Open Geospatial Consortium Inc.

Date: 2005-10-31

Reference number of this OGC® Project Document: **OGC 05-050**

Version: 1.0.0

Category: OpenGIS® Discussion Paper

Editor: Craig Bruce (CubeWerx Inc.)

GML Performance Investigation by CubeWerx

Copyright notice

Warning

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OpenGIS® Discussion Paper
Document subtype: if applicable
Document stage: Approved
Document language: English

This page left intentionally blank.

TABLE OF CONTENTS

1 Scope....1
2 Conformance....1
3 Normative references....1
4 Terms and definitions....1
5 Conventions....2
5.1 Requirement levels....2
5.2 Symbols (and abbreviated terms)....2
6 Encoding formats....2
6.1 Binary-encoding formats....3
6.2 Data-compression formats....3
6.3 Complete format list....4
7 Coordinate-value precision....5
8 Network environments....6
8.1 Local filesystem....6
8.2 LAN and Internet environments with WFS....6
8.3 WFS integration....6
8.4 External codec....7
8.5 Limited-bandwidth network....7
9 Scalability....8
10 Testing approach and environment....8
10.1 Sample application....8
10.2 Test data....9
10.3 Testing methodology....9
10.4 Testing environment....9
10.5 Testing tools....10
10.6 Local-file-system testing....11
10.7 Network simulation....11
10.8 WFS simulation....12
11 VMAP0-data testing....13
11.1 Local file-system testing....13
11.1.1 Built-up areas....13
11.1.2 Inland water bodies....17
11.1.3 Elevation points, single precision....18
11.1.4 Elevation points, double precision....20
11.1.5 Water courses, single precision....21
11.1.6 Water courses, double precision....23
11.1.7 Contour lines....24
11.1.8 Conclusions....25
11.2 LAN testing with high-performance simulated WFS....26

11.2.1 Built-up areas 26
11.2.2 Inland water bodies 26
11.2.3 Elevation points, single precision 26
11.2.4 Elevation points, double precision 27
11.2.5 Water courses, single precision 27
11.2.6 Water courses, double precision 28
11.2.7 Contour lines 28
11.2.8 Conclusions 28
11.3 LAN testing with relational-database WFS 28
11.3.1 Built-up areas 28
11.3.2 Inland water bodies 29
11.3.3 Elevation points, single precision 29
11.3.4 Elevation points, double precision 30
11.3.5 Water courses, single precision 30
11.3.6 Water courses, double precision 30
11.3.7 Contour lines 30
11.3.8 Conclusions 31
11.4 Internet testing with simulated high-performance WFS 31
11.4.1 Built-up areas 31
11.4.2 Inland water bodies 31
11.4.3 Elevation points, single precision 32
11.4.4 Elevation points, double precision 32
11.4.5 Water courses, single precision 32
11.4.6 Water courses, double precision 33
11.4.7 Contour lines 33
11.4.8 Conclusions 33
11.5 Internet testing with relational-database WFS 34
11.5.1 Built-up areas 34
11.5.2 Inland water bodies 34
11.5.3 Elevation points, single precision 34
11.5.4 Elevation points, double precision 35
11.5.5 Water courses, single precision 35
11.5.6 Water courses, double precision 35
11.5.7 Contour lines 36
11.5.8 Conclusions 36
11.6 Dial-up testing 36
11.6.1 Measured results 36
11.6.2 Extrapolated results 37
11.6.3 Conclusions 37

12 MSD3-data testing 37
12.1 Local file-system testing 38
12.1.1 MSD3 aggregation, 3D 38
12.1.2 MSD3 aggregation, 2D 40
12.1.3 AAL015 40
12.1.4 LAP030 41

12.1.5 PAL015....42
12.1.6 Conclusions....43
12.2 LAN testing....44
12.2.1 MSD3 aggregation, 3D....44
12.2.2 MSD3 aggregation, 2D....45
12.2.3 AAL015....45
12.2.4 LAP030....46
12.2.5 PAL015....46
12.2.6 External codec....46
12.2.7 Conclusions....47
12.3 Internet testing....48
12.3.1 MSD3 aggregation, 3D....48
12.3.2 MSD3 aggregation, 2D....48
12.3.3 AAL015....49
12.3.4 LAP030....49
12.3.5 PAL015....49
12.3.6 External codec....50
12.3.7 Conclusions....50
12.4 Dial-up testing....51
12.4.1 MSD3 aggretation, 3D....51
12.4.2 MSD3 aggregation, 2D....51
12.4.3 AAL015....52
12.4.4 LAP030....52
12.4.5 PAL015....52
12.4.6 Conclusions....53

13 GML issues....53
13.1 The trouble with application schemas....53
13.2 GML MeasureType....54
13.3 GML streamability....55
13.3.1 Mid-stream errors....55
13.3.2 Bounding envelope....55
13.3.3 Feature interleaving....55

14 Conclusions....56
14.1 Local file-system testing....56
14.2 LAN testing....57
14.3 Internet testing....57
14.4 Dial-up testing....58

Preface

The Open Geospatial Consortium (OGC) is an international industry consortium of more than 300+ companies, government agencies, and universities participating in a consensus process to develop publicly available geo-processing specifications. This Interoperability Program Report (IPR) is a product of the OGC Web Services Initiative, the objective of which is to provide a vendor-neutral interoperable framework for the web-based discovery and exploitation of geo-processing functions.

The OGC Web Services Initiative is part of the OGC's Interoperability Program: a global, collaborative, hands-on engineering and testing program designed to deliver prototype technologies and proven candidate specifications into the OGC's Specification Development Program. In OGC Interoperability Initiatives, international teams of technology providers work together to solve specific geo-processing interoperability problems posed by Initiative sponsors.

Submitting organizations

This Interoperability Program Report—Performance Investigation is being submitted to the OGC Interoperability Program by the following organization:

CubeWerx Inc.
15 rue Gamelin, Suite 506
Gatineau, QC J8Y 6N5
Canada

Document Contributor Contact Points

All questions regarding this submission should be directed to the editor or the submitters:

Dr. Craig S. Bruce
CubeWerx Inc.

Panagiotis (Peter) A. Vretanos
CubeWerx Inc.

Revision history

Date	Release	Description
2005-05-19	05-050/0.0.1	Initial plan version
2005-05-24	05-050/0.0.2	Revised plan version
2005-07-28	05-050/0.0.n	Initial results reports, additional updates (n=3–9)
2005-10-31	05-050	Final IPR for OWS-3 project
2006-04-24	050	C Reed. Fix copyright, use of OGC, OpenGIS etc. Other editorial changes

Changes to the OpenGIS® Abstract Specification

No revisions to the OpenGIS Abstract Specification are required.

Changes to OpenGIS® Implementation Specifications

No revisions to any OpenGIS Implementation Specifications are required. However, some problems with GML are discussed in clause 13.

Foreword

Attention is drawn to the possibility that some of the elements of OGC 05-050 may be the subject of patent rights. Open Geospatial Consortium Inc. shall not be held responsible for identifying and or all such patent rights.

Introduction

This OpenGIS® Interoperability-Program report proposes and executes methods to evaluate the performance of the use of the Geography Markup Language (GML) as encoded in various ways.

The Geography Markup Language [GML] is an increasingly important feature-encoding format for use in interoperable GIS systems. Unfortunately, the innate text encoding that it uses is both bulky and slow to process. To help GML realize its potential, its encoding efficiency should be made to be comparable with other commonly used feature-encoding formats, such as ESRI® Shapefile [SHAPE] format. To this end, OGC has undertaken to test the compactness and system-throughput capacity of GML as encoded in numerous different ways, including binary encoding in BXML [BXML] and BinXML™ [BINXML] and general compression formats such as GZIP [GZIP] and BZIP2 [BZIP2]. The decision of potential GML adopters becomes much easier when they can have both interoperability and efficiency.

GML Performance Investigation by CubeWerx

1 Scope

This OpenGIS® Interoperability-Program report proposes and executes methods to evaluate the performance of GML as encoded in various different ways.

2 Conformance

Not required in an IP DIPR, IPR or Discussion Paper.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this Interoperability Program Report. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this document (OGC 05-050) are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

[GML] OGC 03-105r1 (February 2004), *OpenGIS® Geography Markup Language (GML) Implementation Specification, version 3.1.0*, <http://portal.opengeospatial.org/files/?artifact_id=4700>.

[GML-SF] OGC 05-033r9 (July 2005), *GML Simple Features Profile, version 0.0.19*, Peter Vretanos (ed.).

[XML] W3C (October 2000), *Extensible Markup Language (XML) 1.0 (Second Edition)*, <http://www.w3.org/TR/REC-xml>.

4 Terms and definitions

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

1.
BinXML™
Binary encoding format/system for XML data.

2.
BXML
Shorthand for “Binary eXtensible Markup Language”. The format is a markup-for-markup-compatible binary encoding of XML data.

3.
Codec
Encoder-decoder pair.

5 Conventions

5.1 Requirement levels

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [KEYWORDS].

5.2 Symbols (and abbreviated terms)

The following symbols and abbreviations are used in this document:

API	Application-Program Interface
BinXML™	Binary-XML encoding system
BXML	Binary eXtensible Markup Language
BZIP2	general compression format
C/C++	C and/or C++ programming languages
DOM	Document Object Model
GML	Geography Markup Language
GZIP	GNU Zip compression format
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
MIME	Multipurpose Internet Mail Extensions
RFC	Request For Comments
SAX	Simple API for XML
URL	Uniform Resource Locator
WFS	Web Feature Service
XML	Extensible Markup Language

6 Encoding formats

It would informative to test the throughput and compactness of a number of feature-encoding formats, including GML-XML, GML-BXML, ESRI® Shapefile, and MapInfo® MIF, both with and without GZIP and BZIP2 compression. The GML version will be 3.1.1 with the Simple Features [GML-SF] profile. With binary encoding, we

believe that the GML format can be much more competitive with binary formats such as Shapefile in terms of compactness and processing throughput. MIF is an all-text format, which would be interesting to compare to GML-XML format.

6.1 Binary-encoding formats

BXML [BXML] is a straightforward patent-unencumbered binary-encoding format for XML data which is designed to be easy to implement and for which an open-source C-language reference implementation is freely available [CWXML]. It is also an OGC Discussion Paper. One of its most important features for our immediate purposes is that it allows lists of floating-point coordinate values to be directly represented as raw-binary arrays. This provides both compactness and system throughput since converting coordinate values to and from a text representation is surprisingly time-consuming.

BinXML™ is a commercial software product for which Galdos Systems is a reseller. Galdos can better explain the strengths of BinXML™.

While a direct comparison of BXML and BinXML™ implemented in the same runtime environment would be desirable, CubeWerx does not have the resources or expertise to implement testing for BinXML™ in the scope of the OWS-3 project even if software licenses were provided. However, an independent BinXML™ implementation in a different software architecture, such as a scripting-language/XML-tool environment relative to CubeWerx's RDBMS/C-language environment would still allow the direct comparison of encoding compactness and would provide an interesting performance characterization of the effectiveness of different encoding methods in the different GML-processing environments as well as a concrete comparison of the architectural approaches overall.

6.2 Data-compression formats

GZIP [GZIP] and BZIP2 [BZIP2] formats do an excellent job of compressing XML data. In one rough example that was tested, GZIP achieved around an 80% compression rate (one-fifth the original size) and BZIP2 achieved around an 85% rate (one-sixth the original size) with GML data that had no unnecessary whitespace. Specialized XML compression methods probably cannot out-compress GZIP and BZIP2 significantly in the general case.

However, the encoding and decoding of GZIP and BZIP2 formats are very CPU-intensive, so the “compaction” of the raw information before the “compression” step can have an large impact on overall system performance. This trade-off is investigated. Binary encodings of XML are considerably more compact than XML itself.

ZIP [ZIP] and **`compress`** [COMPRESS] compression formats are also in general use. ZIP format does not need to be considered in this investigation because it generally uses exactly the same compression method that GZIP does, called “Deflate”. It is also not a streamable format which imposes complications in a streaming client/server environment. The **`compress`** method does not achieve as high a compression rate as GZIP (74%

compression for the example) and, as it turns out, does not execute as quickly as GZIP, so it does not need to be considered in this study. Historically, **`compress`** executed more efficiently than GZIP, however, use and refinement of the method stagnated for many years because of patent issues.

6.3 Complete format list

The complete list of GML-encoding combinations that should be tested for both compactness and throughput is as follows:

Base Format	***Encoding***	***Compression***
GML	XML	none
GML	XML	GZIP
GML	XML	BZIP2
GML	BXML	none
GML	BXML	GZIP
GML	BXML	BZIP2

The GML encodings should be compared to other common feature-encoding formats, plus an extremely efficient one:

Base Format	***Encoding***	***Compression***
ESRI® Shapefile	Binary + Text	none
ESRI® Shapefile	Binary + Text	GZIP
ESRI® Shapefile	Binary + Text	BZIP2
MapInfo® MIF	Text	none
MapInfo® MIF	Text	GZIP
MapInfo® MIF	Text	BZIP2
CubeWerx® MDF	Binary	none
CubeWerx® MDF	Binary	GZIP
CubeWerx® MDF	Binary	BZIP2

MDF is an high-performance format used by CubeWerx to load feature data into the Oracle database system. It is a simple all-binary format in which geometries are stored in OGC Well-Known-Binary [WKB] format and the other properties are stored in fixed-sized fields.

One complication with the Shapefile, MIF, and MDF formats is that they are realized as a collection of multiple files instead of just one file. The represented size will be the sum of the sizes of the constituent files that contain the equivalent information to GML format. In a client/server environment, these files would need to be packaged together in an archive format such as ZIP or TAR [TAR], and the packaging requirement limits the

potential streamability of the formats since the complete first file in the package must be available before the first record of the subsequent file is streamed out from a server.

For our testing purposes, it is sufficient to test the Shapefile and MIF formats only on a single machine to compare them with GML results on a single machine. It would be a burden to incorporate these formats into the WFS interface, though it is technically feasible (when combined with a packaging format as mentioned above).

One capability lacking from GML that Shapefile has is that of random access. BXML and presumably BinXML™ incorporate mechanisms for random access to elements based on identifier attributes, but this is not a spatial index. A spatial index would be needed to access features spatially as Shapefile and MDF allow, but none is provided by the GML format.

7 Coordinate-value precision

An easy way to achieve data compaction in both the XML and BXML representations of GML data is to reduce the precision of the coordinate values in the features. Coordinate values are normally represented as double-precision floating-point values in GIS systems and this is equivalent to about 16 significant decimal digits. However, no real-world data is this accurate and some applications do not even need the full precision of the real data. Sixteen decimal digits is equivalent to 4.4 nanometers (billionths of a meter) of resolution along the equator of the Earth.

The GML encoding should be tested with both double-precision coordinate values as well as single-precision in order to quantify the increased compactness. Single precision is equivalent to about seven significant decimal digits or 2.4 meters of resolution along the equator. BXML encoding has the capability to dynamically select different encodings for XML-Schema [XML-SCHEMA] content identified as “list of double,” such as GML coordinates values.

Intermediate precisions could also be tested. This is easy to do with XML-encoded GML by selecting any integral number of significant decimal digits, though it is slightly more difficult to achieve with a binary encoding. Floating-point doubles would best be used for this encoding with the least-significant bits of the mantissas zeroed out. This data will have the same uncompressed size as with the full double precision, but it will compress better. CubeWerx does not consider it to be worthwhile to investigate intermediate precisions at this time, though in the future, if systems have metadata about the absolute accuracy of the coorindate values in a feature collection, it could automatically select intermediate precisions individually for each feature type.

There is an indirect benefit to always using full double precision values in all calculations in a distributed system in that most computers implement IEEE 754-1995 [FLOATS] floating-point format in hardware so that most systems will compute similar or identical results and will behave more deterministically when making threshold decisions (e.g.,

"**`value >= 1e6`**"). However, the increased compactness of using lesser precision in GML data may be a more important benefit.

8 Network environments

Testing should be conducted in the following environments: local file system (single machine), LAN (Local Area Network), Internet, and limited bandwidth (dial-up).

8.1 Local filesystem

It is very important to test GML-encoding performance on a single machine to determine directly and accurately the performance of the different GML encodings in isolation from confounding factors like network delays. The most scientifically accurate methods should be applied in this experiment and measuring only in a distributed environment would be like measuring the dimensions of a book with a ruler that is marked only in integral feet. Also, the encoding compactness will be the same as in local case as in the distributed case but will be easier to measure in the local case. Testing on a local machine will also be necessary to compare against formats like Shapefile on their "home turf".

8.2 LAN and Internet environments with WFS

GML is normally used to exchange geographic data in the LAN (Local-Area Network) within a single organization and over the Internet between organizations. The purpose of testing in these network environments, considering that raw performance can be measured in a local-machine environment, is to demonstrate interoperability and to assess whether the raw performance differences are actually significant when combined with the network-transfer and WFS-processing costs of the normal GML usage environment.

The LAN and Internet environments have different performance characteristics and represent the common cases for distributed GML-processing systems. Compression almost always makes sense in an Internet environment since modern computers can apply GZIP compression to at least 70 Mbits/sec of text GML data, thereby saturating even high-speed Internet links. However, modern LANs operate at at least 100 Mbits/sec, which imposes a different cost structure on encodings.

8.3 WFS integration

Two methods are available to integrate alternate GML encodings into the WFS interface: the WFS output-format mechanism and the HTTP transfer-encoding mechanism.

The WFS interface provides a mechanism for selecting among different encodings for the feature data that it delivers. The WFS 1.0.0 specification describes an optional attribute called "**`outputFormat`**" on the **`GetFeature`** request for which the default value is "**`GML2`**" in Section 9.2 on page 25. The description on the same page also says:

> The outputFormat attribute defines the format to use to generate the result set. The default value is **GML2** indicating that GML [2] shall be used. Vendor specific formats (including non-XML and binary formats), declared in the capabilities document are also possible. The available types are identified using **<ResultFormat>** tag of the Capabilities document.

In WFS version 1.1.0, the format identifiers are a little different. The example in Section 9.2 shows the string as "text/xml; subtype=3.1.1". To request BXML, the equivalent would be "application/x-bxml; subtype=3.1.1". The Capabilities schema is a little hard to follow, but it looks like the Capabilities tag is **<OutputFormatList>**.

HTTP [HTTP], which the WFS interface is build on top of, also provides a built-in mechanism for requesting GZIP or **compress** encoding of the transferred content. The client can include an "**Accept-Encoding**" MIME-header tag in the request message and the server has the option of responding with a "**Content-Encoding**" MIME-header tag in the reply message that indicates that the body is compressed. The HTTP RFC does not explicitly spell out support for BZIP2 encoding, but it can logically be included by using either "**x-bzip2**" or even "**bzip2**" as the encoding-tag value. The nature of the mechanism makes the use of ad-hoc symbols non-intrusive. BXML encoding could even be requested in this way.

Since this content-encoding mechanism is built-in and optional, any WFS or WFS client has the option to support it and the data transfer will be optimized wherever possible. All WFS clients and servers should support at least this mechanism. Really, CubeWerx finds it rather mind-boggling that any production work would be attempted with GML in the Internet environment without at least using the HTTP mechanism for GZIP compression of the data stream.

8.4 External codec

An external codec is a piece of software that would be placed between a WFS client and server which would accept an XML-encoded GML stream from the WFS server, encode it into an alternate format, transport it to the client site, decode it back into XML-encoded GML and then pass it to the client application.

A general external codec has at least two major problems: it is complicated to implement since it essentially would be a distributed, cascading WFS; it is inefficient since it requires the parsing and generation of the GML data not once but three times, including two times as XML-encoded GML, which would most likely render it less efficient than directly GZIP-compressing a plain XML-encoded GML stream using the HTTP mechanism.

8.5 Limited-bandwidth network

A limited bandwidth environment can serve as a guide for wireless and dial-up cases where limited facilities are available. A suitable speed it 56 kbps. The general

compression methods are expected to have the most significant impact in this environment, since the crucial performance factor will be minimizing the bandwidth usage.

9 Scalability

The scalability of GML data and processing systems should also be investigated. This can be done easily using small, medium, and large data sets of the representative geometry types: points, lines, and polygons. The testing definition of a 'large' data set is one with approximately 1-million features.

A GML-processing system based on utilizing a DOM (Document Object Model) approach to XML processing will have its scalability inherently limited to the number of features that can be stored in machine memory at one time, so GML-processing systems should not be based strictly on the DOM methodology. Systems based on XSLT [XSLT] tools will have the same scalability problems for the same reason.

CubeWerx anticipates no inherent scalability problems with either the representation of or processing of a large number of GML features in its GML-processing systems. The CWXML [CWXML] open-source library utilized by CubeWerx provides a node/subtree mechanism that supplies the convenience of a DOM mechanism with the scalability of the streamable SAX mechanism.

Since the sponsor-supplied MSD3 data totals only in the thousands of features, VMAP0 data is used to conduct scalability testing.

10 Testing approach and environment

10.1 Sample application

The two kinds of "performance" to be measured in the GML Performance Investigations are system throughput (speed) and information-encoding compactness. To test throughput, the various software levels of a representative GML-processing application are measured in a number of usage environments. The compactness can be determined by measuring the size of the same GML feature data as encoded and compressed by the various different means.

A representative GML-processing application is the fetching and rendering of a stream of GML data into a displayable image. The application has the following software levels which can be characterized independently and in combination:

- fetch features from underlying system
- generate features in GML
- transport features

- parse features into internal programming-language structures
- render features into raster

Feature rendering is not strictly necessary for measuring GML performance, but it provides concrete visible proof that all of the features of a data set were actually processed and it supplies an interesting comparison of the performance of two different architectures for SLD [SLD] styling that have been pursued by OGC participants: native implementation vs. XML tools. Most GIS-product vendors including CubeWerx use the native-implementation approach. Galdos uses an XML-tools approach. Proponents have long advocated the benefits of each approach. Also, the software to render features is already available in the testing environments anyway, and the timing of the rendering component can easily be computed and factored out of the GML-transfer components.

10.2 Test data

An MSD3 data set was supplied by the project sponsors which is defined relative to a large central XML Schema and some VMAP0 data was selected by the participants since it has more bulk for reliable testing and it has small schemas. The VMAP0 data selected is described in clause 11 and the MSD3 data is described in clause 12.

10.3 Testing methodology

The general measuring methodology employed is to run each test case four times and discard the first run and average the subsequent three. The first run may involve variable system-caching overheads that would be difficult to measure reliably. In some cases, it is not feasible to perform four runs of every combination because it would take too long.

All time measurements are for real elapsed "wall-clock" time. This is the time of principal concern to most users. (Other conceivable timing measurements include CPU-processing time.)

All figures for file sizes, transfer speeds, use SI unit-multiplier prefixes, which are normally base-10 [SI] but include a special notation for base-2 [SI-BIN]. For example, 1 KB means exactly 1,000 bytes and 1 KiB means exactly 1,024 bytes.

10.4 Testing environment

The computer selected to be the server for the network test cases and the host for the local-file-system test case has the following specifications:

- Processor: AMD® Athlon-64 3200+ (64-bit), 2.01 GHz, 512 KiB cache
- Main memory: 1 GiB dual-channel DDR-400 MHz (PC-3200)
- Hard drive: single 200-GB SATA 3-GB/s channel
- HD sustained read speed: 63.73 MB/sec

- OS: Fedora Core 4 Linux for x86-64 (2.6.12 kernel)
- Compiler: GCC 4.0.1 with -O3 optimization
- File system: Linux **`ext3`** journaling type

Its name is "a64.cubewerx.com".

The computer selected to be the client for network testing is a laptop model with the following specifications:

- Processor: Intel® Pentium-4 Mobile (32-bit), 2.00 GHz, 512 KiB cache
- Main memory: 512 KiB
- Hard drive: single 40-GB ATA U100
- HD sustained read speed: 30.08 MB/sec
- OS: Fedora Core 4 Linux for i686 (2.6.12 kernel)
- Compiler: GCC 4.0.1 with -O3 optimization
- File system: Linux **`ext3`** journaling type

10.5 Testing tools

The following tools are be used in the testing:

- **`cwcat`** – simple command-line file copier
- **`cwdump`** – simple command-line feature-file displayer/copier
- **`cwplot`** – simple command-line feature-file plotter with SLD support
- **`xmlscan`** – simple command-line XML file copier/converter
- CWXML – XML/BXML parser/generator library
- CubeSERV® WFS – Oracle®-based Web Feature Service
- gzip – open-source compression library
- bzip2 – open-source compression library

The **`cwcat`** program is modeled after the Unix® **`cat`** program but has special features including the capability to read from remote URLs (as do all of the "**`cw`**" programs) and to limit the throughput of the data it copies. This latter feature is used to simulate Internet and dial-up networks as described in clause 10.7.

The **cwdump** program includes the capability to generate GML data in a CGI environment and this capability is used to implement the simulated high-performance WFS that is used for some test cases. The implementation is described in clause 10.8.

The **xmlscan** program is included with the open-source CWXML library and is handy for reformatting and 'pretty printing' XML/BXML data.

Version 1.2.3 of the gzip compression library is used. The maximum compression level of '9' is used since the library can produce data at this compression level much faster than the LAN can absorb it, so it is better to maximize the amount of compression. Recent versions of this library include significant performance improvements over older versions. The library-default compression level is '6'.

Version 1.0.3 of the bzip2 compression library is used. The bzip2 compression level of '9' is used, which is the maximum and the library default.

10.6 Local-file-system testing

Reading-speed testing was performed by using a script that translates the source data from the GML-XML distribution format into the format to be tested and then reads the data four times in a loop. After the first read, the file content will be cached in memory, assuming that it fits, and subsequent reads show be as efficient as the format allows.

Writing-speed testing was performed using the same approach as the reading testing, except that the time taken to read from the source-data format (uncompressed GML encoded in XML) is subtracted from the total read+write time, giving only the writing time.

10.7 Network simulation

Early Internet testing was attempted using the real Internet, but the results were far too variable minute-by-minute and day-by-day to produce results that were actually comparable to each other. Instead, this testing is carried out using the CubeWerx internal 100-Mbps LAN with the server CGI shell-script program augmented to pass the outgoing data through **cwcat** used as a precise flow-controlling filter. This filter is so light-weight that it does not influence the results and its operation closely simulates what happens with normal Internet transfers: the outgoing data from the generator program gets queued up in the outgoing socket buffers until the generator process is suspended while the data is slowly emptied onto the outgoing network connection.

The data rate selected for Internet testing is 150 Kbytes/sec, which represents a decent Internet connection between any two points in North America or perhaps the world. Because of the structure of the interaction as a small request with a single large reply, it is not necessary to simulate packet latency (ping times) since TCP/IP is a sliding-window protocol that quite effectively hides packet latency for large transfers, but a 60-millisecond latency is added on the server side anyway.

The data rate selected for Dial-up testing is 5.6 Kbytes/sec, which corresponds to 56 Kbits/sec uncompressed. A response latency of 200 milliseconds is inserted on the server side.

The LAN used is a lightly loaded 100-Mbps Ethernet system with the server and laptop client directly connected to the network. Pinging the server from the client takes an average of 0.141 milliseconds and the sustained HTTP data-transfer rate is 11.70 MB/second.

10.8 WFS simulation

CubeWerx had insufficient time to fully integrate the needs of the MSD3 data into the CubeSERV transactional WFS, so a light-weight simulated WFS was implemented with this capability by building on the **`cwdump`** feature-displaying utility. This utility is so efficient that it serves a second important purpose which is to determine what transfer costs are caused by the network as opposed to the relational-database system used by the full-feature WFS. As it turns out, the RDBMS is quite costly.

The simulated WFS has a simple CGI interface and it reads features from BXML-encoded GML files stored on the server and it recodes them on the fly according to the received request into XML or BXML encoding with GZIP, BZIP2, or no compression. In the case that uncompressed BXML is requested, the full re-coding process is still carried out for timing consistency. The returned GML data references a schema which is also generated by the web application on request to be encoded in the same way as the GML data for the VMAP0 test data and which is copied directly from a pre-encoded schema file on the server for MSD3 data. These methods correspond to the way these data sets would normally be used, since the MSD3 data is defined relative to a fixed central schema.

For example, the following request for XML+GZIP:

http://a64.cubewerx.com/ows3/simfs/simfs.cgi?REQUEST=GetFeature&TYPENAME=builtupa&OUTPUTFORMAT=text/xml%3Bencoding%3Dgzip

will reference its schema as:

http://a64.cubewerx.com/ows3/simfs/simfs.cgi?REQUEST=DescribeFeatureType&TYPENAME=builtupa&OUTPUTFORMAT=text/xml%3Bencoding%3Dgzip

The testing carried out is only to fetch and internally plot the requested data using the **`cwplot`** application, excluding the time taken to write out the resulting PNG image from the internal raster representation. The features received at the client are not stored; each is discarded immediately after plotting.

Thus, the execution time will include reading and re-coding the source data on the server, transferring it over the network, parsing the data into internal C-language feature representations, and plotting it into a C-language raster representation. The translation from GML to internal the C-language structures provides a means of validating the data

in terms of properties and data types, though this is a different from from XML-Schema-based validation.

The plotting speed turns out to be so fast that plotting adds virtually no overhead to the process, so it is safe to include here. Also, the data is fully streamed, so if the client can process the data more quickly than the network can deliver it, the client operations will effectively have no cost. The same is true of the server if it can supply the data faster than the network can absorb it.

11 VMAP0-data testing

The following feature collections from the VMAP0 data are used for testing, in order of the number of features per collection:

Collection	*Description*	*Geom Type*	*Properties*	*Features*	*Vertices*
builtupa	Built-up Areas	Polygon	6	8,346	17.3
inwatera	Inland Water Bodies	Polygon	6	153,358	19.2
elevp	Elevation Points	Point	8	175,880	1.0
watrcrsl	Water Courses	LineString	6	290,528	8.9
coutourl	Contour Lines	LineString	7	1,099,837	30.7

The terms “feature type” and “feature collection” can be used interchangeably here, since each feature collection includes features of exactly one type. The “Properties” column gives the number of properties in the features; the “Features” column gives the number of features in each feature collection; and the “Vertices” column gives the average number of vertices in the geometry of each feature, which indicates how complex the geometries are. (The start/end point of a polygon ring is counted as two vertices.) This data set includes feature-collection sizes of medium volume to large volume. The **contourl** feature collection is suitable for demonstrating the scalability of the processing applications.

The source VMAP0 geometries have single-precision coordinate values, so versions of the **elevp** and **watrcrsl** feature collections were created that have double-precision coordinate values for comparison purposes. Use of the double-precision versions is specifically noted.

11.1 Local file-system testing

11.1.1 Built-up areas

11.1.1.1 Reading

The reading results for the **builtupa** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	6.17	0.533	15,659
GML	XML	indented	7.41	0.547	15,258
GML	BXML	none	2.44	0.151	55,272
GML	XML	gzip	1.15	0.592	14,098
GML	XML	indented + gzip	1.16	0.609	13,704
GML	BXML	gzip	1.07	0.183	45,607
GML	XML	bzip2	1.016	0.882	9,463
GML	BXML	bzip2	1.053	0.404	20,658

The compression type indicated as "indented" is not compression at all but means that the text file was "pretty printed" with an indentation of two spaces for each XML-tag level. As expected, it makes the uncompressed version of the file significantly large; however, it does not add significantly to the gzip-compressed size or time. This extra whitespace compresses very well.

In general, the GZIP compression/decompression method operates faster than expected. It will likely be made faster still in the future when certain hashing-table patents expire.

We fixed a performance bug in the BXML parser during the course of this testing that caused it to unnecessarily convert numeric property values to text before re-converting them back to numbers for the reading application. This resulted in a 40% performance increase.

The GML reader is not that heavily optimized. It uses the node/subtree interface of the CWXML library, which reads in DOM-like subtrees for each feature but only holds one feature in memory at a time. In contrast, the GML writer uses a node-by-node approach and its performance of 117,549 given below shows how much faster that approach can be. The node-subtree approach is easier to program for reading, however, and some modest optimization attempts showed that reading features is harder to optimize than writing them.

11.1.1.2 Reading alternate formats

The reading results for the **`builtupa`** feature collection represented in alternative formats to GML are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
Shapefile	(binary+text)	none	3.70	0.047	178,143
MIF	(text)	none	3.25	0.249	33,558
MDF	(binary)	none	3.62	0.031	267,819

The relevant comparison here is to GML encoded in BXML at 55.3 kF/s and XML at 15.7 kF/s. The MIF text format can be processed twice as fast as XML-encoded GML

and Shapefile can be processed 3.2 times as fast as BXML-encoded GML. This is a little surprising and disappointing, given that Shapefile numeric properties are actually stored in a text representation. CubeWerx MDF format performs 4.9 times as fast as BXML-encoded GML. It should also be noted that both Shapefile and MDF formats always store coordinate values as double-precision floats, but they still out-perform BXML-encoded GML.

The processing performance of these alternate formats when compressed was not tested, but the compressed sizes were measured as follows:

Format	*Encoding*	*Compression*	*Size (MB)*
Shapefile	(binary+text)	gzip	1.45
Shapefile	(binary+text)	bzip2	1.25
MIF	(text)	gzip	1.02
MIF	(text)	bzip2	0.94
MDF	(binary)	gzip	1.49
MDF	(binary)	bzip2	1.17

The compressed sizes are comparable to but not quite as good as the GML compressed sizes, except for MIF format, which is more compact in all cases.

11.1.1.3 Writing

The writing results for the **`builtupa`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	6.17	0.693	12,043
GML	XML	indented	7.41	0.723	11,544
GML	BXML	none	2.44	0.071	117,549
GML	XML	gzip	1.15	1.698	4,915
GML	XML	indented + gzip	1.16	1.685	4,953
GML	BXML	gzip	1.07	0.436	19,142
GML	XML	bzip2	1.016	3.000	2,782
GML	BXML	bzip2	1.053	0.791	10,551

The reported timings for the writing speed are reduced by the 0.533 seconds taken to read the source **`builtupa`** data. There is quite a spike in the performance of the uncompressed BXML case, since this format is very efficient to write.

11.1.1.4 Plotting

The plotting results for the **`builtupa`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	6.17	0.582	14,340
GML	BXML	none	2.44	0.231	36,130
GML	XML	gzip	1.15	0.641	13,020
GML	BXML	gzip	1.07	0.262	31,855

The plotting window is between longitudes -180 and 0 and latitudes -40 to 90 and is size 3600 pixels horizontal by 2600 vertical and the output image type is PNG. The writing time of the PNG file is excluded from the time. The SLD styling symbol uses a light-pink fill (**#EEA9B8**) with a dark-pink outline (**#FFC0CB**). A scaled down version (to reduce document size) of the resulting image is included here:

The data is rather sparse, but this test demonstrates that the full North American coverage has been processed in our testing.

If we eliminate the GML-data-reading time, the internal plotting time is only around 0.080 seconds for a speed of around 104,000 features per second. While this performance is not strictly what is to be measured in this experiment, it would be very interesting to see how well a GML-XSLT-SVG plotting system performs. We suspect not as well. A system needs to be fast at plotting features in order to offer an effective WMS interface.

11.1.2 Inland water bodies

11.1.2.1 Reading

The reading results for the **inwatera** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	120.41	10.494	14,614
GML	BXML	none	47.44	2.802	54,732
GML	XML	gzip	20.81	11.582	13,241
GML	BXML	gzip	19.67	3.433	44,672
GML	XML	bzip2	18.61	17.535	8,746
GML	BXML	bzip2	18.30	7.686	19,953

11.1.2.2 Writing

The writing results for the **inwatera** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	120.41	15.961	9,608
GML	BXML	none	47.44	1.386	110,648
GML	XML	gzip	20.81	38.103	4,025
GML	BXML	gzip	19.67	8.881	17,268
GML	XML	bzip2	18.61	60.603	2,531
GML	BXML	bzip2	18.30	17.276	8,877

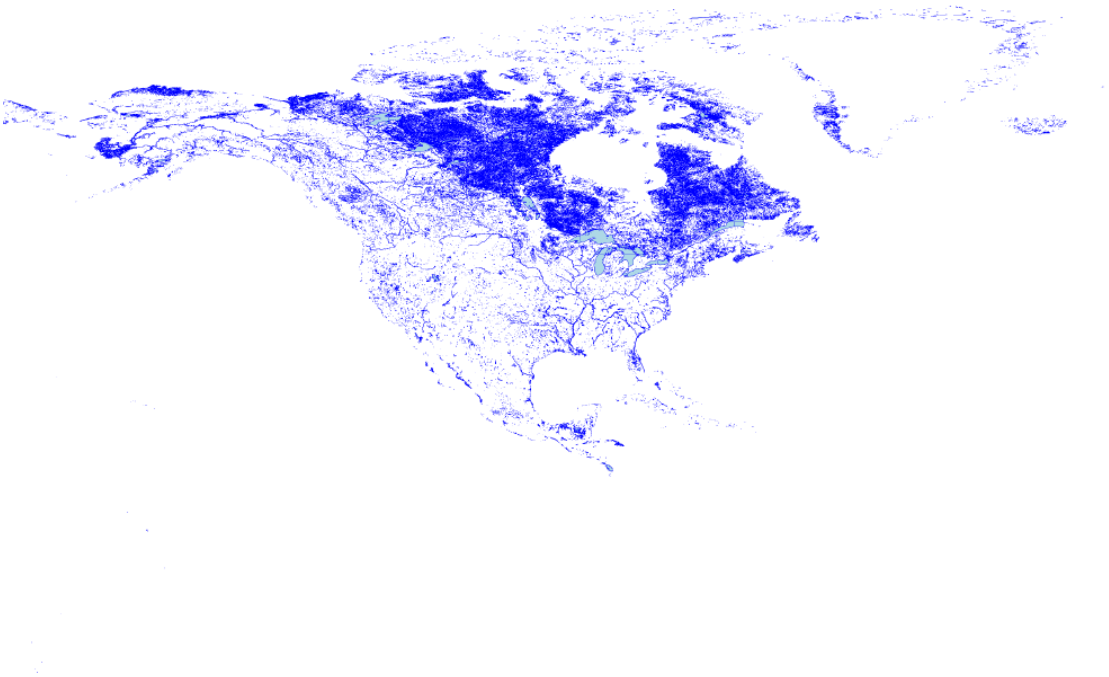
The reported timings for the writing speed are reduced by the 10.494 seconds taken to read the source **inwatera** data.

11.1.2.3 Plotting

The plotting results for the **inwatera** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	120.41	10.656	13,027
GML	BXML	none	47.44	3.560	32,917

The plotting window and image size are the same as for the **builtupa** feature collection, except that the SLD plotting style has a dark-blue stroke (**#0000FF**) over a light-blue fill (**#ADD8E6**):



The raw internal plotting speed is around 202,000 features per second.

11.1.3 Elevation points, single precision

11.1.3.1 Reading

The reading results for the **`elevp`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	62.63	7.193	24,452
GML	BXML	none	25.94	2.133	82,457
GML	XML	gzip	3.60	7.521	23,358
GML	BXML	gzip	3.23	2.321	75,778
GML	XML	bzip2	2.57	8.973	19,601
GML	BXML	bzip2	2.42	3.822	46.018

11.1.3.2 Reading alternate formats

The reading results for the **`elevp`** feature collection represented in alternative formats to GML are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
Shapefile	(binary+text)	none	13.02	0.746	235,724
Shapefile	(binary+text)	gzip	3.34		
Shapefile	(binary+text)	bzip2	3.01		
MIF	(text)	none	13.60	1.493	117,815
MIF	(text)	gzip	3.65		
MIF	(text)	bzip2	3.15		
MDF	(binary)	none	14.07	0.248	618,290
MDF	(binary)	gzip	3.06		
MDF	(binary)	bzip2	2.63		

The time performance of the compressed formats was not tested. The native feature formats are still greatly faster than GML.

11.1.3.3 Writing

The writing results for the **`elevp`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	62.63	4.913	35,799
GML	BXML	none	25.94	1.128	155,922
GML	XML	gzip	3.60	10.644	16,524
GML	BXML	gzip	3.23	6.098	28,842
GML	XML	bzip2	2.57	45.469	3,868
GML	BXML	bzip2	2.42	13.754	12,788

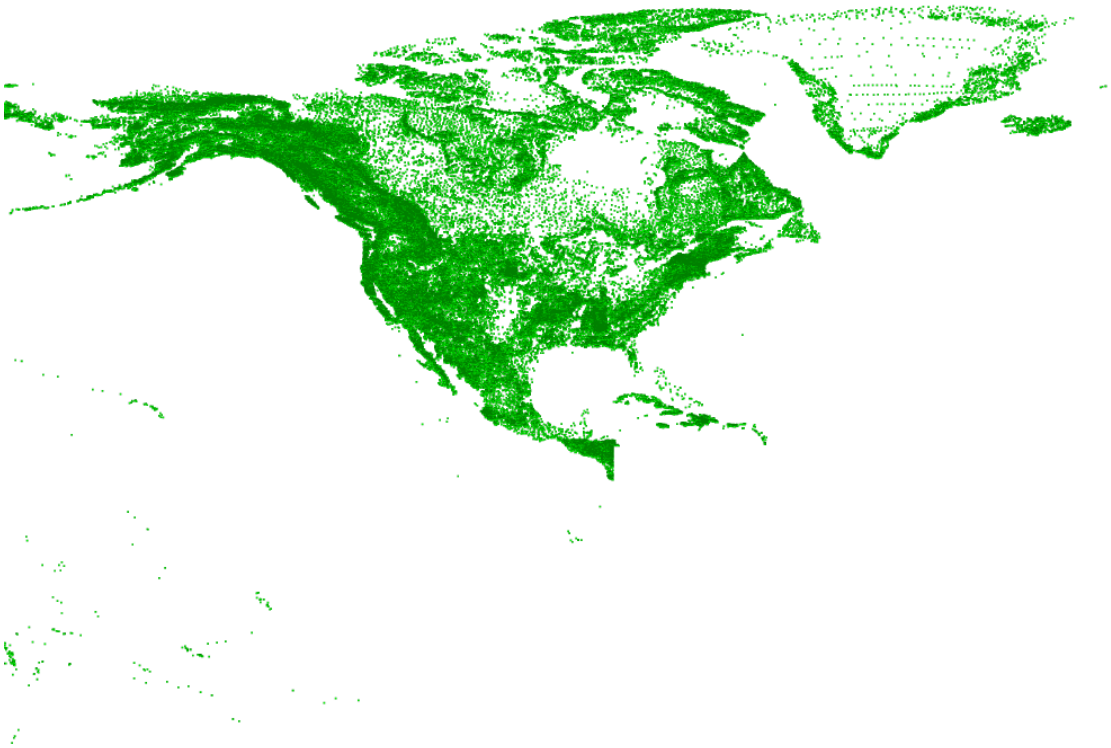
The reported timings for the writing speed are reduced by the 7.193 seconds taken to read the source **`elevp`** data.

11.1.3.4 Plotting

The plotting results for the **`elevp`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	62.63	7.034	25,004
GML	BXML	none	25.94	2.394	73,467

The plotting window and image size are the same as for the **`builtupa`** feature collection, except that the SLD plotting style uses little squares with dark-green stroke over a light-green fill:



The raw internal plotting speed is around 674,000 features per second.

11.1.4 Elevation points, double precision

11.1.4.1 Reading

The reading results for the **`elevp_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	67.89	7.067	24,888
GML	BXML	none	28.05	2.197	80,055
GML	XML	gzip	5.59	7.476	23,526
GML	BXML	gzip	4.64	2.420	72,678
GML	XML	bzip2	4.20	9.579	18,361
GML	BXML	bzip2	3.91	4.098	42,918

The performance is pretty close to the single-precision test case, though the file sizes here are significantly larger.

11.1.4.2 Writing

The writing results for the **`elevp_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	67.89	6.006	29,284
GML	BXML	none	28.05	1.894	92,861
GML	XML	gzip	5.59	12.215	14,399
GML	BXML	gzip	4.64	6.741	26,091
GML	XML	bzip2	4.20	43.871	4,009
GML	BXML	bzip2	3.91	10.262	17,139

The reported timings for the writing speed are reduced by the 7.067 seconds taken to read the source **`elevp_dbl`** data. The double-precision files are significantly larger than the single-precision ones and the processing time is significantly greater.

11.1.5 Water courses, single precision

11.1.5.1 Reading

The reading results for the **`watrcrsl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	146.82	13.053	22,258
GML	BXML	none	58.91	3.329	87,272
GML	XML	gzip	20.59	14.141	20,545
GML	BXML	gzip	19.64	4.049	71,753
GML	XML	bzip2	17.58	19.953	14,561
GML	BXML	bzip2	17.29	8.616	33,720

11.1.5.2 Reading alternate formats

The reading results for the **`watrcrsl`** feature collection represented in alternative formats to GML are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
Shapefile	(binary+text)	none	68.01	1.183	245,632
Shapefile	(binary+text)	gzip	28.18		
Shapefile	(binary+text)	bzip2	22.74		
MIF	(text)	none	61.68	5.658	51,347
MIF	(text)	gzip	18.38		
MIF	(text)	bzip2	16.31		
MDF	(binary)	none	63.07	0.586	495,425
MDF	(binary)	gzip	26.65		

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
MDF	(binary)	bzip2	20.09		

The speed of the compressed representations was not tested. The native feature formats are still greatly faster than the comparable GML encodings.

11.1.5.3 Writing

The writing results for the **`watrcrsl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	146.82	15.483	18,764
GML	BXML	none	58.91	1.901	152,829
GML	XML	gzip	20.59	35.216	8,250
GML	BXML	gzip	19.64	10.813	26,868
GML	XML	bzip2	17.58	87.510	3,320
GML	BXML	bzip2	17.29	24.540	11,839

The reported timings for the writing speed are reduced by the 13.053 seconds taken to read the source **`watrcrsl`** data.

11.1.5.4 Plotting

The plotting results for **`watrcrsl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Read+Plot*	*Tot. Time (s)*	*Speed (F/s)*
GML	XML	none	12.897	14.136	20,552
GML	BXML	none	3.960	5.200	55,871

The plotting window and image size are the same as for the **`builtupa`** feature collection, except that the SLD plotting style uses dark-blue strokes (**`#000080`**):

The raw internal plotting speed is around 460,000 features per second.

11.1.6 Water courses, double precision

11.1.6.1 Reading

The reading results for **`watrcrsl_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	196.62	13.660	21,269
GML	BXML	none	80.71	3.433	84,382
GML	XML	gzip	44.43	15.493	18,752
GML	BXML	gzip	37.18	4.438	65,464
GML	XML	bzip2	38.83	27.288	10,647
GML	BXML	bzip2	38.11	12.748	22,790

11.1.6.2 Writing

The writing results for the **`watrcrsl_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	196.62	23.217	12,514
GML	BXML	none	80.71	3.174	91,534
GML	XML	gzip	44.43	50.845	5,714
GML	BXML	gzip	37.18	13.291	21,859

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	bzip2	38.83	95.444	3,044
GML	BXML	bzip2	38.11	26.140	11,114

The reported timings for the writing speed are reduced by the 13.660 seconds taken to read the source **`watrcrsl_dbl`** data.

11.1.7 Contour lines

11.1.7.1 Reading

The reading results for the **`contourl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,005.92	77.809	14,135
GML	BXML	none	423.08	14.047	78,297
GML	XML	gzip	222.05	85.793	12,820
GML	BXML	gzip	218.26	20.072	54,795
GML	XML	bzip2	193.90	143.972	7,639
GML	BXML	bzip2	189.91	58.142	18,916

The GML-XML-uncompressed case likely includes file-cache-miss effects, since the file size is just short of the amount of physical memory on the machine, but the effect is most likely not significant, since the performance is consistent with the GML-XML-gzip case.

11.1.7.2 Writing

The writing results for the **`contourl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,005.92	150.485	7,309
GML	BXML	none	423.08	21.212	51,850
GML	XML	gzip	222.05	461.263	2,384
GML	BXML	gzip	218.26	72.882	15,091
GML	XML	bzip2	193.90	494.094	2,226
GML	BXML	bzip2	189.91	160.737	6,842

The reported timings for the writing speeds are reduced by the 77.809 seconds taken to read the source **`contourl`** data.

These results most likely include file-cache-miss effects. However, this effect appears to be limited, since if the testing tool is limited to processing only the first 200,000 features

(which will fit into the file-cache memory), the feature throughput is only about 4.5% faster.

11.1.7.3 Plotting

The plotting results for the **`contourl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Read+Plot*	*Tot. Time (s)*	*Speed (F/s)*
GML	XML	none	79.182	80.279	13,700
GML	BXML	none	18.374	19.255	57,120

The plotting window covers the whole world and the image is 3600 pixels across by 1800 (with a scaled-down version included here). The SLD plotting style uses brown strokes (**`#808000`**):

The raw internal plotting speed is around 254,000 features per second.

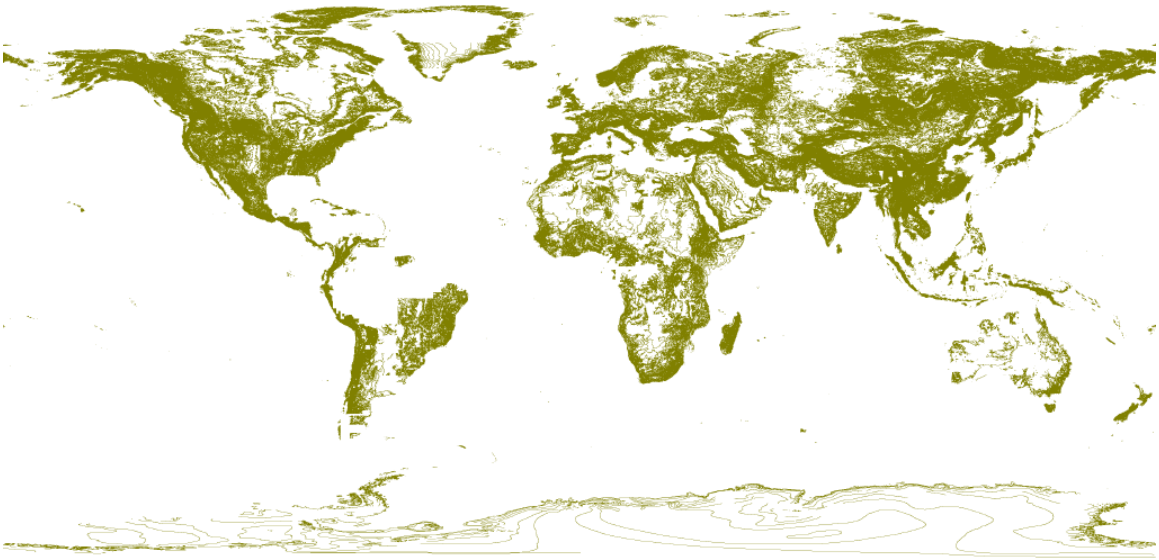
11.1.8 Conclusions

Uncompressed BXML provided the fastest reading, writing, and plotting performances for local file-system testing. This is the expected result. Averaging the single-precision test cases, BXML format files were only 40% as large as the uncompressed-XML files; they were read 4.0 times as fast for an average of 71,606 features/second; and they were written 8.2 times as fast for an average of 117,760 features/second.

Using compression in the local file system just slows down the performance. This would only be recommended to conserve space. The compressed file sizes between BXML and XML are quite similar, but the compressed BXML files could be read and written much faster than the compressed XML files.

Using double precision coordinate values produces files that are significantly larger and take significantly more time to process. This is as expected.

The BXML encoding of GML falls far short of the reading performance of Shapefile and CubeWerx MDF formats. Shapefile, the golden standard, can be read an average of 3.0



times as fast as the BXML-encoded GML format.

The plotting performance of the CubeWerx SLD renderer is astonishingly high.

11.2 LAN testing with high-performance simulated WFS

11.2.1 Built-up areas

The plotting results for the **`builtupa`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	6.17	1.590	5,249
GML	BXML	none	2.44	0.693	12,043
GML	XML	gzip	1.15	1.972	4,232
GML	BXML	gzip	1.07	0.768	10,867
GML	XML	bzip2	1.016	3.763	2,218
GML	BXML	bzip2	1.090	2.127	3,924

There are competing factors in the determination of the performance in this environment: the encoding, the compressing and decompressing, and the network bandwidth. The network in this case is fairly high in bandwidth and compression is fairly expensive. The client-side CPU utilization showed up as being between 8% and 43%, with the former for uncompressed data and the latter for bzip2-compressed data.

11.2.2 Inland water bodies

The plotting results for the **`inwatera`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	120.41	29.130	5,265
GML	BXML	none	47.44	11.157	13,745
GML	XML	gzip	20.81	39.650	3,868
GML	BXML	gzip	19.67	12.472	12,296
GML	XML	bzip2	18.61	64.052	2,394
GML	BXML	bzip2	19.17	25.739	5,958

11.2.3 Elevation points, single precision

The plotting results for the **`elevp`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	62.63	19.174	9,173

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	BXML	none	25.94	7.017	25,065
GML	XML	gzip	3.60	17.369	10,126
GML	BXML	gzip	3.23	8.617	20,411
GML	XML	bzip2	2.57	48.273	3,643
GML	BXML	bzip2	2.44	16.966	10,367

It is interesting that the XML+gzip case is faster than the XML-uncompressed case, as opposed to most other feature types where the uncompressed data is faster. In this case, the GML file contains a much greater amount of redundancy than the other feature types since the quasi-random coordinate data occupies a much smaller portion of the file and the XML tags a much greater portion. The file ends up being compressed to such a degree that the faster transfer speed improves the overall performance. Or, it could be that the GZIP compressor is disproportionately slow at compressing quasi-random numbers.

11.2.4 Elevation points, double precision

The plotting results for the **`elevp_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	67.89	19.574	8,985
GML	BXML	none	28.05	7.074	24,863
GML	XML	gzip	5.59	17.881	9,836
GML	BXML	gzip	4.64	9.542	18,432
GML	XML	bzip2	4.20	51.201	3,435
GML	BXML	bzip2	3.91	18.038	9,751

11.2.5 Water courses, single precision

The plotting results for the **`watrcrsl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	146.82	35.075	8,283
GML	BXML	none	58.91	12.089	24,032
GML	XML	gzip	20.59	39.050	7,440
GML	BXML	gzip	19.66	14.892	19,509
GML	XML	bzip2	17.58	92.006	3,158
GML	BXML	bzip2	18.08	33.215	8,747

11.2.6 Water courses, double precision

The plotting results of the **watrcrsl_dbl** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	196.62	39.754	7,308
GML	BXML	none	80.71	13.361	21,744
GML	XML	gzip	44.43	53.609	5,419
GML	BXML	gzip	37.18	16.282	17,843
GML	XML	bzip2	38.83	114.080	2,547
GML	BXML	bzip2	38.11	48.259	6,020

11.2.7 Contour lines

The plotting results for the **contourl** feature collection is as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,005.92	206.250	5,333
GML	BXML	none	423.08	62.596	17,570
GML	XML	gzip	222.05	401.249	2,741
GML	BXML	gzip	218.80	85.491	12,865
GML	XML	bzip2	193.89	541.616	2,030
GML	BXML	bzip2	198.18	220.583	4,986

11.2.8 Conclusions

Uncompressed BXML still gives the best performance in the LAN environment with the simulated WFS, performing an average of 2.8 times as fast as XML encoding over the seven test cases.

Gzip compression is much more competitive in this environment and was actually faster than uncompressed data in a couple of instances. Bzip2 compression is still too CPU-bound to be competitive in this environment.

11.3 LAN testing with relational-database WFS

LAN testing with a relational-database WFS is carried out using the CubeSERV WFS accessing feature from Oracle relational tables.

11.3.1 Built-up areas

The plotting results for the **builtupa** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	6.17	27.114	308
GML	BXML	none	2.31	25.436	328
GML	XML	gzip	1.16	26.814	311
GML	BXML	gzip	1.14	25.725	324
GML	XML	bzip2	1.03	27.675	302
GML	BXML	bzip2	1.13	25.974	321

These results are somewhat disappointing and show that accessing a relational database incurs a substantial time overhead. The close similarity of all the timings regardless of encoding format show that accessing this data from the database has a fixed cost of around 25 seconds relative to a few seconds of difference for processing the different encodings, rendering the encoding unimportant in this environment.

11.3.2 Inland water bodies

The plotting results for the **`inwatera`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	120.41	463.5	331
GML	BXML	none	46.03	469.0	327
GML	XML	gzip	21.30	490.5	313
GML	BXML	gzip	20.72	469.4	327
GML	XML	bzip2	19.05	504.1	304
GML	BXML	bzip2	19.64	471.9	325

11.3.3 Elevation points, single precision

The plotting results for the **`elevp`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	67.38	164.6	1,068
GML	BXML	none	28.10	163.9	1,073
GML	XML	gzip	5.53	182.3	965
GML	BXML	gzip	6.27	187.7	937
GML	XML	bzip2	4.50	204.4	861
GML	BXML	bzip2	4.78	181.5	969

The performance here is better than for other geometry types because points are simpler geometry types and they are stored in a simpler way in the database accessed by the WFS.

11.3.4 Elevation points, double precision

The plotting results for the **`elevp_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	72.64	164.9	1,067
GML	BXML	none	30.21	166.9	1,053
GML	XML	gzip	7.54	182.7	963
GML	BXML	gzip	7.03	186.6	943
GML	XML	bzip2	6.15	204.5	860
GML	BXML	bzip2	5.61	173.3	1,015

11.3.5 Water courses, single precision

The plotting results for the **`watrcrsl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	146.82	849.1	342
GML	BXML	none	54.71	839.1	346
GML	XML	gzip	21.22	894.7	325
GML	BXML	gzip	21.08	878.4	331
GML	XML	bzip2	18.04	910.1	319
GML	BXML	bzip2	18.86	875.4	332

11.3.6 Water courses, double precision

The plotting results for the **`watrcrsl_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	196.61	889.6	327
GML	BXML	none	76.51	848.4	342
GML	XML	gzip	44.83	910.9	319
GML	BXML	gzip	38.57	880.2	330
GML	XML	bzip2	39.30	941.5	309
GML	BXML	bzip2	39.95	894.0	325

11.3.7 Contour lines

The plotting results for the **`contourl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,005.92	3,384	325
GML	BXML	none	407.54	3,244	339
GML	XML	gzip	226.34	3,676	299
GML	BXML	gzip	223.72	3,365	327
GML	XML	bzip2	199.03	3,677	299
GML	BXML	bzip2	199.68	3,423	321

11.3.8 Conclusions

The GML encoding did not have a great effect in this testing. Retrieving the feature data from Oracle was the major bottleneck in this environment.

11.4 Internet testing with simulated high-performance WFS

11.4.1 Built-up areas

The plotting results for the **`builtupa`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	6.173	41.256	202
GML	BXML	none	2.435	16.338	510
GML	XML	gzip	1.151	7.857	1,062
GML	BXML	gzip	1.069	7.274	1,147
GML	XML	bzip2	1.015	7.589	1,099
GML	BXML	bzip2	1.090	7.828	1,066

There are competing factors in the determination of the performance in this environment: the encoding, the compressing and decompressing, and the network bandwidth. The network in this case is fairly low in bandwidth and compression is relatively inexpensive, so file size is the major determinate of performance.

One would need to be crazy not to compress GML data being sent over the Internet.

11.4.2 Inland water bodies

The plotting results for the **`inwatera`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	120.41	802.728	191
GML	BXML	none	47.44	316.314	485
GML	XML	gzip	20.81	138.957	1,104

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	BXML	gzip	19.67	131.248	1,168
GML	XML	bzip2	18.61	124.725	1,230
GML	BXML	bzip2	19.17	128.348	1,195

11.4.3 Elevation points, single precision

The plotting results for the **`elevp`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	62.63	417.583	421
GML	BXML	none	25.94	173.034	1,016
GML	XML	gzip	3.60	24.329	7,229
GML	BXML	gzip	3.23	21.810	8,064
GML	XML	bzip2	2.57	48.571	3,621
GML	BXML	bzip2	2.44	17.220	10,213

The XML+bzip2 case may seem anomalously slow, but this is what was reliably measured. The major factor would be the bulkiness of the data input to the expensive bzip2-compression algorithm.

11.4.4 Elevation points, double precision

The plotting results for the **`elevp_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	67.89	452.651	389
GML	BXML	none	28.05	187.104	940
GML	XML	gzip	5.59	37.557	4,683
GML	BXML	gzip	4.64	31.181	5,641
GML	XML	bzip2	4.20	51.197	3,435
GML	BXML	bzip2	3.91	26.564	6,621

11.4.5 Water courses, single precision

The plotting results for the **`watrcrsl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	146.82	978.816	297
GML	BXML	none	58.91	392.769	740
GML	XML	gzip	20.59	137.466	2,113

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	BXML	gzip	19.66	131.205	2,214
GML	XML	bzip2	17.58	117.729	2,468
GML	BXML	bzip2	18.08	121.026	2,401

11.4.6 Water courses, double precision

The plotting results for the **`watrcrsl_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	196.62	1,310.726	222
GML	BXML	none	80.71	538.089	540
GML	XML	gzip	44.43	296.354	980
GML	BXML	gzip	37.18	247.932	1,172
GML	XML	bzip2	38.83	259.576	1,119
GML	BXML	bzip2	38.11	254.603	1,141

11.4.7 Contour lines

The plotting results for the **`contourl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,005.92	6,705.465	164
GML	BXML	none	423.08	2,820.376	390
GML	XML	gzip	222.05	1,480.280	743
GML	BXML	gzip	218.80	1,458.621	754
GML	XML	bzip2	193.89	1,292.842	851
GML	BXML	bzip2	198.18	1,321.149	832

The simulated high-performance network is efficient enough to saturate the network in all cases except for using bzip2 compression, but it comes close even then.

11.4.8 Conclusions

Compression is the most important factor in this test environment. Since the compressed sizes of BXML and XML are quite similar, the performance was also quite similar. Using bzip2 compression frequently gave the best performance, though bzip2 applied to XML was substantially slower than the other encodings since BXML is slow to execute and XML gives the compression method a bulky input stream.

11.5 Internet testing with relational-database WFS

11.5.1 Built-up areas

The plotting results for the **builtupa** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	6.17	42.523	196
GML	BXML	none	2.31	24.880	335
GML	XML	gzip	1.16	26.855	311
GML	BXML	gzip	1.14	25.659	325
GML	XML	bzip2	1.03	31.207	267
GML	BXML	bzip2	1.13	31.749	263

The network was the bottleneck in the uncompressed-XML test case but retrieving the features from the database was the bottleneck in the rest of the cases. The throughput was about 145 kbytes/sec in the uncompressed-XML case but only 93 kbytes/sec in the uncompressed-BXML test case (significantly less than the network bandwidth). Applying compression increases the elapsed time since retrieving features from the database is already a CPU-intensive activity, and compression adds to the CPU-power bottleneck. The XML+bzip2 test case achieves a network utilization of only 33 kbytes/sec. The other feature types below follow a similar pattern.

11.5.2 Inland water bodies

The plotting results for the **inwatera** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	120.41	804.4	191
GML	BXML	none	46.03	448.8	342
GML	XML	gzip	21.30	489.6	313
GML	BXML	gzip	20.72	467.8	328
GML	XML	bzip2	19.05	566.5	271
GML	BXML	bzip2	19.64	567.1	270

11.5.3 Elevation points, single precision

The plotting results for the **elevp** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	67.38	449.9	391
GML	BXML	none	28.10	188.5	933

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	gzip	5.53	181.8	968
GML	BXML	gzip	6.27	186.6	943
GML	XML	bzip2	4.50	203.1	866
GML	BXML	bzip2	4.78	188.3	934

11.5.4 Elevation points, double precision

The plotting results for the **`elevp_dbl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	72.64	484.9	363
GML	BXML	none	30.21	202.1	870
GML	XML	gzip	7.54	184.3	954
GML	BXML	gzip	7.03	187.4	938
GML	XML	bzip2	6.15	208.4	844
GML	BXML	bzip2	5.61	194.1	906

11.5.5 Water courses, single precision

The plotting results for the **`watrcrsl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	146.82	983.2	295
GML	BXML	none	54.71	834.1	348
GML	XML	gzip	21.22	887.1	328
GML	BXML	gzip	21.08	870.2	334
GML	XML	bzip2	18.04	953.5	305
GML	BXML	bzip2	18.86	954.0	305

11.5.6 Water courses, double precision

The plotting results for the **`watrcrsl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	196.61	1,313.0	221
GML	BXML	none	76.51	846.1	343
GML	XML	gzip	44.83	912.4	318
GML	BXML	gzip	38.57	879.3	330
GML	XML	bzip2	39.30	1,094.2	266

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	BXML	bzip2	39.95	1,094.2	266

11.5.7 Contour lines

The plotting results for the **`contourl`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,005.91	6,714	164
GML	BXML	none	407.54	3,299	333
GML	XML	gzip	226.34	3,669	300
GML	BXML	gzip	223.72	3,376	326
GML	XML	bzip2	199.01	4,477	246
GML	BXML	bzip2	199.68	4,489	245

11.5.8 Conclusions

Using a relational database shows less overhead in the Internet environment, though full network utilization is not achieved for the non-uncompressed-XML test cases. The other encodings are compact enough that the CPU and database access on the server is the bottleneck.

11.6 Dial-up testing

This testing was carried out using the same simulated method as with the Internet testing. In this case, the link speed was set to 5,600 bytes/second to correspond to a 56-kbit/second low-speed link. No built-in compression was simulated over the link.

11.6.1 Measured results

The plotting results for the **`builtupa`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	6.172	1,102.382	7.57
GML	BXML	none	2.435	435.063	19.18
GML	XML	gzip	1.151	205.945	40.53
GML	BXML	gzip	1.069	191.193	43.65
GML	XML	bzip2	1.015	182.290	45.78
GML	BXML	bzip2	1.089	195.392	42.71

Transferring any volume of data over the link is obviously very slow and is completely dominated by the link speed. The CPU utilization on the client was shown as 0% to 2%.

Very limited plotting performance was measured for the **`inwatera`** feature collection:

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	120.41	21,499	7.13

This test case ran for approximately six hours.

11.6.2 Extrapolated results

The testing for the dial-up case is extremely time-consuming and the results have been demonstrated to be determined by the link speed, so presented here is an extrapolation of the testing of all seven feature collections being concatenated together (2,194,357 features):

Format	*Encoding*	*Compression*	*Size (MB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,606.46	286,868	7.65
GML	BXML	none	666.56	119,029	18.44
GML	XML	gzip	318.22	56,825	38.62
GML	BXML	gzip	304.22	54,325	40.39
GML	XML	bzip2	276.71	49,413	44.41
GML	BXML	bzip2	280.92	50,164	43.74

If all these test cases were executed once in practice, it would take over seven days.

11.6.3 Conclusions

Dial-up links are very slow, so the GML encoding with the greatest compression will have the highest throughput. In this case, bzip2 gives the best compression and therefore, the best throughput. The bzip2 file sizes are slightly smaller for XML than BXML, so XML+bzip2 gives the best throughput.

12 MSD3-data testing

It would not be feasible or greatly useful to perform manual testing on all of the individual feature types, so four instances were chosen for manual testing to represent the MSD3 data set: three feature collections of varying geometry type plus an aggregation of all the feature types. The aggregation was obtained simply by concatenating together all of the **`<gml:featureMember>`** elements of the supplied GML feature collections. The schema references were also changed to be locally available and offered up with the different encodings of the test instances (e.g., XML, BXML, compressed). The source data is all three-dimensional, but two-dimensional versions are also tested.

The following feature collections from the MSD3 data are used for testing:

Collection	*Description*	*Geom Type*	*Properties*	*Features*	*Vertices*
`MSD3`	MSD3 Aggregate	various	33.0	7,448	8.5

Collection	*Description*	*Geom Type*	*Properties*	*Features*	*Vertices*
AAL015	Building Areas	MultiSurface	45	857	7.2
LAP030	Roads	MultiCurve	20	1,444	5.5
PAL015	Building Points	MultiPoint	45	2,888	1.0

The "Properties" column gives the average number of properties per feature present in the **MSD3** collection, excluding the topology property (since it is not processed). The "Vertices" column is the average number of vertices per geometry for all feature collections.

Two versions of the MSD3 schema were available for testing: the full version and a version with all of the **<appinfo>** metadata trimmed out of it. The major reporting of results is for the trimmed version of the schema with additional comments about the performance with the full schema.

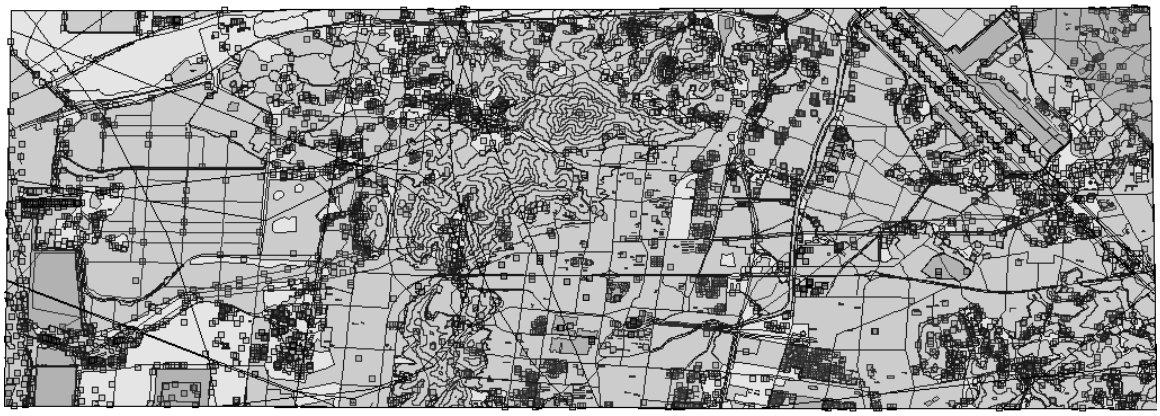
The testing of 3D coordinate values versus 2D coordinate values is reported only for the MSD3-aggregation feature collection. The performance difference between 2D and 3D is not that large and the aggregation case shows the difference sufficiently.

12.1 Local file-system testing

The GML and schema files were tested using local files.

12.1.1 MSD3 aggregation, 3D

The MSD3 aggregate feature collection has the following appearance when plotted with



translucent fills and no feature-type-specific styling:

The reading results for the **MSD3** feature collection using the trimmed MSD3 schema are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,511	1.621	4,594
GML	BXML	none	7,172	0.876	8,466
GML	XML	gzip	626	1.715	4,342
GML	BXML	gzip	929	0.926	8,046
GML	XML	bzip2	499	2.485	2,997
GML	BXML	bzip2	819	1.416	5,261

Parsing the schema takes a significant amount of the elapsed time. The schema is supplied with in the same encoding and compression format as the GML data. The parsing time for the trimmed and full MSD3 schemas are as follows, with schema sizes in KB and parsing times in seconds:

Format	*Encoding*	*Compress*	*Trim Size*	*Trim Time*	*Full Size*	*Full Time*
Schema	XML	none	1,337	0.318	8,056	0.815
Schema	BXML	none	581	0.238	3,696	0.433
Schema	XML	gzip	28	0.329	446	0.873
Schema	BXML	gzip	21	0.241	307	0.460
Schema	XML	bzip2	14	0.411	204	1.442
Schema	BXML	bzip2	12	0.273	190	0.715

The writing results for the **`MSD3`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,511	1.030	7,231
GML	BXML	none	7,172	0.335	22,226
GML	XML	gzip	626	1.943	3,833
GML	BXML	gzip	929	1.030	7,234
GML	XML	bzip2	499	11.656	639
GML	BXML	bzip2	819	6.458	1,153

The source-data reading time of 1.621 seconds is subtracted from the write timings above. Note that no new schema file is written out here since the generated GML data refers to a preexisting schema file.

The XML encodings of the compressed data are significantly smaller than the BXML versions. This is likely because BXML format compacts the data with the primary goal being for speed of processing whereas GZIP and BZIP2 formats compact data with the primary goal being minimization of size.

12.1.2 MSD3 aggregation, 2D

The reading results for the **MSD3_2D** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,241	1.577	4,735
GML	BXML	none	6,645	0.845	8,815
GML	XML	gzip	562	1.664	4,477
GML	BXML	gzip	831	0.906	8,224
GML	XML	bzip2	447	2.413	3,087
GML	BXML	bzip2	725	1.356	5,494

There does not appear to be a great performance advantage in using 2D coordinates over 3D ones. This is presumably because the extra coordinate dimension takes a relatively small amount of space compared to all of the attributes. The difference would be larger if the geometries in the MSD3 data included significantly more vertices.

The writing results for the **MSD3_2D** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,241	0.921	8,084
GML	BXML	none	6,645	0.325	22,925
GML	XML	gzip	562	1.808	4,119
GML	BXML	gzip	831	0.812	9,171
GML	XML	bzip2	447	11.631	640
GML	BXML	bzip2	725	6.386	1,166

The writing time excludes the 1.577 seconds required to read the source data to be written.

12.1.3 AAL015

The AAL015 feature collection (Building Areas) has the following appearance:

The reading results for the **AAL015** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,984.1	0.570	1,504
GML	BXML	none	999.5	0.383	2,237
GML	XML	gzip	63.9	0.589	1,454
GML	BXML	gzip	86.3	0.397	2,158
GML	XML	bzip2	48.1	0.781	1,097
GML	BXML	bzip2	82.1	0.502	1,707

As discussed with the **MSD3**-aggregate case, parsing the schema takes a substantial amount of the elapsed time. However, since this feature collection is much smaller and the schema-parsing time is fixed for each different encoding, schema parsing now takes the majority of the elapsed time.

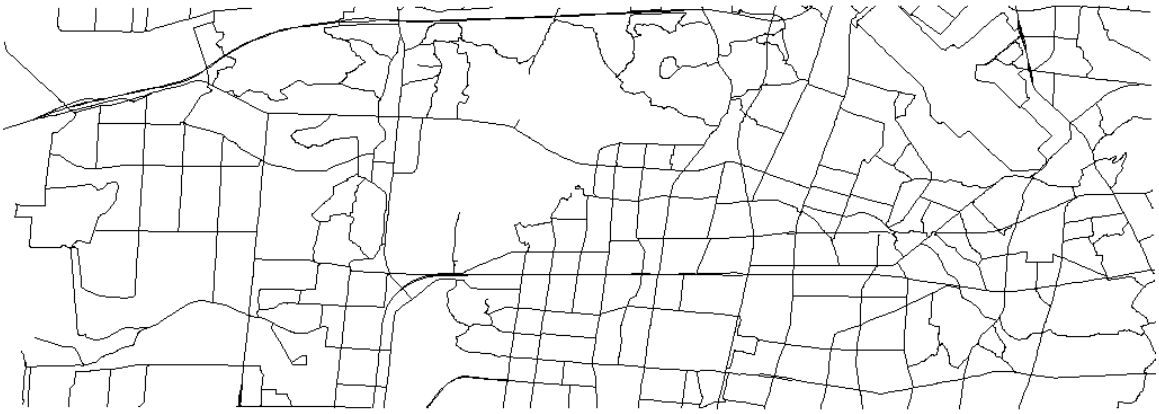
The writing results for the **AAL015** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,984.1	0.130	6,613
GML	BXML	none	999.5	0.054	15,730
GML	XML	gzip	63.9	0.234	3,655
GML	BXML	gzip	86.3	0.118	7,269
GML	XML	bzip2	48.1	1.770	484
GML	BXML	bzip2	82.1	0.950	902

The 0.570 seconds taken to read the source data is removed from the writing times.

12.1.4 LAP030

The LAP030 feature collection has the following appearance:



The reading results for the **LAP030** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,629	0.527	2,742
GML	BXML	none	754	0.383	3,770
GML	XML	gzip	102	0.589	2,451
GML	BXML	gzip	140	0.397	3,637
GML	XML	bzip2	74	0.781	1,849
GML	BXML	bzip2	117	0.502	2,877

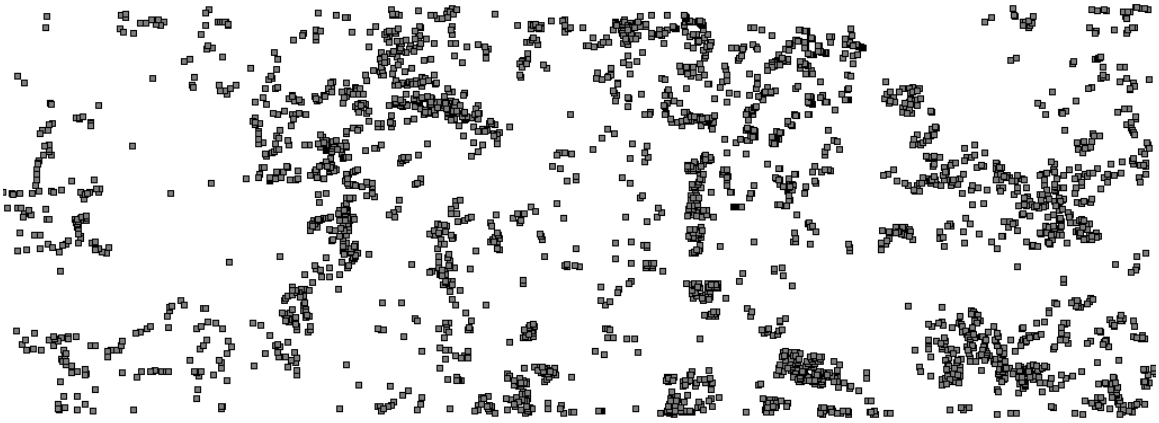
The writing results for the **LAP030** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,629	0.173	8,367
GML	BXML	none	754	0.098	14,742
GML	XML	gzip	102	0.277	5,205
GML	BXML	gzip	140	0.161	8,974
GML	XML	bzip2	74	1.813	796
GML	BXML	bzip2	117	0.993	1,454

The writing times exclude the 0.527 seconds taken to read the source data.

12.1.5 PAL015

The PAL015 feature collection has the following appearance (with the points rendered as small squares):



The reading results for the **PAL015** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	5,580.4	0.931	3,102
GML	BXML	none	2,710.9	0.566	5,103
GML	XML	gzip	70.7	0.976	2,959
GML	BXML	gzip	71.9	0.588	4,911
GML	XML	bzip2	41.9	1.309	2,207
GML	BXML	bzip2	54.0	0.733	3,735

The writing results for the **PAL015** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	5,580.4	0.292	9,896
GML	BXML	none	2,710.9	0.173	16,737
GML	XML	gzip	70.7	0.537	5,375
GML	BXML	gzip	71.9	0.285	10,136
GML	XML	bzip2	41.9	5.367	538
GML	BXML	bzip2	54.0	3.184	907

The writing times exclude the 0.931 seconds used to read the source data.

12.1.6 Conclusions

Uncompressed BXML encoding gives the best performance in the local-file-system test case. Since the MSD3 3D case actually includes the data for the other test cases, it is the most representative one. The BXML format was 1.85 times as fast as XML and was only 53% as large for using the trimmed schema.

A large performance factor with this data is the schema, since it is so large relative to the data. Two versions were tested, a trimmed version and the full version which includes a great deal of metadata. Both versions include definitions for hundreds of feature types that are not present in the test data set.

The BXML version of the trimmed schema is 43% as large as the XML version and can be processed 1.34 times as fast. The BXML version of the full schema is 46% as large as the XML version and can be processed 1.88 times as fast. Parsing the trimmed schema takes 27% of the BXML execution time.

Compression gives poor performance in this environment because the compression algorithms are slow to execute. However, if storage size is the crucial restriction, then XML+bzip2 gives the best compression, though it is much slower to execute than gzip compression.

The BXML compressed files are substantially larger than the XML compressed file. The reason for this is not known.

Processing 2D data versus 3D data did not make a substantial difference in either the processing time or the file size. This is likely because the features have relatively few vertices per geometry and have a large number of properties, so the extra coordinate in the 3D data does not take a substantial amount of extra space.

12.2 LAN testing

The BXML format includes a central symbol table and a mechanism to make any text content reference a string in this table. The advantage is that a literal string value need appear in the BXML file only once and it can be referenced by a compact index value thereafter. The option is enabled in the BXML generator for the MSD3 network cases to make all generated content strings use this central symbol table for greater compactness.

The reason this option was not enabled in the file-system testing is that there appears to be a performance bug in the CWXML library that makes processing referenced strings a little slower than literal strings. However, in the network environment, the increased compactness of the volume of data sent over the network provides better performance than using the literal strings.

12.2.1 MSD3 aggregation, 3D

The plotting results for the **`MSD3`** feature collection with the trimmed schema are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,512	4.855	1,534
GML	BXML	none	3,753	2.201	3,385
GML	XML	gzip	627	4.098	1,817

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	BXML	gzip	910	2.263	3,291
GML	XML	bzip2	499	13.055	571
GML	BXML	bzip2	776	4.391	1,696

Schema parsing takes a significant amount of the processing time (sizes in KB, times in seconds):

Format	*Encoding*	*Compress*	*Trim Size*	*Trim Time*	*Full Size*	*Full Time*
Schema	XML	none	1,336.7	0.706	8,056	2.003
Schema	BXML	none	350.2	0.445	2,263	0.796
Schema	XML	gzip	27.7	0.615	446	1.654
Schema	BXML	gzip	26.0	0.425	235	0.700
Schema	XML	bzip2	14.1	0.831	204	2.984
Schema	BXML	bzip2	16.9	0.463	160	1.037

12.2.2 MSD3 aggregation, 2D

The plotting results for the **`MSD3_2D`** feature collection with the trimmed schema are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,241	4.653	1,601
GML	BXML	none	3,225	2.148	3,467
GML	XML	gzip	563	3.974	1,874
GML	BXML	gzip	819	2.063	3,611
GML	XML	bzip2	447	12.977	574
GML	BXML	bzip2	729	4.262	1,748

Again, the 2D case is not greatly more efficient than the 3D case. The third-dimension coordinate values occupy a relatively small portion of the overall file sizes.

12.2.3 AAL015

The plotting results for the **`AAL015`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,984.2	1.640	4,542
GML	BXML	none	437.6	1.029	7,239
GML	XML	gzip	64.1	1.571	4,740
GML	BXML	gzip	84.5	1.032	7,218

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	bzip2	48.1	3.775	1,973
GML	BXML	bzip2	82.0	1.429	5,212

12.2.4 LAP030

The plotting results for the **`LAP030`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,692.2	1.532	4,861
GML	BXML	none	424.6	1.010	7,374
GML	XML	gzip	101.7	1.453	5,126
GML	BXML	gzip	136.5	1.004	7,415
GML	XML	bzip2	74.2	3.254	2,289
GML	BXML	bzip2	120.2	1.375	5,416

12.2.5 PAL015

The plotting results for the **`PAL015`** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	5,580.6	2.672	2,787
GML	BXML	none	973.9	1.326	5,618
GML	XML	gzip	70.8	2.634	2,827
GML	BXML	gzip	67.9	1.418	5,251
GML	XML	bzip2	42.0	6.508	1,144
GML	BXML	bzip2	55.8	2.556	2,914

12.2.6 External codec

An external codec (encoder-decoder, external to the main GML-consuming program) was added to the LAN testing script by adding a Unix shell statement of the form:

`xmlscan -url` *data_url* **`-pack | cwplot -f -`** *parameters*

This executes the **`xmlscan`** program which reads the GML data from the server as a separate process from the **`cwplot`** program which plots the data. The **`xmlscan`** program reads the data in whatever format is requested from the server but always delivers it to the **`cwplot`** program as uncompressed XML, so this arrangement operates as an "external codec" for delivering GML data to a client application that does not understand BXML or the compression formats. A caveat is that it would be very awkward to intercept the retrieval of the schemas, so only the GML data is subjected to the external codec, not the schema access. Also, this external codec is implemented only

on the client side; the external codec could be extended onto the server side as discussed in clause 8.4 which would impose greater costs and complications for recognizing and translating coordinate values and other numbers.

This arrangement is functionality equivalent to the direct **`MSD3`** test, except the external codec will create more execution overhead on the client from managing multiple processes and in interprocess pipe and from executing an extra decoding and encoding step. Additionally, the extra encoding step will be for regular XML and the decoding step inside of the **`cwplot`** program will always be for regular XML-encoded GML.

The plotting results for the **`MSD3`** 3D feature collection using the external codec are as follows (sizes in KB, times in seconds, speeds in features/second):

Format	*Encoding*	*Compress*	*Size*	*Time*	*Speed*	*Lost Time*	*Lost Speed*
GML	XML	none	13,512	6.777	1,099	1.922	435
GML	BXML	none	3,753	5.928	1,257	3.727	2,128
GML	XML	gzip	627	6.091	1,223	1.999	594
GML	BXML	gzip	910	5.708	1,305	3.445	1,986
GML	XML	bzip2	499	13.316	559	0.261	12
GML	BXML	bzip2	776	8.071	923	3.680	773

The results show that the use of an external codec imposes significant throughput costs in the Internet environment and the benefits of using BXML in this environment are largely nullified. The lost performance would be significantly larger if the schema fetching was also run through an external codec.

Note that the XML-none figure does not really belong here since the main application in this arrangement can consume regular XML directly, so an external codec is not really needed in this case.

12.2.7 Conclusions

The uncompressed BXML encoding gives the best performance in the LAN test case, same as with the VMAP0 data. The BXML format was 2.2 times as fast as XML and was only 28% as large for using the trimmed schema. Using the symbol-table BXML mechanism makes the BXML files substantially smaller here than in the file-system testing where it was turned off.

Schema processing is a large factor with the MSD3 data. The BXML verison of the trimmed schema is 26% as large as the XML version and can be processed 1.59 times as fast. The BXML version of the full schema is 28% as large as the XML version and can be processed 2.5 times as fast. Parsing the trimmed schema takes 20% of the BXML execution time. The schemas compress extremely well.

Compression performs better in this environment than in the file-system environment because there is a trade-off between processing time taken to compress the data and the effectively increased network throughput of the compressed data. However, the LAN is fast enough that the processing time is more costly. The BXML compressed data files are substantially larger than the XML compressed file. The reason for this is not known.

Using an external codec adds a substantial overhead in this environment and tends to even out the timings because of the large fixed overhead of the fixed XML-GML generate/parse steps.

12.3 Internet testing

12.3.1 MSD3 aggregation, 3D

The plotting results for the **`MSD3`** feature collection with the trimmed schema are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,512	99.002	75
GML	BXML	none	3,753	27.400	272
GML	XML	gzip	627	6.042	1,233
GML	BXML	gzip	910	6.910	1,078
GML	XML	bzip2	499	14.001	532
GML	BXML	bzip2	776	7.448	1,000

Schema parsing takes a significant amount of the processing time (sizes in KB, times in seconds):

Format	*Encoding*	*Compress*	*Trim Size*	*Trim Time*	*Full Size*	*Full Time*
Schema	XML	none	1,336.7	9.335	8,056	54.222
Schema	BXML	none	350.2	2.762	2,263	15.573
Schema	XML	gzip	27.7	0.732	446	3.618
Schema	BXML	gzip	26.0	0.621	235	2.084
Schema	XML	bzip2	14.1	1.013	204	3.318
Schema	BXML	bzip2	16.9	0.649	160	1.663

12.3.2 MSD3 aggregation, 2D

The plotting results for the **`MSD3_2D`** feature collection with the trimmed schema are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,241	97.146	77

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	BXML	none	3,225	23.781	313
GML	XML	gzip	563	5.282	1,410
GML	BXML	gzip	819	6.320	1,179
GML	XML	bzip2	447	13.897	536
GML	BXML	bzip2	729	6.889	1,081

12.3.3 AAL015

The plotting results for the **AAL015** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,984	22.119	39
GML	BXML	none	438	5.216	164
GML	XML	gzip	64	1.982	432
GML	BXML	gzip	85	1.458	588
GML	XML	bzip2	48	4.017	213
GML	BXML	bzip2	82	2.194	391

12.3.4 LAP030

The plotting results for the **LAP030** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,692	20.161	72
GML	BXML	none	425	5.133	281
GML	XML	gzip	102	1.840	785
GML	BXML	gzip	136	1.376	1,056
GML	XML	bzip2	74	3.476	415
GML	BXML	bzip2	120	2.357	613

12.3.5 PAL015

The plotting results for the **PAL015** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	5,581	46.100	63
GML	BXML	none	974	8.802	328
GML	XML	gzip	71	3.065	942
GML	BXML	gzip	68	2.088	1,383

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	bzip2	42	6.770	427
GML	BXML	bzip2	56	3.095	933

12.3.6 External codec

An external codec was added to the Internet testing script using the method described in clause 12.2.6 for the GML data but not the schema. The plotting results for the **MSD3** 3D data with the external data codec are as follows (sizes in KB, times in seconds, speeds in features/second):

Format	*Encoding*	*Compress*	*Size*	*Time*	*Speed*	*Lost Time*	*Lost Speed*
GML	XML	none	13,512	98.554	76	-0.448	-1
GML	BXML	none	3,753	27.216	274	-0.184	-2
GML	XML	gzip	627	7.245	1,028	1.203	205
GML	BXML	gzip	910	8.250	903	1.340	175
GML	XML	bzip2	499	14.027	531	0.026	1
GML	BXML	bzip2	776	9.814	759	2.366	241

The results here are quite different from the LAN test case. The primary difference is that in this case, the client CPU is largely idle in the direct-connection case, so the resources are readily available to execute the external codec and while keeping up with the server. There is no obvious explanation for why the uncompressed cases actually experience performance improvements, though there may be some subtle effects in the network-simulation implementation.

The cases where the performance is significantly worse is probably related to the fixed overhead of executing the decode-generate-decode steps on the client. The break-even point seems to be around the 14-second period of the XML-bzip2 case, though in this case, the server would be very busy executing the expensive bzip2-compression algorithm. Indeed, in this case, the server can only generate 37 kbytes/sec over the simulated 150-kbyte/sec link. The server is nearly able to saturate the link in the gzip test cases, and so would deliver the maximum concentration of features per second. In the uncompressed cases, the network is the bottleneck, so the number of features per second that the client needs to process is restricted.

12.3.7 Conclusions

Compression is crucial to performance in the Internet environment because of the limited network bandwidth. However, while bzip2 compression gives the greatest compactness, gzip gives the better throughput.

The XML+gzip case gives the best throughput in this environment. The portion time taken parsing the schema is substantially lower in this environment than in the LAN environment because the schemas compress incredibly well.

Using an external codec adds greatly less overhead than in the LAN environment because the client machine has much more idle time while waiting for data to come from the network that it can spend on the fixed cost of the fixed XML-GML generate/parse steps.

12.4 Dial-up testing

Dial-up testing is carried out using a simulated 5.6-kbyte/sec uncompressed link.

12.4.1 MSD3 aggretation, 3D

The plotting results for the **MSD3** feature collection with the trimmed schema are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,512	2,627.3	2.8
GML	BXML	none	3,753	707.7	10.5
GML	XML	gzip	627	112.5	66.2
GML	BXML	gzip	910	163.7	45.5
GML	XML	bzip2	499	90.6	82.2
GML	BXML	bzip2	776	140.1	53.2

Schema parsing takes a significant amount of the processing time (sizes in KB, times in seconds):

Format	*Encoding*	*Compress*	*Trim Size*	*Trim Time*	*Full Size*	*Full Time*
Schema	XML	none	1,336.7	239.27	8,056	1,440.63
Schema	BXML	none	350.2	63.11	2,263	405.30
Schema	XML	gzip	27.7	5.56	446	80.74
Schema	BXML	gzip	26.0	5.23	235	42.72
Schema	XML	bzip2	14.1	3.24	204	37.71
Schema	BXML	bzip2	16.9	3.70	160	29.34

12.4.2 MSD3 aggregation, 2D

The plotting results for the **MSD3_2D** feature collection with the trimmed schema are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	13,241	2,578.9	2.9
GML	BXML	none	3,225	614.1	12.1
GML	XML	gzip	563	102.2	72.9
GML	BXML	gzip	819	147.8	50.4

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	bzip2	447	94.9	78.5
GML	BXML	bzip2	729	134.9	55.2

12.4.3 AAL015

The plotting results for the **AAL015** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,984	577.3	1.5
GML	BXML	none	438	116.0	7.4
GML	XML	gzip	64	12.7	67.3
GML	BXML	gzip	85	15.7	54.7
GML	XML	bzip2	48	11.7	73.4
GML	BXML	bzip2	82	20.0	42.8

12.4.4 LAP030

The plotting results for the **LAP030** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	1,692	517.3	2.8
GML	BXML	none	425	113.7	12.7
GML	XML	gzip	102	19.3	74.8
GML	BXML	gzip	136	25.0	57.8
GML	XML	bzip2	74	15.3	94.5
GML	BXML	bzip2	120	26.9	53.7

12.4.5 PAL015

The plotting results for the **PAL015** feature collection are as follows:

Format	*Encoding*	*Compression*	*Size (KB)*	*Time (s)*	*Speed (F/s)*
GML	XML	none	5,581	1,212.0	2.4
GML	BXML	none	974	211.6	13.6
GML	XML	gzip	71	13.8	209.3
GML	BXML	gzip	68	13.0	222.2
GML	XML	bzip2	42	19.1	151.2
GML	BXML	bzip2	56	17.3	167.3

12.4.6 Conclusions

With a dial-up link, compression is crucial to performance. Whatever encoding method produces the fewest bytes to represent a feature collection will be the fastest, pretty much regardless of how much processing the encoding method takes. In the case of the MSD3 data, the best encoding method for dial-up links is XML+bzip2.

13 GML issues

In the course of carrying out this performance study, some problem areas with using GML were identified.

13.1 The trouble with application schemas

There are two general approaches to handling the schemas associated with GML data. One way is to auto-generate a custom-tailored schema for each GML file that is generated from the information available from the source feature format, as was done for the VMAP0 test data set, and the other way is to use an external, centralized schema, as was done for the MSD3 test data set.

Using a custom-tailored schema is the approach taken with most other feature formats, with the schema information normally embedded within the data file(s). A vulnerability with this approach is the fidelity with which the schema information is carried from one representation to another. Sometimes, the sizes of properties or letter case of names are changed, and most feature formats have tricky idiosyncrasies. For example, in Shapefile format, numeric types are actually stored as strings, and a numeric type with width=9 and scale=3 can actually store any number that can be represented in nine characters. Some tools even write numbers in scientific notation.

Using the centralized-schema approach has problems also. One major problem is that the centralized schema may include a great number of feature-type definitions and a great deal of metadata that is not relevant to the purpose at hand. For example, the MSD3 central schema includes definitions for 443 different feature types even though the sponsor-supplied data includes only 93 feature types and 83% of the bulk of the full schema contains metadata which is not relevant to the performance-testing activities of the OWS-3 project. The schema is so large that parsing it takes a significant portion of the time to read the MSD3 data and it even takes the majority of the time for smaller feature collections.

The other major problem with centralized application schemas is that it is easy to include arbitrary formatting declarations that general-purpose GML client and server applications cannot understand or follow. This problem is present with custom-generated schemas also, though the arbitrariness is limited in practice to the types of declarations that are portable between common representation formats. For instance, if the data is held in a relational-database format using simple types (plus a geometry), then the generated schema will reflect this simplicity.

Handling arbitrary schema declarations may be an unsolvable computing problem in the general case and is at least a complex artificial-intelligence problem since in order for a computer to transform data from one representation into an arbitrary new one, it needs to "understand" the semantics of the information, and arbitrary semantics probably are not representable in a declarative way.

Schema-processing in the centralized-schema approach is particularly onerous because the schema *theoretically* needs to be parsed and understood in order to even write the GML data correctly, whereas with the custom-generated schema approach the GML writer is easily and statically preprogrammed to generate its output GML in accordance with the schema that it generates. In practice, however, vendors will not implement a capability to "understand" the schema of the GML data they are writing; they will instead simply implement application-specific hacks into their GML generators to handle any arbitrariness in the application schemas.

It is also possible to implement any arbitrary transformations needed for generated data using a hand-made application-specific transformation script as a post-processing step and to do the reverse in the same way on the client side. However, beyond the manual effort involved in creating and managing the transformation programs to service all of the different feature types in a data set, portable transformation programs are generally written in XSLT, which, given the way that XSLT operates, is likely to be inefficient and may not work at all for large feature sets like the Contour Lines of the VMAP0 test data since XSTL implementations tend to store the entire XML stream in memory, which also imposes streamability limitations. It is theoretically possible to optimize XSLT to behave in a streamable fashion where the transformation to be carried out permits, but this optimization is not yet available in real-world XSLT implementations, but even if it is available, the XML stream still needs to be parsed and generated an additional time.

However, there is a fairly easy way around the problems of dealing with arbitrary XML schemas, which is to use simplified canonical profiles of GML and XML-Schema, such as the GML Simple-Features Profile [GML-SF]. If both the application schema and the GML generator conform to the profile (or specific compliance level of future versions of the profile), and if the source format for the generator retains the typing information of the schema with the full fidelity of the schema profile, then the generated GML will always conform to the central application schema.

13.2 GML MeasureType

A practical problem with **`gml:MeasureType`** is that the attributes take a fairly large amount of space in the GML data files, whereas the measurement unit for individual properties all appear to be constant in the MSD3 data. It would be much more space-efficient if the if the fixed unit associated with each measurement property in the MSD3 data could be specified once in the schema definition or metadata rather than being repeated in-line with every property instance.

The presence of the attribute on every property instance also imposes the need on consumers to re-map and retain this information for every measurement property instance

when the data is transformed into another feature format in order to maintain the fidelity of the data, since the consumer will not know in advance (or really, ever) that the unit is actually fixed. This information may be awkward to re-map and will be time- and space-consuming in other feature representations. This author is not aware of any other format that includes a variable-unit mechanism.

13.3 GML streamability

A format is "streamable" if it can be generated, transfered, and consumed without any undue storage between the end points. Streamability is desirable because it minimizes processing delays and storage costs throughout the system. However, there are a few features of GML that interfere with its streamability.

13.3.1 Mid-stream errors

A GML feature collection can include features from numerous different feature types and the WFS interface allows requests to be made for multiple feature types and this can cause streamability issues on the server side since many systems implement the storage of features of different types using a separate internal feature collection for each different type. Also, all OWS requests require the response to be either a valid document of the requested format or to be a specially coded exception report.

The way that a query that includes multiple feature types is normally implemented is to process the feature-type queries in order and generate the GML output. But this poses a big problem if any errors are encountered in the intermediate steps, since if any errors are encountered at any point during the GML-generation process, an exception report must be generated instead of a GML document. GML includes no mechanism for reporting an error after the generation of the stream has begun.

In order for a generating application to be (mostly) safe is to generate all of the output data to a temporary file and then copy this file to the network, but then the generation time is wasted. This problem is compounded in the Internet environment because the network is usually the bottleneck and it will be idle while the temporary file is generated.

Really, the only sane approach for WFS implementers is to cheat in some way.

13.3.2 Bounding envelope

The requirement that the bounding envelope be stated at the beginning of a feature stream is desirable for clients, but it imposes restrictions on streamability for servers. The only way to be sure that the tightest envelope is stated in the general case is to pre-scan all of the features according to all of the query constraints.

13.3.3 Feature interleaving

Another streamability issue is that features in a GML feature collection may appear in any order and be mixed together. This poses no problems for GML generators since they can choose any order which is convenience, but it does pose problems for GML

consumers since many client applications cannot process a sequence of more than one feature type at a time. If only one feature type is in a stream, a clever client with the one-feature-type-at-a-time constraint can process it in one pass, but this is not possible if more than one feature type is present in the stream. The client would need to either store or re-retrieve the stream when it is ready to process the next feature type.

GML should include some mechanism to indicate whether or not the features of a type are contiguous and/or appear in the order of the feature types in the schema. This would allow simple applications to process a GML stream one feature-type at a time in one pass.

14 Conclusions

14.1 Local file-system testing

Testing in the local file system showed that binary encoding using BXML was around four times as fast for reading and eight times as fast for writing GML data compared to XML for the VMAP0 test data. The performance figures for the MSD3 data showed a lower overall improvement because the parsing of the bulky application schema is conflated with the data encode/decode speeds. When factored out, the BXML reading is about twice as fast and the writing is about three times as fast. The MSD3 data has many more properties than the VMAP0 data.

We also need to be cognizant in general that the CWXML parser is extremely well optimized for scanning XML and BXML realized in other environments may be even more efficient than comparable XML parsers since the BXML format is quite easy to process efficiently.

The BXML files are around 40% the size of the XML files and can be smaller still if space-saving options are enabled, and there are design tweaks that can be applied to future versions of BXML to increase compactness. When compression is applied, the BXML files are frequently around the same size as the XML files, though sometimes the BXML files can be significantly larger, as with the MSD3 data files. The XML encoding of MSD3 has the advantage that the source data is rounded to ten significant decimal digits, but these are not round values when represented in binary, which reduces the compressibility of the BXML representation. The compressed BXML files are still substantially faster to process than the compressed XML files, especially with the relatively efficient GZIP method.

Using double-precision coordinate values with the VMAP0 data produced files that were significantly larger and slower to process than using single-precision coordinates. However, using 2D coordinate values instead of 3D coordinate values with the MSD3 data did not make a significant difference in size or processing speed, likely because the coordinate values occupy a relatively small portion of the overall MSD3-data size.

The BXML encoding of GML falls far short of the performance available with other feature-file formats. Shapefile can be read about three times faster than BXML-encoded GML.

14.2 LAN testing

The speed and size savings of BXML translate well to the LAN environment. The uncompressed BXML encoding is still the most efficient, performing 2.8 times as fast with transfers of the VMAP0 data and 2.2 times as fast with the MSD3 data.

Utilizing an relational database system as the feature storage system imposes an unexpected high burden and is the major bottleneck in the LAN environment. The overhead is so high that the effects of the different encodings tend to be largely muted, though BXML still technically performs the best. CubeWerx will be investigating optimizations. The use of a dual-processor server would like improve throughput significantly, since the Oracle activity during feature retrieval seems to be split between two processes which compete for CPU time. (The server in the test environment has only one processor.)

Using an external codec with an efficient GML generator imposes substantial costs in this environment because of the fixed cost of the additional XML-GML parse/generate step(s).

14.3 Internet testing

In the Internet environment, compression is very important because of the limited bandwidth, 150 Kbytes/sec in the test case. Since the compressed sizes of BXML and XML data are quite similar, their performance was also quote similar. While BZIP2 gave the greatest amount of compression, GZIP provides the greatest throughput and GZIP is a ubiquitous format.

The sponsors of this project, including NGA, should insist that all of their software suppliers equip their GML-processing clients and servers with HTTP-based GZIP-compression support. Using the HTTP mechanism allows support to be optional for both client and server while providing compression when both client and server support it. GZIP is suitable for compressing streamed data on-the-fly.

Using an relational database system to server the features slowed down the transfer significantly, though not to the degree experienced in the LAN environment. Using either BXML encoding or compression gives similar throughput since the server in this case cannot saturate the network link except in the XML case, where the network becomes the bottleneck instead of the database system.

Using an external codec with an efficient GML generator degrades performance, though not to the same degree as in the LAN case since the network is now the bottleneck and the client has more otherwise-idle CPU time with which to execute the external codec process.

14.4 Dial-up testing

In the dial-up network case, compression is crucial. BZIP2-compressed XML generally gives the best performance in this environment since it generally gives the best compression.

Summary of key GML-testing variables

(Informative)

Encoding: XML, BXML, [BinXML™ tested by Galdos].

Compression: none, GZIP, BZIP2.

Precision: 16 digits or 7 digits.

Network: none (local filesystem), LAN, Internet, dial-up.

Scalability: hundreds, tens-of-thousands, or 1-million features.

Bibliography

[BASE64] IETF RFC 1521 (September 1993), *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, N. Borenstein, et al., <http://www.ietf.org/rfc/rfc1952.txt>.

[BINXML] Expway, BinXML™ binary-XML encoding system.

[BXML] OGC 03-002r8 (May 2003), *Binary-XML Encoding Specification version 0.0.8*, Craig Bruce, <http://www.opengeospatial.org/docs/03-002r8.pdf>.

[BZIP2] (June 2000), *The bzip2 and libbzip2 official home page*, Julian Seward, <http://www.digistar.com/bzip2/>.

[COMPRESS] Unix **`compress`** file format, Unix manual.

[CWXML] CubeWerx (May 2005), *CWXML Library*, Craig Bruce (ed.), <http://www.cubewerx.com/main/cwxml/>.

[FLOATS] IEEE 754-1985 (1985), *Standard for Binary Floating-Point Arithmetic*, <http://grouper.ieee.org/groups/754/>.

[GZIP] IETF RFC 1952 (May 1996), *GZIP File Format Specification Version 4.3*, L. Peter Deutsch, <http://www.ietf.org/rfc/rfc1952.txt>.

[HTTP] IETF RFC 2616 (1999), *Hypertext Transfer Protocol—HTTP/1.1*, R. Fielding, et al., <http://www.ietf.org/rfc/rfc2616.txt >.

[KEYWORDS] IETF RFC 2119 (March 1997), *Key words for use in RFCs to Indicate Requirement Levels*, Scott Bradner, <http://www.ietf.org/rfc/rfc2119.txt>.

[PNG] PNG (2003), *PNG: Portable Network Graphics: A Turbo-Studly Image Format with Lossless Compression*, Greg Roelofs, et al, <http://www.libpng.org/pub/png/>.

[SHAPE] ESRI® (July 1988), *ESRI® Shapefile Technical Description*, <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>.

[SI] System International base-10 prefixes, <http://www.bipm.fr/en/si/prefixes.html>.

[SI-BIN] System International base-2 prefixes, <http://physics.nist.gov/cuu/Units/binary.html>.

[SLD] OGC 02-070 (August 2002), *Styled Layer Descriptor*, Bill Lalonde (ed.), <https://portal.opengeospatial.org/files/?artifact_id=1188>.

[TAR] Unix Tape Archive utility, **`tar`**, Unix manual.

[WFS] OGC (2004) 04-094, *Web Feature Service (WFS) Implementation Specification*, Peter Vretanos (ed.), <https://portal.opengeospatial.org/files/?artifact_id=8339>.

[WKB] OGC (May 1999) 99-049, *Simple Features Implementation Specification For SQL*, clause 3.3, Keith Ryden (ed.), <http://portal.opengeospatial.org/files/?artifact_id=829>.

[XML-SCHEMA] W3C (May 2001), *XML Schema Part 0: Primer*, David C. Fallside (ed.), <http://www.w3.org/TR/xmlschema-0/>.

[ZIP] PKWARE (April 2005), *ZIP File Format Specification*, <http://www.pkware.com/business_and_developers/developer/popups/appnote.txt>.