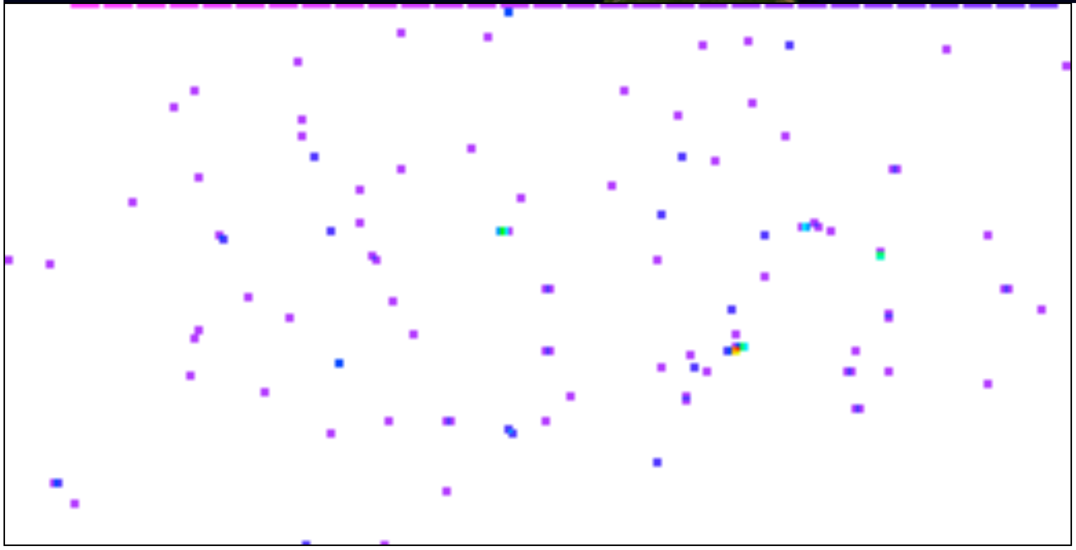
```
total size: 106846
composition: %elems = 22.8, %att(names) = 2.5, %att(values) = 5.6,
%text/c_data = 68.5, %other = 0.5
elements: total count = 701, differents = 29, min. occ. = 1, max. occ.
= 25, avg. occ. = 24.2, avg. size = 34.8 bytes
attributes: total count = 263, avg. size = 10.2 bytes (names) & 22.8
bytes (values)
text / c_data: %coords = 96.7, %dates = 2.6, %other = 0.7
```



6.2.3.4 f2_navaids_alaska_310k.xml

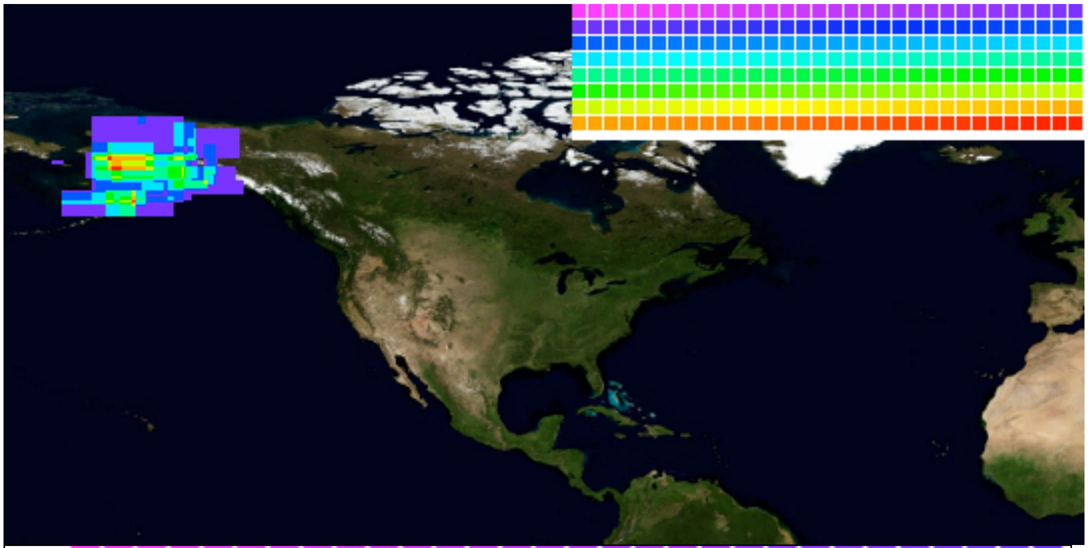
```
total size: 313548
composition: %elems = 56.3, %att(names) = 4.9, %att(values) = 27.0,
%text/c_data = 10.8, %other = 1.0
elements: total count = 4967, differents = 42, min. occ. = 1, max. occ.
= 153, avg. occ. = 118.3, avg. size = 35.6 bytes
attributes: total count = 1565, avg. size = 9.7 bytes (names) & 54.0
bytes (values)
text / c_data: %coords = 42.8, %dates = 33.7, %other = 23.4
```

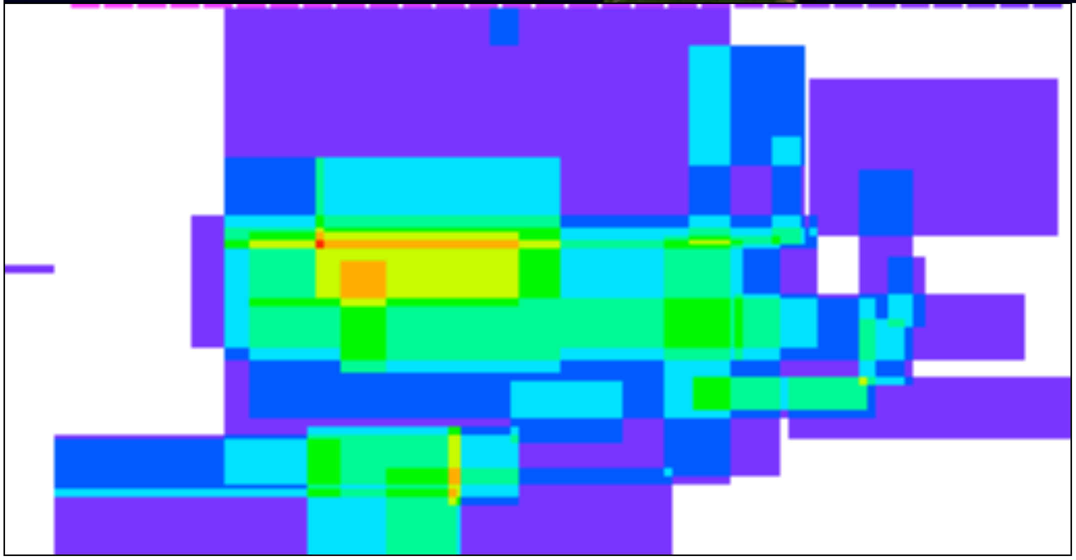




6.2.3.5 f2_route_alaska_50k.xml

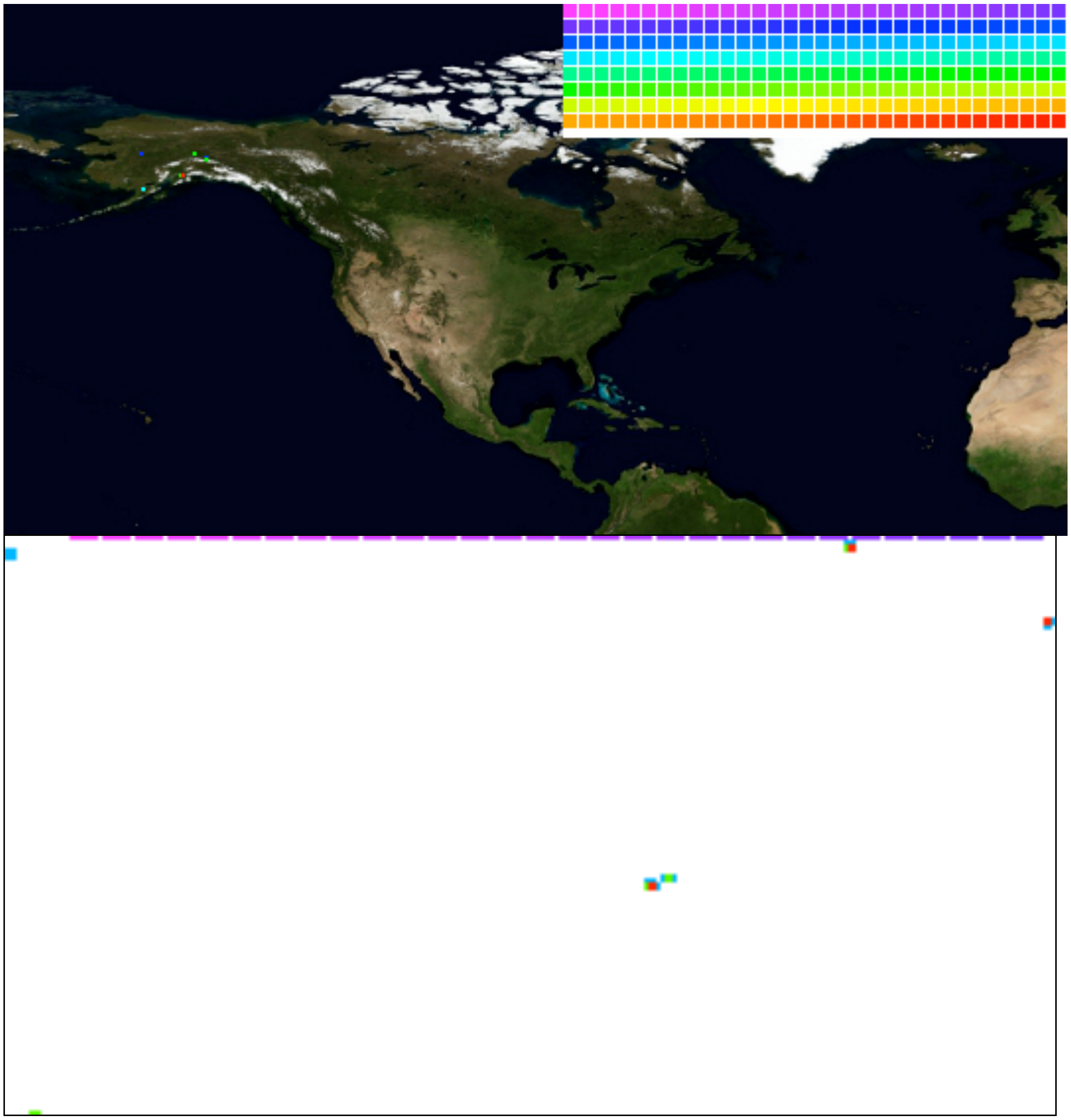
```
total size: 50284
composition: %elems = 64.8, %att(names) = 6.4, %att(values) = 16.9,
%text/c_data = 10.6, %other = 1.3
elements: total count = 919, differents = 28, min. occ. = 1, max. occ.
= 34, avg. occ. = 32.8, avg. size = 35.5 bytes
attributes: total count = 319, avg. size = 10.1 bytes (names) & 26.6
bytes (values)
text / c_data: %coords = 44.4, %dates = 48.0, %other = 7.7
```





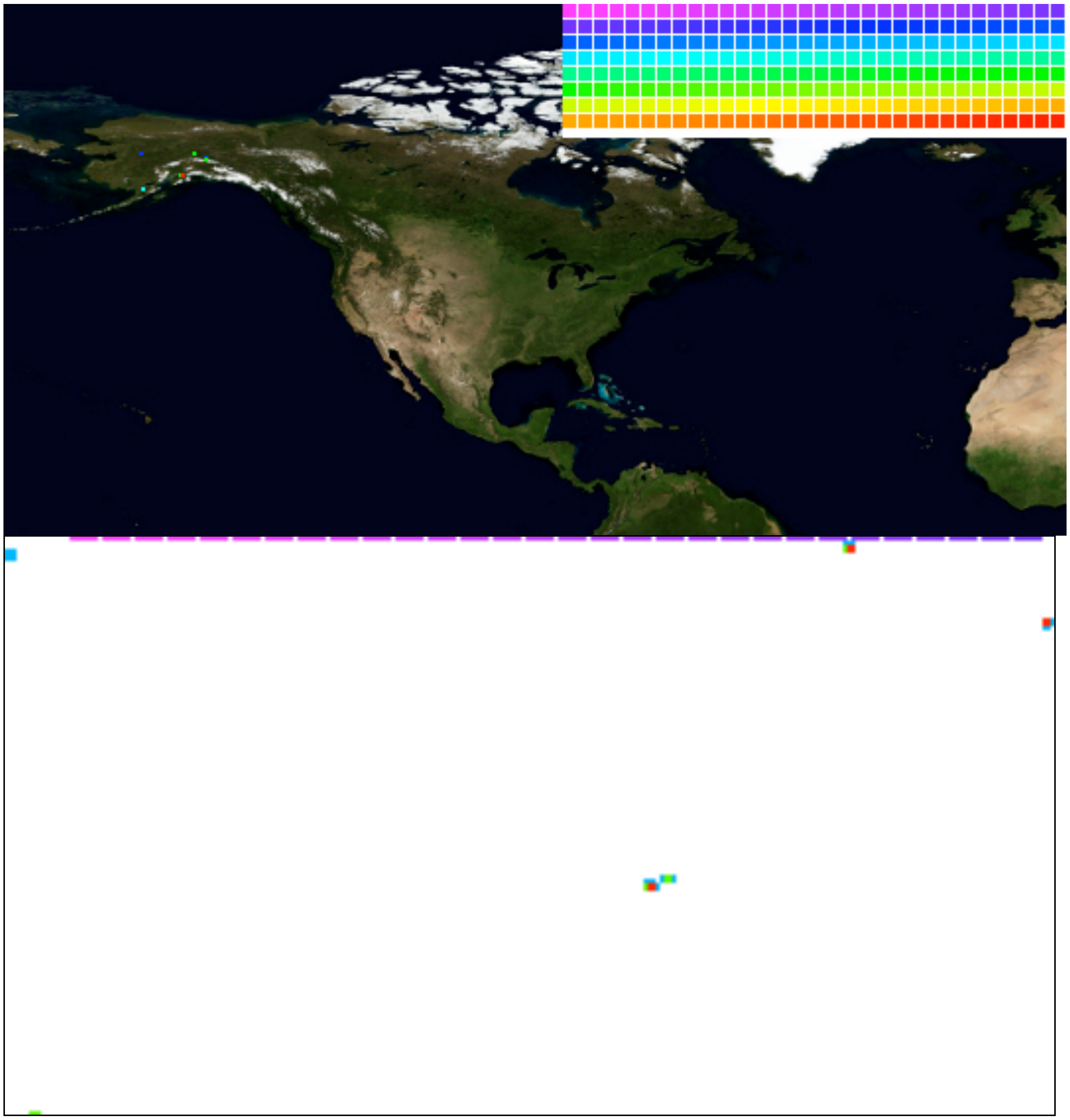
6.2.3.6 f2_runway_elements_alaska_40k.xml

```
total size: 39024
composition: %elems = 41.6, %att(names) = 4.9, %att(values) = 16.7,
%text/c_data = 35.7, %other = 1.1
elements: total count = 455, differents = 36, min. occ. = 1, max. occ.
= 14, avg. occ. = 12.6, avg. size = 35.7 bytes
attributes: total count = 197, avg. size = 9.6 bytes (names) & 33.0
bytes (values)
text / c_data: %coords = 91.6, %dates = 7.5, %other = 0.9
```



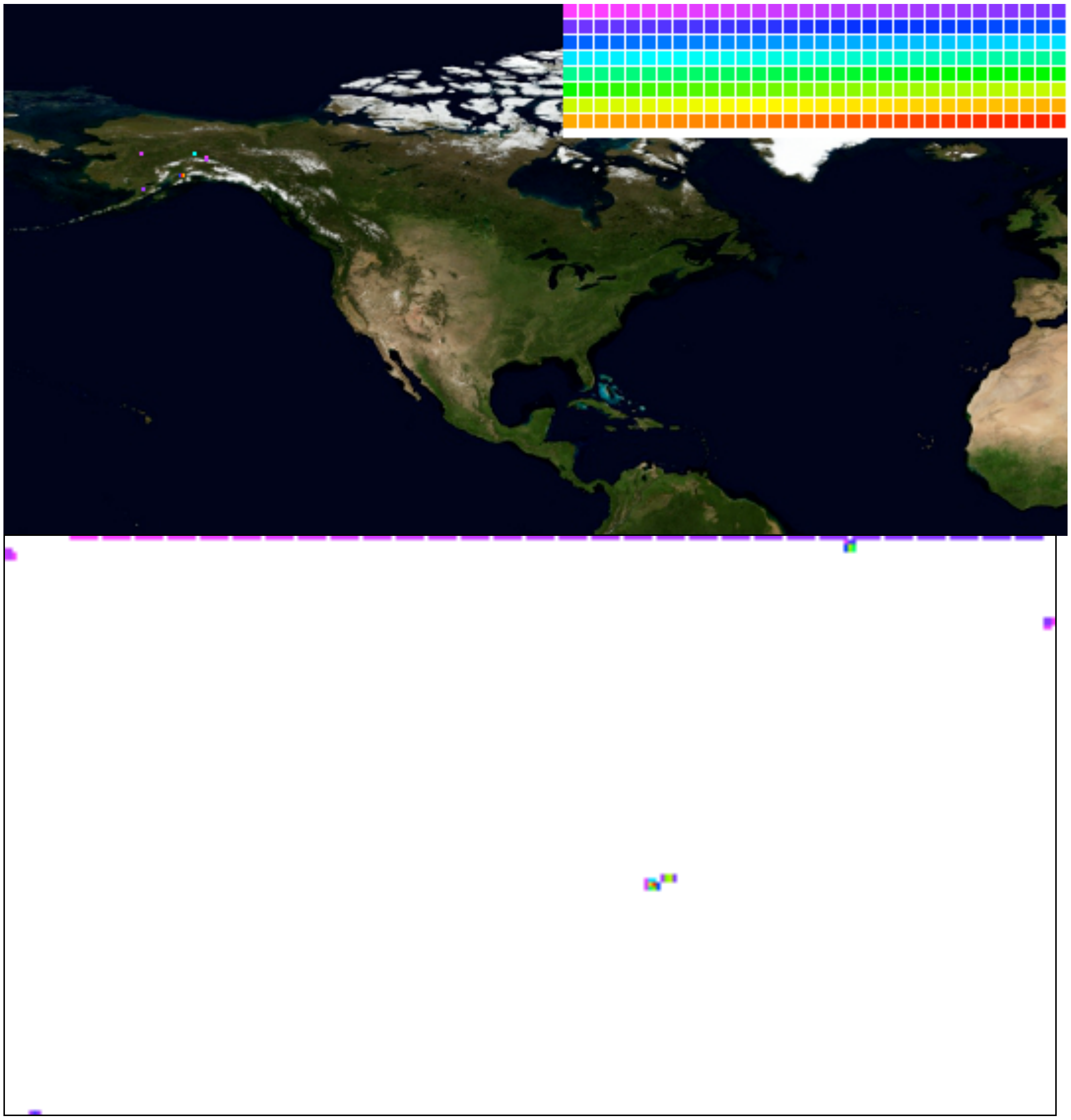
6.2.3.7 f2_runways_alaska_22k.xml

```
total size: 21988
composition: %elems = 61.7, %att(names) = 6.2, %att(values) = 20.7,
%text/c_data = 10.0, %other = 1.5
elements: total count = 361, differents = 31, min. occ. = 1, max. occ.
= 14, avg. occ. = 11.6, avg. size = 37.6 bytes
attributes: total count = 141, avg. size = 9.7 bytes (names) & 32.2
bytes (values)
text / c_data: %coords = 45.8, %dates = 47.7, %other = 6.5
```



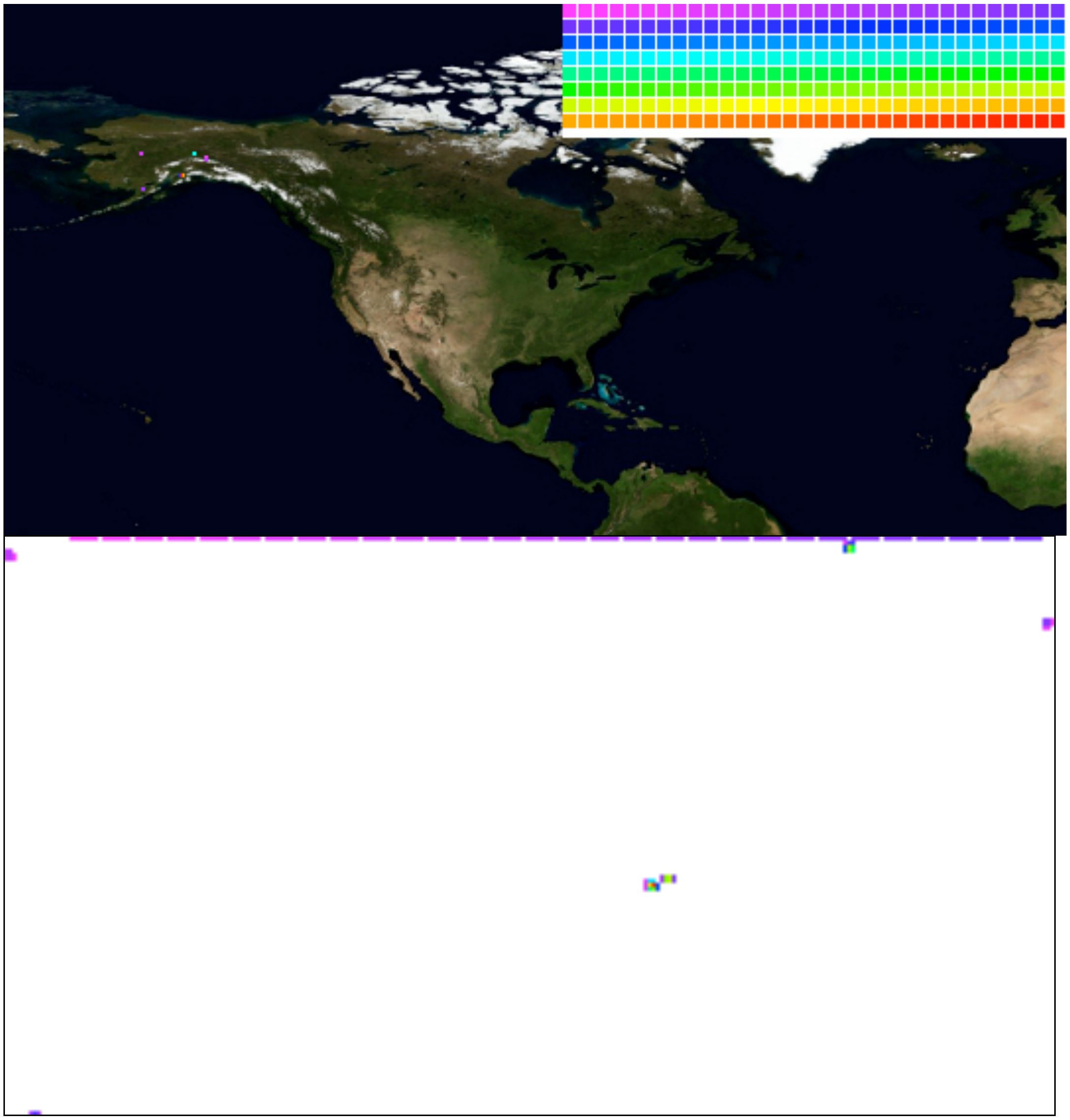
6.2.3.8 f2_taxiway_elements_alaska_730k.xml

```
total size: 737057
composition: %elems = 37.9, %att(names) = 4.0, %att(values) = 15.7,
%text/c_data = 41.6, %other = 0.8
elements: total count = 7455, differents = 34, min. occ. = 1, max. occ.
= 231, avg. occ. = 219.3, avg. size = 37.5 bytes
attributes: total count = 3016, avg. size = 9.7 bytes (names) & 38.3
bytes (values)
text / c_data: %coords = 94.0, %dates = 5.6, %other = 0.3
```



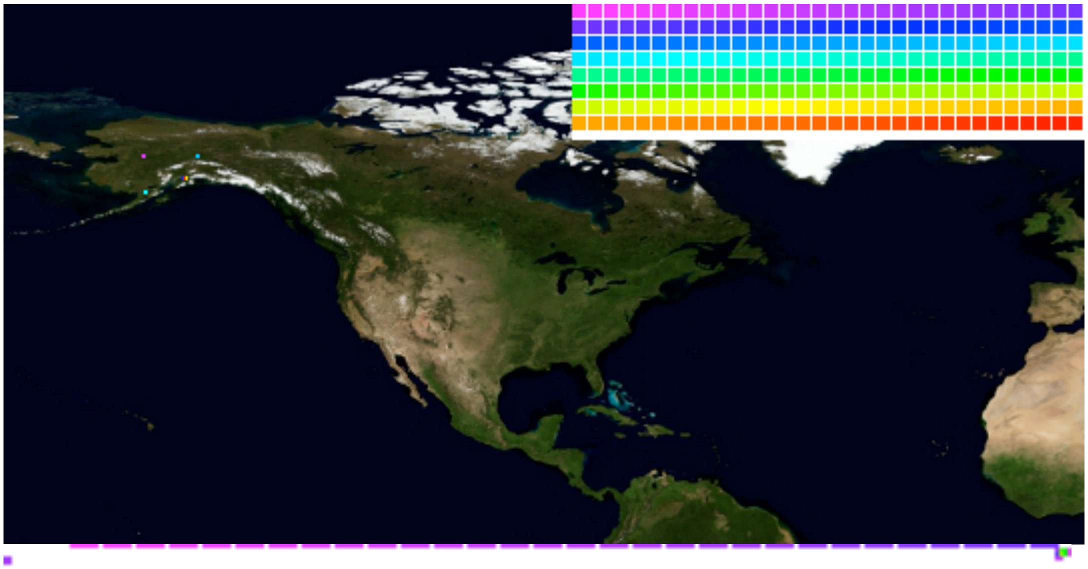
6.2.3.9 f2_taxiways_alaska_375k.xml

```
total size: 388601
composition: %elems = 61.3, %att(names) = 5.9, %att(values) = 21.1,
%text/c_data = 10.4, %other = 1.2
elements: total count = 6238, differents = 29, min. occ. = 1, max. occ.
= 231, avg. occ. = 215.1, avg. size = 38.2 bytes
attributes: total count = 2323, avg. size = 9.9 bytes (names) & 35.4
bytes (values)
text / c_data: %coords = 35.0, %dates = 42.9, %other = 22.1
```



6.2.3.10 f2_vertical_structure_alaska_230k.xml

```
total size: 232073
composition: %elems = 58.0, %att(names) = 5.4, %att(values) = 23.9,
%text/c_data = 11.7, %other = 1.1
elements: total count = 3509, differents = 30, min. occ. = 1, max. occ.
= 125, avg. occ. = 117.0, avg. size = 38.3 bytes
attributes: total count = 1263, avg. size = 9.9 bytes (names) & 43.9
bytes (values)
text / c_data: %coords = 35.8, %dates = 34.6, %other = 29.6
```



6.2.3.11 Analysis

Complexity:

If we based the complexity on the number of different elements we can find in the file, we get this chart list of complexity:

- Airspaces: 60
- Navaids: 42
- RunWays Elements: 36

- Taxiways elements: 34
- RunWays: 31
- Vertical structures: 30
- Taxiways: 29
- Geo Borders: 29
- Routes: 28

Composition:

We place in “c-data” everything not included inside elements. It’s not necessary free-text, it could be dates, coordinates, enumerates… any type or sequence of types defined in XSD. The composition of every XML is split into percentage of:

- Elements (tags)
- Attributes (names and values)
- C_DATA (coordinates, dates and the rest)

After analysis of composition, we made 2 groups of features:

- A group using intensively coordinates made of:
  - Airspaces, Geo Borders, TaxiWays Elements, and RunWays Elements
- A second group using only few coordinates made of:
  - Navaids, RunWays, TaxiWays, Routes, Vertical Structures

The first group should benefit of EXI Schema knowledge or CWXML for conversion of float or double to binary.

Entropy:

On this axis, we consider the importance taken by elements compared to the one taken by attributes and c_data. We notice some files with very long attributes values, sometimes paired with a high presence of attributes in the file:

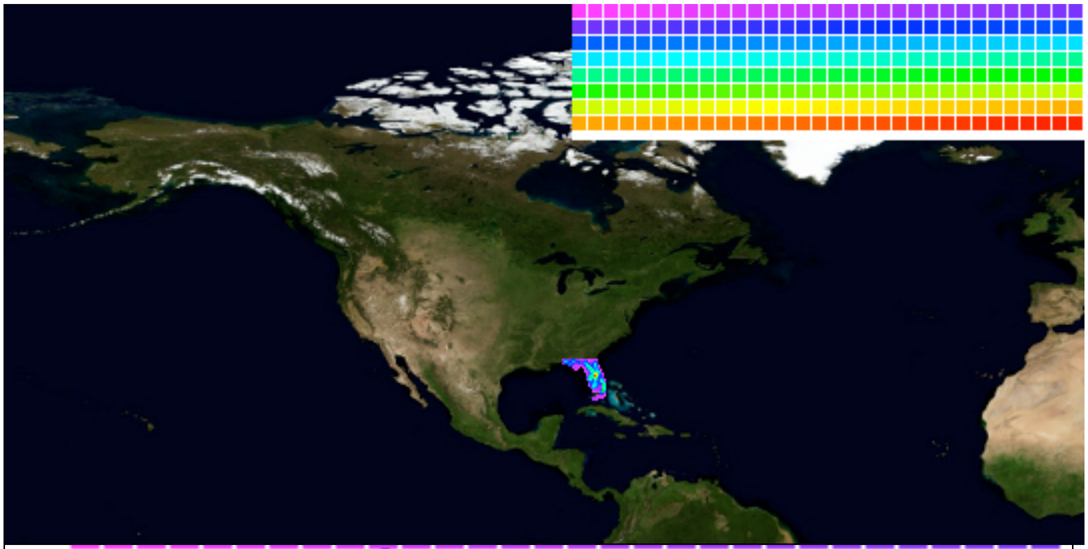
- Vertical structures and Navaid present longs attributes values, with a high ratio of space taken by attribute in the file. This will certainly be an issue for EXI grammar based compression.

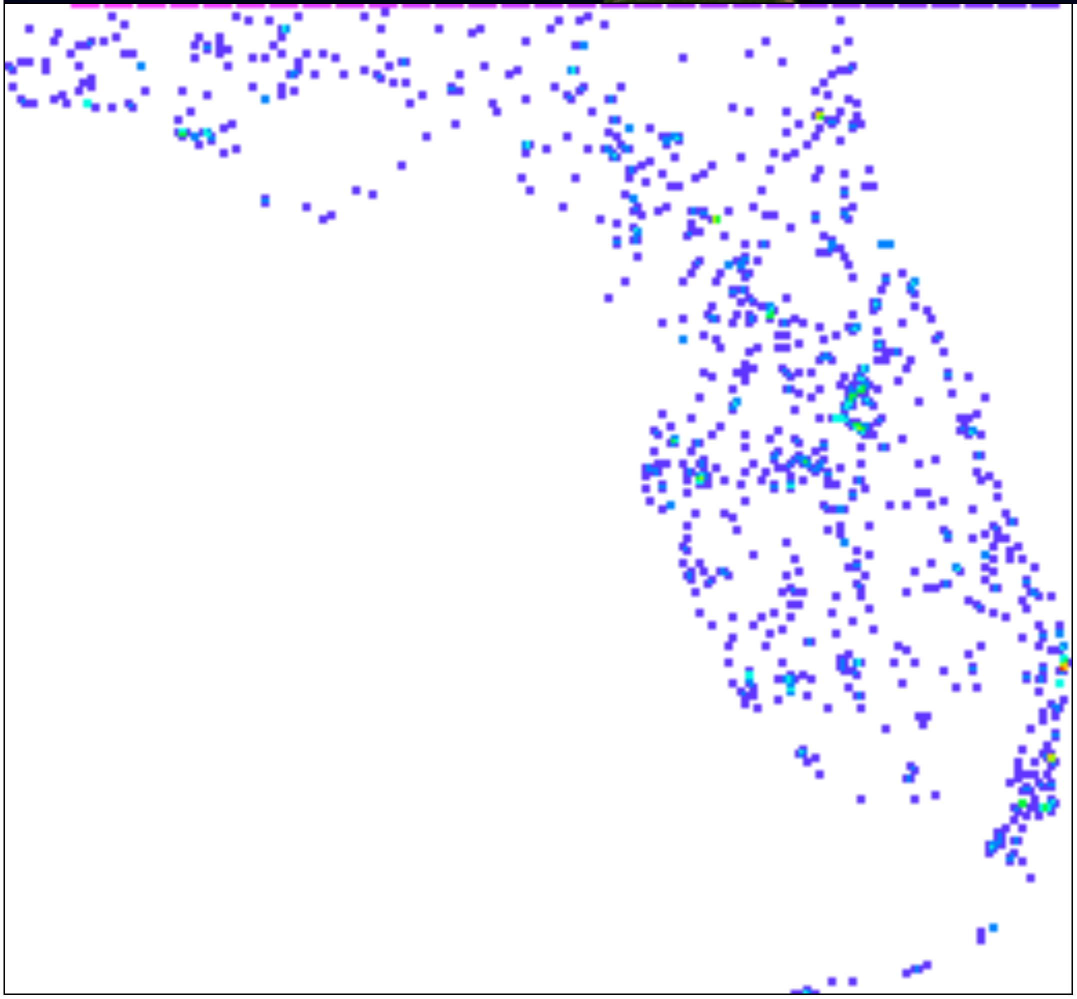
- RunWays, Route and TaxiWays present a majority of space occupied by elements and also a few number of different elements, short attributes, and not much c_data. This is good for all candidates, and especially for EXI with both grammar and post-compression.

6.2.4 Large files (>1MB)

6.2.4.1 f3_airports_from_florida_2_2m.xml

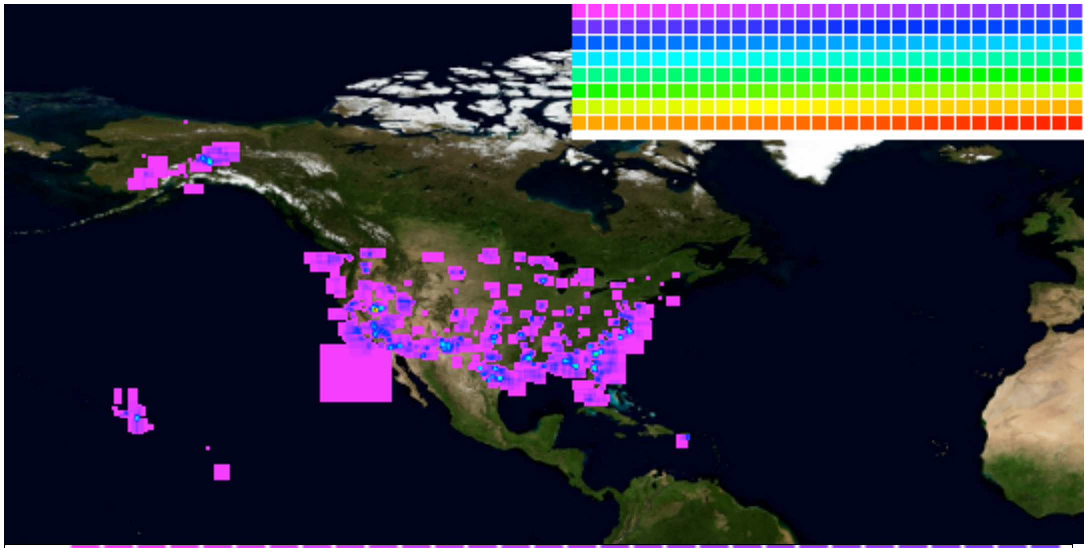
```
total size: 2146231
composition: %elems = 64.6, %att(names) = 5.2, %att(values) = 19.2,
%text/c_data = 9.8, %other = 1.1
elements: total count = 37444, differents = 41, min. occ. = 1, max.
occ. = 1858, avg. occ. = 913.3, avg. size = 37.0 bytes
attributes: total count = 12157, avg. size = 9.2 bytes (names) & 33.9
bytes (values)
text / c_data: %coords = 44.5, %dates = 33.4, %other = 22.1
```

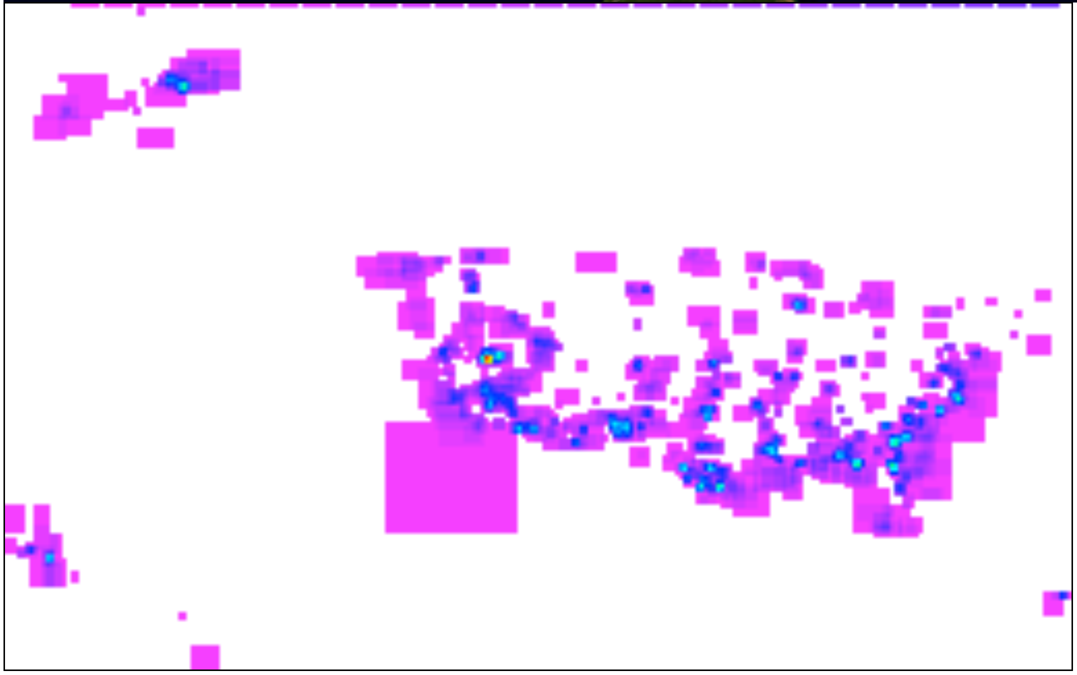




6.2.4.2 f3_airspaces.xml

```
total size: 7036816
composition: %elems = 61.0, %att(names) = 3.9, %att(values) = 15.9,
%text/c_data = 18.0, %other = 1.1
elements: total count = 129511, differents = 100, min. occ. = 1, max.
occ. = 21022, avg. occ. = 1295.1, avg. size = 33.1 bytes
attributes: total count = 33058, avg. size = 8.4 bytes (names) & 33.8
bytes (values)
text / c_data: %coords = 26.0, %dates = 0.1, %other = 73.8
```





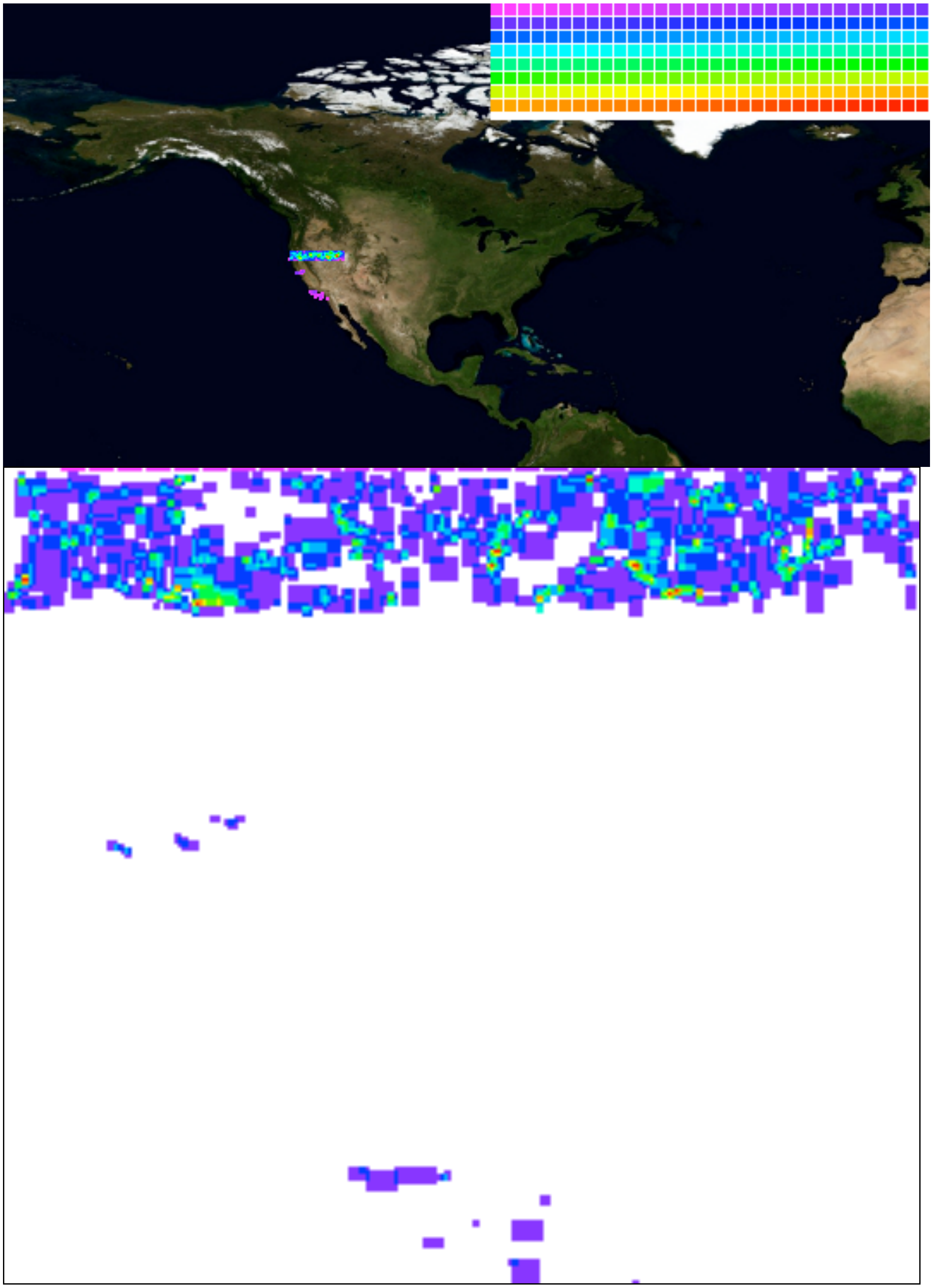
6.2.4.3 f3_estonia-ows8.xml

```
total size: 3761035
composition: %elems = 62.2, %att(names) = 4.9, %att(values) = 11.1,
%text/c_data = 20.6, %other = 1.2
```

```
elements: total count = 84910, differents = 395, min. occ. = 1, max.
occ. = 25505, avg. occ. = 215.0, avg. size = 27.5 bytes
attributes: total count = 22543, avg. size = 8.2 bytes (names) & 18.4
bytes (values)
text / c_data: %coords = 69.1, %dates = 9.5, %other = 21.4
```

6.2.4.4 f3_geo_borders_calif_nev_1_5m.xml

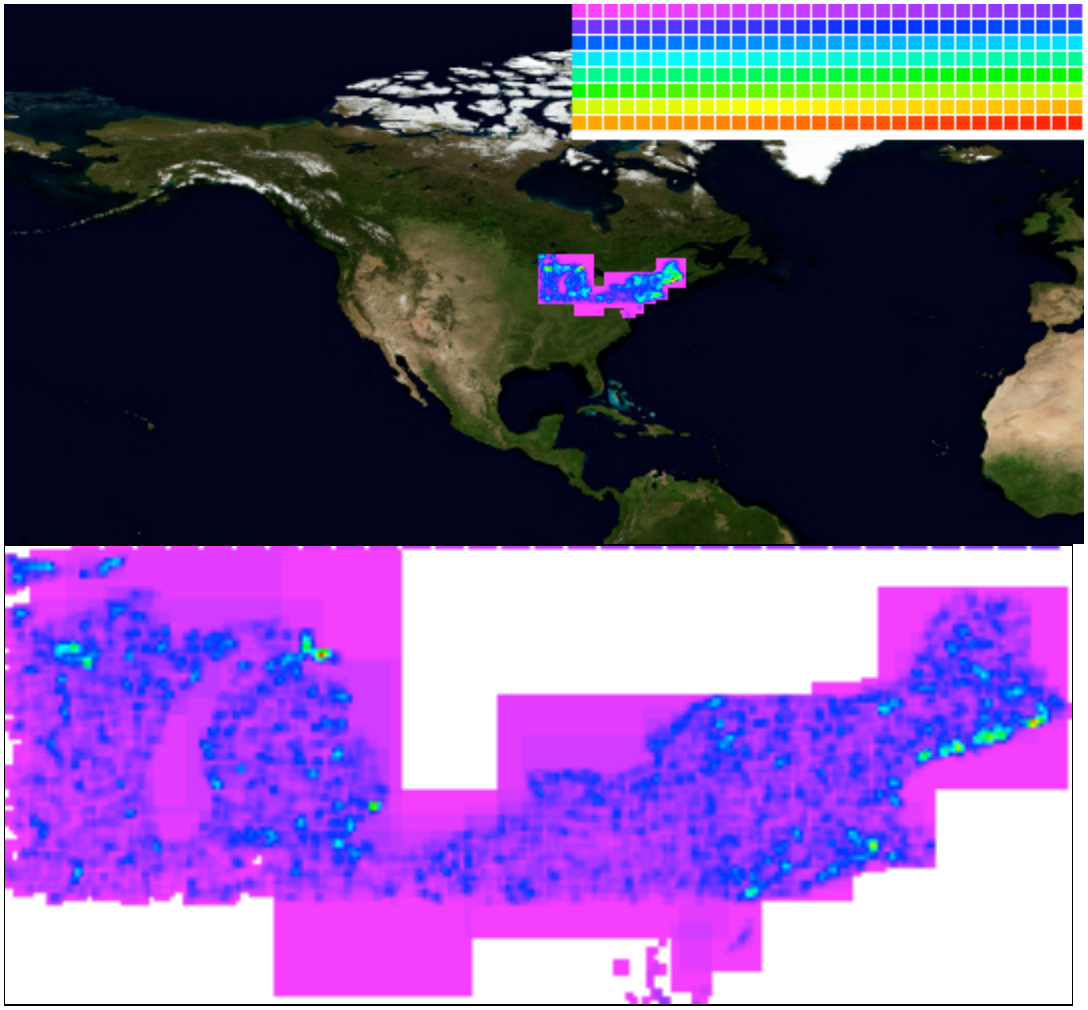
```
total size: 1563628
composition: %elems = 40.6, %att(names) = 4.2, %att(values) = 9.5,
%text/c_data = 44.8, %other = 0.8
elements: total count = 18220, differents = 29, min. occ. = 1, max.
occ. = 656, avg. occ. = 628.3, avg. size = 34.9 bytes
attributes: total count = 6573, avg. size = 10.1 bytes (names) & 22.6
bytes (values)
text / c_data: %coords = 90.9, %dates = 7.0, %other = 2.1
```



6.2.4.5 f3_geo_borders_megalop_20m.xml

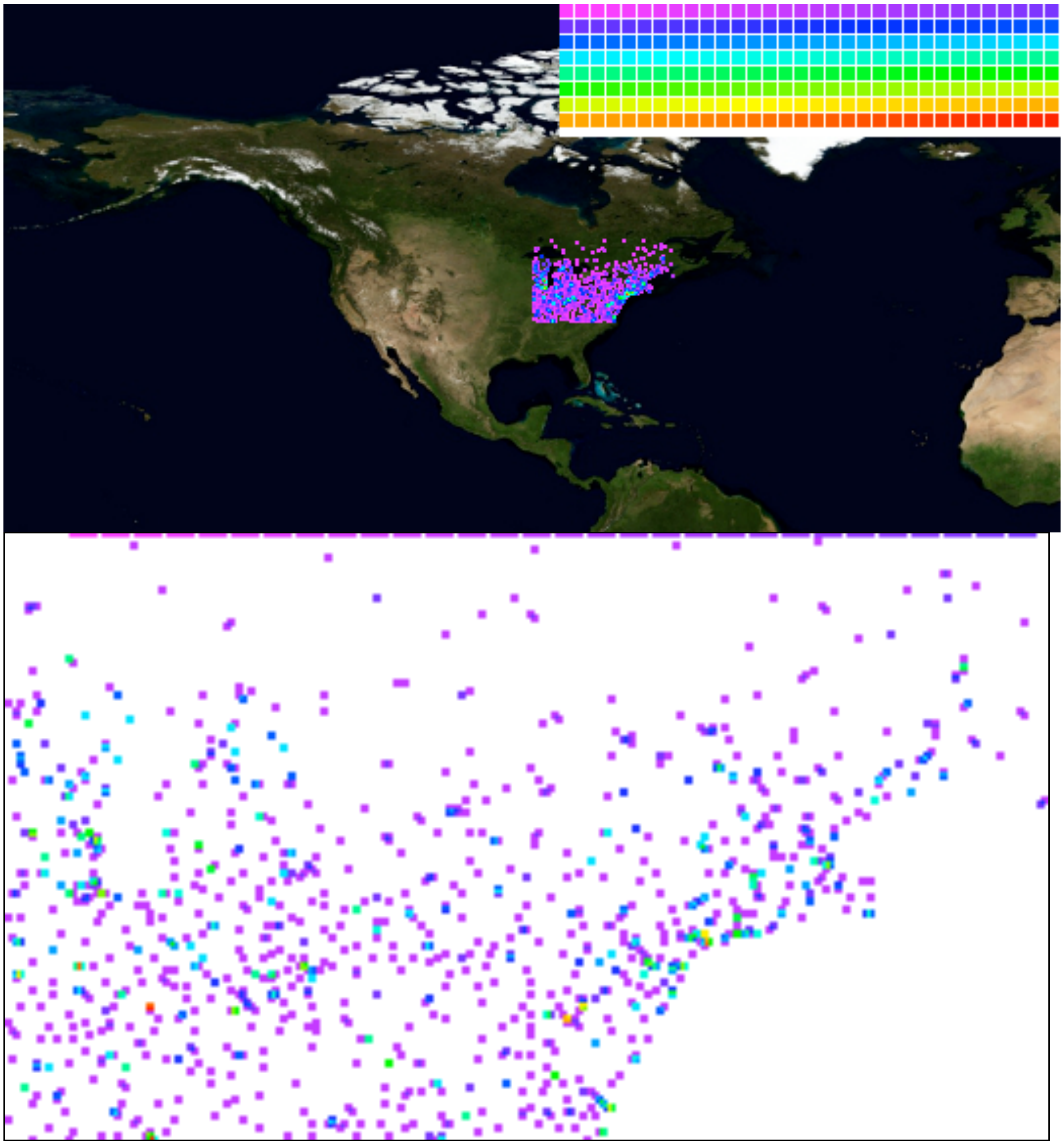
```
total size: 19802453
composition: %elems = 38.4, %att(names) = 4.0, %att(values) = 9.0,
%text/c_data = 47.8, %other = 0.8
```

```
elements: total count = 217052, differents = 29, min. occ. = 1, max.
occ. = 7890, avg. occ. = 7484.6, avg. size = 35.0 bytes
attributes: total count = 78913, avg. size = 10.1 bytes (names) & 22.6
bytes (values)
text / c_data: %coords = 92.4, %dates = 6.2, %other = 1.4
```



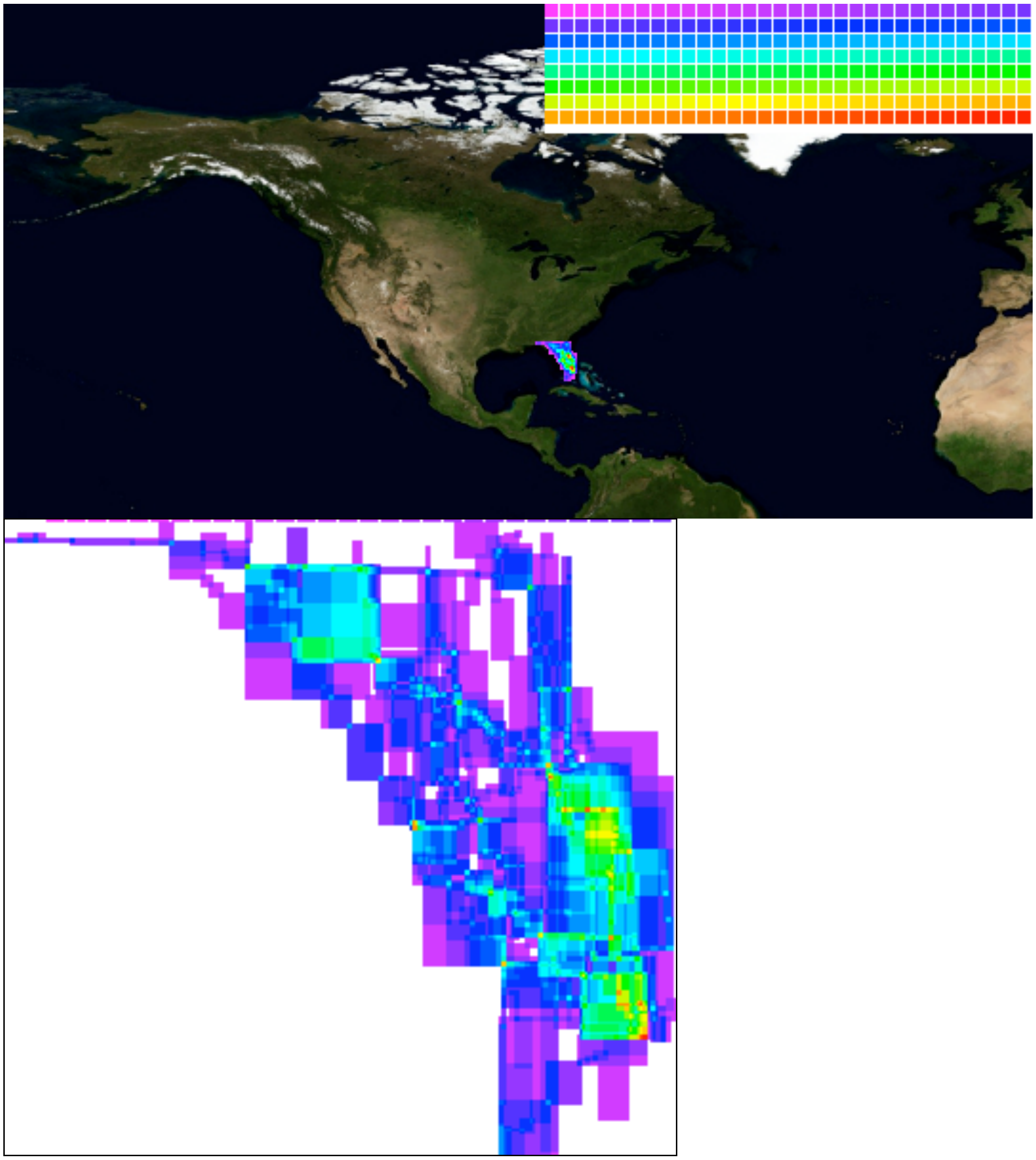
6.2.4.6 f3_navaids_megalop_3_6m.xml

```
total size: 3594804
composition: %elems = 57.0, %att(names) = 5.3, %att(values) = 25.3,
%text/c_data = 11.4, %other = 1.1
elements: total count = 58183, differents = 42, min. occ. = 1, max.
occ. = 1880, avg. occ. = 1385.3, avg. size = 35.2 bytes
attributes: total count = 18934, avg. size = 10.0 bytes (names) & 48.0
bytes (values)
text / c_data: %coords = 39.4, %dates = 34.5, %other = 26.1
```



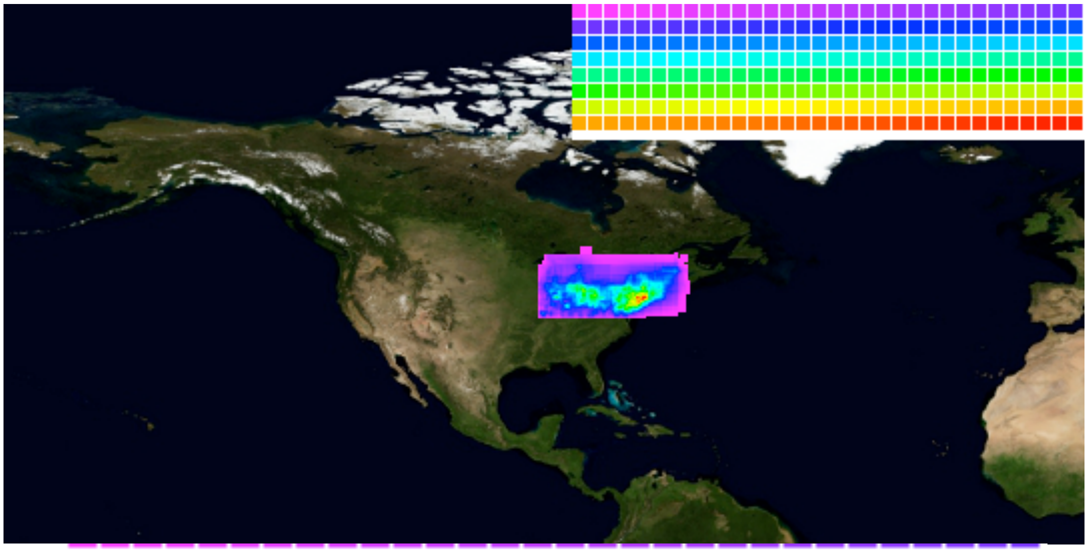
6.2.4.7 f3_route_segment_florida_1_2m.xml

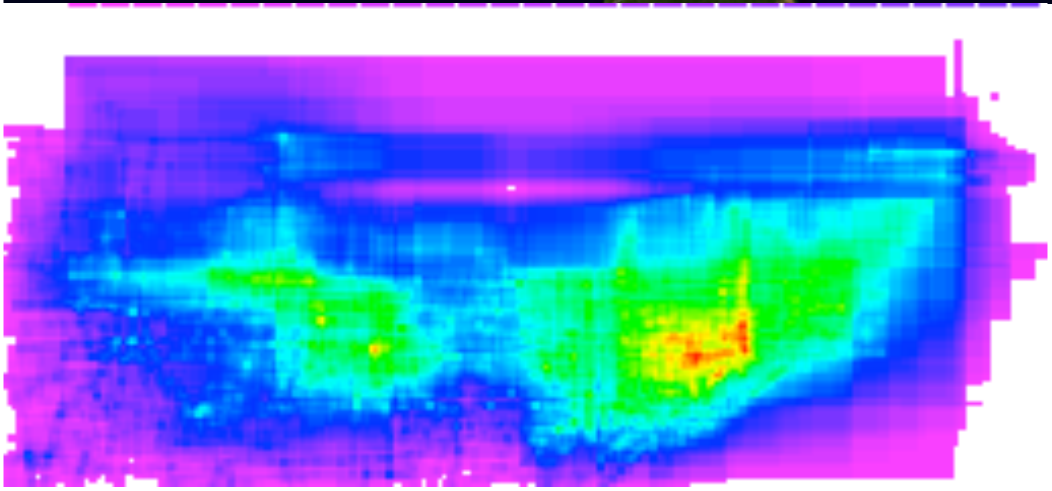
```
total size: 1204582
composition: %elems = 60.3, %att(names) = 7.9, %att(values) = 17.3,
%text/c_data = 12.8, %other = 1.6
elements: total count = 20314, differents = 34, min. occ. = 1, max.
occ. = 1098, avg. occ. = 597.5, avg. size = 35.8 bytes
attributes: total count = 9895, avg. size = 9.7 bytes (names) & 21.1
bytes (values)
text / c_data: %coords = 70.5, %dates = 26.8, %other = 2.7
```



6.2.4.8 f3_route_segment_megalop_15m.xml

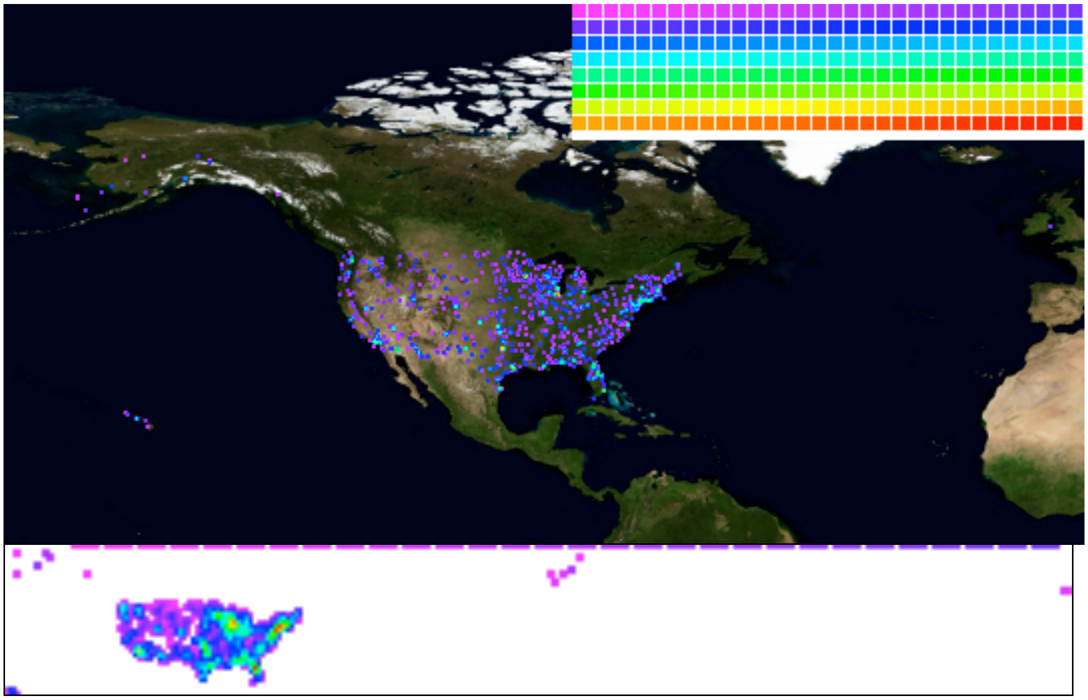
```
total size: 14452017
composition: %elems = 60.2, %att(names) = 7.9, %att(values) = 17.4,
%text/c_data = 12.9, %other = 1.6
elements: total count = 243424, differents = 34, min. occ. = 1, max.
occ. = 13158, avg. occ. = 7159.5, avg. size = 35.8 bytes
attributes: total count = 118435, avg. size = 9.7 bytes (names) & 21.2
bytes (values)
text / c_data: %coords = 70.6, %dates = 26.6, %other = 2.8
```





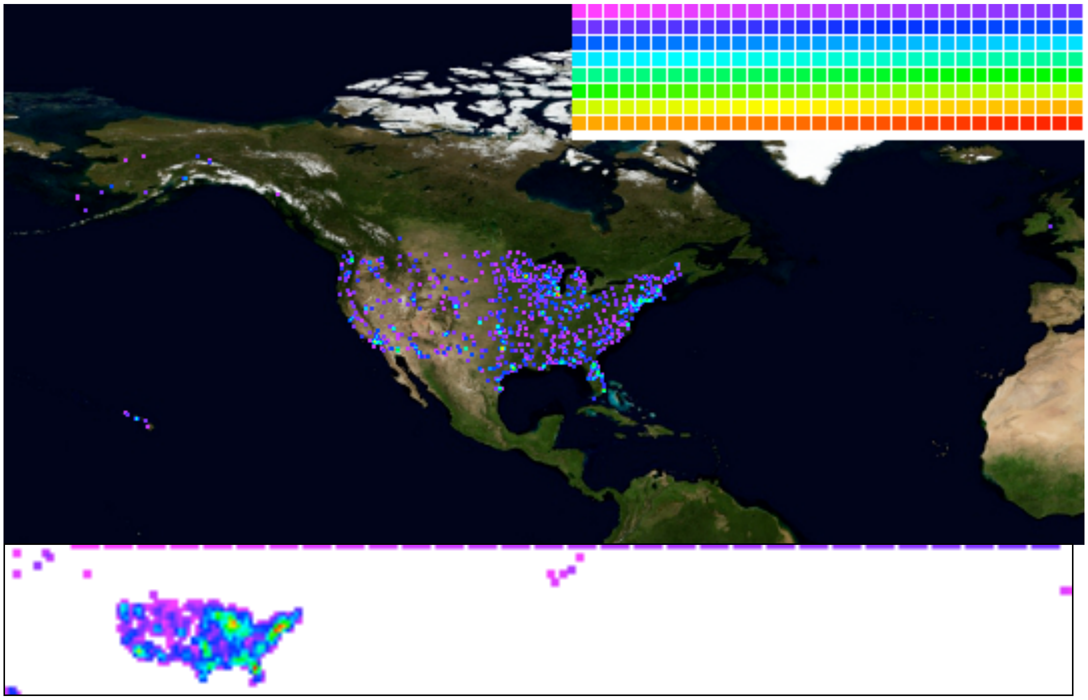
6.2.4.9 f3_runway_elements_all_4_4m.xml

```
total size: 4434489
composition: %elems = 43.4, %att(names) = 4.6, %att(values) = 16.4,
%text/c_data = 34.6, %other = 1.0
elements: total count = 53633, differents = 37, min. occ. = 1, max.
occ. = 1625, avg. occ. = 1449.5, avg. size = 35.9 bytes
attributes: total count = 21687, avg. size = 9.4 bytes (names) & 33.5
bytes (values)
text / c_data: %coords = 91.0, %dates = 7.9, %other = 1.0
```



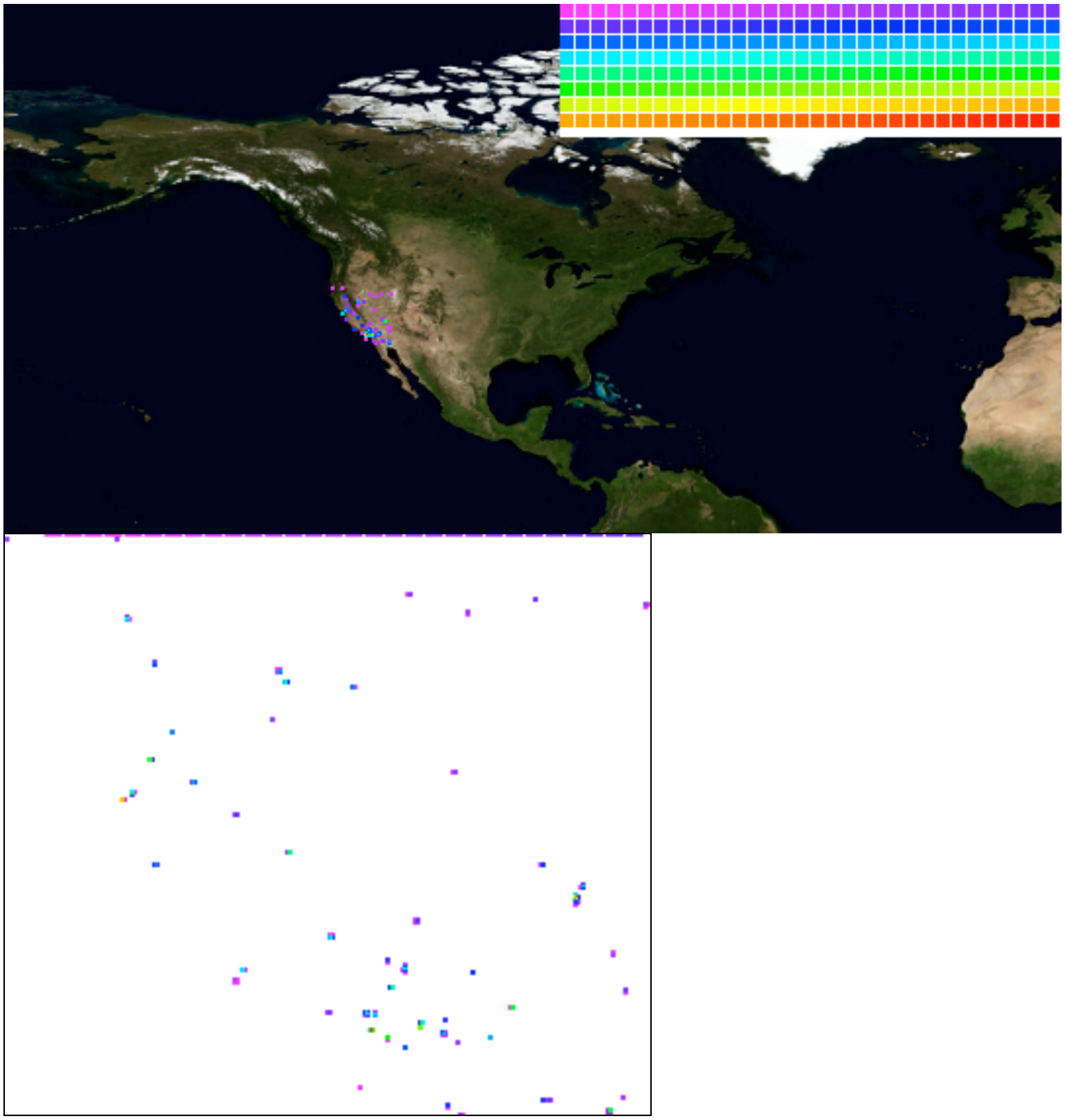
6.2.4.10 f3_runways_all_2_5m.xml

```
total size: 2581599
composition: %elems = 63.9, %att(names) = 5.6, %att(values) = 19.4,
%text/c_data = 9.9, %other = 1.2
elements: total count = 43642, differents = 47, min. occ. = 1, max.
occ. = 1635, avg. occ. = 928.6, avg. size = 37.8 bytes
attributes: total count = 15288, avg. size = 9.4 bytes (names) & 32.8
bytes (values)
text / c_data: %coords = 44.0, %dates = 47.9, %other = 8.1
```



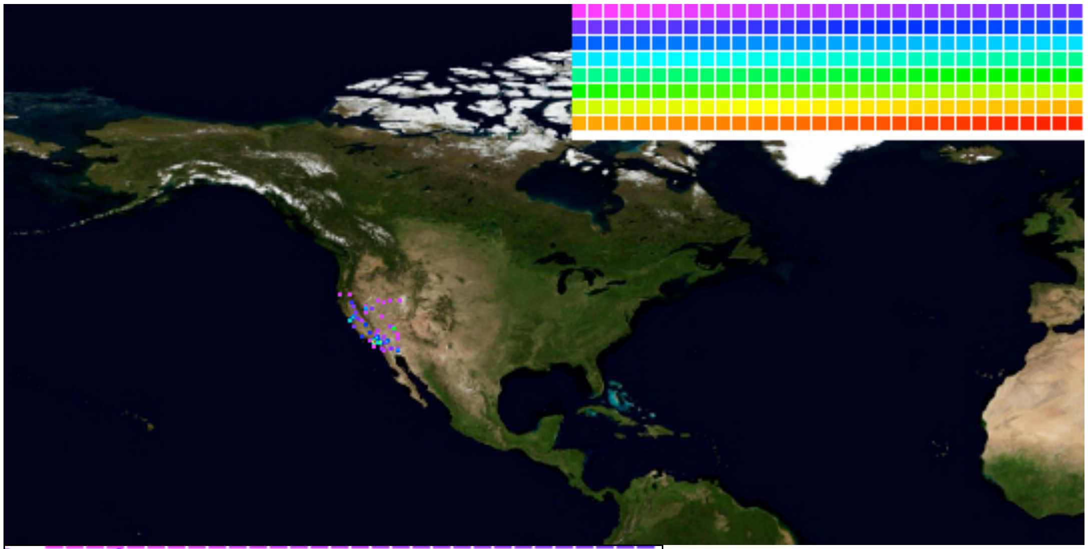
6.2.4.11 f3_taxiway_elements_calif_nev_5_4m.xml

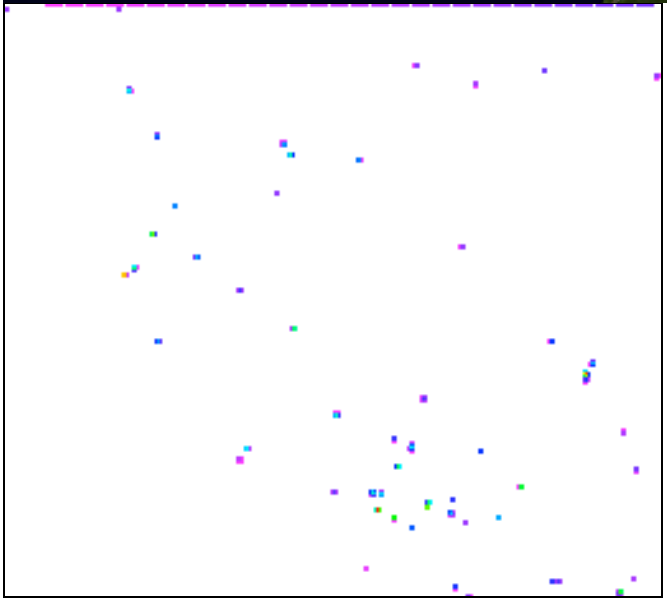
```
total size: 5410260
composition: %elems = 41.6, %att(names) = 4.3, %att(values) = 17.1,
%text/c_data = 36.0, %other = 0.9
elements: total count = 60083, differents = 34, min. occ. = 1, max.
occ. = 1862, avg. occ. = 1767.1, avg. size = 37.5 bytes
attributes: total count = 24219, avg. size = 9.7 bytes (names) & 38.3
bytes (values)
text / c_data: %coords = 92.4, %dates = 7.2, %other = 0.4
```



6.2.4.12 f3_taxiways_calif_nev_3m.xml

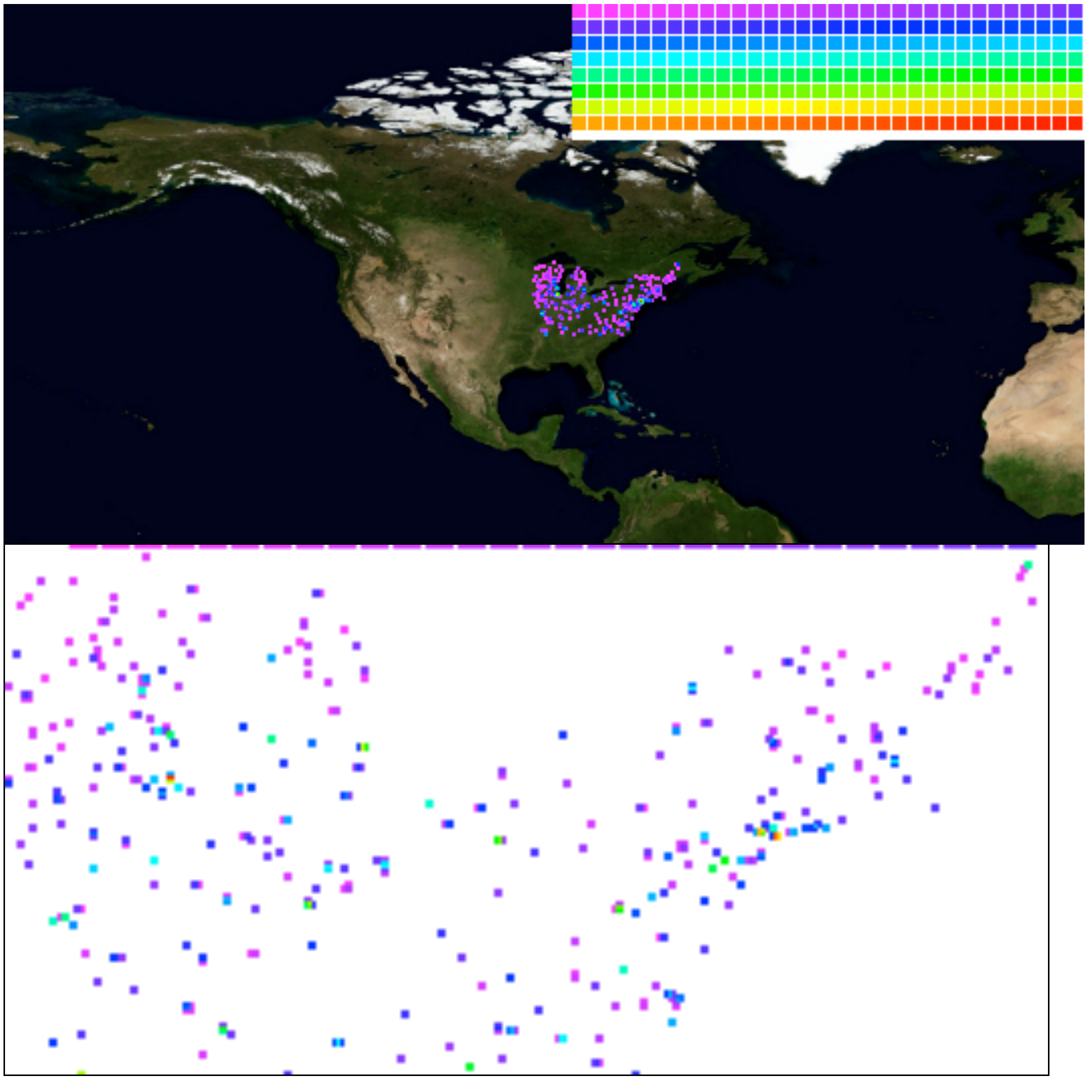
```
total size: 3143078
composition: %elems = 61.6, %att(names) = 5.9, %att(values) = 21.0,
%text/c_data = 10.4, %other = 1.2
elements: total count = 50706, differents = 29, min. occ. = 1, max.
occ. = 1862, avg. occ. = 1748.5, avg. size = 38.2 bytes
attributes: total count = 18633, avg. size = 9.9 bytes (names) & 35.3
bytes (values)
text / c_data: %coords = 34.8, %dates = 42.7, %other = 22.4
```





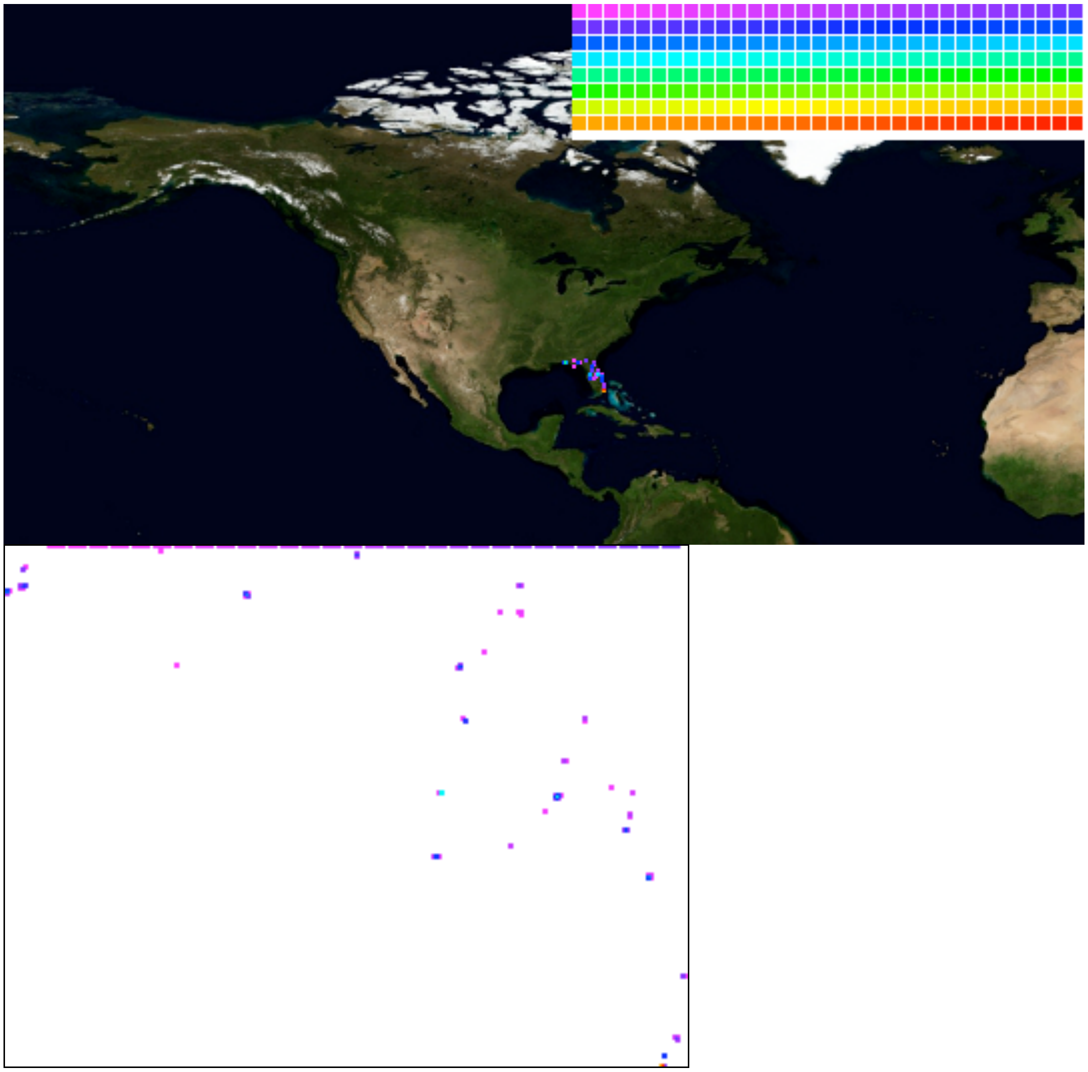
6.2.4.13 f3_taxiways_megalop_12m.xml

```
total size: 12041044
composition: %elems = 61.6, %att(names) = 5.9, %att(values) = 21.0,
%text/c_data = 10.3, %other = 1.2
elements: total count = 194235, differents = 29, min. occ. = 1, max.
occ. = 7160, avg. occ. = 6697.8, avg. size = 38.2 bytes
attributes: total count = 71613, avg. size = 9.9 bytes (names) & 35.3
bytes (values)
text / c_data: %coords = 33.8, %dates = 43.4, %other = 22.8
```



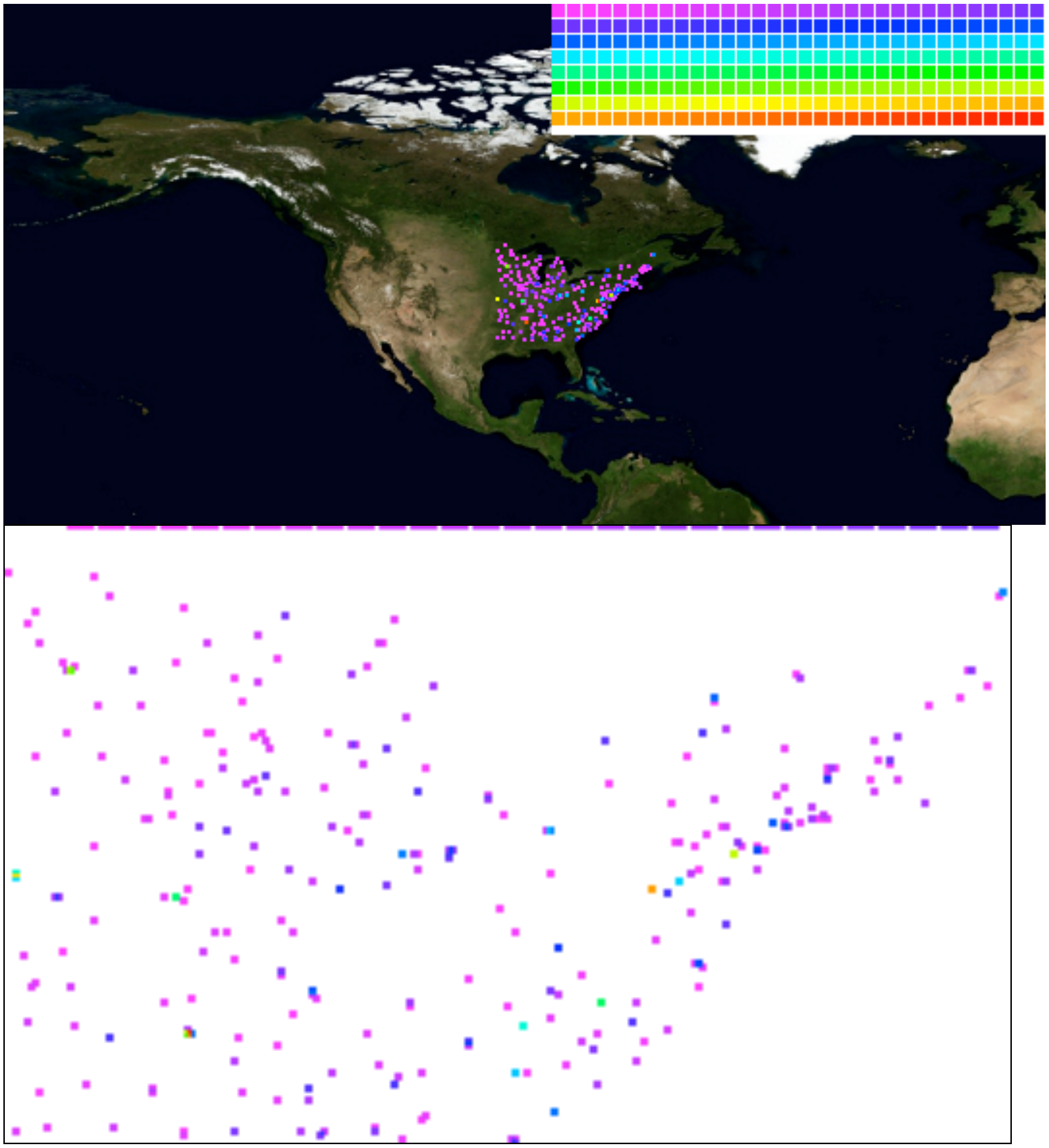
6.2.4.14 f3_vertical_structure_florida_1_1m.xml

```
total size: 1150632
composition: %elems = 58.2, %att(names) = 5.4, %att(values) = 23.8,
%text/c_data = 11.5, %other = 1.1
elements: total count = 17474, differents = 35, min. occ. = 1, max.
occ. = 622, avg. occ. = 499.3, avg. size = 38.3 bytes
attributes: total count = 6235, avg. size = 9.9 bytes (names) & 44.0
bytes (values)
text / c_data: %coords = 34.8, %dates = 35.2, %other = 30.0
```



6.2.4.15 f3_vertical_structure_megalop_8m.xml

```
total size: 7802553
composition: %elems = 58.2, %att(names) = 5.4, %att(values) = 23.8,
%text/c_data = 11.5, %other = 1.1
elements: total count = 118646, differents = 35, min. occ. = 1, max.
occ. = 4222, avg. occ. = 3389.9, avg. size = 38.3 bytes
attributes: total count = 42257, avg. size = 9.9 bytes (names) & 44.0
bytes (values)
text / c_data: %coords = 34.6, %dates = 35.4, %other = 30.0
```



6.2.5 Technical files

Only AIXM files are analyzed here, other files are purely “technical” and do not have any “statistical” interest.

6.2.5.1 f4_all_features_from_family_2_in_disorder.xml

```
total size: 2741692
composition: %elems = 47.1, %att(names) = 4.6, %att(values) = 16.3,
%text/c_data = 31.0, %other = 0.9
```

```
elements: total count = 35362, differents = 106, min. occ. = 1, max.
occ. = 1193, avg. occ. = 333.6, avg. size = 36.5 bytes
attributes: total count = 12842, avg. size = 9.9 bytes (names) & 34.8
bytes (values)
text / c_data: %coords = 85.3, %dates = 10.5, %other = 4.2
```

6.2.5.2 f4_all_features_from_family_2_sorted_by_feature.xml

```
total size: 2741692
composition: %elems = 47.1, %att(names) = 4.6, %att(values) = 16.3,
%text/c_data = 31.0, %other = 0.9
elements: total count = 35362, differents = 106, min. occ. = 1, max.
occ. = 1193, avg. occ. = 333.6, avg. size = 36.5 bytes
attributes: total count = 12842, avg. size = 9.9 bytes (names) & 34.8
bytes (values)
text / c_data: %coords = 85.3, %dates = 10.5, %other = 4.2
```

6.2.5.3 doubles.xml

```
total size: 22366
composition: %elems = 0.1, %att(names) = 0.0, %att(values) = 0.1,
%text/c_data = 99.7, %other = 0.1
elements: total count = 29, differents = 29, min. occ. = 1, max. occ. =
1, avg. occ. = 1.0, avg. size = 35.4 bytes
attributes: total count = 23, avg. size = 11.1 bytes (names) & 28.5
bytes (values)
text / c_data: %coords = 100.0, %dates = 0.0, %other = 0.0
```

We use quite a batch of other technical files, but we don't mention them here because they do not have any interest for the benchmark as they were used to investigated bugs.

6.2.6 Infuence of AIXM encoding schemes of the various WFS servers

During the phase of selection of AIXM input files to class into the 4 families, we encountered some divergences between the way WFS servers output or "encode" AIXM, from their internal database for similar WFS requests. The following chapters address those differences, and how they may impact the compression performance.

The "exposition" of those differences to the various candidates are shown by some special scenarios (inside the technical family), and results can be seen from the archive file report_compaction_for_aixm_various_encoding.zip.

6.2.6.1 Formating

6.2.6.1.1 Indenting

An XML output is fine to read once formatted, but in general the client do not need this human oriented formatting because it will not read directly the raw text, but will use a lib to do it (sax, stax, dom,...). Space, carriage returns, line feeds and tabulation are unnecessary as deflate will try to make a past reference only on a 3 bytes match, so theses characters could cost a lot, specially if they do not occur for each element in the same way. A past reference will cost less if it is near and not too long. So if you have for instance <gml:pos> elements indented different ways, it will certainly cost some raw space not to be compressed in the symbol flow. For instance:

```
        <gml:pos>24.7533333 59.4480556</gml:pos>
        ...
                <gml:pos>24.7533333 59.4480556</gml:pos>
        ...
        <gml:pos>22.5095889 58.2230694</gml:pos>
```

will cost raw white space before the second <gml:pos> line

Snowflake: generally one different line for each element, except for some shorts sequence including C-data / text , e.g.:

```
<aixm:upperLimit uom="FL">179</aixm:upperLimit>
<gml:posList srsDimension="2" count="x">-99.6836111 47.4166667 ...
</gml:posList>
<gco:DateTime>2010-05-10T00:00:00.000Z</gco:DateTime>
```

Comsoft: same as Snowflake with additional indenting (one space per depth level) and a double CR between features

```
  <aixm-message-5.1:hasMember>
   <VerticalStructure gml:id="ID000002">
    <gml:identifier codeSpace="http://www.comsoft.aero/cadas-
aimdb/caw">17b32e8d-955d-4ed5-9d26-879db3d717d9</gml:identifier>
    <timeSlice>
     <VerticalStructureTimeSlice gml:id="ID000004">
      <gml:validTime>
       <gml:TimePeriod gml:id="ID000003">
        <gml:beginPosition>2010-04-08T00:00:00.000Z</gml:beginPosition>
```

Luciad: Same as Comsoft but with 2 spaces instead of one and no double CR between features

```
  <wfs:member>
    <ns5:Airspace gml:id="urn.uuid.55464991-797a-4ffc-85c0-
08fd9fa573c9">
      <gml:identifier codeSpace="http://www.faa.gov/nasr">55464991-
797a-4ffc-85c0-08fd9fa573c9</gml:identifier>
      <ns5:timeSlice>
        <ns5:AirspaceTimeSlice gml:id="urn.uuid.55464991-797a-4ffc-
85c0-08fd9fa573c9_1">
          <gml:validTime>
```

```
            <gml:TimePeriod gml:id="urn.uuid.55464991-797a-4ffc-85c0-
08fd9fa573c9_2">
              <gml:beginPosition>2011-03-10-05:00</gml:beginPosition>
```

Candidate exposition:

□ GZIP: Yes

□ Fast Info Set : Yes

□ BXML: Yes

□ EXI: No

Policy for benchmarking:

Canonization of XML, we will drop all formatting to be fair with gzip.

6.2.6.1.2 Autoclosing elements, aka <toto/>

Good for performance, but only if always used in the same way and not both

Snowflake: used for

```
aixm:associatedAirportHeliport
aixm:associatedRunway
aixm:associatedTaxiway
aixm:clientAirspace
aixm:EnRouteSegmentPoint (also used in the 2 tags version)
aixm:LinguisticNote (also used in the 2 tags version)
aixm:ownerOrganisation
aixm:pointChoice_fixDesignatedPoint
aixm:pointChoice_navaidSystem
aixm:routeFormed
aixm:servedAirport
aixm:serviceProvider
aixm:SurfaceCharacteristics (also used in the 2 tags version)
aixm:Timesheet (also used in the 2 tags version)
gmd:contact
gmd:identificationInfo
```

Comsoft: only used for gml:endPosition

Luciad: never used

Candidate exposition: (understanding to the presence of both <toto></toto> and <toto/> tags)

- GZIP: Yes
- Fast Info Set : No
- BXML: Yes
- EXI: No

Policy for benchmarking:

Canonization of XML, we will drop all formatting to be fair with gzip.

6.2.6.2 NameSpace referencing

As allowed by W3C (http://www.w3.org/TR/xml-names/), you can use a default namespace, scoping an element and its children. This saves some space. Also, namespaces references in elements and attributes can use a substitution value thanks to the xmlns:xx special attribute. For instance, both declarations are correct:

```
<html:html xmlns:html='http://www.w3.org/1999/xhtml'>
<html:html xmlns:h='http://www.w3.org/1999/xhtml'>
```

And "h" costs less space than "html"

Snowflake: namespace are always used for both elements and attributes, with their original name

Comsoft: AIXM is the default namespace, and others ns are used with their original name

Luciad : AIXM is the default namespace and others ns are used in a short version, generally no more than 3 chars, e.g:

```
default => urn:us:gov:dot:faa:aim:saa:5.1

ns0 => urn:us:gov:dot:faa:aim:saa:sua:5.1
ns1 => http://www.isotc211.org/2005/gts
ns2 => http://www.isotc211.org/2005/gco
ns3 => http://www.isotc211.org/2005/gss
ns4 => http://www.isotc211.org/2005/gsr
ns5 => http://www.aixm.aero/schema/5.1
ns6 => http://www.isotc211.org/2005/gmd

fes => http://www.opengis.net/fes/2.0
```

```
gml => http://www.opengis.net/gml/3.2
ows => http://www.opengis.net/ows/1.1
wfs => http://www.opengis.net/wfs/2.0
xlink => http://www.w3.org/1999/xlink
xsd => http://www.w3.org/2001/XMLSchema
xsi => http://www.w3.org/2001/XMLSchema-instance
```

Candidate exposition:

□ GZIP: Yes

□ Fast Info Set : Yes

□ BXML: Yes

□ EXI: No

6.2.6.3 BBOX

Only Snowflake server outputs boundary box gml envelope for each feature. This cost some space because BBOXs differ between features. E.g:

```
<gml:boundedBy>
  <gml:Envelope srsName="urn:ogc:def:crs:OGC:1.3:CRS84">
    <gml:lowerCorner>-116.018777777778
38.5067222222222</gml:lowerCorner>
    <gml:upperCorner>-113.571555555556
40.6228888888889</gml:upperCorner>
  </gml:Envelope>
</gml:boundedBy>
```

6.2.6.4 GML:ID

We find some different species of GML:ID depending the WFS server. Some of them around 50 characters will be hard to compress.

Snowflake: species / ranked by occurrence (x stand for hexadecimal character)

```
  41248 urn-x:owsx:snowxlxkx:VIxxxxxxx
  40001 urn-x:owsx:snowxlxkx:tsrivxrs.xxxxx
  40001 urn-x:owsx:snowxlxkx:tp:rivxrs.xxxxx
  40001 urn-x:owsx:snowxlxkx:rivxrs.xxxxx
  40001 urn-x:owsx:snowxlxkx:gxomrivxrs.xxxxx
  31246 RtxSxg_xxxxx
  22955 urn-x:owsx:snowxlxkx:RSxOxxxxxxxx
  20910 urn-x:owsx:snowxlxkx:xmxx_txxiwxy.xix-xxxxxxxx_xxxxxxxxxxx_xxxx
  20910 urn-x:owsx:snowxlxkx:twx:xmxx_txxiwxy.xix-
xxxxxxxx_xxxxxxxxxxx_xxxx
  20910 urn-x:owsx:snowxlxkx:twx:ts:xmxx_txxiwxy.xix-
xxxxxxxx_xxxxxxxxxxx_xxxx
  20910 urn-x:owsx:snowxlxkx:twx:tp:xmxx_txxiwxy.xix-
xxxxxxxx_xxxxxxxxxxx_xxxx
  20910 urn-x:owsx:snowxlxkx:twx:sx:xmxx_txxiwxy.xix-
xxxxxxxx_xxxxxxxxxxx_xxxx
  20910 urn-x:owsx:snowxlxkx:twx:gxom:xmxx_txxiwxy.xix-
xxxxxxxx_xxxxxxxxxxx_xxxx
  20910 urn-x:owsx:snowxlxkx:ts:xmxx_txxiwxy.xix-
xxxxxxxx_xxxxxxxxxxx_xxxx
  20910 urn-x:owsx:snowxlxkx:tp:xmxx_txxiwxy.xix-
xxxxxxxx_xxxxxxxxxxx_xxxx
  20910 urn-x:owsx:snowxlxkx:sx:xmxx_txxiwxy.xix-
xxxxxxxx_xxxxxxxxxxx_xxxx
  19590 GxOM_RSxVIxxxxxxx_VIxxxxxxx
  18149 urn-x:owsx:snowxlxkx:xnxxnroutxsxg:VIxxxxxxx
  18149 urn-x:owsx:snowxlxkx:stxrtxnroutxsxg:VIxxxxxxx
  15623 RtxSxg_xxxxx_RtxSxg_xxxxx
  13425 urn-x:owsx:snowxlxkx:xvxil:nxsr_xrp.xxxxx
  13425 urn-x:owsx:snowxlxkx:xonxom:nxsr_xrp.xxxxx
  13425 urn-x:owsx:snowxlxkx:xirxrxxtxhxr:nxsr_xrp.xxxxx
  13425 urn-x:owsx:snowxlxkx:usxgx:nxsr_xrp.xxxxx
  13425 urn-x:owsx:snowxlxkx:ts:nxsr_xrp.xxxxx
  13425 urn-x:owsx:snowxlxkx:tp:nxsr_xrp.xxxxx
  13425 urn-x:owsx:snowxlxkx:nxsr_xrp.xxxxx
  13425 urn-x:owsx:snowxlxkx:gxom:nxsr_xrp.xxxxx
  12064 urn-x:owsx:snowxlxkx:xity:nxsr_xrp.xxxxx
   9696 urn-x:owsx:snowxlxkx:xmxx_oxstxxlx.xix-xxxxxxxx_xxxxxxxxxxx_-
xxxx
   9696 urn-x:owsx:snowxlxkx:vs:xmxx_oxstxxlx.xix-
xxxxxxxx_xxxxxxxxxxx_-xxxx
   9696 urn-x:owsx:snowxlxkx:ts:xmxx_oxstxxlx.xix-
xxxxxxxx_xxxxxxxxxxx_-xxxx
   9696 urn-x:owsx:snowxlxkx:tp:xmxx_oxstxxlx.xix-
xxxxxxxx_xxxxxxxxxxx_-xxxx
   9696 urn-x:owsx:snowxlxkx:gxom:xmxx_oxstxxlx.xix-
xxxxxxxx_xxxxxxxxxxx_-xxxx
   9052 urn-x:owsx:snowxlxkx:VIxxxxxxxx
   9000 urn-x:owsx:snowxlxkx:tsrivxrs.xxxx
   9000 urn-x:owsx:snowxlxkx:tp:rivxrs.xxxx
   9000 urn-x:owsx:snowxlxkx:rivxrs.xxxx
   9000 urn-x:owsx:snowxlxkx:gxomrivxrs.xxxx
   9000 RtxSxg_xxxx
   8018 urn-x:owsx:snowxlxkx:xvxil:nxsr_xrp.xxxx
   8018 urn-x:owsx:snowxlxkx:xonxom:nxsr_xrp.xxxx
   8018 urn-x:owsx:snowxlxkx:xirxrxxtxhxr:nxsr_xrp.xxxx
   8018 urn-x:owsx:snowxlxkx:usxgx:nxsr_xrp.xxxx
   8018 urn-x:owsx:snowxlxkx:ts:nxsr_xrp.xxxx
   8018 urn-x:owsx:snowxlxkx:tp:nxsr_xrp.xxxx
   8018 urn-x:owsx:snowxlxkx:nxsr_xrp.xxxx
   8018 urn-x:owsx:snowxlxkx:gxom:nxsr_xrp.xxxx
   7051 urn-x:owsx:snowxlxkx:xity:nxsr_xrp.xxxx
   5741 urn-x:owsx:snowxlxkx:xnxxnroutxsxg:Vxxx_xx_x
```

```
5741 urn-x:owsx:snowxlxkx:Vxxx_xx_x_TP
5741 urn-x:owsx:snowxlxkx:Vxxx_xx_x
5741 urn-x:owsx:snowxlxkx:Vxxx_xx
5741 urn-x:owsx:snowxlxkx:stxrtxnroutxsxg:Vxxx_xx_x
4806 urn-x:owsx:snowxlxkx:xnxxnroutxsxg:VIxxxxxxxx
4806 urn-x:owsx:snowxlxkx:stxrtxnroutxsxg:VIxxxxxxxx
4500 RtxSxg_xxxx_RtxSxg_xxxx
4364 urn-x:owsx:snowxlxkx:ROxxxxxxxxROU
```

…. it goes below 4000 after this point

Comsoft: gml:id are all on the form Idddddd , with d a decimal character

Luciad: species / ranked by occurrence (x stand for hexadecimal character)

```
8568 urn.uuix.xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_x
6450 urn.uuix.xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_x_x
4775 urn.uuix.xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_xx
 952 urn.uuix.xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
 597 urn.uuix.xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx_x_xx
   5 Ix_xIRSPxxx_TIMxSHxxT_xxxx
   2 xonx_xxxlusionx
   2 Ix_xIRSPxxx_LxYxR_LxVxLS_xxxx
    and a lot more of individual namings after this point
```

6.2.6.5 Time

Snowflake: 2011-01-01T00:00:00.000Z format

Comsoft: 2011-01-01T00:00:00.000Z format

Luciad: 2011-01-01T00:00:00.000+01:00 format

Candidate exposition:

The dateTime type (cf. http://www.w3.org/TR/xmlschema-2/#dateTime) is only exploited by EXI (cf. http://www.w3.org/TR/exi/#encodingDateTime). For the others, only deflate will be able to compress the textual string representing date and time.

6.2.6.6 Coordinates

We find both use of poslist and pos, of course poslist is better

For compression issues: is double precision mandatory on wfs's response?

Compression of floating point is hard whatever algorithm is used. As coordinates are expressed in degrees, even for longitude, the 23 bits of mantissa permit to cover a ground resolution of 1.7m on the equator. At +/- 180 deg (180 takes 8 bits (7 as a fractional part), 16 bits remain to cover in worth case one degree at the equator or so 111km, so 111/2^16 is the resolution).

This resolution goes bellow 1m for most of the airports (only the ones in the pacific (between +128 and -128 degrees) will suffer from 1.7m limitation)

Snowflake:

Usage of both pos and poslist, in double precision, also use lowerCorner and upperCorner, eg:

```
<gml:posList srsDimension="2" count="14">-90.7784118652344
29.0497417449951 -90.7805099487305
<gml:pos>-102.328245555556 46.3591641666667</gml:pos>
<gml:lowerCorner>16.47888889 52.13061389</gml:lowerCorner>
<gml:upperCorner>16.72713333 52.52638889</gml:upperCorner>
```

Comsoft:

Only use of pos in simple precision, eg.

```
      <name>ESTONIA_LATVIA</name>
      <type>STATE</type>
      <border>
       <Curve gml:id="ID013845"
srsName="urn:ogc:def:crs:OGC:1.3:CRS84">
        <gml:segments>
         <gml:LineStringSegment>
          <gml:pos>27.3222222 57.5483333</gml:pos>
          <gml:pos>27.3019444 57.5513889</gml:pos>
          <gml:pos>27.2858333 57.5505556</gml:pos>
                    ... and so on
```

Luciad:

Use of both in simple precision, also startAngle and endAngle:

```
<gml:pos>-118.238 36.525</gml:pos>
<gml:pos>-118.239 36.529</gml:pos>
<gml:posList>-146.2666667 64.9997222 -146.0833333 64.9997222 …
<gml:startAngle uom="deg">228.6</gml:startAngle>
<gml:endAngle uom="deg">34.94149</gml:endAngle>
```

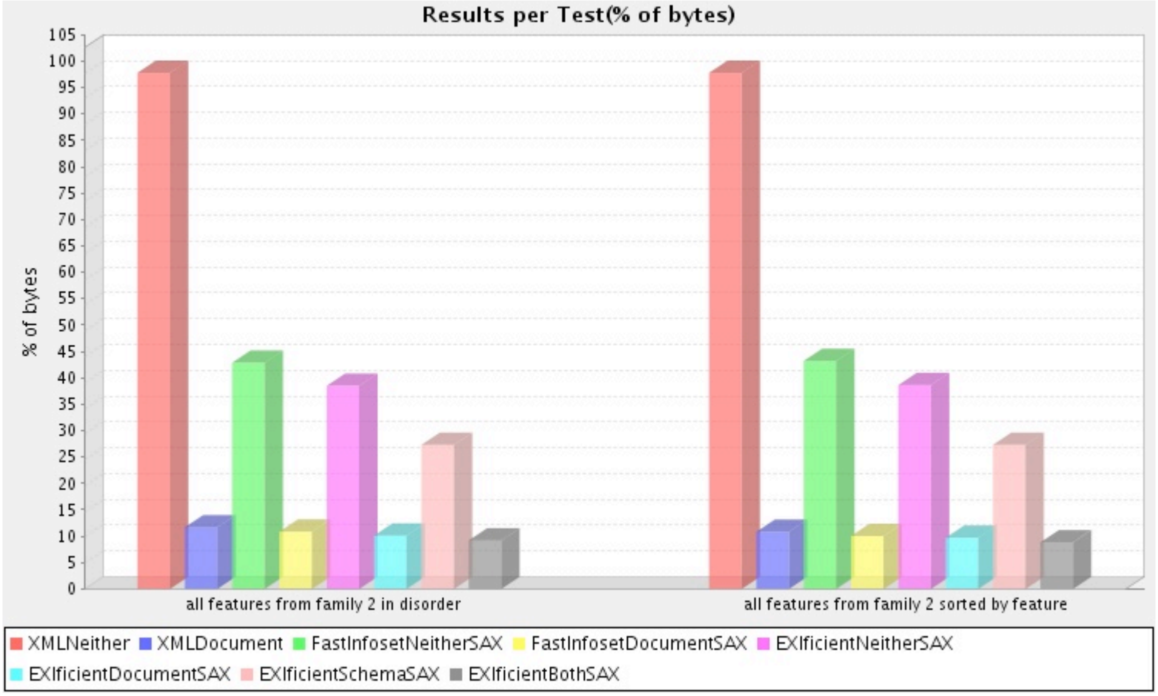
Candidate exposition:

All candidates are impacted by float or double representation. BXML is impacted by both (because they store data in IEEE-754). GZIP and FIS are less impacted because if some digits are not used, they won't be output in textual representation, and they basically do not know that what they are reading is actually a floating point number. EXI uses variable length integer encoding for both exponent and mantissa, and treats double and float as double. This topic is more discussed in 7.1.1.4.1 where is described a test centered on coordinates alone.

6.2.6.7 Order of features

If we request many different features on the same WFS request, are they output in ordrer (let's say all routes, then all route_segments,...) or are they interlaced by features (a route, then all segments for this route, then next route and so on) ?

As we only have airspaces from Luciad, a flat file containing Estonia from Comsoft, we cannot make comparisons on this aspect. But certainly it would be predominant for compression performance. The technical family contains one test related to order, based on features from family 2, and result show a 5-10% improvement when we sort features by type (only for deflate post-compression algorithms).



As histograms may look the same, we notice the following gains using ordered features :

□ 8% for Fast Info Set with deflate

□ 7% for deflate alone

□ 4% for EXI with deflate with or without schema knowledge

Candidate exposition:

□ GZIP: Yes

- Fast Info Set : Yes with deflate
- BXML: Yes with deflate
- EXI: Yes, because of post-compression which will work better if successive tokens are the same, instead of been interlaced.

7 Results

All Japex reports can be found with this document, in HTML format. On the next paragraph, we just present histograms, and reference figures when needed.

7.1 Brute compaction performance

These measure concern only the gain of space obtained using the compression provided by each candidate with various options. The size of file after compression is expressed in percentage of the original file size.

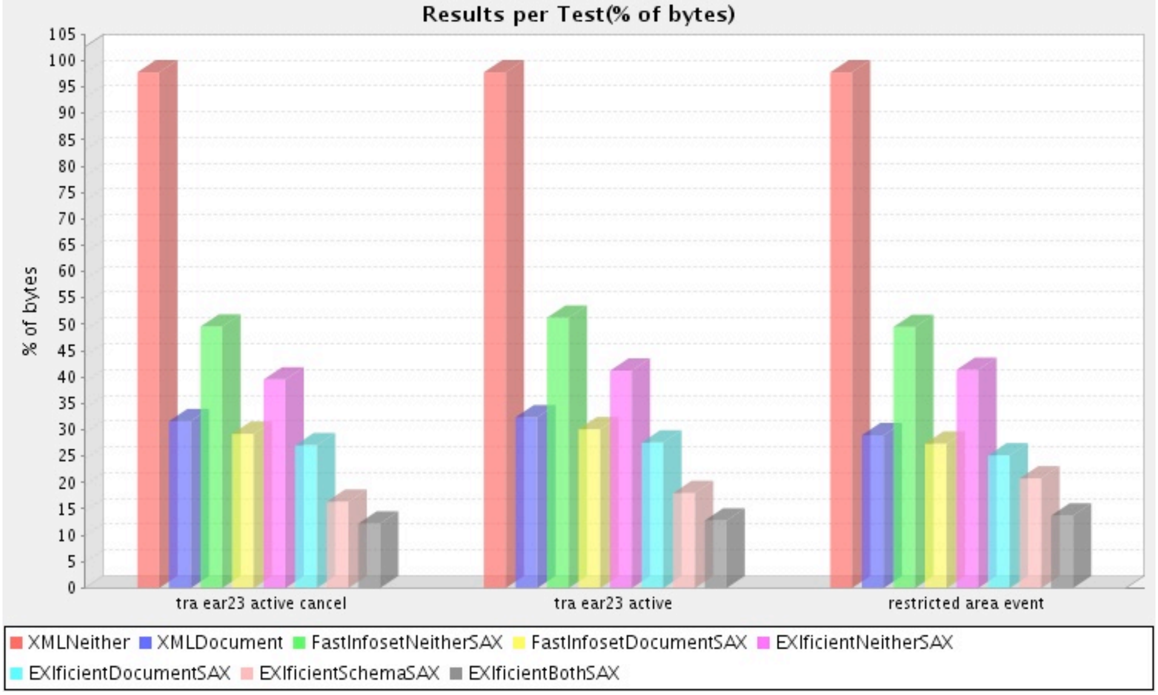
Note that for Java SAX candidates (all of Java candidates), verification is made about the validity of compression by decompressing and comparing it with original content using SAX parser.

7.1.1 Figures by family

AIXM sample files shown on graphs are sorted by size, growing.

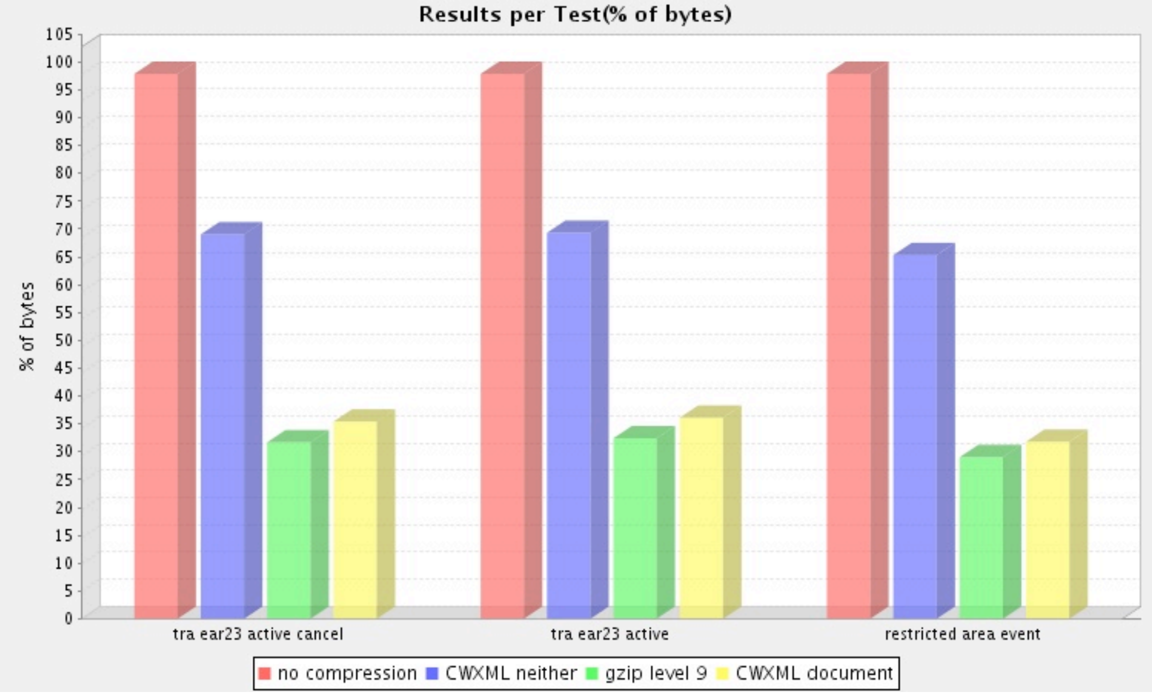
7.1.1.1 First family (D-NOTAMS)

Java candidates:



EXI with both schema knowledge and deflate post-compression brings an average reduction to 13% of the original size of the file. This level of compression allows D-NOTAMs to lower under the 1KB level.

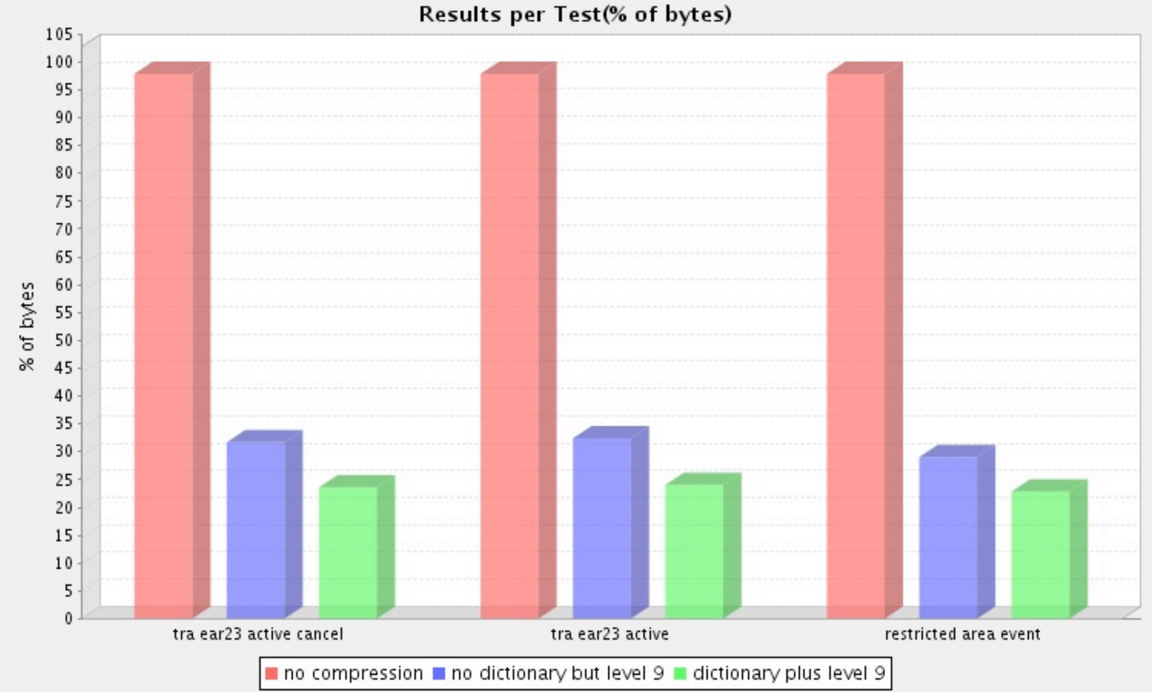
Native candidate CWXML:



With deflate post-compression, CWXML performs worst than bare deflate with level 9. Maybe the CWXML output doesn't suit deflate and the header must weight too much.

Without deflate, CWXML performs worst than FI and EXI, because it works on byte boundaries where FIS use PER and EXI a bit output code.

Deflate with a dictionary:

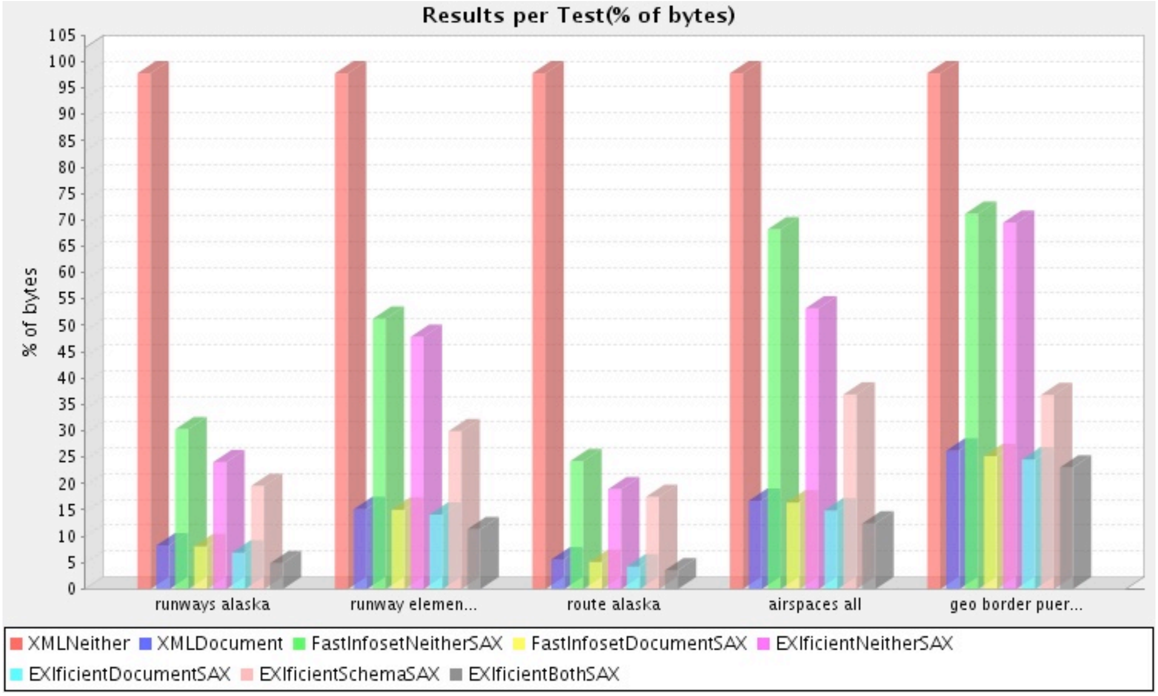


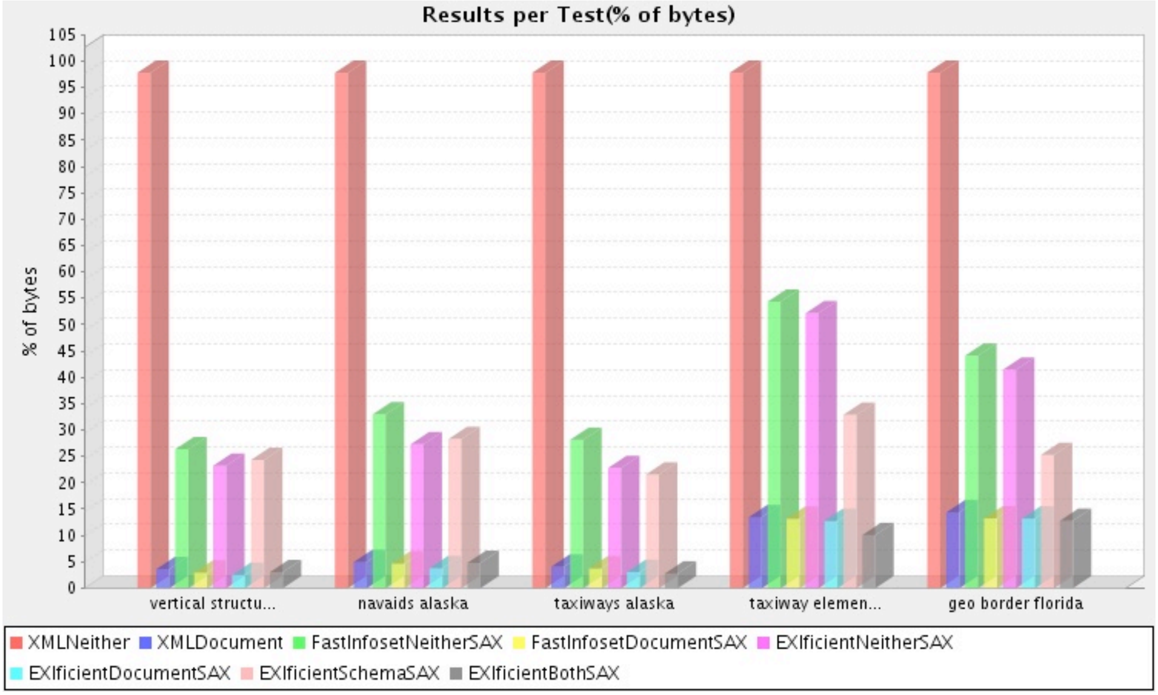
Adding a 28KB dictionary to deflate provides an additional compression of 30% for D-NOTAMS. This performance places deflate+dictionary ahead EXI without schema knowledge but with compression and FI + deflate too.

As giving a dictionary based on AIXM schemas to deflate is a bit like providing schema knowledge, this result is not surprising.

7.1.1.2 Second family

Java candidates:





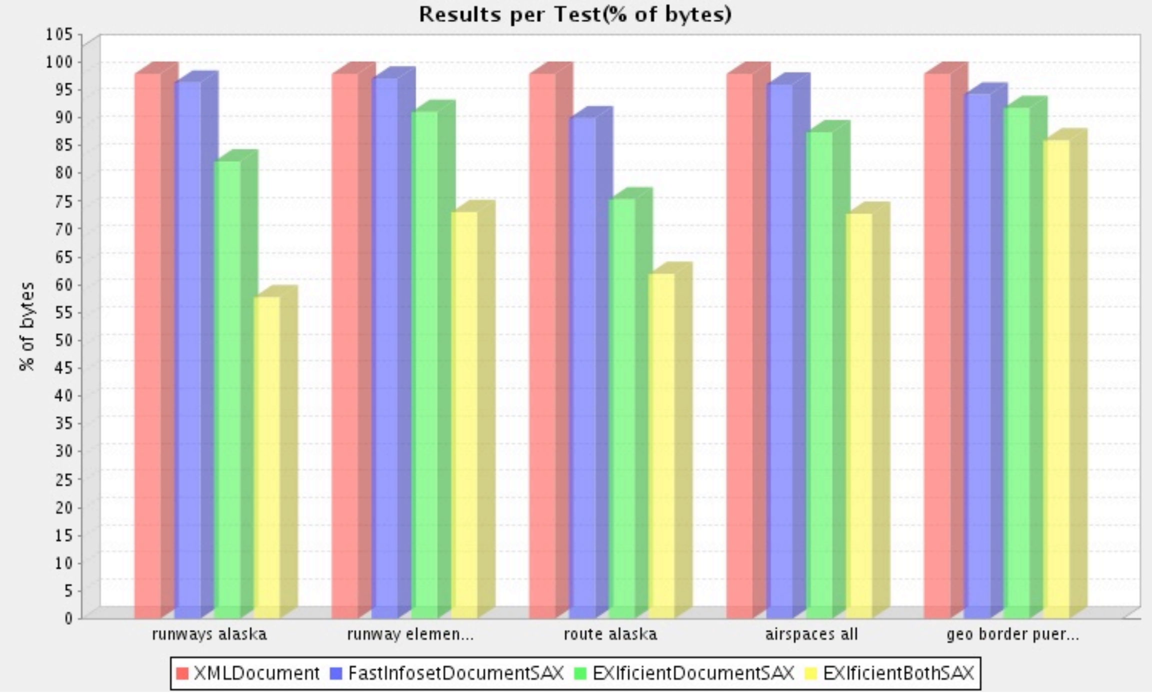
The first good news, is that EXI performs always better than FI even without schema. That means the default bit-encoding of EXI performs better than FI's BER.

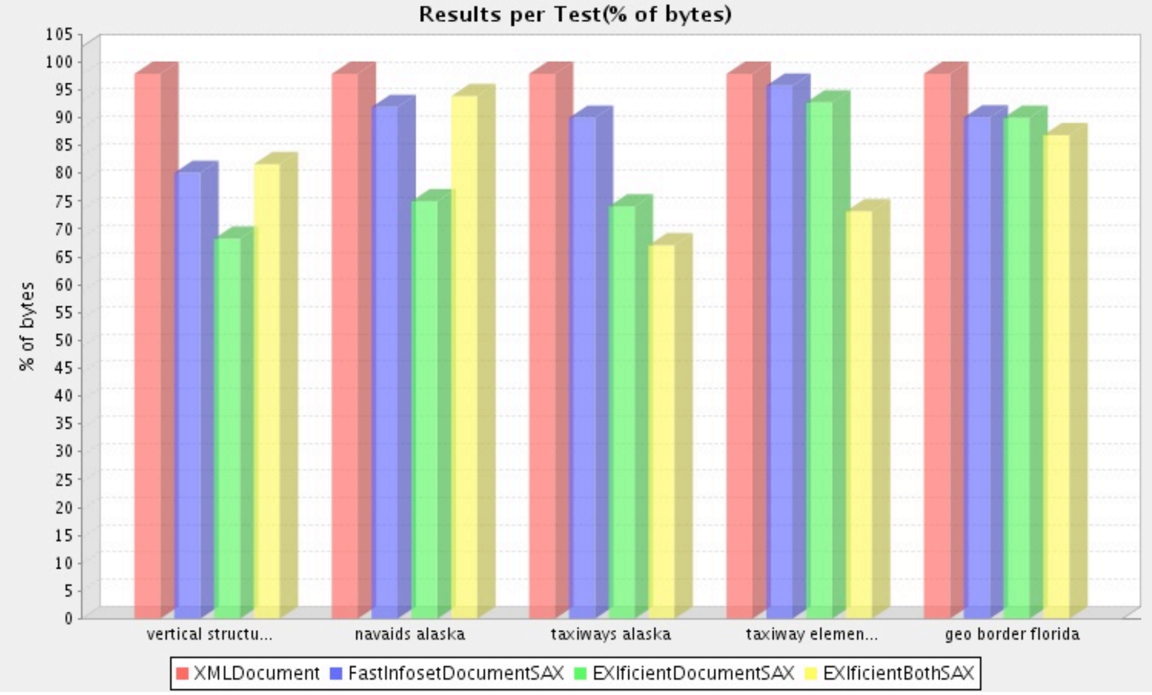
Without post-compression, we see that runways, routes, vertical structures, navaids and taxiways offer a good compression ratio below 30% for FI and EXI. Looking back at analysis, this must be due to the small presence of coordinates inside those files (cf. 6.2.3.11)

When we look at the differences between EXI with schema and without, we see a major improvement of performance due to schema only for runways elements, taxiways elements, geoborders, and airspaces. Analysis shows these 4 files share a huge consumption of coordinates. So when EXI knows it deals with float / double numbers (same for it), it compresses better. This is good, but as we are going to discover it on the next page, this advantage annihilates itself when deflate intervene.

Java candidates with post-compression only:

As differences between non post-compress data and post-compress one is important, the following graphs shows only "post-compress" candidates. Results are indexed on gzip -9 (100%):





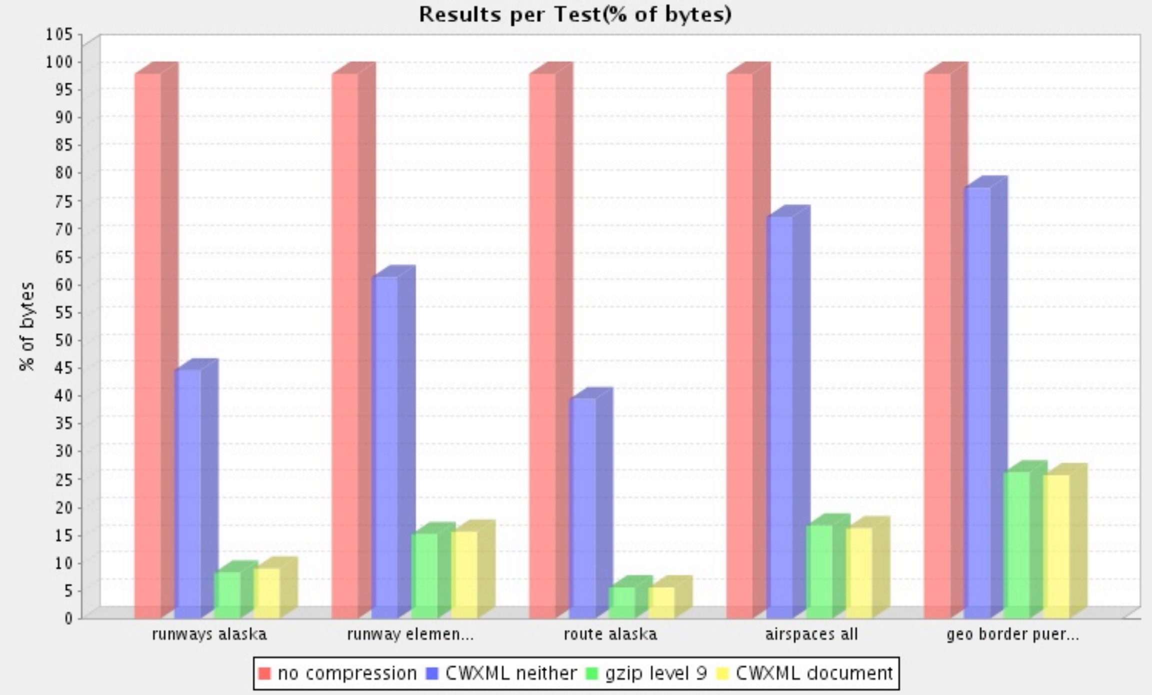
The bad news is that FI or EXI only brings few extra-compression compared to gzip -9. EXI with schema knowledge and deflate remove only 15 to 40% of the file size obtained by pure deflate.

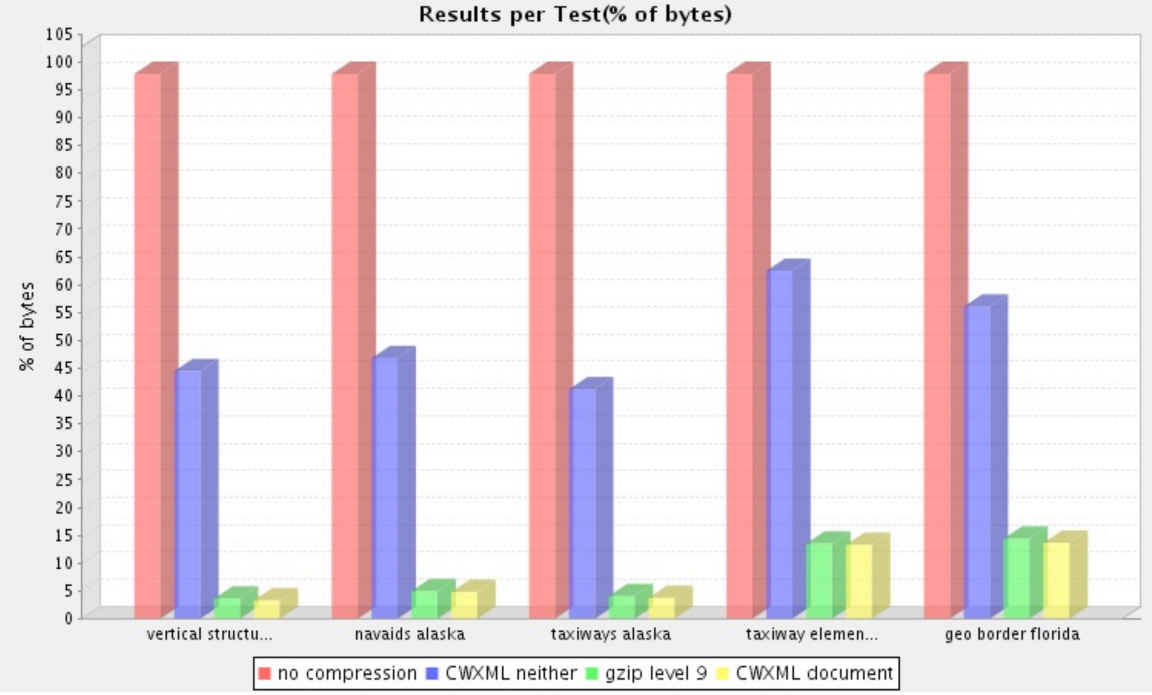
More surprisingly, in both cases (vertical structure and navaids), the knowledge of the schema is a handicap for EXI which performs better without schema than with it. The analysis already tells us those 2 files presented long attribute names. This attribute predominance deserves EXI and its grammar because there are too few elements to get advantage of the grammar rules. This trend was already noticeable without post-compression, but is more visible with post-compression, FI performing better than EXI with schema.

Regarding coordinates handling, the differences around 40% noticed between both EXI candidates without deflate for geo borders, airspace, taxiways and runways elements is shrunk by deflate, EXI encoded doubles aligned on byte boundaries being more difficult to compress as their ascii counterparts.

Still this is good news, EXI without schema performs always better than FI, with or without compression.

Native candidates:



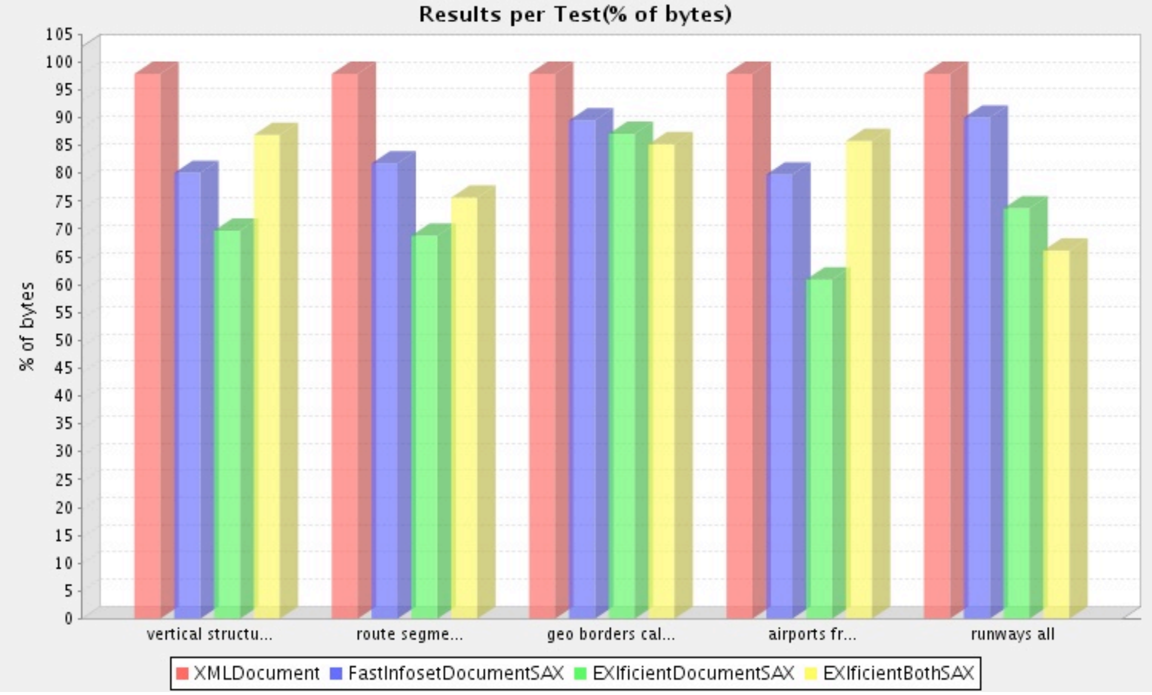


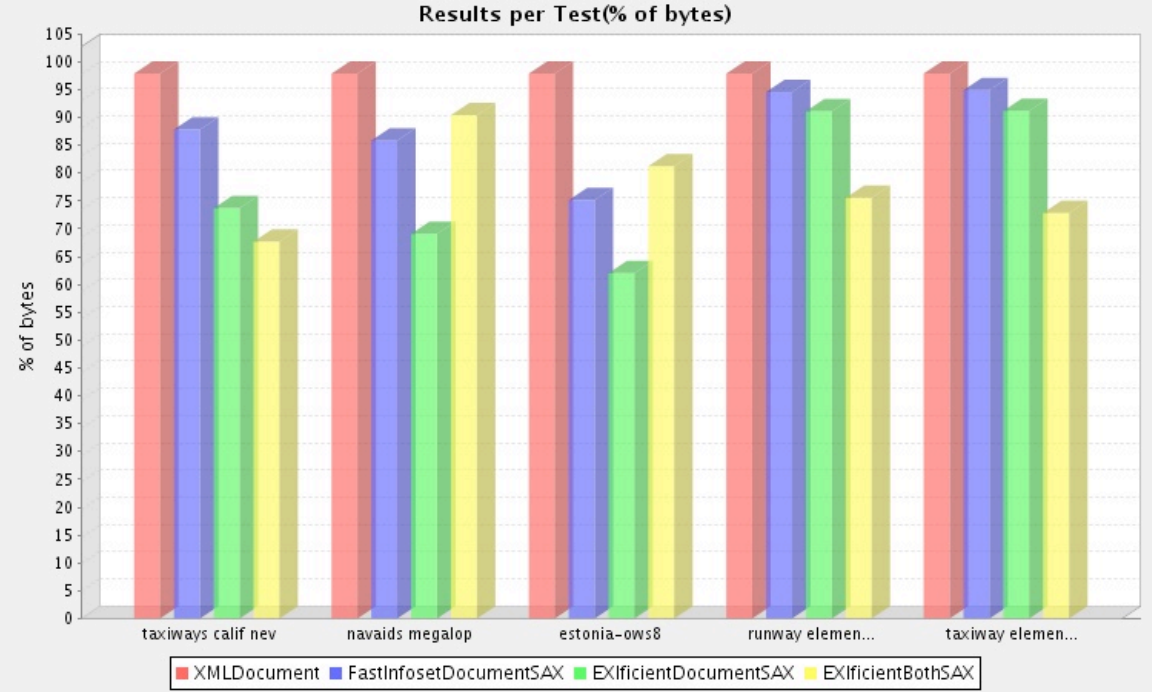
The header thread seemed correct for CWXML which succeeded to perform better (a little bit) than GZIP on this family. CWXML can take advantage of its dictionaries and present to gzip data more suitable for compression. The difference between gzip and CWXML being so small, we won't use it for family 3. Same for dictionary based deflate whose advantages disappear near totally for this family and the next one, as back-references from deflate cannot go further than 32KB before, so for any file over 32KB, the dictionary loose its edge, because impossible to reference.

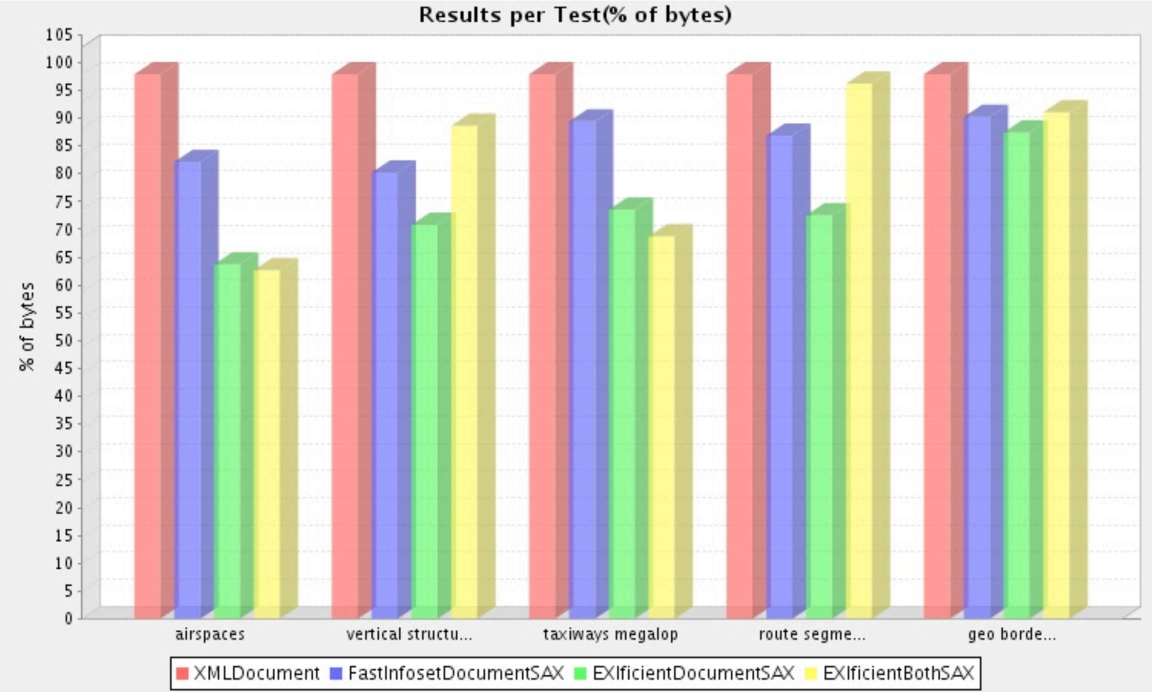
7.1.1.3 Third family

Java candidates with post-compression:

As java candidates do not offer any difference on compression compared to family 2 without compression, we directly show the post-compression results:

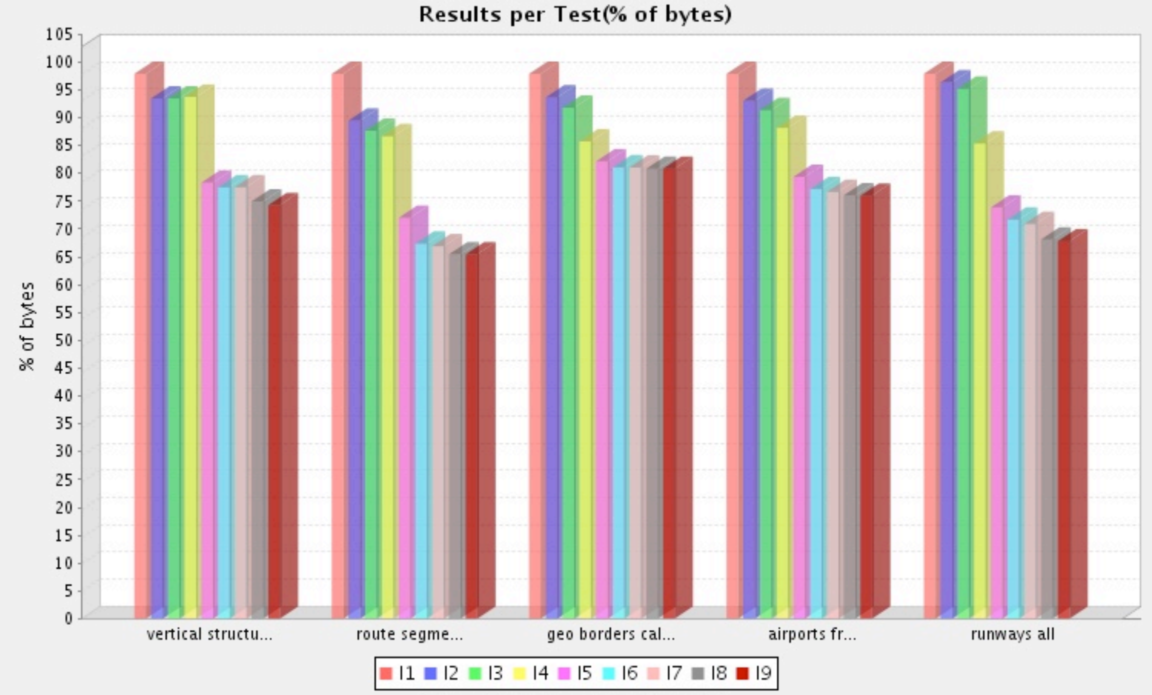


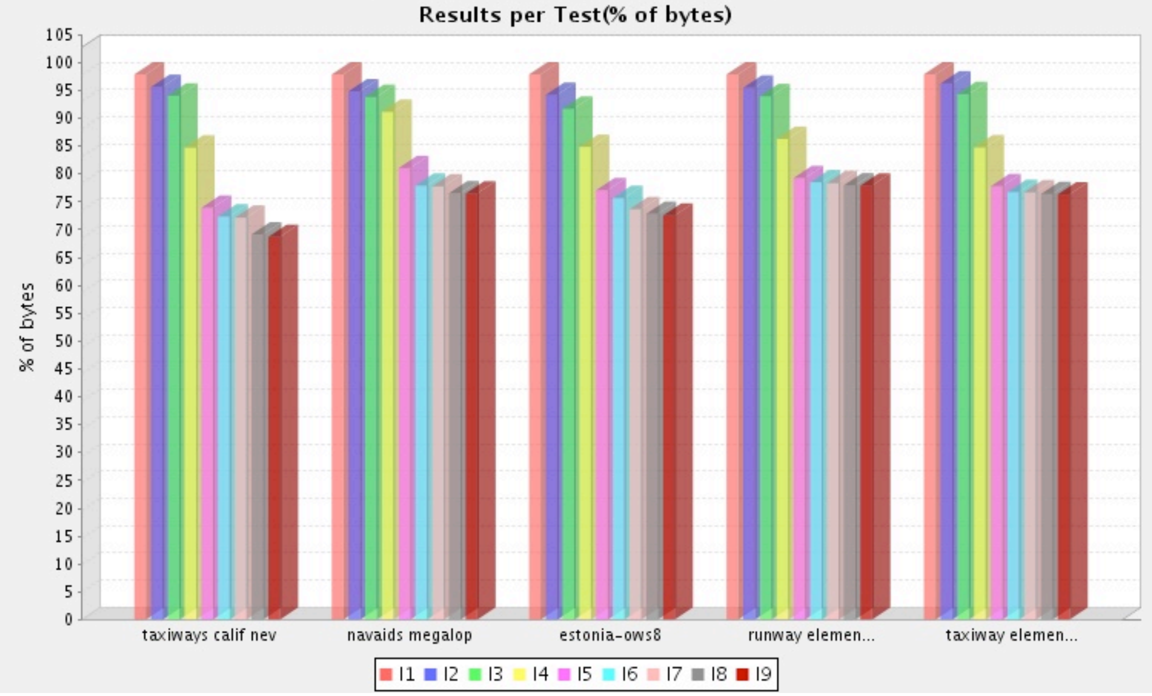


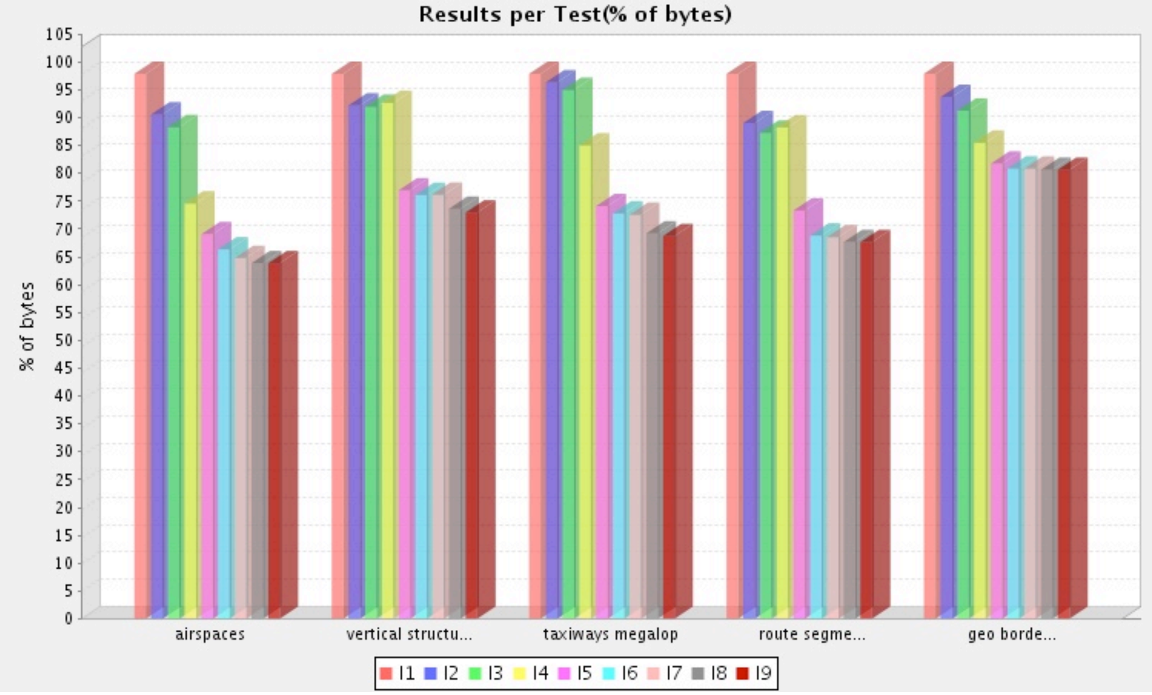


There is no big surprise in this chart, compared to family 2. Runways, Taxiways and Luciad Airspaces offer a good level of compaction for EXI compared to Gzip. This confirms the efficiency of EXI for XML file using a limited set of elements and few C-DATA (in AIXM case coordinates and dates).

Deflate level incidence:


Results per Test(% of bytes)
% of bytes
105
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
vertical structu...
route segme...
geo borders cal...
airports fr...
runways all
I1
I2
I3
I4
I5
I6
I7
I8
I9


Results per Test(% of bytes)
% of bytes
105
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
taxiways calif nev
navaids megalop
estonia-ows8
runway elemen...
taxiway elemen...
I1
I2
I3
I4
I5
I6
I7
I8
I9

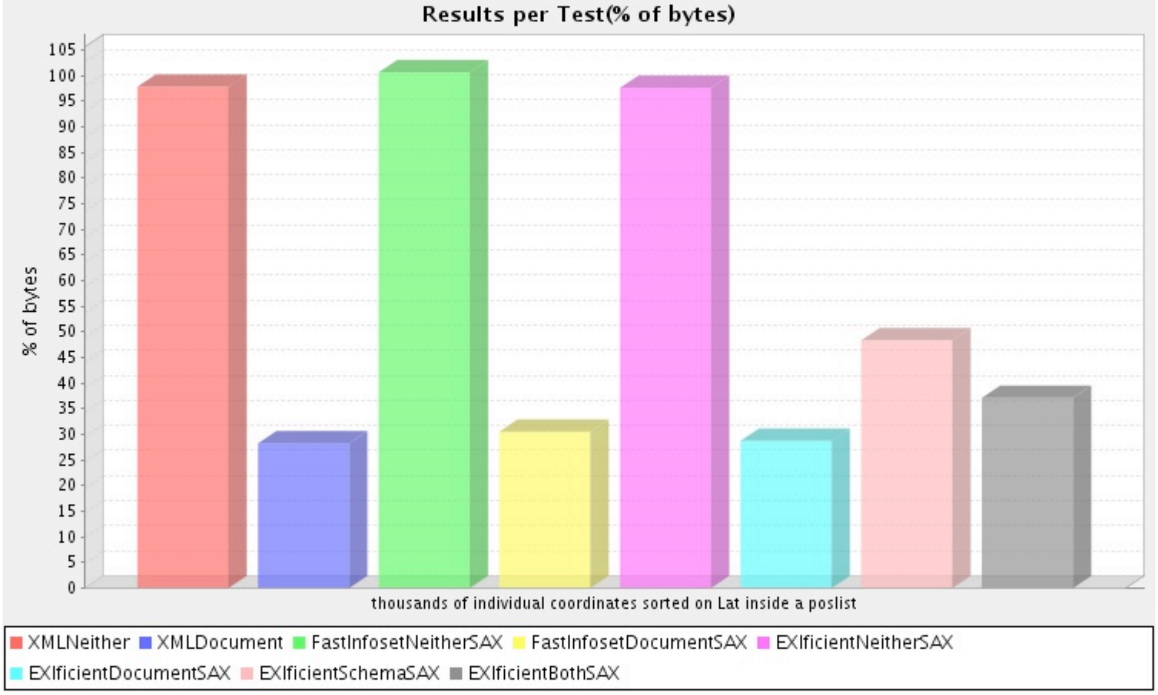


As you can see, level 5 or 6 is sufficient in most cases; additional compression brought by higher levels costs a lot of CPU for only few percents compaction gain.

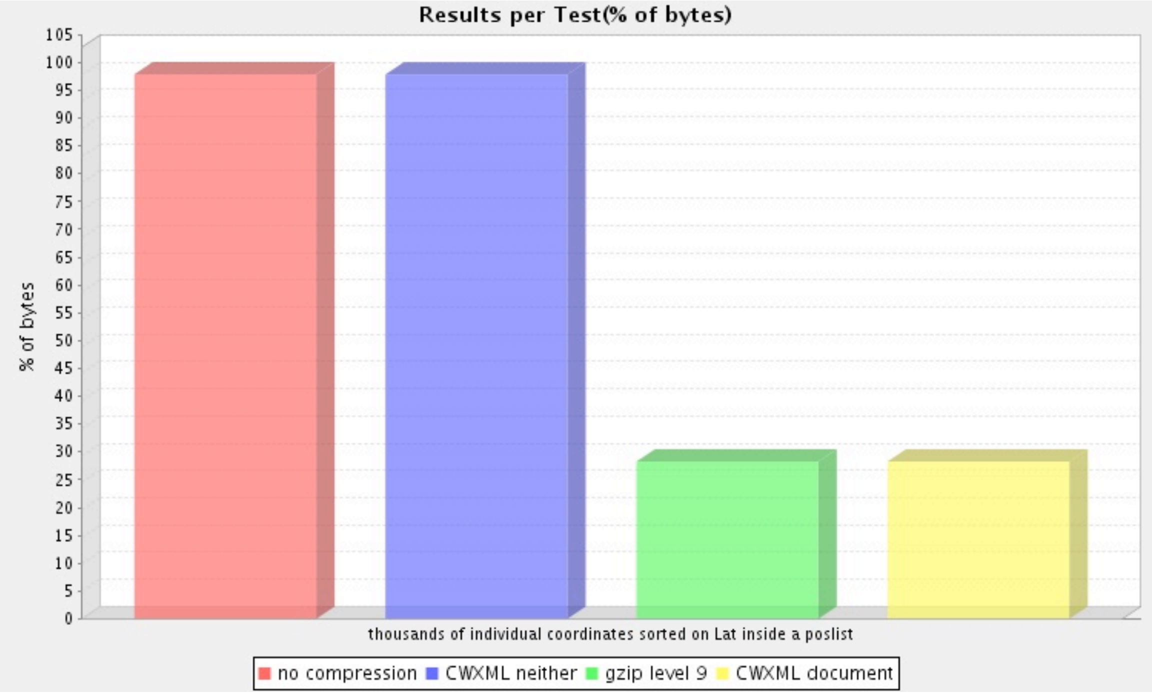
7.1.1.4 Thourth family

7.1.1.4.1 Lot of doubles

Java candidates:



Native candidates:



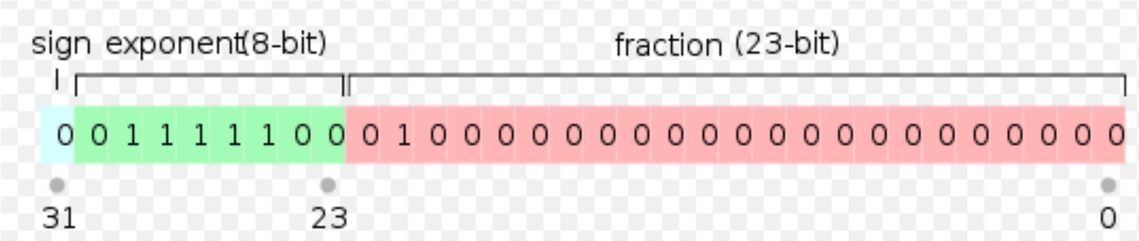
Interpretation:

All candidates "without deflate" perform nearly the same when dealing with floating point numbers except EXI with schema knowledge. When we look closely at data and count characters, we see that a typical coordinate is:

-105.013035 40.2083175

It uses double, the whole pattern spaning on 23 characters.

- IEEE 754 (CWXML) uses 8 bytes for each double, it will take just 16 bytes to store the raw data. If you add one additional byte for separation you reach 17 bytes compared to 23. The compression is only around 25%. IEEE 754 binary format is made in such a way it doesn't please deflate which needs to find at least 3 consecutive identical bytes to begin to consider placing a back-reference. For float it's hard because the format is not byte aligned.



For doubles we will have 11 bits of exponent (not byte boundaries either) and mantissa 52 bits (possible to get 6 consecutives bytes this time).

- EXI use its own way to store floating points numbers (exi:double or just Float) which covers both xsd:float and xsd:double. The “Float” datatype representation is two consecutive integers (signed):
    - The first Integer represents the mantissa of the floating point number
    - and the second Integer represents the base-10 exponent of the floating point number

This representation relies on integer compression, which is the following:

- A Boolean for the sign (a single bit when deflate is not used, a full byte otherwise)
- An unsigned integer (for the absolute value of the integer), which is encoded as a sequence of bytes terminated by a byte with its most significant bit set to 0. The value of the unsigned integer is stored in the least significant 7 bits of the bytes as a sequence of 7-bit bytes, with the least significant byte first.

As you see this storage using 7 real bits in 8 bits envelope can be bad for performance when using full fractions, as 23 bits of mantissa will make 4 bytes in EXI. So basically when using deflate on a double, the input material provided by EXI could cost 8 bytes of mantissa, 2 bytes of exponent and 2 bytes of sign (total 12 bytes). That means that EXI doubles could take much place as ascii data equivalent.

7.1.2 Interpretation of results

All results points to one trend. Coordinates are difficult to swallow by all algorithms. Even worst, GZIP is doing a bit better than competition dealing with numbers in ascii…

7.2 CPU consumption

The time of processor used by each candidate is measured just between the start end the end of an encoding or a decoding phase. To get maximum usage of cache, we run a batch of runs before taking the measure. Grammar computations for EXI, file loading into memory, dictionary loading for gzip, and so on are made before the measurement. This works for memory measurements too.

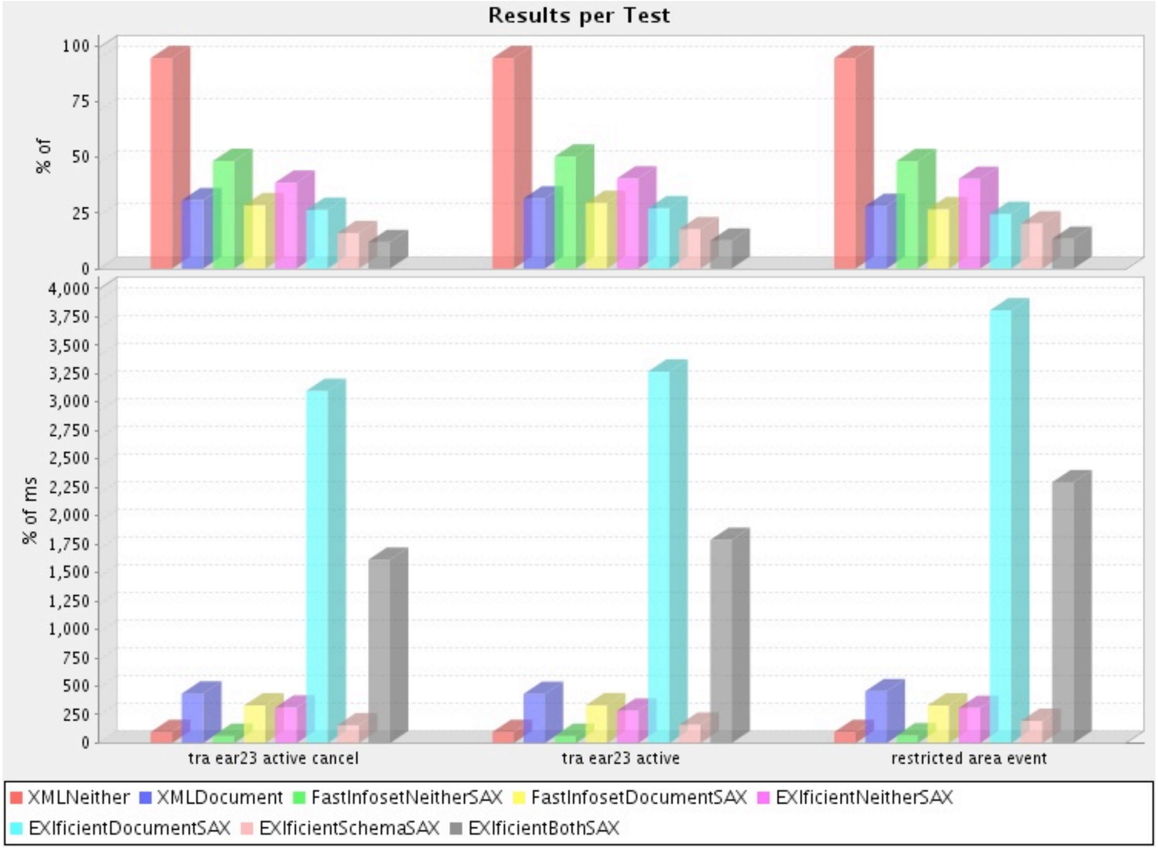
Histograms show on 2 graphs, the compaction and just below the ratio of time needed by a specific compression/decompression algorithm compared to the first candidate (generally raw SAX parsing for java candidates, or gzip for compressed java candidates).

7.2.1 Figures by family

As CWXML is outranged by all of java candidates, we stop giving figures for CPU and memory for CWXML. Same idea for gzip + dictionary for families 2 and 3, because the usage of a dictionary for files much bigger than 32KB doesn't add any value.

7.2.1.1 First Family (D-NOTAMs)

7.2.1.1.1 Encoding



Points to notice:

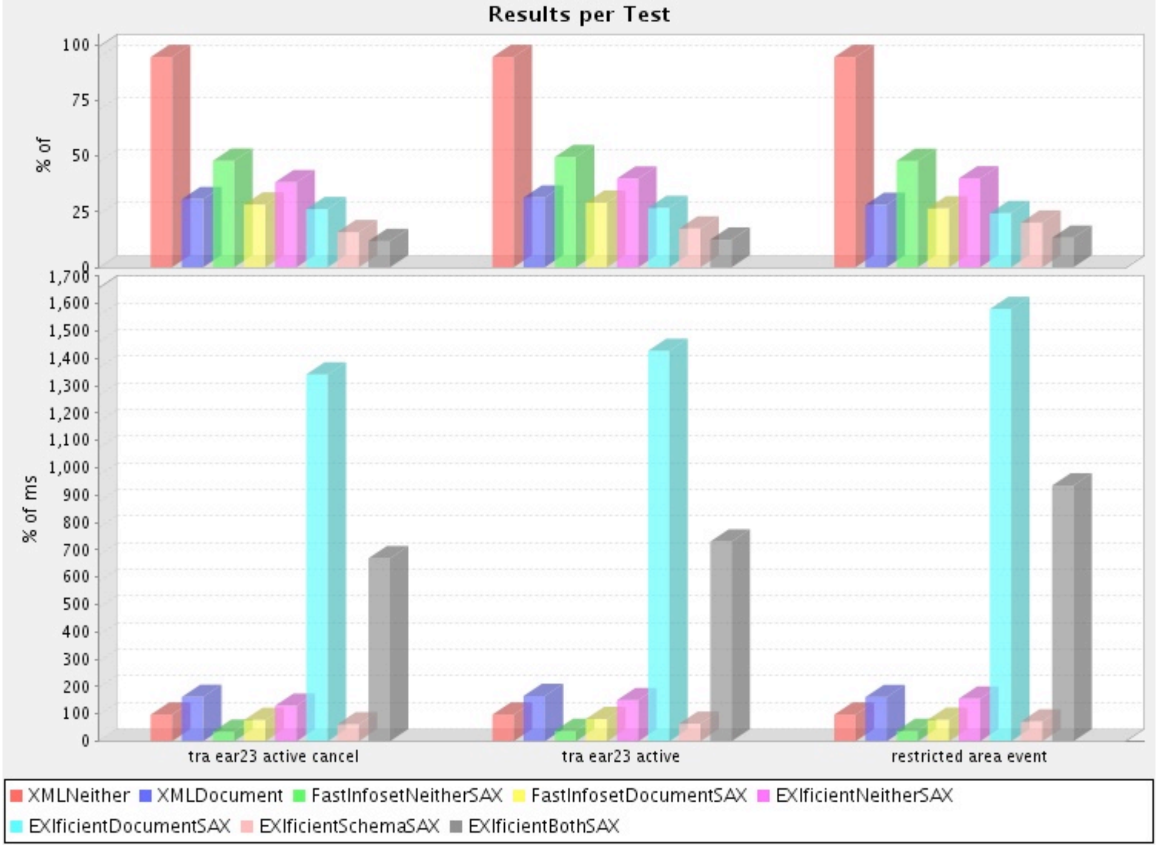
□ FIS without deflate is faster than raw SAX parsing

□ Time of deflate (zlib based) is proportional to the inner compression of the first stage (FIS < SAX)

- The deflate algo used by EXIficient is very long and also proportional to the input data (20x slower than raw SAW parsing)

- EXI with schema but without deflate performs well (average of 131us for SAX, 88 for FIS, and 234 for EXI), but cannot beat FIS in the ratio compaction/time

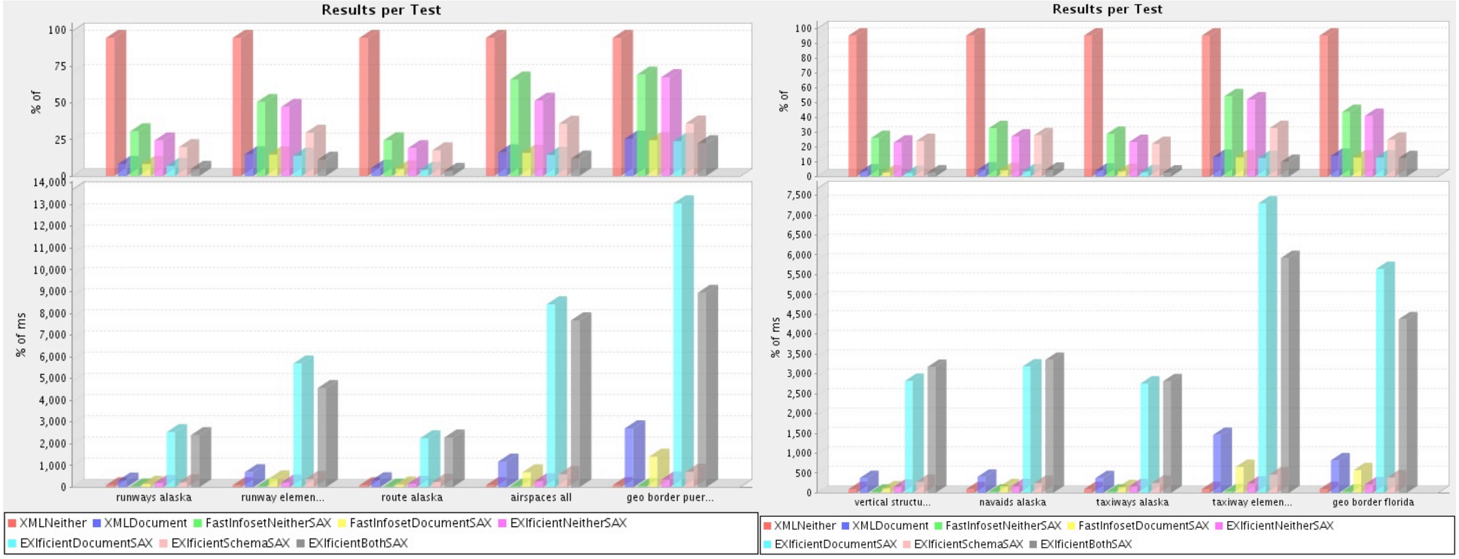
7.2.1.1.2 Decoding



Once again FIS is the big winner, deflate impact is lesser than for compression (ratio 5 to 2), and EXI's deflate is still very bad. For the FIS / EXI+schema match, the difference is decreasing (78us for FIS and 142 for EXI), both offering an equivalent compaction/time performance.

7.2.1.2 Second family

7.2.1.2.1 Encoding



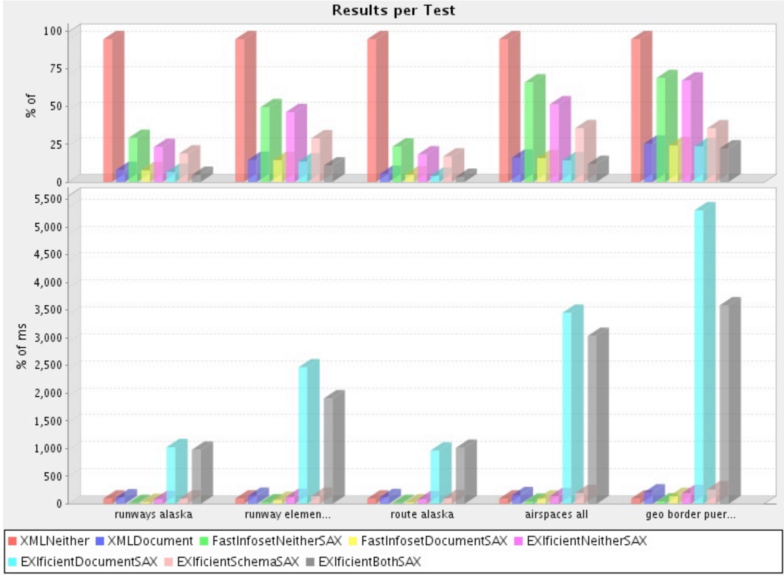
□ EXI deflate is digging its grave, with ratio reaching 100x the time of SAX

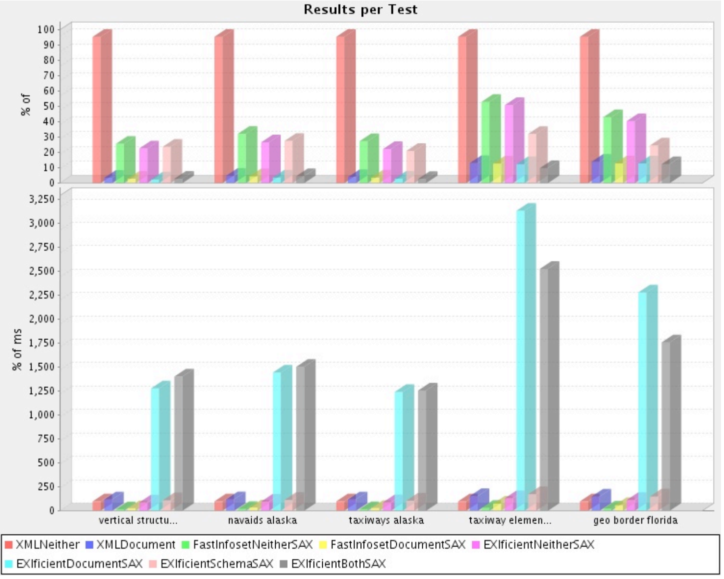
□ Zlib deflate level 9 cost between 3 an 15x the time of SAX (average 8x)

□ FIS without deflate is twice faster than SAX and fourth time slower when using deflate level 9.

□ EXI with schema is 4x slower than SAX

7.2.1.2.2 Decoding





Near the same ratio as for family 1 when comparing compression to decompression:

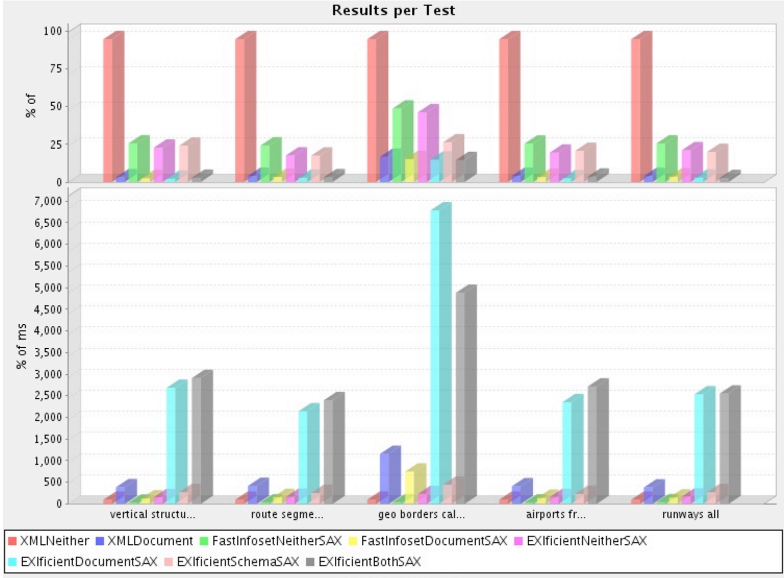
□ FIS is 4 time faster than SAX without deflate and only twice when using deflate

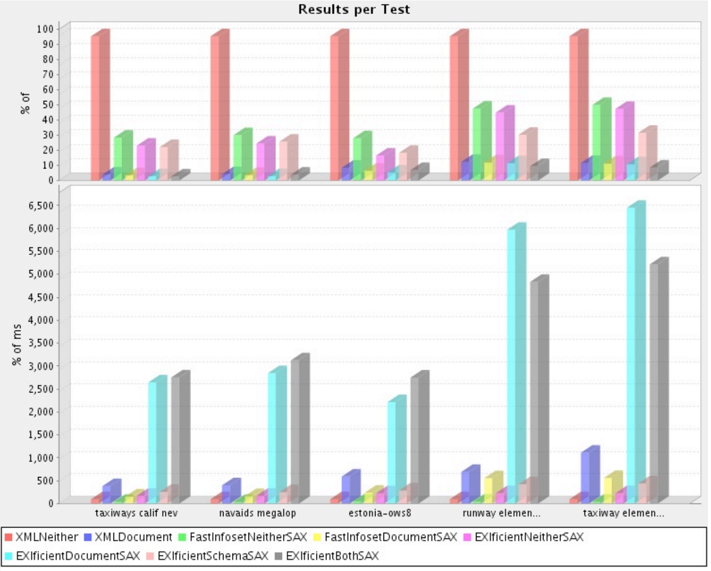
□ Deflate cost is only 40%

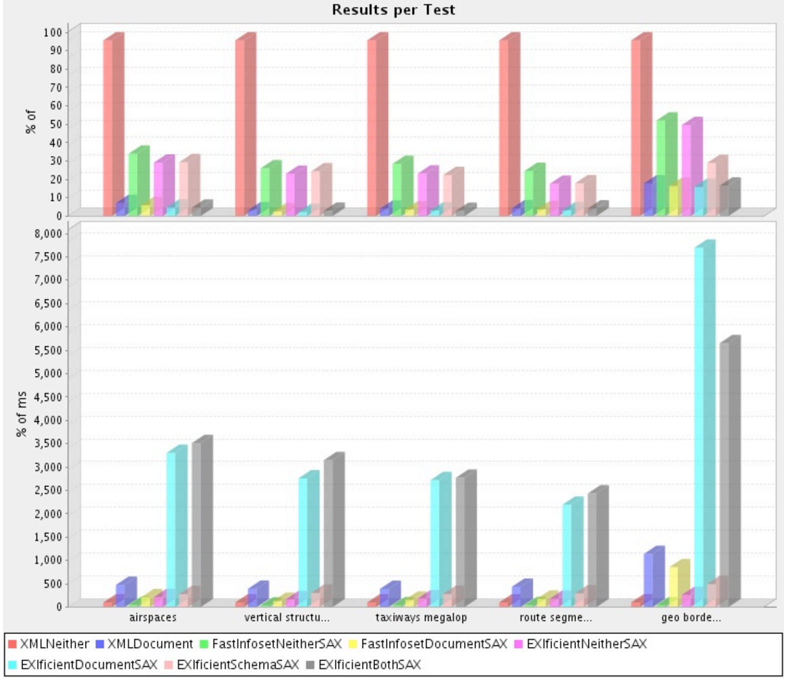
□ Same metric 40% for EXI without deflate

7.2.1.3 Third family

7.2.1.3.1 Encoding







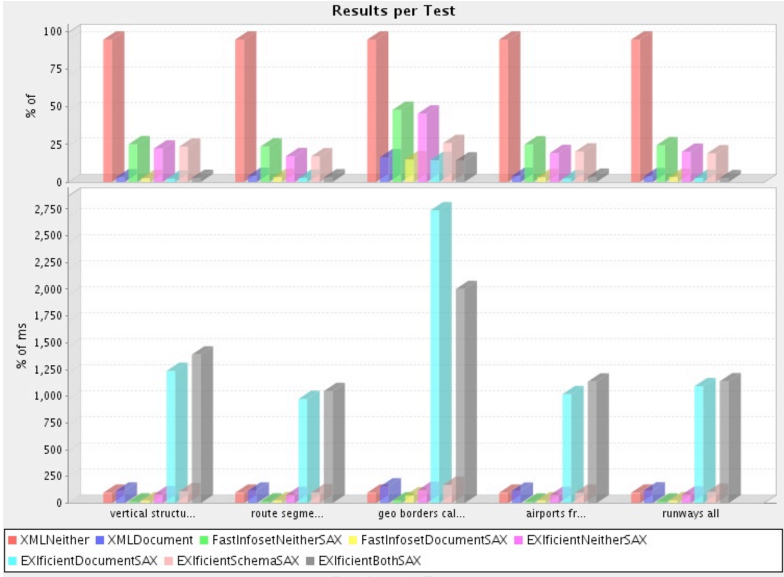
Figures are in ad equation with family 2:

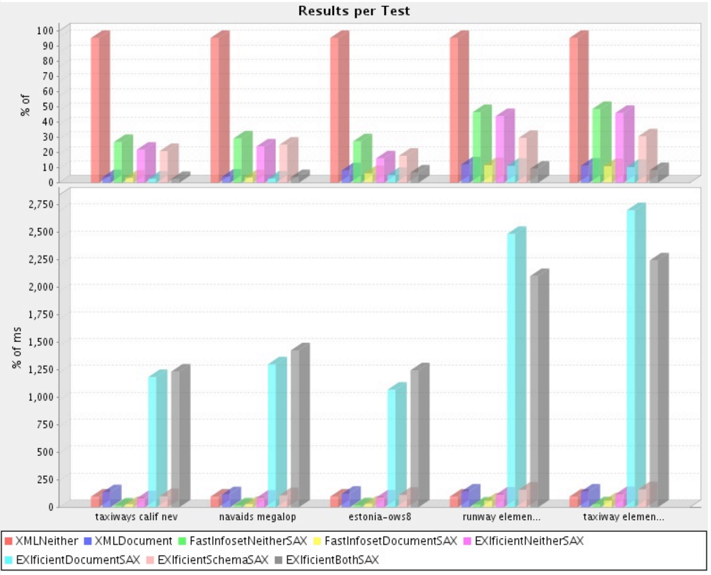
- FIS is 2.4 times faster than raw SAX
- FIS with deflate and EXI with schema but without deflate performs the same, 3.4 times slower than raw SAX.

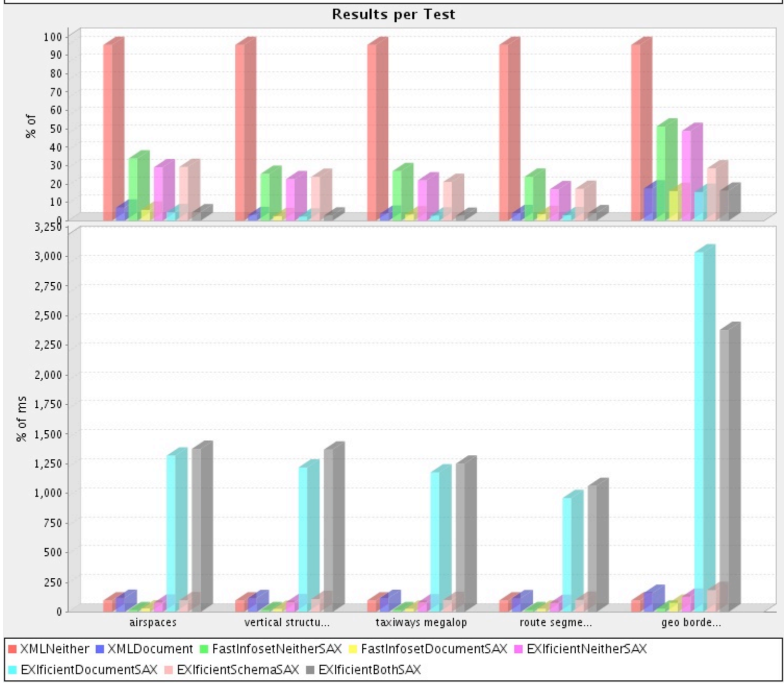
- Deflate level 9 cost is 6x compare to raw SAX

- FIS with deflate is the big winner of the ratio compaction/time:
  - An overall compression of 12x on all data from family 3
  - A throughput of 20MBps for input data (using SAX)
  - A throughput of 1.6MBps for encoded data

7.2.1.3.2 Decoding







Here also, figures are very compatibles with the ones from family 2:

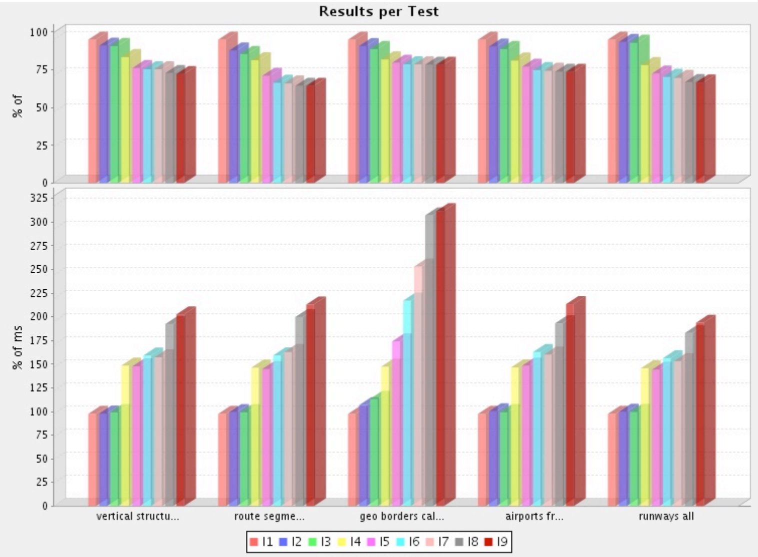
- FIS wins the prize: 5x faster than SAX, only 2.4x with deflate
- EXI with schema performs the same as GZIP and 30% slower than SAX

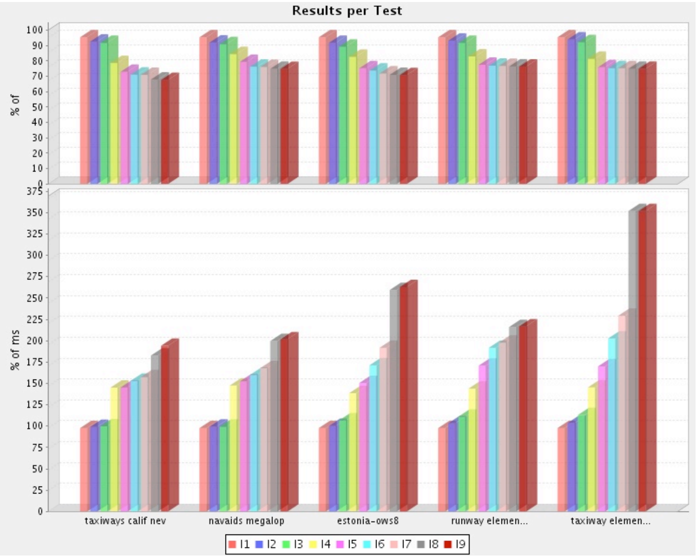
- Here also FIS performs very fast: read compressed data at 8MBps and output SAX tokens for a parser at 104MBps (a gigabit link). Anyway theses figures remain theoretical because the encoder cannot perform at this speed.

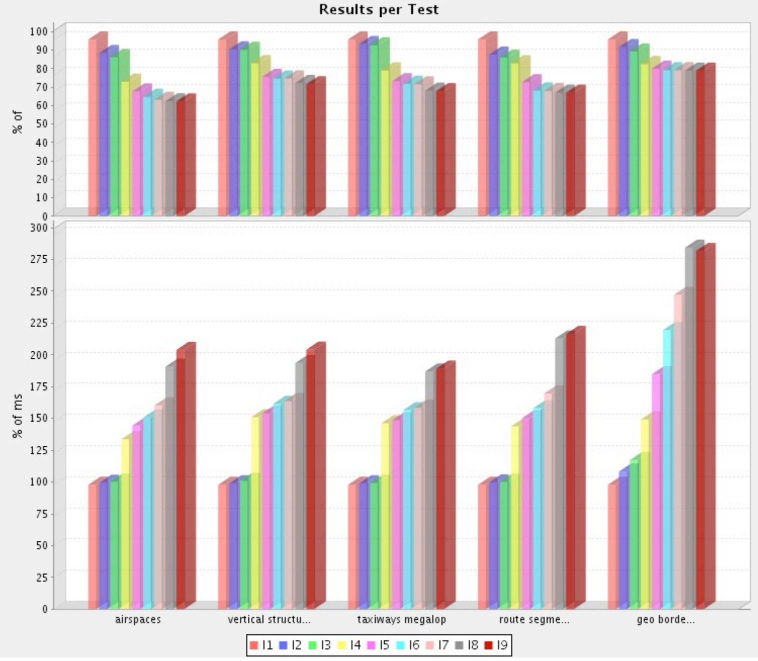
7.2.2 GZIP levels incidence for family 3

As the appreciation of gzip levels is better on large files, we give the CPU features of GZIP only for family 3. All levels are evaluated related to level 1.

7.2.2.1.1 Encoding



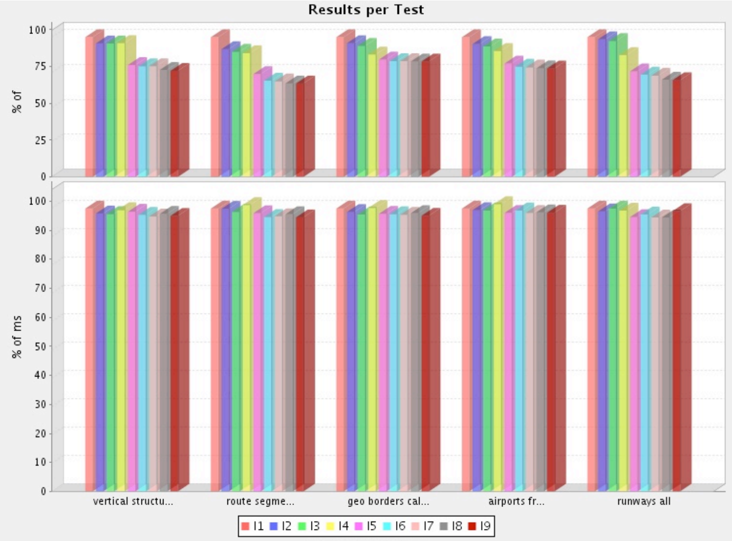


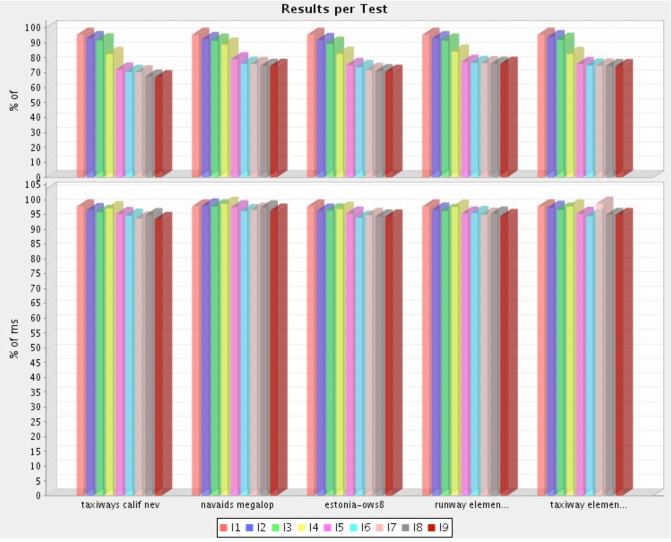


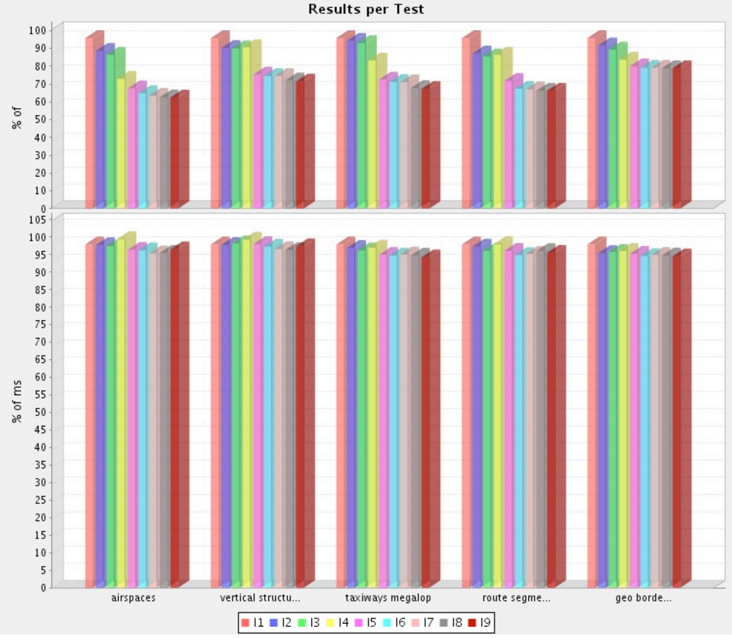
As you can see, on a server, the level 9 costs only between 2 and 4 times the cost of level 1. That’s the reason why we chose to use level 9 for java candidates (rax SAX + deflate, FIS + deflate).

Anyway level 5/6 is enough for a decent compression and avoids using too much additional CPU for only few percents of additional compaction.

7.2.2.1.2 Decoding







The time is very slow and is roughly linked to the size of the input data (the more compression, the smaller size of input), but globally the differences are unnoticeable.

7.2.3 Projection on maximum throughput of candidates

Based on an average value on encoding all files from family 3, we get these results:

Candidate	encoding rate (Mbytes/s)	output rate of encoded symbols (Mbytes/s)

Candidate	encoding rate (Mbytes/s)	output rate of encoded symbols (Mbytes/s)
XMLNeither (raw SAX)	68	68
XMLDocument (deflate level 9)	11	1
FastInfosetNeitherSAX (FIS without deflate)	163	61
FastInfosetDocumentSAX (FIS with deflate level 9)	20	1.6
EXIficientNeitherSAX (EXI without schema, without deflate)	33.5	10.9
EXIficientDocumentSAX (EXI without schema, with deflate)	1.7	0.13
EXIficientSchemaSAX (EXI with schema, without deflate)	20.1	5.2
EXIficientBothSAX (EXI with schema, with deflate)	1.9	0.15

As the server we use was excellent for 2006 but not up to date in 2011, maybe a simple projection can give you a performance double compared to the one represented here.

7.3 Memory footprint

7.3.1 For initialisation of candidate (static)

Most of candidates do not make any preparation before processing an encoding or a decoding step. Generally, this just load a few global data into memory, but who pass unnoticeable compare to java behavior, and for garbage collector.

Only EXI is preparing its work by parsing the schemas of the files it will have to compress/uncompress. This preparation phase is long (circa 5 seconds) and consumes memory (25MB).

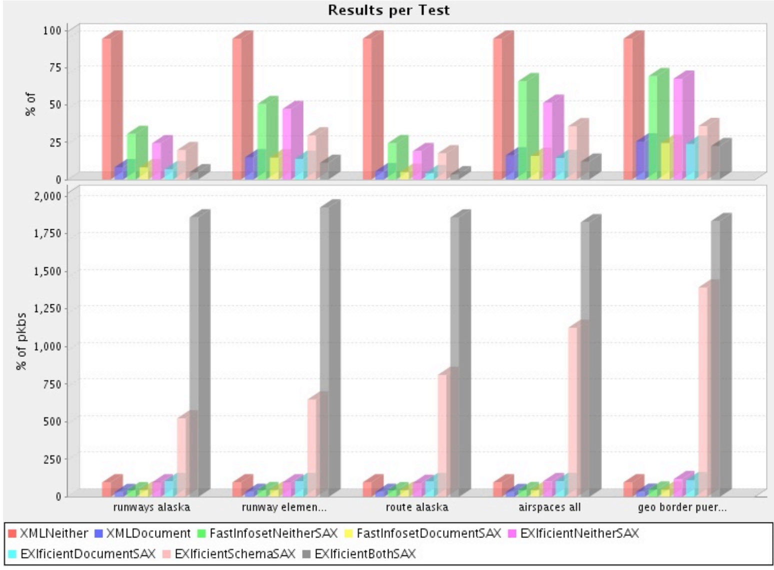
As AIXM schema weight around 1.6MB, the ratio between grammar and schema seems to be around 15x.

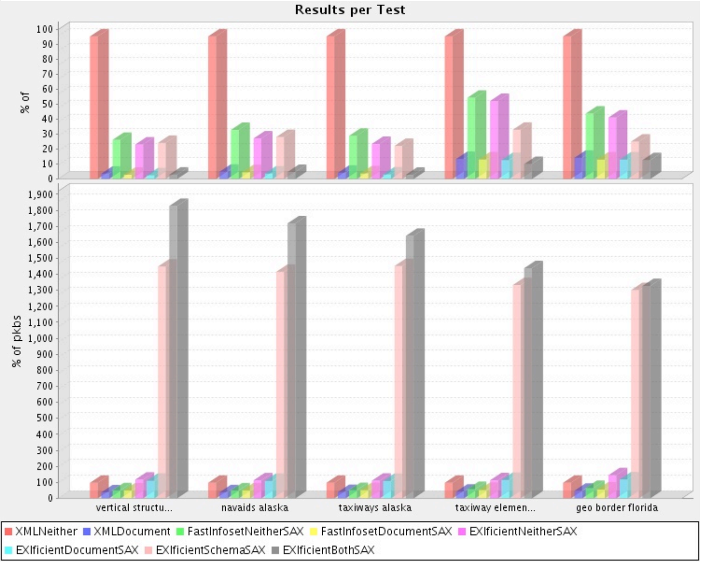
7.3.2 For a run (dynamic)

Most of candidates do not consume any noticeable amount of memory to perform encoding or decoding. Anyway when deflate (and level 9) add to the process, there is always some memory consumed as deflate store the past 32KB of data, and a hashtable with a key of any succession of 3 bytes possible and all references as values.

The following figures are all computed for family 2, to lower the incidence of the garbage collector (works on memory banks, on blocks) and the incidence of output buffer.

7.3.2.1 Encoding





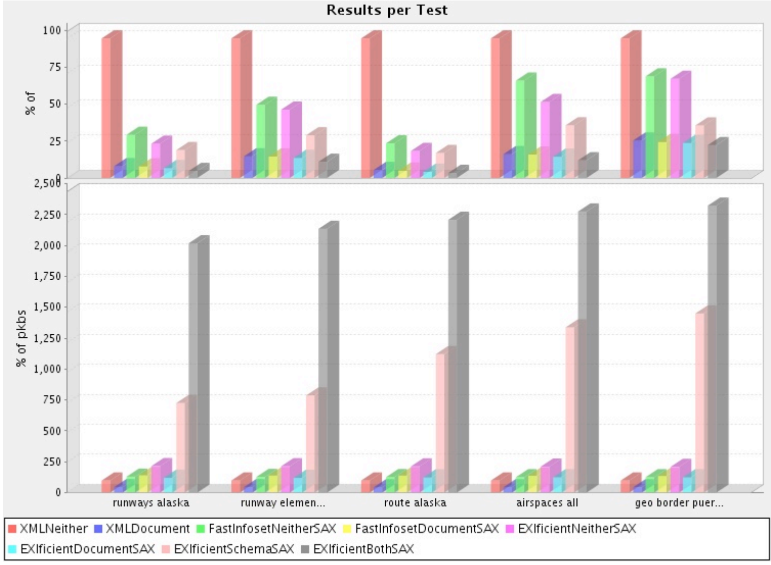
EXI is consuming a lot of memory related to other algorithms, especially grammar handling and deflate post-compression. As the memory used by the output buffer is taken in consideration, the raw SAX consumes more than gzip and FIS. In a network streaming option, this would not be the case.

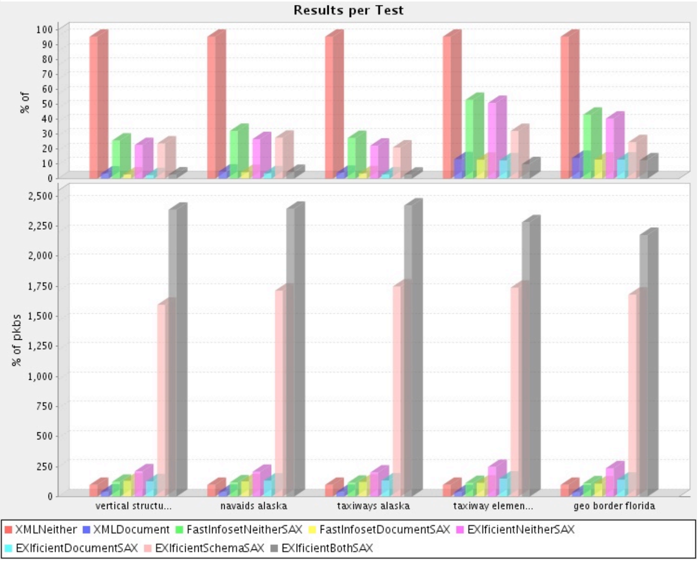
Anyway, EXI with schema and deflate is consuming too much:

- Around 150-200 MB for family 1,
- Around 275-300 MB for family 2,
- Around 250-500 MB for family 3

As Japex memory consumption measures are not so realistic (because the garbage collection control is very complex, on successive runs), we will just remember the huge memory consumption of EXI without trying to giving a law. The initial grammar computation (25MB) doesn't weight much compare to the one instantiated to perform compression / decompression operations (150MB even for very small files).

7.3.2.2 Decoding





The conclusion is exactly the same as for compression, and figures for EXI remains huge, even worse:

- Around 140-190 MB for family 1,
- Around 250-350 MB for family 2,
- Around 320-500 MB for family 3

7.4 Integration cost, ease of usage

All candidates are really easy to integrate with SAX, on a stream base, excepted CWXML because it only supports file processing or file mapped to memory but not real streams.

Using Schema with EXI adds a bit of complexity as you have to place in the right place all xsd files and take care of relative links between files.

7.5 Data integrity / Safety

<u>Deflate / GZIP</u>:

Deflate does no offer a specific protection against corruption, and detection is generally made on the GZIP level, where a ADER32 checksum is added. Eventually a changed bit could pass undetected or generate an impossible sequence that will make the decoder crash. In a flow option where decoding occurs as soon as data is received to feed SAX parser, the data integrity cannot be 100% assured.

FIS:

As FIS use PER for encoding, an error in the bit stream could be detected if applying on a length field or value field. In the others cases it could be missed.

EXI:

EXI does not use error correcting codes, nor detection. Event codes, like Huffman codes can be altered without necessarily raising an error from the decoder.

All candidates suffer from poor/inexistent error detection in the compressed flow, so they have to rely on lower layers (TCP/IP, Ethernet,…) to detect any error. For datalink, strong checksum are generally used (like on AVLC/VDL2), so it's not a big issue.

7.5.1 Quality of code

As deflate and FI are well spread since years, our attention is concentrated on EXI.

Exificient code lacks of comments, the only documentation around being the one generated from javadoc.

7.5.2 Complexity

Deflate is a complex algorithm, using Huffman codes, back references with various costs, … but as the code used by java is from zlib, where C code is quite short and was heavily tested, we can probably consider an error in zlib like very improbable.

FI is very simple in principle, excepted the PER encoding which is quite complex, but as the code to handle PER is supposed to be generated by an ASN.1 compiler, the probability to face a bug is reduced.

EXI is complex as it uses a proto-grammar and reduction rules to eliminate duplicate productions. The output module uses channels, multiplexers and blocks. Well as EXI is not yet a mainstream library, it's still possible to find bugs or limitations (as we made the experience during this benchmark)

7.5.3 Experience return

We were positively surprised by the handling of schema by Exificient. It works from the first time with all AIXM XSD (GML and so on) and intensively uses Xerces/XS API. Anyway, we were puzzled by the time needed to build the grammar, and the memory consumption of EXI. The deflate algorithm used by exificient is too slow. That's a very dark spot, because in all cases EXI without schema / deflate brings more compression than FI for an acceptable CPU consumption. If coupled with zlib, it would be a very interesting perspective.

FI and EXI are very easy to put in place or integrate to a SAX parser / XML reader.

8 Perspective on real world use cases

8.1 Best compaction candidate for small files or “datalink” messages

8.1.1 Understanding onboard systems constraints and datalink limitations

8.1.1.1 ATC Datalink “en route”

If cabin communications are relatively open, datalink communications between DSP and cockpit are very structured and driven between 4 main actors: Boeing, Airbus, Arinc and SITA. Buses between CMU (datalink user) and VDR (radio) is also very structured and does not evolve very often. Every time a new datalink media is proposed, 10 years run between the first running mockup and a global installation in a majority of planes. That’s why, DataLink are never up-to-date compared to the kind of communications publicly available (LTE, wifi, high speed internet through satellite with a dish antenna,…)

As Airframe manufacturers and DSP begin to install VDL-2 to replace POA DL, the throughput of VDL-2 is only 31.5 kbps compared to legacy 2.4kbps ACARS. Of course in dense area multiple frequencies can be used (up to 3) to increase the global throughput.

So it’s reasonable to think that in the next years, a bandwidth beneath 100kbps is the maximum reachable for VHL datalink or SatCom using omni-directional antennas. New datalinks (both VHF (LDACS1,2) and SatCom (IRIS)) are studied by SESAR, but they are still at very early stages and won’t offer MB/s by plane throughput anyway.

As VHF bandwith is shared between all the plane of a region (radius around 100NM, and a sender cannot use the radio channel too much time for a message regarding fairness to other planes communications.), the time when we will be able to send big volumes of data to a flying plane has not come yet.

8.1.1.2 AOC / AAC datalink grounded

However, when the plane is grounded or flying not too fast and near the ground, some commercial techno can be used such as LTE, WiMax, or even Wifi. Some companies already offer such services like Thales/GateSync or Teledyne.

Anyway theses usages as they may interest companies willing to synchronize their EFB, or update their IFE once grounded, present no much interest for D-NOTAM updates as they are ground based, deployed only on some planes, on some airports, for only several companies.

8.1.1.3 AOC datalink “en route”

These AOC communications are quite new, developed to provide internet access onboard and are mostly based on 3 media:

- Terrestrial antennas, revamping a mobile phone techno, enhanced to support plane speed and distance (10km altitude)
- Geostationary satellite (GEO) and classical DVB/S(2) + RCS
- Low orbit satellite constellations (soon MEO too)

8.1.1.3.1 Terrestrial antennas

Actually only one such solution exists, and is called the gogo-biz (previously AirCell). Several communication towers are installed on the US territory under major fly routes and provide kind of ADSL like internet aboard. A special air/ground antenna has to be installed under the plane to be able to receive and emit data.

Of course this service cannot be transposed for global ATC datalink, since it is just present on ground on profitable airways (to oceanic coverage, isolated/remote places…)

In addition as this service operates in L-band, which is a very expansive band, and very regulated in every country, a global use is very improbable.

8.1.1.3.2 DVB-S / RCS

Since the boeing initiative of ‘connection by boeing’ in early 2000, it was possible to get an internet access in a plane, like a particular living in a remote place far from PSTN DSL. This costs a special antenna on the top of the plane, very heavy, to replace the common parabolic dish used for TV reception / internet by satellite.

Several dynamic antennas exist, using different technologies. As the plane is moving, the antenna has to figure out how to point to the right satellite. This innovation is made possible trough the separate usage of multiples antennas (dipoles) and a lot of software to be able to delay / amplify differently the reception of each element to provide the direction. Same alchemy for emission, as an antenna (even a network antenna) is always symmetric for emission or reception.

Right now all commercial offers (row 44, matsushita, …) use Ku Band, but some antenna already exist to use Ka Band (like the one from DLR’s Santana project) allowing even higher bandwidth (multiple gigabits per seconds).

Those antennas are still not used for ATC, because they are heavy and cannot equip any plane (only big ones with enough room on the roof), and their difficulty to operate on polar routes where the angle required to point GEO is too highfor the antenna . As most of flights between Asia and US use such airways, this is a problem.

8.1.1.3.3 Satellite constellations on GEO

Those constellations present the advantage of being seen from the ground always at the same place in the sky. This simplifies a lot the maintenance of the link between the base station and the satellites. A big dish can be pointed to the satellite, this latest managing all the planes it sees on its visible half of the globe. To get a decent coverage, you need 3 satellites to avoid 90° angles on the ground. So a basic constellation just asks for 3 satellites separated from 120° from each over on GEO to offer a world coverage (with near 90° on poles off course…). Inmarsat propose such a solution (even 4 satellites to increased bandwidth over Europe)

Those satellites are mostly relays between planes and the ground but are quite different of basic TV DVB satellites as they can receive data from a plane equipped with an omnidirectional antenna at near 40,000 km from the satellite. This asks for a lot of small antennas, very directive with small lobes and a specific work on signal amplification and noise reduction. Their initial usage was for mobile phone.

Some countries have this kind of satellite, not necessarily with global coverage and global beams. Thuraya's network covers (with 3 satellites), a part of Africa, Europe and a part of Asia including Australia. Japan had such a program for aviation (MTSAT).

These GEO constellations are already used for ATC (SITA, ARINC) through 2400 bps ACARS on Inmarsat SBB.

8.1.1.3.4 Satellite constellation on LEO (soom MEO)

The main problem with a low orbit constellation is their mobility. In fact such a satellite moves so fast (an earth orbit in few hours, 100 min for Iridium), that you can see it only for 10 minutes max, so frequent hand-off are part of the game. As those constellation cover the whole world (including polar area), they require a lot of satellites (66 for Iridium, 48 for Global Star) to cover the entire moving globe and not offer a moving hole. So those constellations are very expensive. Another problem to address is the visibility from ground, as they move fast, only one satellite is visible from a ground station at a given time, so they have to relay the information along the constellation.

The main advantage of LEO is the proximity, as you don't need the same power or amplification when you deal with planes only 600 km under you (to compare to the 36000 km from GEO). That proximity allows micro satellite (under 1000 kg) with smaller solar panels, and reduced antennas.

Iridium is already used for ATC (Arinc).

Many projects are ongoing for the next 10 years, on both LEO (Iridium Next) and MEO (google project called O3B) and should bring high speed internet access for mobiles using omnidirectionnal antennas (so aviation should be a target).

8.1.1.4 Cost of airborne embeded software

As software assurance level (SWAL) asked for avionic software is higher onboard than for what you can design for ground IT service, the quality of the libraries / components brought onboard is predominant and have to comply to ED12B and DO-178C (common to both EUROCAE and RTCA). So compression is mandatory but not at any cost, because you need to prove the stability and reliability of your software.

8.1.1.5 Bandwith constraint

In the case of this study, uploading a D-NOTAM on an en-route "standard" plane asks for low bandwidth datalink and using the scarce remaining bandwidth let to AOC communications by ATM communications

That's the reason why compression is mandatory; the smaller will be the data to send the better. As most of datalink techno are using frames/paquets, the goal would be to keep data small enough to fit in one single frame/paquet/time slot.

POA: 2400 bps by frequency shared between all planes on a 100 miles radius

AOA: 31kbps by frequency shared by all plane in a 100 miles radius and with ATN.

Inmarsat BGAN: between 200 and 432 kbps by channel (up to 2) and by plane, depending on the antenna type (active or not) (or up to four 64Kbps channel for swift 64).

Iridium: 2400 bps by plane

8.1.1.6 Message size for each datalink

As POA limits messages to a maximum of 16 blocs of 220 characters, it's difficult to imagine sending a message bigger than 3KB. As VHF communication is noisy, the more frames you send, the less probable it is to retrieve all pieces.

For ATN which relies on X.25 and connected communication with detection of lost messages, you can send bigger messages. As ATN relies on AVLC and on VDL2, burst frames contains only 249 bytes, so the less burst you use, the better.

SatCom use slots, and as SatCom is mainly thought for phone communications, you find small slots.

The SBB service of Inmarsat provide both IP and packet modes (ISDN), but no data is publicly available concerning a frame size.

Iridium uses messages up to 1960 bytes (SBD) cf. [IRIDUM_SBD]

8.1.2 Best candidate

Regarding the message size data, and the result obtain for compression of D-NOTAM, we recommend using EXI with both schema and deflate post-compression.

Anyway, even if the compression provided by this candidate is the best one around (reduction from 5321 bytes to 711 bytes), a D-NOTAM will still span on 3 VDL2 burst frames, or 4 POA blocs. As the probability to have a corrupted message increases with the numbers of frames used, the goal is to fit into only one frame. D-NOTAM won’t fit in less than 220 characters.

But this compression level is enough to send a full D-NOTAM through Iridium’s SBDS.

8.2 Best compaction for synchronization of databases across ground network

The recommendations given in this chapter uses intensively the measures of maximum throughput of the candidates given in 7.2.3.

8.2.1 Slow network links (<1Mbits/s)

As such a low speed of 1Mbps (like an anemic DSL line), you could use almost any algorithm of compression. If your server doesn’t serve more than 1 client at a time (average), you could go for EXI with both schema and deflate (0.15 MBps will just make enough to fulfill the bandwidth).

8.2.2 Fast network links (>1Mbit/s)

Between 1 and 10 Mbits/s (regular DSL line, or a rented EtherLink access with limited bandwidth), EXI with deflate is not more an option, excepted if you have between 5 and 10 clients connected simultaneously in average and a server with a least 8 cores (HT included) or more.

But in most cases, for regular server and a limited number of clients (with no overlap of requests), we recommend to use Fast Info Set with deflate or simply deflate alone (with level between 5 and 9). As this latest choice is already implemented in HTTP servers, it will cost you nothing.

8.2.3 High speed network links (>10Mbit/s)

With a bandwidth on the 100Mbps range, closer to a LAN access, the usage of deflate is no more possible, excepted if the server is multi-core (at least 8) and that more than 5 clients are placing requests at the same time. In this scenario, you can still use FI but maybe you will need to reduce the level of post-compression brought by deflate to a level 1-4 (1 is doing quite some compression at a very low cost).

In a standard usage, you can consider using FI without deflate, or EXI without schema and without deflate (who will provide a bit more compression).

If you are more in the lower bound of the 10-100Mbps range, you can also consider using EXI with schema and without deflate

To sum up the 10-100Mbps range is difficult to address simply. You must try to stick to deflate as much as possible as it will provide the best compression and thus allowing you to maximize the quantity of XML sent through the same wire (even if you have to lower the deflate level to 1). If you server cannot tolerate the CPU additional consumption, then you have no choice and have to deal with FIS or EXI but without deflate. As those 2 formats are bit aligned, you should disable HTTP deflate compression from HTTP negotiation, because the result will be bad and cost extra CPU for no result.

8.2.4 Very high speed links (~circa 1Gbps)

On a gigabit (or more) LAN, you will be limited by both Java (network bindings) and SAX. That leaves only 2 choices: raw XML or FI without deflate. HTTP deflate compression have to be inhibited in these cases.

9 Looking forward, improvements

9.1 Toileting D-NOTAMs before emission

As you could notice when looking closely to a D-NOTAM, you find the same information in many places (features impacted + timeslices, but also in raw text). Maybe this redundancy can be worked around.

Also a lot of GML Ids are used; maybe some of them are unnecessary for an EFB.

9.2 Improving coordinates handling compression

The bad but not so unexpected result from this ER, is that none of the evaluated candidate is made for GML. Off course GML relies on XML, but its specificity is to handle coordinates. And as we could notice, deflate works better on coordinates as strings than on a sophisticated representation of them (EXI or IEEE754).

A good candidate for GML would have to cope with lists of coordinates, understand the dimension scaling and the precision needed.

Our guess, is that if EXI would treat coordinates (gml:posList and so on) apart and give them a specific compression layer (like the golomb/rice coding done on audio data of both stereo channels by FLAC, coupled with a MTF algorithm to sort coordinates in the first place), we could allow EXI to perform 50% better than today. But this candidate will be a specific EXI for GML, so a fork of the W3C EXI main branch of specification).

Another way to bring compression would be to modify float numbers directly in the XML file to made them more likely to be compressed by deflate (like by storing the differences between the current couple of coordinates and the previous one read, or by issuing a change of reference frame to reduce the size of ascii coordinates) but it wouldn't be as efficient.

9.3 Getting a little more compression using simple algorithms

As D-NOTAMs are short (less than 10KB), some existing algorithms could be adapted to get an additional compression performance, enough to match a specific media (like iridium's SDBS). Our experiment with deflate using a previously filled dictionary based on AIXM's XSDs shows that we could reduce a D-NOTAM to 1272 bytes (instead of 1671 using deflate alone). That kind of arrangement permits to re-use a well known compression algorithm with a recognized software assurance and reaching a compaction size goal without turning to a sophisticated algorithm with no experience return and no software assurance warranty.

10 Conclusion

EXI with schema knowledge of AIXM coupled with deflate post-treatment is very adapted for compaction of D-NOTAMs, as D-NOTAMS do not use much coordinates in their structure. This combination allows reducing a D-NOTAM to only 700-750 bytes, or only 13% of their original size. Such a size makes their transfer through datalinks possible, and will make EFB aware of the inner AIXM nature of NOTAMs.

Anyway 700 bytes are still a lot of bytes, and could make say to AIXM detractors that the same textual NOTAM would weight less, and that they could easily figure out a binary dedicated format to reduce this size under 100 bytes. This is certainly truth, but the goal of D-NOTAM is to make a bridge between AIP and NOTAMs, to make NOTAMs look friendly on a EFB screen (like showing a specific NAVAID equipment dead, a portion of route closed, or a full airspace reserved for military usage). In this context Digital does not mean compression, but more service.

Regarding general AIXM compression, EXI doesn't add any specific value when dealing with raw GML data (coordinates) or big amount of data where deflate take the lead over EXI schema knowledge, AIXM staying too complex. As our benchmark suggests, Fast Info Set with a selected deflate level should accommodate most of use cases and bring both speed and low memory footprint. This conclusion could change if EXI (at least Exificient) could improve its deflate processor, but right now it's too slow to compete with FIS on a WFS server side.

Annex B

XML Schema Documents

1. XML Schema for Japex test suite (from Japex 1.2.2), suitable if you want to create your own test using the platform:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.sun.com/japex/testSuite"
    xmlns:tns="http://www.sun.com/japex/testSuite"
    elementFormDefault="qualified"
    xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
    jxb:version="2.0">

    <!-- Adds the suffix Element to avoid name clashes -->
    <xsd:annotation>
        <xsd:appinfo>
            <jxb:schemaBindings>
                <jxb:nameXmlTransform>
                    <jxb:elementName suffix="Element"/>
                </jxb:nameXmlTransform>
            </jxb:schemaBindings>
        </xsd:appinfo>
    </xsd:annotation>

    <!-- Description element - typically HTML content -->
    <xsd:element name="description">
        <xsd:complexType mixed="true">
            <xsd:complexContent>
                <xsd:restriction base="xsd:anyType">
                    <xsd:sequence>
                        <xsd:any processContents="skip" minOccurs="0"
                            maxOccurs="unbounded" namespace="##other"/>
                    </xsd:sequence>
                </xsd:restriction>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>

    <!-- Parameter and parameter groups -->
    <xsd:element name="param">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:restriction base="xsd:anyType">
                    <xsd:attribute name="name"  type="xsd:string"
use="required"/>
                    <xsd:attribute name="value" type="xsd:string"
use="required"/>
                </xsd:restriction>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="paramGroup">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref="tns:ParamOrParamGroup" minOccurs="1"
maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string"
use="required"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:group id="ParamOrParamGroup" name="ParamOrParamGroup">
        <xsd:choice>
            <xsd:element ref="tns:param"/>
            <xsd:element ref="tns:paramGroup"/>
        </xsd:choice>
    </xsd:group>

    <!-- Test cases and test case groups -->
    <xsd:element name="testCase">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref="tns:ParamOrParamGroup" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string"
use="required"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="testCaseGroup">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref="tns:ParamOrParamGroup" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:group ref="tns:TestCaseOrTestCaseGroup"
minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string"
use="required"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:group id="TestCaseOrTestCaseGroup"
name="TestCaseOrTestCaseGroup">
        <xsd:choice>
            <xsd:element ref="tns:testCase"/>
            <xsd:element ref="tns:testCaseGroup"/>
        </xsd:choice>
    </xsd:group>

    <!-- Drivers and driver groups -->
    <xsd:element name="driver">
        <xsd:complexType>
```

```
            <xsd:sequence>
                <!-- Optional description for the driver -->
                <xsd:element ref="tns:description" minOccurs="0"/>
                <xsd:group ref="tns:ParamOrParamGroup" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string"
use="required"/>
            <xsd:attribute name="normal" type="xsd:boolean"
default="false"/>
            <xsd:attribute name="extends" type="xsd:string"
use="optional"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="driverGroup">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref="tns:ParamOrParamGroup" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:group ref="tns:DriverOrDriverGroup" minOccurs="1"
maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string"
use="required"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:group id="DriverOrDriverGroup" name="DriverOrDriverGroup">
        <xsd:choice>
            <xsd:element ref="tns:driver"/>
            <xsd:element ref="tns:driverGroup"/>
        </xsd:choice>
    </xsd:group>

    <!-- Test suite -->
    <xsd:element name="testSuite">
        <xsd:complexType>
            <xsd:sequence>
                <!-- Optional description for the testsuite -->
                <xsd:element ref="tns:description" minOccurs="0"/>

                <!-- Zero or more params groups or params -->
                <xsd:group ref="tns:ParamOrParamGroup" minOccurs="0"
maxOccurs="unbounded"/>

                <!-- One or more driver groups or drivers -->
                <xsd:group ref="tns:DriverOrDriverGroup" minOccurs="1"
maxOccurs="unbounded"/>

                <!-- One or more test case groups or test cases -->
                <xsd:group ref="tns:TestCaseOrTestCaseGroup"
minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" 
use="required"/>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```