Validation Processor
Implementation Report

Submitted To: Program Manager
GeoConnections
Victoria, BC, Canada

Submitted By: Jody Garnett
Brent Owens
Refractions Research Inc.
Suite 400, 1207 Douglas Street
Victoria, BC, V8W-2E7
jgarnett@refractions.net
Phone: (250) 383-3022
Fax: (250) 383-2140

Table of Contents

1 INTRODUCTION

This document describes the changes we have made during the implementation of our initial design, and the future changes that are planned. Our initial design held up very well during implementation and most changes are minor and will have very little impact on existing code.

2 VALIDATION PROCESSOR

2.1 Overview

The Validation Processor manages the Feature Validation and Integrity Validation Tests. Every feature that is modified is checked against the list of validation tests and has the appropriate tests run against it. A Visitor design pattern is used to keep track of the results of the validation tests. When the tests have been completed, the Visitor is returned with all of the results to determine if the updated features have passed the tests. If the features failed, the user is given the list of errors.

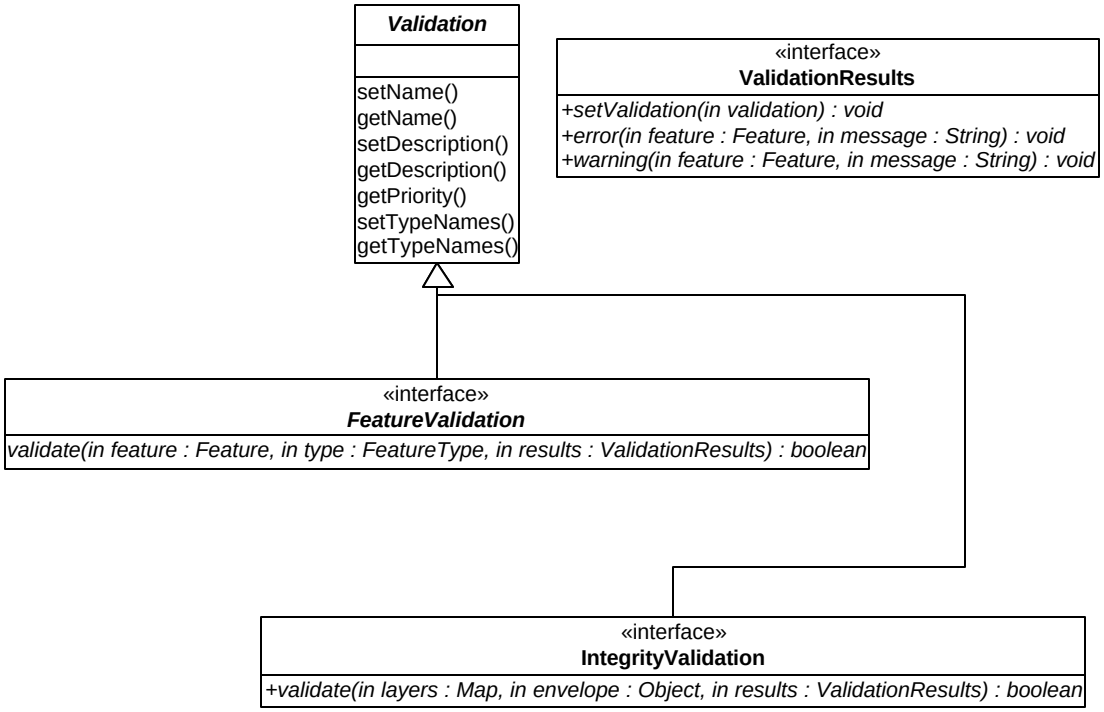

Figure 1 - Validation API

2.2 Recommendations

Our initial design held up to the implementation with very few improvements that are needed for future development. This section documents our planned modifications.

2.2.1 Simplify the runFeatureTests() Method

Currently, runFeatureTests requires that the user pass in the FeatureType of the FeatureCollection that is being validated. However, the FeatureType can be obtained from the FeatureCollection and thus is not needed from the user.

Original

```
Public void runFeatureTests(FeatureType type, FeatureCollection collection,
ValidationResults results);
```

New

```
Public void runFeatureTests(FeatureCollection collection, ValidationResults
results);
```

2.2.2 Implement Test Cost

In order to allow efficient validation tests to fail on dirty features early on, before expensive tests are run, we have proposed the implementation of a test *cost* that would be determined by the validation test programmer. The cost would rank the validation tests so the validation processor can sort them. This feature has been in our design but has not yet made it into implementation.

2.2.3 Exception Handling in ValidationResults

Validation tests have the ability to throw exceptions to indicate that the test was not performed. These errors are not related to the Validation rules specified by the Validation plug-ins and the user should be warned of this event. This will speed up the debug process of plug-ins. We plan to have the exceptions recorded into the ValidationResults Visitor and appear as a type of error, separate from validation errors, for the user to track down.

3 VALIDATION TEST IMPLEMENTATIONS

3.1 Overview

We have implemented various validation plug-ins to test the functionality of our Validation Processor. These plug-ins are loaded and configured, then run against the features that are being modified.

3.1.1 Feature Validation

Feature Validation tests are performed on one feature at a time and have no ability to obtain information from other features. These validations are usually relatively simple and quick to perform.

3.1.2 Integrity Validation

Integrity Validation tests are performed across multiple features and possibly multiple feature types. These validations can be quite costly and are therefore performed after the feature validation tests so that the validation processor has a chance to quit early on.

3.2 Implemented Validations

We have implemented and tested several Validations for both Feature and Integrity situations.

Test Name	Validation Type	Description
IsValidGeometry	Feature Validation	Tests to see if the Geometry is valid.
IsSingleSegmentLine	Feature Validation	Tests to see if a line is made of only two points (one segment).
NoSelfIntersectionLine	Feature Validation	Tests to see if a line intersects itself.
NoSelfOverlappingLine	Feature Validation	Tests to see if a line overlaps itself. This test is not yet fully operational.
LinesNotIntersecting	Integrity Validation	Tests to make sure that no lines, anywhere, are intersecting.
UniqueFID	Integrity Validation	Tests to make sure that no features have the same feature ID.

3.3 Recommendations

3.3.1 Design enhancements

We will be implementing more functionality into the Integrity Validations to allow them to pull from the database with the Envelope bounding box they were provided. Right now the existing tests require all of the features to be passed into the integrity validations in order to perform testing.

3.3.2 Additional Validation Implementations

We also plan to implement more validation tests to fulfill the requirements of the DRAQA (Digital Road Atlas Quality Assurance) test suite and ESRI's ArcGIS Geodatabase. These tests include:

- Area boundary must be covered by boundary
- Boundary must be covered
- Contains Point
- Endpoint must be covered
- Must be covered
- Must be covered by boundary
- Must be covered by endpoints
- Must be covered by feature class
- Must be properly inside polygons
- Must cover each other
- Must not have dangles
- Must not have gaps
- Must not have Pseudo-nodes
- Must not intersect or touch interior
- Must not overlap
- Must not overlap with
- Point must be covered by Line
- Value gap check
- Value overlap check
- Angle size check
- Attribute specification check

3.3.3 Caching

Integrity tests are the most costly of the validations because they span many Features and FeatureTypes. In order to improve performance, we plan to cache pre-built Feature networks, and make spatial indexes available to all Integrity tests. This will prevent the validations from generating this critical information each time they are called.