Validation Plug-In API
Design Document

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) 885-0632
Fax: (250) 383-2140

TABLE OF CONTENTS

1 INTRODUCTION

The Validating Web Feature Server uses a series of validation tests to ensure spatial database integrity across transaction operations.

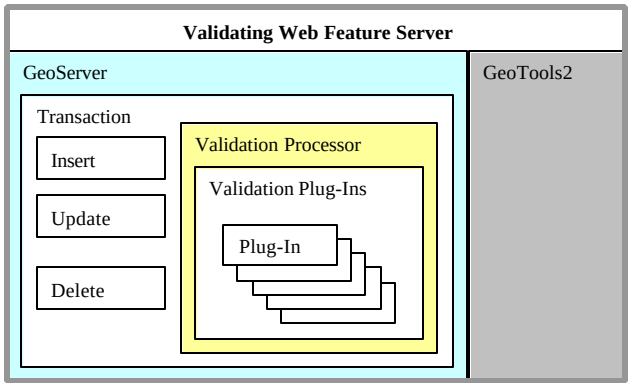

Figure 1: Validating Web Feature Server

Validation Tests are implemented by the Plug-In API described in this document.

This document is intended for Validation Plug authors. For an alternate viewpoint, and the border issue of WFS validation, please see Validating Web Feature Server Design Document.

2 VALIDATION PLUG-INS

A Validation Plug-In implements a user-defined test on feature information. A Plug-In definition is used to map the user's test to the Java class implementing the test.

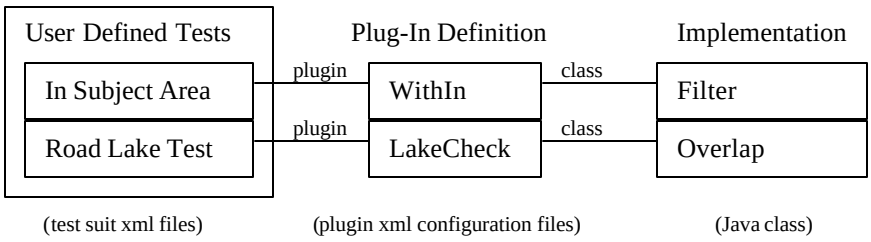

Figure 2: Validation Plug-Ins

For more information on test and Plug-In configuration please see the Validation Language Document.

2.1 User Defined Validation Tests

Consider the following user defined test:

```
Name: In Subject Area
Description: Ensures roads are within the subject area
PlugIn: WithIn
FeatureType: road
Rectangle: (A Bounding Box describing the subject area)
```

This test ensures that all road features are limited to the subject area.

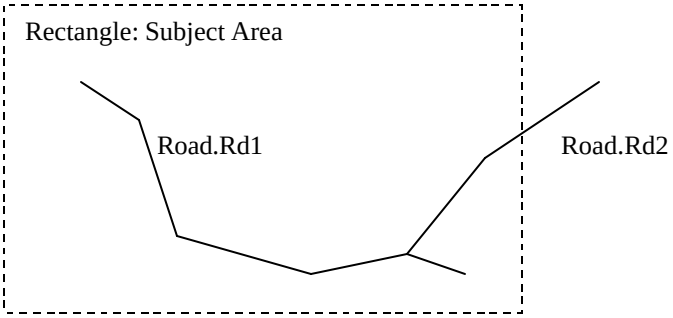

Figure 3: InSubjectArea Road Test

In the above diagram:

- InSubjectArea will pass Road.Rd1 as it is within the Subject Area
- InSubjectArea will fail Road.Rd2 as it outside the Subject Area

By declaring tests, rather than providing a script, we minimize configuration complexity and provide room for application based optimization.

2.2 Plug-In Definition

A Plug-In Definition is used to associate this validation test with the Java class used to perform the test.

```
Name: WithIn
Description: Ensures features of type FeatureType are located in Rectangle
Class: org.geoserver.validation.plugins.Filter
Required Arguments:
  FeatureType
  Rectangle
```

The Validation Processor will use an instance of the Filter class to check any modifications or additions to the road featureType.

2.3 Implementation

A Validation Plug-In is implemented as a Java Bean. Two kinds of Validation Plug-In are supported each with its own Java Interface.

2.3.1 FeatureValidation Interface

The FeatureValidation Interface is used by Plug-Ins providing per feature validation tests.

Example of Per Feature Validation Tests:

- attribute tests: valid speed limit, name checks, valid road type
- topology tests: closed polygon

A FeatureValidation Plug-In process a single feature at time.

2.3.2 Integrity Validation Interface

The IntegrityValidation Interface is used by Plug-Ins providing tests that verify the relationships between features.

Example Integrity Tests.

- Gap checks: on road or river networks
- Intersection Tests: ensure Road intersections occur only on nodes
- Overlap: ensure Roads and Rivers do not overlap
- Attribute Tests: ensure rivers flow downhill

An Integrity Validation Plug-In process many features at a time.

3 VALIDATION PLUG-IN DESIGN

3.1 Design Guidelines

Plug-Ins should be designed with the following guidelines:

- **Single Purpose:**
  A Plug In performs a single validation test
- **User Level:**
  A Plug-In performs a user-level test. Configuration should be expressed in terms meaningful to a GIS user, rather than a developer.

The goal of these guidelines is:

- To create Validation Plug-Ins as reusable components
- To define Validation Plug-Ins that are meaningful to users

If the user is tempted to write a script it is likely that they are trying to construct a higher-order User Level test. We should take this as an opportunity to provide, document and test a new Validation Plug-In.

4 VALIDATION PROCESSOR

The Validation Process is used by the Validating Web Feature Sever to execute Validation Tests.

4.1 Transaction Operation

The Validation Processor performs tests at several points over the course of a Transaction Operation.

	FeatureValidation	IntegrityValidation
Pre Insert	×	
Post Update	×	
Pre Commit		×

- **Pre Execution of Insert Transaction Elements:**
  FeatureValidation Tests can be performed on individual features specified in the Insert Element
- **Post Execution of Update Transaction Elements:**
  FeatureValidation Tests can be performed on Features modified by the Update Element.
- **Post Transaction Operations / Pre Commit:**
  IntegrityValidation Tests will need to be performed prior to committing the Transaction.

4.2 Test Scheduling

The Validation Plug-In API provides a Plug-In Priority that is used in ordering the execution of tests. This will allow less computationally expensive tests a chance to fail Features early, giving the more expensive tests less work to do.

4.3 Reporting Results

The Validation Processor does not produce single pass or fail, but many.

The Web Feature Server Implementation Specification allows transactions to complete with partial success (multiple failures in a single request).

5 VALIDATION PLUG-IN API

The Validation Plug-In API defines a small number of interfaces. A Validation Plug-In is implemented as a Java Bean supporting one of the two provided Interfaces: FeatureValidation or IntegrityValidation.

The Java Bean Specification requires:

- Beans provided a parameterless constructor.
- Bean properties access via get/set methods
- Bean Metadata be provided by a BeanInfo class

Java Beans are well supported by development tools and provide an easy starting point for those wishing to create Validation Plug-Ins.

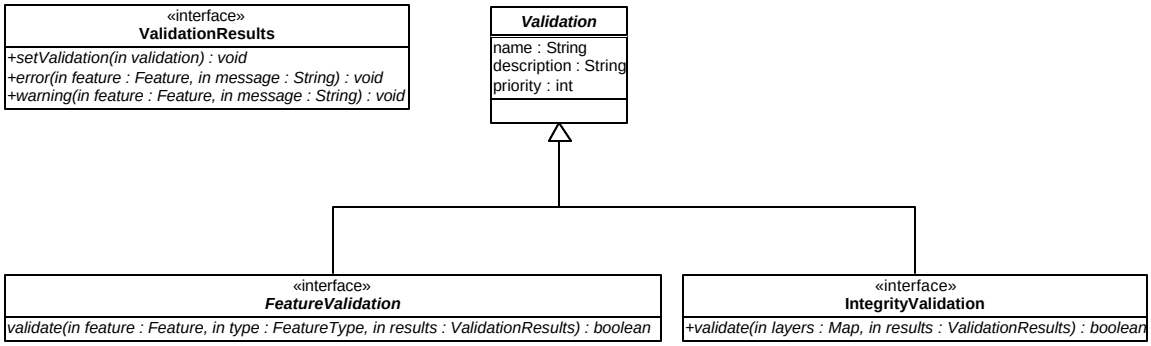

Figure 4: Validation API

The geotools2 library is used to represent feature information. This provides a common representation of feature information with the GeoServer project.

5.1 Validation

```
public interface Validation {
  public String setName( String name );
  public String getName();
  public String setDescription( String description );
  public String getDescription();
  public int getPriority();
  public String[] getFeatureTypes();
}
```

This interface defines how validation tests are described for deployment.

The getFeatureTypes() method is used by the Validation Processor to determine when the Validation Test needs to be applied.

Future work may extend this interface to meet the requirements of a User Interface based configuration tool.

5.2 FeatureValidation

```
public interface FeatureValidation extends Validation {
  public boolean validate( Feature feature,
                           FeatureType type,
                           ValidationResults results
  );
}
```

Defines per feature validation tests used to check new or modified features. The FeatureType is provided to allow access to schema and metadata information.

5.3 IntegrityValidation

```
public interface IntegrityValidation extends Validation{
  public boolean validate( Map layers,
                           ValidationResults results );
}
```

This API is used to test a Spatial Database for consistency.

5.3.1 FeatureType Access

The getFeatureTypes() method provides a list of the FeatureTypes involved in a Integrity Validation Test.

To Execute an IntegrityValidation test the Validation Processor will need to gather Feature Information for the test to run against:

1. For Each Feature Type the Validation Processor will gather the required features into a FeatureCollection.
2. Each Feature Collection will be placed into a Map by FeatureType name.
3. This Map will be used with the validate(layers, results) method to perform the validation test.

How the Validation Processor gathers the feature information will initially be based on bounding boxes defining the effected area.

Should a more specific gathering process be required, we consider changing to a strategy pattern.

5.4 ValidationResults

```
public interface ValidationResults {
   public void setValidation(Validation validation );
   public void error( Feature feature, String message );
   public void warning( Feature feature, String message );
}
```

ValidationResults are used to collate the results of the validation process. An interface is used to allow different applications to provide their own facilities for responding to validation errors.