gbXML Validator Software Specification

and

Software Programming Implementation

v 1.0.0



# Introduction

The goal of this testbed is to make a tool that validates gbXML geometry files created primarily by cad tools such as Autodesk’s Revit, AutoCAD 3D, Bentley Microstation, Graphisoft ArchiCAD, and SketchUp, among others. Ideally, the testbed will serve as a resource for CAD vendors engineers to be able to enhance the accuracy and reliability of gbXML document creation for the green building community at large. This specification goes into detail regarding the overall project goals, and also includes the proposed solutions to meet these stated goals.

Difference between unit test and test file….

Validation, in the context of a gbXML file, can conceptually be broken into two parts: schema validation and data validation, which will both be described in depth in this specification. In its most basic form, this tool, referred to throughout this document as “the validator”, is designed to be a web service where those parties interested in gbXML validation may upload their gbXML files to a webservice that validates both the schema and the data, and will return a report to the user identifying whether the test passes or fails the criteria set forth by the simulation design community engaged in this testing.

Pass and fail criteria are determined by comparing a gbXML file uploaded by the user to a standard gbXML file that has been pre-edited and vetted for accuracy. This testing technique is conducted, in concept, much in the way that a kilogram mass may be compared to the world standard kilogram for the engineering community that uses SI units. Validation of a test object to a standard object, is a more complex topic for file validation, since many variables are required to be tested, but in concept, the notion of a standard object vs. a test object is the same. This specification will detail how the validator has been structured to satisfy the needs of a geometry validator in the context of architecture and green building simulation tools.

Special thanks to the United States Department of Energy, the Pacific Northwest National Laboratory, Autodesk, Bentley, IES, Glumac Consulting Engineers, Carmel Software Coroporation, and Stanford University for providing guidance and support for this project. Special thanks to the US Department of Energy for providing a grant to make this work possible.

Key Contributors:

Chien Si Harriman, Carmel Software Corporation

Stephen Roth, Carmel Software Corporation

Amir Roth, Department of Energy

Krishnan Gowri, Pacific Northwest Energy Laboratory

# Human User Interaction Requirements

1. *User must clearly be able to understand the gbXML Test Case being evaluated.*
2. *User requires a summary of key findings of the unit test results, and the overall performance of their gbXML file (aka – overall “fitness” score).*
3. *User must receive clear feedback in a human readable format that clearly indicates pass/fail of the unit tests.*
4. *User must clearly understand what actions are being performed during a unit test, the criteria that determine a passed unit test, and the criteria that constitute failing the unit test.*

## User Education Level

The user is intended to be any user, though primarily these users will have some familiarity with gbXML. The user is aware that the gbXML Test Cases exist and does not require a deep explanation of each, just a verbal overview of the Test Case with an optional image. Also, users are aware of the nature of the validator, i.e. – that the validator is comparative…that standard files exist, have names, and can be selected as the means to validate their own gbXML file.

## UI Flow Requirements

* User arrives at the website and prompted to go to the upload page.
* Once on the upload page, the user is given the choice of Test Cases to choose from.
* After selecting a Test Case, the user sees a short description of the Test Case. (fulfills Human User Interaction Requirement 1)
* They may browse for their gbXML file on their local machine.
* User “Uploads” using a button or some other means
* Validator returns the results of the data validation
  + User sees summary table describing the state of the unit test (pass or fail) and the overall “fitness” of their gbXML file (fulfills Human User Interaction Requirement 2)
  + User sees tables describing the results of each individual test
    - Colors will be used in the tables to indicate passing or failing (fulfills Human User Interaction Requirement 3)
  + User may click on any unit test result to understand more clearly why that particular test may have passed or failed (Fulfills Human User Interaction Requirement 4)

The proposed solution to these UI Flow Requirements will receive more treatment in the section below outlining the UI.

# XML Data Validation Requirements

This section describes the overall concept that shall be employed to validate the user’s gbXML files, which are uploaded by vendors or others interested understanding the quality of the gbXML file against standards of quality set forth by the gbXML community.

*There shall be “levels” of validation which ensure any gbXML is:*

* *valid XML*
* *valid gbXML,*
* *geometrically accurate*
* *geometrically perfect*

Each level of validation requires a different set of algorithms to ensure any given gbXML file meets the overall standards of quality. As each test progresses to a higher level of validation, the geometric precision required overall increases.

Below is a flow diagram showing, overall, how the validator shall manage the various levels of validation.

XML Schema Validations

XML Content Validations

PASS?

no

Fail with

messages shown in HTML

Basic Content Tests

Detailed Content Tests

Pass/Fail Criteria Filter

Report Pass/Fail with

messages shown in HTML



gbXML.org

Standard FIle

High Level Tests



yes

user gbXML

Test file

Figure 1: High level functionality of the gbXML Validator.

## Important Schema Validation Requirements

*The schema validator shall employ existing industry-standard XML validation tools available in .NET to first ensure the gbXML file uploaded for validation contains valid XML that meets the current standards set forth by the XML W3C working groups.[[1]](#footnote-1)*

*The schema validator shall employ existing industry-standard XML validation tools available in .NET to secondly ensure the gbXML file uploaded for validation meets the current gbXML requirements as per the latest XSD document available from gbXML.org.[[2]](#footnote-2)*

## Testing Requirements

Testing the gbXML file can be grouped into two general categories, as shown in Figure \_\_ above. *High-Level Tests*—where the XML is generally checked for properly formed XML along with some basic unit tests of easily-accessible values found in the gbXML file, and *Detailed Tests*—unit tests that focus more specifically on specific geometry-related validation, such as comparing the geometric coordinates, orientations, and adjacency conditions of the standard file and the user’s uploaded file. These detailed tests are more complex in nature.

### High-Level Test Requirements

High Level tests shall first include tests for the schema, which in essence look for deficient XML structure. Some examples of deficient XML structure include:

* missing tags
* malformed tags
* lack of closing tags
* unnecessary white space
* etc…

At this phase of the validation, the validator also tests for deficient gbXML, by comparing the uploaded test file to the latest XSD released by gbXML.org. These High-level Tests outlined above might also be referred to as “Schema Validation”.

***Take Note****: Any gbXML file uploaded onto the web service, that does not meet these Schema Validation tests, will be promptly exited and the user warned that the tests cannot be continued until the XML is valid XML, and also basic valid gbXML.*

As per Figure 1, should these Schema Validations be successful, the content of the test gbXML file will then be checked. The validator bundles some **Basic Content** checks, along with the Schema validation in the High Level Tests category. Conceptually, this can be allowed due to the fact that the tests are fairly simple in nature, hence “High Level” is an appropriate term to describe these Basic Content checks. These tests are simple checks of values, element counts, and required spaceId matching.

The following list outlines what is tested in these “Basic Content” tests when the test file is uploaded and has passed the XML Schema Validation tests. Each of these values culled from the uploaded XML file is tested against (compared to) the standard file). This is shown conceptually in Figure 1 above

* Building Floor Area (total square feet)
* Number of Spaces (occurrences of the Space Element)
* Number of Building Stories
* Z-Height of the Building Story
* The Building Story PolyLoop normal vector orientation
* SpaceId Match – ensures that spaceId values in the standard at test files are identical (CANNOT FAIL)
* Space Floor Area (square feet of each space)
* Space Volume (cubic feet of each space)
* Total Number of Surfaces
* Total Number of each of the surfaceType Enums:
  + Number of Surfaces that are Exterior Walls
  + Number of Surfaces that are Underground Walls
  + Number of Surfaces that are Interior Walls
  + Number of Surfaces that are Interior Floors
  + Number of Surfaces that are Roofs
  + Number of Surfaces that are Shading Surfaces
  + Number of Surfaces that are Air Surfaces
* Total Number of Openings
* Total Number of each of the openintType Enums:
  + Number of Openings that are Fixed Windows
  + Number of Openings that are Operable Windows
  + Number of Openings that are Fixed Skylights
  + Number of Openings that are Operable Skylights
  + Number of Openings that are Sliding Doors
  + Number of Openings that are Air Openings

*These basic content tests shall be summarized in a distinct, separate table for the user to review once the test is complete.*

*Basic tests can generally fail and the detailed tests will still carry forward, to the greatest extent possible. It is not necessary for all the basic tests to pass prior to starting the detailed tests. (See establishment of pass/fail criteria below for further details about pass/fail criteria).*

***Take Note:*** *The spaceId Match cannot fail. If it fails, the validator will cease and reports to the user that until this issue has been resolved by the user the validator will not proceed.[[3]](#footnote-3)*

### Detailed Test Requirements

Detailed Tests achieve greater depth than the High Level tests, looking at each individual **Geometric Object** in the standard file.

**Geometric Objects** tested by this version of the validator include only gbXML Surface elements and their descendants, which includes gbXML Opening elements. Those familiar with gbXML understand that Surface elements (of various types (loosely described here as) walls, roofs, floors) are the parent objects of Opening elements (of various types (loosely described here as) windows, skylights, doors). Surfaces are, in turn children of the Space elements already tested in the High Level tests. For this phase of validator development, it has been determined that the information contained within these elements contains all necessary geometric information to create a detailed thermal simulation model of sufficient geometric fidelity for HVAC load calculations, and energy simulations, with the potential for 3-dimensional accuracy that is sufficient to produce reasonably accurate simulation results.[[4]](#footnote-4)

*Surface:*

* *geometry detail*

*Space:*

* *floor area*

*Opening:*

* *geometry detail*

Figure : Space, Surface, and Opening data is sufficient for determining geometric accuracy.

*These detailed tests must ensure that information contained in a given gbXML file, contains a sufficient level of geometric accuracy, only to the extent that load calculations and energy simulations may be carried forward. Watertight enclosure testing of volume surface elements, as may be required for lighting simulations and CFD, is not required for this phase*.

Furthermore, though the standard file may have watertight enclosures and a high degree of spatial accuracy, the user’s gbXML file does not need to match the exact coordinates of the standard file elements exactly. This ensures that some allowances be given to the vendors seeking certification. There is also the fact that certain vendors only produce egg-shell geometries with no construction thickness, which may make the user’s gbXML file slightly different than the standard file geometry. See “Tolerances” below.

*Tolerances should be built into the validator to allow some variance in the test file geometric attributes, to provide the validator with flexibility to declare minor imperfections, still, of sufficient quality.*

The validator shall parse and analyze each Surface and Opening element that describes the building spatial geometry using an algorithm that attempts to match a standard file surface with its suitable equivalent in the test file. Should an adequate match be found, the test will be given a “Pass” mark. Should a match not be found, the algorithm will declare that surface to have no counterpart and the test for that surface shall be considered “Failed”.[[5]](#footnote-5) The results of these passed and failed tests will be stored in a summary list for each Surface and Opening in the standard file, giving the user the additional ability to seek more detail about how the validator has determined whether the test passed or failed. [[6]](#footnote-6)

*For every Surface and Opening in the standard file, a report shall be generated both summarizing and clarifying whether the unit test has passed or failed, and why the validator has found or failed to find a suitable match in the uploaded user gbXML file.*

Even if a match for a Surface or Opening element cannot be found, the validator will still test subsequent Surfaces and Openings until every Geometric object has searched for a counterpart. Users requested this feature, because, conceptually, this should allow for the user to gain more insight into the testing procedure and potentially derive greater meaning and context from the tests.

*The validator must run to completion for all Surfaces and Openings in the standard file, even if a match for a given Geometric Object cannot be found.*

There is only one condition that will prevent the detailed tests from running to completion. If the coordinates that describe the PolyLoop of a Geometry Object are determined to form a non-planar object, or if the coordinates form a self-intersecting polygon then the detailed test shall be declared “Failed” immediately and the no detailed tests will be performed.

*For both Surface and Opening PolyLoop coordinates, the coordinate set must define a planar (and non self-intersecting polygon. If the validator determines that the PolyLoop coordinates define a non-planar object or a self-intersecting polygon, the detailed tests will not be performed. All Surfaces and Openings must meet this principle.*

*The detailed tests shall analyze all attributes of Surface and Opening Elements that are fundamental to the geometric description of a building for energy simulation. It will ignore features such as construction type at this phase of the validator, since this phase of the validator development focuses solely on analyzing geometry characteristics.*

The detailed tests look more closely at the geometric properties of the gbXML file, attempting to discover how accurate the user’s uploaded gbXML file geometry is relative to the standard file. Since the user’s gbXML file should have been created “with the same dimensions located in the same point in the global reference frame”[[7]](#footnote-7), it should be possible to test pure geometric definitions to determine how accurate the two models are to each other.

Theoretically, it should only be necessary to review the PolyLoop coordinates embedded in each Surface and and Opening gbXML element, since each respectively contains all the pure geometric information needed to compare two files. The validator does initiate a routine to compare absolute geometry coordinates in the standard and test files, but….many engineering software vendors do not necessarily utilize the PolyLoop coordinates in order to re-construct geometry data. Instead, the more simulation-specific geometry terminology

* Width
* Height
* Azimuth
* Tilt
* Insertion Point

node values are commonly consumed by the engineering vendor community due to their simplicity and ease of use. In addition, most buildings today are designed using rectilinear geometry components, which are most easily described using these commonly used simulation specific terminology. For this reason, both the coordinate geometry as well as these values are used to check the accuracy of the test file geometric accuracy.

*The validator shall utilize both PolyLoop coordinates as well as more “high-level” Element values in order to deem how accurate a user’s gbXML geometry information is, compared to a standard file.*

As the validator seeks to find an appropriate match in the test file, if it should find a suitable object (whether Surface element or Opening element)…it does not have to match the standard surface explicitly. However, this does extend beyond simply measurement tolerances. Interviews with users and CAD vendors indicated that, in many cases, geometry does not come through cleanly when drawn in the CAD tool, and that a single surface in the CAD tool may be broken into several smaller surfaces by the CAD tool as the geometry is translated from its native format to the gbXML format. Sometimes this “breaking” of the surface may be intended, or in other cases, unintended (the production of slivers). When the act of subdividing a surface is the exception rather than the rule for planar surfaces, it is a very common occurrence for a curved surface to be automatically transformed into planar surfaces for analysis surfaces, with no clear industry guidance as to how these planar surfaces should be created. Thus, one wall may be fragmented into several smaller walls, that, when summed, are the thermodynamic equivalent of one single surface.[[8]](#footnote-8) The wall has not been drawn this way by the user, but fragmented by the CAD tool. Given that there may be no “standard” method to conduct this simplification, the notion of thermodynamic similarity takes on the primary means by which accuracy may be defined. The validator team shall hold these ideas as the future litmus tests for success in future geometry validators.

*The Detailed Tests should be designed such that (if not in this version but in future versions):*

* *the notions of thermodynamic similarity can be accurately assessed*
* *the tests accommodate a wide range of thermodynamically similar objects*
* *future accommodations for highly complex geometry tests are possible without massive structural overhaul to the underlying code*

In this current phase of validator development, it is acceptable if these guidelines above cannot be fully implemented. It is acceptable if it is assumed that a single Geometric object in the standard file be matched with only a single Geometric object in the user’s gbXML file.

*The detailed tests shall be required to find only a single suitable match for each Geometric object the standard file. However, the software should display some structure that will make thermodynamic similarity matching possible in the future.*

### Tolerances

Tolerances should be established for every XML item tested in the gbXML test file, including both high level and low-level tests. Tolerances are required for both traditional measurable units (ft, inches, etc) and counts (the number of times an item appears in the gbXML file.

Below shall be the tolerances as they are currently configured in the validator. Each test shall use the same global tolerances.

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Value being Tested** | **Tolerance** | **Unit** |
| **Space** |  |  |  |
|  | Floor Area | 2.50% | % of standard floor area |
| **Story** |  |  |  |
|  | PolyLoop Z-coordinate | 0.1 | feet |
|  | Outward Normal Vector | 2.5 | degrees from standard's vector |
| **Basic Counts** |  |  |  |
|  | Space Count | 0 | number of spaces |
|  | Level Count | 0 | number of levels |
|  | Surface Count | 0 | # of occurrences of Surface |
|  | Exterior Wall Count | 0 | # of Surfaces = ExteriorWall |
|  | Interior Wall Count | 0 | # of Surfaces = InteriorWall |
|  | Interior Floor Count | 0 | # of Surfaces = InteriorFloor |
|  | Air Surface Count | 0 | # of Surfaces = Air Type Wall |
|  | Shading Device Count | 0 | # of Surfaces =Shade |
|  | Opening Count | 0 | # of occurrences of Opening |
|  | FixedWindow Count | 0 | # of Openings = FixedWindow |
|  | OperableWindow Count | 0 | # of Openings =OperableWindow |
|  | FixedSkylight Count | 0 | # of Openings = FixedSkylight |
|  | OperableSkylight Count | 0 | # of Openings = OperableSkylight |
|  | SlidingDoor Count | 0 | # of Openings =SlidingDoor |
|  | NonSlidingDoor Count | 0 | # of Openings = NonSlidingDoor |
|  | Air Opening Count | 0 | # of Openings = Air Opening |
| **Surface** |  |  |  |
|  | Surface Azimuth | 2.5 | degrees from standard's value |
|  | Surface Tilt | 2.5 | degrees from standard's value |
|  | Surface PolyLoop Coordinate | 0.5 | ft |
|  | Surface Height | 0.5 | ft |
|  | Surface Width | 0.5 | ft |
|  | Surface Area | 2.50% | % of standard surface area |
|  | Surface Insertion PointX | 0.5 | ft |
|  | Surface Insertion PointY | 0.5 | ft |
|  | Surface Insertion PointZ | 0.5 | ft |
| **Opening** |  |  |  |
|  | Parent Surface Tilt | 2.5 | degrees from standard's value |
|  | Parent Surface Azimuth | 2.5 | degrees from standard's value |
|  | Opening Area | 2.50% | % of standard opening area |
|  | Opening Height | 0.5 | ft |
|  | Opening Width | 0.5 | ft |
|  | Opening PolyLoop Coordinate | 0.5 | ft |
|  | Opening Insertion PointX | 0.5 | ft |
|  | Opening Insertion PointY | 0.5 | ft |
|  | Opening Insertion PointZ | 0.5 | ft |

These tolerances are reconfigurable but may require approval of the gbXML.org steering body prior to changes.

Other tolerance requirements:

*The tolerances shall be easily configurable by an administrator.*

*A typical user cannot configure the tolerances.*

*The tolerances do not have to be made editable in the web service, but may be hard coded into the software, though this feature may have a web interface to edit the tolerances in future application.*

*The tolerances must be reported to the user by the validator after the completion of a test.*

## Requirements for Determining if a Test File Passes/Fails

Given that the test file geometry must only be capable of meeting the standard file geometry within given tolerances, as well that CAD tools may create a thermodynamically similar surface sets of objects, and thirdly that it is common for CAD tools to generate what are commonly referred to as “slivers” when translating geometry to gbXML…pass/fail criteria may be more complex than expected. It is required, then, that flexible pass/fail criteria are established for each test.

It has already been identified that certain unit tests, if failed, will result in the entire user test file declared as failed.

* Schema Validation
* Surface PolyLoop coordinates producing a non-planar object or self-intersecting object
* Opening PolyLoop coordinates producing a non-planar object or self-intersecting object
* spaceId values not matching in the standard and test file

*Beyond these strict requirements for the unit tests listed, it shall be possible that a single test may fail, yet overall, the gbXML file uploaded by the user may still be considered of passing quality.*

The motivation for doing this is similar in theory to letter grades in educational curriculum. This specification invokes the word “fitness” to describe the overall quality and precision of the user’s uploaded gbXML file. In this current phase, given the lack of user interaction with the validator, it is not how many levels of fitness may be required to determine success or failure.

*The user interface, at the end of unit tests, shall make it clear that a failed unit test is acceptable for a given test, allowing the gbXML file as a whole, to be of Passing quality.*

*The user interface, at the end of unit tests, shall make it clear that if all unit tests pass within the allowable tolerances, that this denotes a more precise result.*

To think of this in context, generally, the High-Level basic content tests have the most flexibility in terms of allowing a given test to fail, yet still overall, the test file may pass. Since slivers may be common occurrences when creating gbXML from a CAD tool, surface counts may often times not match the standard file when uploaded to the validator. Yet, if each Opening finds a suitable match, then perhaps the gbXML file as a whole, is suitable.

It merits feedback from the user base to determine exactly what may constitute pass/fail overall for the fitness score, and the criteria may be different depending upon the Test Case in question.

*The validator shall have the option to provide each test case with its own hashtable or other means to declare which unit tests are required to pass in order for the fitness score overall to be declared perfect/pass/fail.*

*The validator shall allow the administrator to configure this hashtable criteria for each Test Case to determine when a user’s file is Perfect/Pass/Fail*

## Requirements Implementation (Coding Approach)

This gbXML testbed has been written in .NET Framework 4, utilizing ASP.NET as the html and javascript backbone for displaying the HTML. We do not include all of the MSDN libraries utilized, but include below a description of the custom classes created and the code modules created for the validator

Classes

Figure : Class Diagram - gbXML Validator.

DOEgbXML Basics Class

-standard units of measure

-tolerances

DOEgbXML Report Class

-messages reported to user

-unit test pass/fail bool

Main Methods

DOEgbXML Surface Class

-object class mirrors gbXML

DOE XML + gbXML Schema Validation

DOEgbXML Opening Class

-object class mirrors gbXML

DOE gbXML Parser (geometry checks)

DOEgbXML TestCriterai Class

-fitness requirements

DOEgbXML TestDetail Class

-stores static information about each Test Case (descriptions, etc)



gbXML XSD

DOE XML + gbXML Schema Validation



User gbXML File

if successful

Standard gbXML File

DOE gbXML Parser (geometry checks)

STOP: Display Failure

High Level Basic Content

STOP: Display Failure

Planar Surface Tests

Detailed Surface Tests

Planar Opening Tests

STOP: Display Failure

Detailed Opening Tests

Display Unit Test Results

Report Fitness

Determine Overall Fitness

STOP

Figure : High Level Flow Diagram, gbXML Validator.

### High-Level Test Algorithms/Procedures – Schema Validation

Schema validation of a user’s gbXML test file is accomplished through the use of a .NET 4.0 XmlReader[[9]](#footnote-9) instance, with settings for the reader determined by XmlReaderSettings[[10]](#footnote-10). This XmlReader instance settings determine which warnings and error messages will be thrown as the validator reads the users’ test files. This XmlReader is explicitly instructed by the Schemas Property[[11]](#footnote-11), as to which XSD file is used. The XSD is stored locally on the validator’s web service. Neither the in-line schema contained in the users’ gbXML file, nor the schema location hints found in users’ gbXML document are utilized to pull schema information.

XmlReaderSettings:

* Validation Type = Schema
* Validation Flags = ProcessSchemaLocation & ReportValidationWarnings
* ValidationEventHandler[[12]](#footnote-12) is utilized

With the XmlReader, ValidationEventHandler, and ValidationType set to Schema, events may be produced while the XmlReader is active, during the Read event. Exceptions returned by the ValidationEventHandler are gathered and collected, and making use of the Severity property returned by the handler, the exceptions are broken down into two types, Warnings or Errors. The gbXML validator reformats the exceptions based upon Severity and copies the event into html, making the messages available to the user directly following the test.

There may be an error that cannot be properly handled by the ValidationEventHandler. This will report back to the user the moniker “Big error” which will be reported in HTML to the user. This error is neither an exception of Severity=Warning nor Error. This “Big Error” is due to serious structural deficiencies in the users’ gbXML file and the source of the error cannot be handled by the .NET classes.

**Whether the ValidationEventHandler returns an exception of Severity Warning, Severity Error, or if a Big Error is thrown. In all three cases, the users’ test file will be declared Failed.**

This approach meets the requirements set forth in Important Schema Validation Requirements. There is insufficient labor and manpower to produce a customized library of Schema Validation tools, thus the .Net framework has been employed and the XML portion of the validator is thusly limited to the capabilities provided in these families.

### High-Level Test Algorithms/Procedures – Basic Content

*Total Building Area*

*Building Space Count*

*Building Story Count*

*Building Story Z-height*

*Building Story PolyLoop RHR*

must pass

*SpaceId Value Match*

*Space Area Value Match*

*Space Volume Value Match*

*Surface Count Comparison*

*ExteriorWall Count Comparison*

*InteriorWall Count Comparison*

*InteriorFloor Count Comparison*

*Roof Count Comparison*

*Shade Count Comparison*

*Air Surface Count Comparison*

*Opening Count Comparison*

*FixedWindow Count Comparison*

*OperableWindow Count Comparison*

*FixedSkylight Count Comparison*

*OperableSkylight Count Comparison*

*SlidingDoor Count Comparison*

*Air Opening Count Comparison*

*NonSlidingDoor Count Comparison*

Figure : High Level gbXML tests. Displayed in order of execution in the test bed.

After the XML Validation has successfully occurred, if successful, the High Level Basic Content unit tests are performed. The figure above shows each of the unit tests that are run and subsequently reported in its own summary table.[[13]](#footnote-13) These tests and subsequent presentation fulfill the requirements for the High Level Basic Content tests previously described in the software specification.

The gbXML Elements tested in High Level unit tests are given above in figure ??. All of these unit tests utilize a simple pattern that culls the desired data from both the users’ test file and the standard file, one at a time. This means that, rather than store all the High-Level Basic Content in two objects, one each for the standard and user’s file, each gbXML file is mined for the same data at runtime, only retrieving the data nodes needed for the High Level Uests, i.e. – the gbXML files are mined each time a High-Level test is called.

Conclusion: ?Not using objects does potentially have some long term drawbacks and potential performance implications, but given the simplicity of extracting data from XML files and the simplicity of the data, retrieved, performance was not a concern in this phase of the project. Modifying the code to store data in two objects (a standard file object and a test file object) that can be compared just once, once they have been populated with the desired information from the standard and test files, is achievable without massive modification to the code.

For each desired piece of data in the High-Level Basic Content comparisons, the data is extracted from each the standard and test files using an MSDN XPath expression[[14]](#footnote-14) to extract identical content from both files. The data collected is then compared and messages are returned to the user indicating:

1. Whether the comparison passed or failed
2. The differences between the information extracted.
3. The tolerances for the test, if available.

Results of the high level tests are stored in a table below the overall fitness score, and reported to the user after the completion of the tests (see section on the results page for more information, pg. \_\_\_ ).

### Detailed Test Algorithms/Procedures

#### Flow Diagram of Detailed Testing Procedures

MatchedOpening Id Dict \*

MatchedSurface Id Dict \*

List of Objects

List of Surfaces \*

List of Objects

Convert XML data to Object + store all objects

Convert XML data to Object + store all objects

List of Openings\*

Surface Planar Tests

Opening Planar Tests

Find Opening Matches

Find Surface Matches

Figure : High Level Flow Diagram of Detailed Tests

The figure above shows an overview of how the Detailed unit tests operate. Items listed with asterisks (\*) are collections that are available in memory while the validator runs, which are discarded when the validator completes and exits. Currently, the validator begins by converting the XML in the standard and user uploaded gbXML file to Objects that have attributes identical to the relevant descendants and attributes of the Surface elements of these gbXML files. These objects are subsequently stored via in-memory lists (dynamic arrays). Before the matching routine begins, the user’s polyLoops are checked to ensure that they form planar objects. If the planar tests pass (explained in more detail below), then the arrays are subsequently called into a matching routine, that, for each standard surface object, tries to find a match in the user gbXML object list. Every time it finds a match, it stores the surfaceId of the standard surface that was matched, and a list of all surface Ids in the user file that matched, in a global dictionary element where the standard surface Id is the key and the list of matched user surfaceId’s is the value in the key value pair. Surfaces that have no suitable match in the standard file are not stored in this dictionary. The routine also creates a log that can be displayed to the user to show where the matching routine failed when seeking a match in the user’s data.

In a similar manner, the openings in the standard and user file are also turned into objects that are first stored locally as lists (dynamic arrays). Before the opening matching begins, the PolyLoops of the user’s Opening elements are checked to ensure that they form planar objects. If the planar tests pass (explained in more detail below), then the arrays are subsequently called into a matching routine, that, for each standard opening object, tries to find a match in the user gbXML opening object list. Every time the routine finds a match, it stores the openingId of the standard opening that was matched, and a list of all openingIds in the user file that matched, in a global dictionary element where the standard openingId is the key and the list of matched user openingId’s is the value in the key value pair. Openings that have no suitable match in the standard file are not stored in this dictionary. The routine also creates a log that can be displayed to the user to show where the matching routine failed when seeking a match in the user’s data.

Also note in the Figure that the routine for finding openings also relies on the global list of Surface objects and the MatchedSurfacesId Dictionary described. The reason the global list of Surface objects is because the Opening element is a child of a surface. In order to truly understand the geometric properties of a given opening (its tilt, azimuth, and coordinates in the global reference frame, to be precise), each opening when processed in the routine must pull information about its parent, which is subsequently analyzed to find a match. The MatchedSurfacesId is a simple shortcut that is employed by the routine to simplify the geometry analysis when finding an opening match. This is to overcome the use case where a multi-storey building with identical floor plates and building shell are modeled. This use case will be described in more detail below.

These objects could be stored in a database, easily.

These detailed unit tests begin by gathering all of the Geometric Objects and their relevant information (stored in its attributes and descendants) into iterable objects for both the standard file and the test file. The validator, being programmed in .NET 4.0[[15]](#footnote-15) transfers the XML data into a specially constructed class with attributes identical to the gbXML data structure. Each geometric object is stored as a separate instance, each instance being placed in an iterable object.

Test File Iterable Object

for Surfaces

Standard File Iterable Object

for Surfaces

su-1

su-1

surface 1

su-1

su-1

surface 1

Test File Iterable Object

for Openings

Standard File Iterable Object

for Openings

su-1

su-1

opening 1

su-1

su-1

opening 1

All of the data is extracted from the gbXML file and placed into four different iterable objects, two for the standard file (one iterable object for Surfaces and Openings each) and two for the Test File (one iterable object for Surfaces and Openings each).

#### Procedure for checking if PolyLoops are Planar

Once these iterable objects have been created, each Surface and Opening PolyLoop in the test file[[16]](#footnote-16) is checked to ensure the PolyLoops’ coordinates of each Geometric object form a planar surface [and that the coordinates do not form a self-intersecting polygon.] The planar test algorithm is as follows.

*The planar test is required to look at each coordinate in the PolyLoop and verify that it is in the same plane as all the other coordinates in its PolyLoop coordinate group, with an arbitrary first coordinate in the coordinate set.*

To determine if a coordinate is planar or not with its neighboring coordinates, a simplified method was used, taking sets of three coordinates at a time, creating two vectors from the three coordinates, and one resultant normal vector from the two vectors. These normal vectors are collected successively as each new set of coordinates are gathered, forming two new vectors and a new resultant normal vector. For any given number of coordinates *n*, there shall be *n-2* normal vectors formed. All normal vectors are collected, and compared.

For each set of three coordinates in the PolyLoop[[17]](#footnote-17), get the vector formed by the right hand rule of the vectors formed by these three coordinates. Store this right hand rule (RHR) vector as a unit vector. Continue in this fashion until all coordinates in the PolyLoop have been used to create vectors, whose cross products are found to determine an outward normal, storing the right hand rule vector each time. Cross product calc example:

(x4,y4,z4)

(x3,y3,z3)

(x1,y1,z1)

(x2,y2,z2)

Vector1-2 = (x2 – x1) x + (y2 – y1) y + (z2 – z1) z

Vector2-3 = (x3 - x2) x + (y3 – y2) y + (z3 – z2) z

(x2 – x1) (y2 – y1) (z2 – z1)

(x3 – x2) (y3 – y2) (z3 – z2)

Normal Vector = det =

(y2 – y1)(z3 – z2) – (z2 – z1)(y3 – y2) x

(z2 – z1)(x3 – x2) – (x2 – x1)(z3 – z2) y

(x2 – x1)(y3 – y2) – (y2 – y1)(x3 – x2) z

Figure : Example right hand rule calculation for any given set of coordinates.

The normal vector calculations will continue successively until all coordinates have been used to form a vector. In the example above, one additional set of two(2) vectors and a resulting normal vector will be created with (x2,y2,z2), (x3,y3,z3), and (x4,y4,z4).

To generalize to the case where there might be more than four coordinates, we extend to five coordinates as an example, which also forms n-2 normal vectors, stored in RHR Vector Storage in the example diagram below.

RHR Vector Storage

Get RHR

coord 1, coord 2, coord 3

GetPolyLoop Coordinates

coord 1: (x1,y1,z1)

coord 2: (x2,y2,z2)

coord 3: (x3,y3,z3)

coord 4: (x4, y4,z4)

coord 5: (x5,y5,z5)

Get RHR

coord 2, coord 3, coord 4

Get RHR

coord 3, coord 4, coord 5

if the two vectors are parallel:

xcomponent == xcomponent of neighbor AND

ycomponent == ycomponent of neighbor AND

zcomponent == zcomponent of neighbor

check RHR and the neighboring vector ahead

RHR Vector Storage

or if the two vectors are anti-parallel:

xcomponent == -1\*xcomponent of neighbor AND

ycomponent == -1\*ycomponent of neighbor AND

zcomponent == -1\*zcomponent of neighbor AND

continue checking RHR Vectors until all checked

planar coordinates

else:

not planar : FAIL

To summarize, this algorithm will create at least one normal vector (for a *n=3* sided surface, there are n-2 normal vector ), up to generally *n-2* normal vectors for *n* coordinates, that are stored in an iterable object such as a list. Each of the vectors is stored in a list and are then are compared to one another, and as long as the vector components of all RHR vectors form vectors that are *exactly* parallel or anti-parallel, the coordinate set is assumed to form a planar surface. Even a minor deviation from parallel or anti-parallel will fail this test.

Note that not every vector is compared with every other vector, just its neighbor. Since each vector pair is checked sequentially, if each pair is parallel or anti-parallel, they by the associative law, if each pair is parallel or anti-parallel, all vectors are parallel or anti-parallel.

The reason for comparing all vectors created by the PolyLoop coordinates is due to the need for *each and every* coordinate to lie within the same plane, with no deviations in the *entire* PolyLoop set, thus a rational framework for testing all coordinates in the PolyLoop needed to be devised. In testing the application, it has been found that the planar tests must make an allowance for both parallel and antiparallel normal vectors because there is a common situation when the PolyLoop coordinates formed a plane that was not a square or rectangle, but a plane that has a jagged profile. These types of profiles are particularly common for floor plates. As the algorithm planar test successively analyzes sets of three coordinates, it is possible that the normal vector formed by these coordinates will not be parallel to the other normal vectors. However, if the coordinates lie in the same plane as the rest of the coordinate set, the normal vector should be exactly anti-parallel.

There is also information about planar tests TestPlanarSurfaceTest() and TestPlanarOpeningTest() in the tracking log below.

After determination that the each set of PolyLoop coordinates are planar, tests are conducted on first the Surface and then the Opening Elements and their children to ensure that the test file closely matches the standard file.

#### Procedure for Analyzing Geometric Accuracy of Surface Objects

After several different experimental testing methodologies, the gbXML team determined that the Detailed Test algorithms should be designed to ensure that thermodynamic similarity should be a consideration when extending the testbed interface, and that the detailed tests shall be structured to accommodate more complex geometry as noted in the sections above (pg?). Though not each of the requirements could be met in this phase of the validator’s implementation, it is deemed that the approach taken will form a solid code base upon which these future requirements can be met.

With this in mind, the surface geometry validator has been structured in the following manner:

for each test surface…find matches for the standard surface

su-1

su-1

surface 1

standard file surface iterable

test file surface iterable

su-1

su-1

surface 1

surface 1

for each standard surface…

for the standard surface, search all test file surfaces that have the same adjacency relationships

Adjacency Matching

of those surfaces found above, keep only the surfaces that have the same surfaceType enumeration

surfaceType Enumeration

Azimuth and Tilt

of surfaces remaining from above, keep only the surfaces that have azimuth and tilt within tolerance

of surfaces remaining from above, keep only the surfaces that have surface area within tolerance

Wall Area

Absolute PolyLoop Coordinates

of those surfaces remaining from above, determine if the PolyLoop coordinates match

Insertion Point Coordinates

of those surfaces remaining from above, determine if the insertion point coordinates match

Figure : Sequence of Detailed Test function calls for a Surface Element Test.

The figure above shows the logical progression of function calls for Detailed Tests performed on each surface in the standard file. At a high level, this progression could be explained as a set of calls that take each individual surface from the standard file, and searches all test surfaces for a match.

The search algorithm progressively calls functions that look for matches that demand higher orders of accuracy, with an underpinning seeking thermodynamic similarity first, and geometric accuracy second. Starting at the beginning of the search, the test seeks only for similar neighboring relationships and returns all surfaces that have the same neighboring relationships. Of this collection of returned surfaces from the test file, a second test is performed, keeping only those surfaces that have the same neighboring relationships AND similar tilt and azimuth. This is a refined collection, which is subsequently passed to the next function, where the collection is filtered further by calculated wall area. This remaining collection now only contains surfaces that have similar neighboring relationships AND similar tilt and azimuth AND similar surface areas. By definition, this remaining collection contains all thermodynamically similar surfaces that match the given standard test surface under consideration.

One could argue, at this stage, that the remaining tests are only to ensure accuracy of geometry information. Detailed coordinate tests are carried forth to ensure that the surface does indeed closely match the standard file, first at the PolyLoop coordinate level, and next for the definition of the insertion point. If all of these tests pass, and only one surface remains at the end of the coordinate tests, then the Surface in the standard file has successfully found a match in the test file.

#### Explanation of each Procedure Call for Surface Matching

##### Adjacency Matching

The adjacency match searches each test file surface to find whether the AdjacentSpaceId’s spaceIdRef attribute values match the standard surface in question. If a match is found, the test surface is added to a temporary collection. There is no limit as to the number of temporary surfaces that may match, so long as a match can be successfully established.

Because this test using the value of the spaceIdRef, it is critical that the spaceId values match exactly in both the standard file and the test file. If they do not match, then this test will always fail, since the validator uses the spaceId reference to establish adjacency relationships. The validator could have attempted to use a more complex method, allowing the CAD vendor to name the spaces whatever they would have liked, but this proved to be a far more complex task that may be an enhanced feature of the tool in the future.

The adjacency matching algorithm does take into account the ability to match adjacencies for each of the different surfaceType enumerations available in the latest version of the schema.

##### SurfaceType Matching

The surfaceType match takes the temporary collection of test surfaces found above, and now searches it to find those that have the same surfaceType enumeration as the standard file. For example, if the standard surface is of Type “InteriorWall”, then all test surfaces in the collection that also have type=”InteriorWall” will remain. The rest will be discarded. A new collection will be created that keeps all test surfaces that match. These surfaces have the same surfaceType AND AdjacentSpaceId attribute values.

The collection created to store those surfaces that matched the AdjacentSpaceId attribute values (from the Adjacency Matching step above) is destroyed.

##### Azimuth and Tilt Matching

This new second collection of test surfaces is then again searched to find only those surfaces whose tilt and azimuth match within the allowable tolerances specified by the program administrator. Both the surface and the tilt are both tested at once, so a surface in the test collection that has the same tilt but different azimuth will still fail. Surfaces that typically are considered to have an arbitrary azimuth (e.g. – floors) must still match the azimuth of the standard file or else the surface will be thrown away.

It is sometimes assumed that surfaces with large changes in surface azimuth are still thermodynamically similar. Typical industry calculations will allow the azimuth to be +/- 45o from the actual surface azimuth and still be considered thermodynamically similar. This test has much tighter tolerances because of the overall philosophy of the test, which has very tight tolerances for coordinate geometry accuracy.

Those surfaces in the test collection that meet the azimuth and tilt criteria are stored in a new collection, that contains all surfaces that match the azimuth AND tilt AND surfaceType enumeration AND adjacentSpaceId values as the standard file in question. The old collection formed in the matching step above, is destroyed.

This matching scheme is the first occasion where tolerances come into play in the detailed surface tests.

##### Surface Area Matching

To match the surface area of the standard file, it might at first pass seem simple enough to only use the values stored in the Height and Width elements that are children of the Surface element, and multiply them together. However, this test goes one step further than this and can calculate the area of any complex, non-intersecting polygon. This is to accommodate the case where the surface description is more complex, perhaps an irregular polygon that is not well-described by the use of simply the Height and Width element values.

This matching scheme starts by reviewing the coordinates of the standard file PolyLoop, and makes an assessment as to whether or not the PolyLoop coordinates form a rectangle or square. If the coordinates do form a rectangle or square, then the matching scheme by default will simply look at the Height and Width values in the collection of Test surfaces, keeping only those test surfaces that have Width and Height values that are within the allowable tolerance of the Standard surface being evaluated. Those test surfaces that are within the allowable tolerance are stored in a new collection and the old collection from the matching scheme above is destroyed.

If it is determined that the PolyLoop coordinates do not form a square or rectangle, then the Surface Area matching scheme goes through a lengthier process of using Green’s theorem on the PolyLoop coordinate list to calculate the area of first the standard surface and then the area of each of the test surfaces still remaining from the collection created in the previous matching step. Those test surfaces that can match the standard surface area within the allowable tolerance are stored in a new collection, and the old collection from the previous matching step is subsequently destroyed at the end of this matching test. See below for more treatment on the implementation of Green’s theorem.

##### PolyLoop Coordinate Matching

At this stage, there should typically be only one test surface remaining in the latest collection formed in the previous matching step, but it is possible that more than one could still remain. Each of the remaining candidates will have its PolyLoop coordinates compared to the PolyLoop coordinates of the standard file. If all the coordinates in one of the test file surfaces matches the Standard file, then this surface goes on to finally have its insertion point coordinates checked. If a test surface cannot match even a single of the standard surface coordinates, then this matching test fails, the test surface is discounted, and the next test surface is checked.

*If no test surfaces pass the PolyLoop coordinate matching test, then it is deemed that this Standard surface has no match, and the detailed Surface test will fail*.

This coordinate matching scheme works simply. The test does not attempt to enforce a particular order for the surface PolyLoop coordinates…neither does it require the PolyLoop coordinates of the Test File surface to be in the same order as the Standard File surface in question, nor does it require that the order of the Test File surface PolyLoop coordinates be in a counter-clockwise direction. It simply matches the coordinates.

For each coordinate in the Standard File Surface, this coordinate’s X,Y,Z coordinates are checked against all coordinates in the Test File surface candidate. If a match is found that is within the allowable tolerance, the next coordinate in the Standard File Surface is chosen, and again, its X,Y,Z coordinates are checked against all coordinates in the Test File surface candidate. If each coordinate in the Standard File Surface finds a match with a given Test File Surface, the two are assumed to have matched.

The reason it takes this approach is because additional coordinates may be inserted into a given PolyLoop when CAD tools translate to gbXML. These extra coordinates are simply ignored by this algorithm, giving the CAD vendor more leniency when the code checks for coordinate matches. Since surface area has already been matched, we can assume that the additional coordinates do not greatly affect the overall surface.

do any match?

coord 1: (x1,y1,z1)

coord 2: (x2,y2,z2)

coord 3: (x3,y3,z3)

coord 4: (x4, y4,z4)

coord 5: (x5,y5,z5)

Test File surface

Standard File surface

coord 1: (x1,y1,z1)

coord 2: (x2,y2,z2)

coord 3: (x3,y3,z3)

coord 4: (x4, y4,z4)

coord 5: (x5,y5,z5)

Test File surface

no...fail test surface.

New Test Surface

Start Test from

Standard File Coord 1

yes…try to match

next Standard File

Surface Coord

Coordinate

Match?

if all Standard File

Surface Coords Match:

Pass

if all Surface Coords do not Match && all

Test File Surfaces are checked:

Fail

##### Insertion Point Match

With the one surface that remains (if any) from the surface coordinate match above, the insertion point coordinates of the Standard File surface and the Test File surface are compared. If the Test File X,Y,Z coordinates are within the allowable tolerance of the Standard File X,Y,Z insertion point, then this match will pass and the entire detailed Surface test will be declared to Pass.

#### The Use of Green’s Theorem

To be more precise, the equations that have been used in the validator to calculate the area of any polygon with ordered coordinates (that do not form a self-intersecting polygon) are adaptations of Green’s theorem which is more typically employed when one wishes to calculate the area between two line integrals of a 2-D graph. Though we use the term Green’s theorem, we do not use integrals per se to solve this problem, but instead use a well-known simplification that adheres to the principles of Green’s Theorem. The formula given in the figure below, with example geometry and a solution.



Figure : Formula for calculating the area of a polygon defined with two coordinates, whose polygons are ordered (clockwise or counterclockwise) and whose coordinates do not form a self-intersecting polygon.[[18]](#footnote-18)

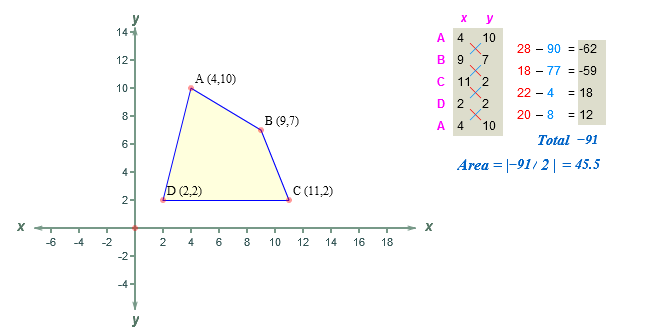


Figure : Example of calculation of irregular non self-intersecting polygon defined with coordinates in sequential counter clockwise order.[[19]](#footnote-19)

The equation and example above may only be used for a polygon defined with two coordinates only—there is no known extension into three dimensions where the plane (though 2-D by definition of a plane) is defined with 3 coordinates. Each plane as defined in the gbXML PolyLoop, therefore, has to be transformed into a 2-D plane prior to using this calculation method.

Transformation of 3-D coordinates to 2-D is trivial for equations where the surface as defined in gbXML is parallel to the X, Y, or Z global reference frame axes. It is more difficult for the case where the surface description is not aligned with X,Y, or Z global reference frame axes. The transformation technique used in the validator first checks to see if the surface is aligned with the reference frame axes, and if so, performs a simplified transformation, otherwise it performs a more complex transformation that is based on work done previously in the computer graphics industry for similar problems.

First, determine how the surface is aligned:

Find the normal vector of the surface, using the right hand rule

If the normal vector = [1,0,0] then the surface is parallel to the YZ-Plane

* Perform the following transformation: set all X-coordinates in the PolyLoop coordinates to zero. Pass only the Y and Z coordinates and proceed as if the surface now has only two coordinates (Y,Z). Solve for area.

If the normal vector = [0,1,0] then the surface is parallel to the XZ-Plane

* Perform the following transformation: set all Y-coordinates in the PolyLoop coordinates to zero. Pass only the X and Z coordinates and proceed as if the surface now has only two coordinates (X,Z). Solve for area.

If the normal vector = [0,0,1] then the surface is parallel to the XY-Plane

* Perform the following transformation: set all Z-coordinates in the PolyLoop coordinates to zero. Pass only the X and Y coordinates and proceed as if the surface now has only two coordinates (X,Y). Solve for area

else Perform the following transformation

* Take the normal vector and declare it to be a new “local Z-axis” for the surface (which is still defined in the global reference frame)
* Take the cross product of the normal vector and the global X-axis (normal x [1,0,0]) to get the new “local Y-axis”, which is still defined in the global reference frame)
* Take the cross product of the new local-Y axis and the surface normal vector (local Z-axis) to get the new “local X-axis, which is still defined in the global reference frame))

(at this stage we now have a new local reference frame for the surface)

* ensure that all local axes are unit vectors
* arbitrarily pick one of the coordinates to be the new origin for the surface (0,0)

to find the X and Y of each subsequent coordinate:

* find the X, Y, Z difference between the origin and the coordinate in question using both the coordinates’ global reference frame definitions, described as a vector with an x, y, z component. Let’s call this the difference vector
* the new X coordinate = local X-Axis  difference vector
* the new Y coordinate = local Y-Axis  difference vector
* pass these new coordinates to the algorithm to determine the area. Solve for area.

#### Procedure for Analyzing Geometric Accuracy of Opening Objects

The process for analyzing the detailed geometry accuracy of the opening objects is very similar to the detailed surface checks. Below is a flow diagram showing how the each Opening element in the standard file is matched with the user’s uploaded file.

Note that we do not cover planar tests here, as they are similar and perform the same function as for the surface planar tests.

test file opening iterable

opening 1

su-1

su-1

su-1

su-1

opening 1

for each standard opening…

opening 1

standard file opening iterable

for each test surface…find matches for the standard surface

Parent Az and Tilt Matching

find all openings in user’s file that have similar parent azimuth and tilt

opening Type

of those matched above , keep all that have the same opening type as the standard opening

Azimuth and Tilt

of those matched above, all should have an azimuth at tilt of zero. Candidates that fail are removed

Area Match

of those matched above, an area test will filter out those that have dissimilar area out of tolerance

PolyLoop Coord Match

of those matched above, use planar geometry cords to keep only the one that matches the standard

opening

Insertion Point Match

of those matched above (should be only one match), the insertion points are checked against

the standard opening. If this passes, the opening Match declares “Pass”

Deeper explanation of these tests as above, to be completed.

## Anticipated Future Revisions

There is a future provision to allow more than one match of a Geometric object, which will be available in a future release of the validator.

In the future, it is anticipated that other Geometric Objects (most notably Shell Geometry and Space Boundaries) will also be included in the validator. At this phase of the project, enclosure checking…the process of ensuring the model is watertight without overlapping surfaces and coordinates, is slated to be added to the geometry verification tool. It is considered that

The structure of the detailed tests, it is believed, could be adapted to match thermodynamically similar surfaces with much more relaxed constraints on the accuracy of the actual PolyLoops. Research and future efforts are the only certainty that this function calling flow can be adapted to these uses.

Finally, the validator is intended to be tested against increasingly complex geometry conditions and larger models. More work must be done to make the process of creating standard test files, checking the files, and “healing” mistakes a priority if the validator is to be effective at a wide range of scales. Thus far, without a suite of tested tools, the construction of large standard models has been an entirely manual process with assistance from a BIM authoring tool. There must be a way to better analyze and fix large files that are broken and do not meet the gbXML standard.

The anticipated release of these additional features is not known at this time.

|  |  |  |
| --- | --- | --- |
| gbXML Validator Features | Validator Version | Anticipated Date of Release |
| Surface, Opening Elements Checking | 1.0.0 | March 31, 2013 |
| Shell Geometry, Space Boundary Enclosure Checks |  |  |
| Thermodynamic Similarity Checking |  |  |
| Enhanced gbXML file creation and automated repair |  |  |

1. See <http://www.w3.org/XML/Core/#Publications> for XML W3C requirements [↑](#footnote-ref-1)
2. As of this writing, the most current XSD is Version 5.10 <http://gbxml.org/currentschema.php> [↑](#footnote-ref-2)
3. The validator requires this spaceId match to ensure that the remaining code can successfully be employed. Onus is placed on the user of the validator (CAD vendor, other interested party) to ensure the spaceId values match. [↑](#footnote-ref-3)
4. No previous work could be cited that ensures the accuracy of this statement. This is considered state of the art in the field of energy and HVAC engineering. [↑](#footnote-ref-4)
5. Fulfills Human User Interaction requirement 3 [↑](#footnote-ref-5)
6. Fulfills Human User Interaction requirements 2, 4 [↑](#footnote-ref-6)
7. This is a requirement in each of the Test Cases. Refer to test cases for more instruction. [↑](#footnote-ref-7)
8. The notion of thermodynamic similarity did dominate several discussions, as it is common industry knowledge that, surfaces that are not precisely described may be suitably used to produce an accurate analysis, so long as the orientation, total surface area, and absolute coordinates are within sufficient levels of accuracy in order to achieve meaningful results from the analysis. [↑](#footnote-ref-8)
9. http://msdn.microsoft.com/en-us/library/system.xml.xmlreader(v=vs.100).aspx [↑](#footnote-ref-9)
10. http://msdn.microsoft.com/en-us/library/system.xml.xmlreadersettings.validationtype(v=vs.100).aspx [↑](#footnote-ref-10)
11. http://msdn.microsoft.com/en-us/library/system.xml.xmlreadersettings.schemas(v=vs.100).aspx [↑](#footnote-ref-11)
12. http://msdn.microsoft.com/en-us/library/system.xml.xmlvalidatingreader.validationeventhandler(v=vs.100).aspx [↑](#footnote-ref-12)
13. Not all surfaceType enumerations nor openingType enumerations are checked in the validator at this stage. The Test Cases do not use these enumerations and so are outside of scope for this phase. [↑](#footnote-ref-13)
14. The XPath expression for each of the Basic Content comparisons is given below in the Code Library section. [↑](#footnote-ref-14)
15. Using C# and Visual Studio 2011 [↑](#footnote-ref-15)
16. Standard File has already been vetted [↑](#footnote-ref-16)
17. There must be at least three coordinates in the PolyLoop to define a plane. [↑](#footnote-ref-17)
18. From Wikipedia: <http://en.wikipedia.org/wiki/Polygon> [↑](#footnote-ref-18)
19. Taken from: <http://www.mathopenref.com/coordpolygonarea.html> [↑](#footnote-ref-19)