|  |
| --- |
| User Guide for |
| DuckHawk WFSValidator |
| GML Schema Validation Tool  Stefan Hansen  3/4/2009 |



**LISAsoft Pty Ltd**

**Sydney**

Suite 112, The Lower Deck  
Jones Bay Wharf  
19-21 Pirrama Rd  
Pyrmont NSW Australia 2009

Ph: +61 2 8570 5050  
Fax: +61 2 8570 5099  
  
**Adelaide**Ph: +61 8 8425 8050  
Fax: +61 8 8425 8099  
  
**Melbourne**Ph: +61 3 8680 3250  
Fax: +61 3 8680 3299

User Guide

# Introduction to the Testing Framework DuckHawk

DuckHawk is an open source testing framework that enables the development of automated reliability, load, performance, stress, error-handling and conformance tests. It can be used to automatically test any web service or application.

DuckHawk is written in Java and built upon JUnit 3. Using JUnit test runner functionality allows the framework to integrate with most commonly used build systems such as Maven and Ant and be used for continuous integration testing during product development. The tests themselves are also written in Java and use a design similar to JUnit 3 tests. DuckHawk allows configuring the test parameters through simple configuration files, increasing the flexibility of the developed test systems.

DuckHawk can perform various types of testing (e.g., performance, reliability, etc.), but also provides modules for conformance testing (XML schema validation). Tests using the conformance testing module send requests to the target server and validate the returned XML response documents against an XML Schema.

DuckHawk collects all test results and passes them on to test listeners, which can perform processing, analysis, and generate reports. At the moment two main implementations of test listeners exist; an XML-report generator and a human readable HTML-report generator.

## Requirements

* Java 1.5 or better

# DuckHawk for WFS Validation

DuckHawk has been extended to support GML validation of XML from a WFS server, including validation against locally specified Schematron rules. Besides adding GML Validation specific test classes, the modifications allowed the validation of responses received from the targeted web-services.

## The WFS Validation Test Suite

The WFS Validation test suite provides example test cases that target the modified Duckhawk functionality and Schematron rules. The following section provides a detailed overview of the tests and rules included in DuckHawk’s GML Validation module.

### Tests

DuckHawk WFSValidator comes with a small set of example test cases. The provided examples demonstrate how tests are written and how to set them up. They can be used as templates for creating additional tests. Currently tests for CityGML and AIXM are included.

#### List of Provided Tests

The CityGML tests are described in the following table:

|  |  |
| --- | --- |
| GityGML Feature Type | Parameters |
| utds:Fence | maxFeatures="2" |
| utds:Building | maxFeatures="2" |
| utds:AircraftTransportationComplex | maxFeatures="2" |
| utds:StorageTank | maxFeatures="2" |
| utds:Road | maxFeatures="2" |

The AIXM tests are described in the following table:

|  |  |
| --- | --- |
| AIXM Feature Type | Parameters |
| aixm:AirportClearanceService | maxFeatures="10" |
| utds:VerticalStructure | maxFeatures="10" |
| utds:Runway | maxFeatures="10" |
| utds:Apron | maxFeatures="10" |

### Schematron Rules

Included in the DuckHawk WFSValidator module are several Schematron files containing generic rules for validating the responses of the target servers. Some of these rules are part of OGC standards and some are written to validate general parts of an XML document (e.g., validation of dictionaries).

The CityGML Schematron files are described in the following tables:

|  |  |  |
| --- | --- | --- |
| File Name | Origin | Comment |
| codeSpaceValidation.sch | Created for the OWS6 project | Validates values against referenced code spaces. |
| GMLConstraints.xml | Extracted from the GML specification | This currently not part of the validation process. |
| referentialIntegrity.sch | Taken from the City GML specification |  |
| utds\_test.sch | Created for the OWS6 project |  |

The AIXM Schematron files are described in the following tables:

|  |  |  |
| --- | --- | --- |
| File Name | Origin | Comment |
| codeSpaceValidation.sch | Created for the OWS6 project | Validates values against referenced code spaces. |
| GMLConstraints.xml | Extracted from the GML specification | This currently not part of the validation process. |
| aixm\_test.sch | Created for WFSValidator | To be replaced by a full set of rules called referentialIntegrity.sch. |

## Installing the Test Suite

The test suite comes packed in a ZIP-archive. Extract this archive to a folder on the systems from which you want to run the tests.

### List of Files

The ZIP-archive contains the following files:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | | | Purpose |
| log4j.properties | | | Logging properties |
| WFSValidator.properties | | | Test suit properties |
| jar | WFSValidator-1.0-SNAPSHOT-jar-with-dependencies.jar | | Java application |
| bin | WFSValidator.bat | | Batch file to run the project |
| docs | WFS Validator User Guide 1.0.docx | | This document |
| reports | xml | | Folder for XML reports |
| html | | Folder for HTML reports |
| Schematron | **CityGML** | codeSpaceValidation.sch | Code space validation rules |
| GMLConstraints.xml | Schematron rules for GML (not in use) |
| referentialIntegrity.sch | Schematron rules for City GML |
| iso\_schematron\_skeleton\_for\_saxon.xml | XSL transformation from Schematron to XSLT |
|  | **AIXM** | codeSpaceValidation.sch | Code space validation rules |
|  |  | GMLConstraints.xml | Schematron rules for GML (not in use) |
|  |  | iso\_schematron\_skeleton\_for\_saxon.xml | XSL transformation from Schematron to XSLT |
|  |  | aixm\_test.sch | Schematron rules for AIXM |
| tests | **CityGML** | WFSValidatorGenericTest.csv | Generic test file defining CityGML tests |
|  | **AIXM** | WFSValidatorGenericTest.csv | Generic test file defining AIXM tests |

## Configuring the Test Suite Context

The file WFSValidator.properties defines the context in which the tests will be executed. It is located in the root-directory of the installation.

The parameters listed below have to be set within this file:

|  |  |  |
| --- | --- | --- |
| Parameter | Example | Explanation |
| host | host = 127.0.0.1 | URL of the host. |
| port | Port = 8080 | Port of the server. |
| serverPath | serverPath = deegree2/ogcwebservice | Path to service interface. |
| testsConfigDir | testsConfigDir = AIXM/tests | Location of the test parameters. |
| testsConfigurationFile | testsConfigurationFile = OWS6GenericTest.csv | Name of the file containing the test parameters. If no file is specified, each file in the testsConfigFolder will be treated as a stored response and validated. |
| schematronFolder | schematronFolder = AIXM/schematron/ | Location of Schematron files. |
| schematronFilesExtension | schematronFilesExtension = sch | Files extension of Schematron rule files. |
| schematronTransformer | schematronTransformer = iso\_schematron\_skeleton\_for\_saxon.xml | XSLT file for Schematron transformation. |
| reportXmlDir | reportXmlDir = reports/xml/ | Folder for XML reports. |
| reportHtmlDir | reportHtmlDir = reports/html | Folder for HTML reports. |

## Running the Test Suite

To start the test suite simply execute this command in the root directory of the installation:

**java -jar .\jar\** **WFSValidator-1.0-SNAPSHOT-jar-with-dependencies.jar**

After entering the command the tests will be executed. DuckHawk will generate log-files and reports, but also will output internal information on the screen. After the execution DuckHawk will return to the command-line.

**PLEASE NOTE:** DuckHawk’s internal process uses exceptions to report any errors. Therefore, any validation error will result in the output of an exception message in the console. This does not indicate an error within DuckHawk.

## Output

### Reports

Duckhawk’s WFSValidator module produces two types of reports. All of the data collected during a test run is stored in XML reports. Each time a test suite is executed DuckHawk generates a summary report and a report for each of the executed test classes.

As the main purpose of the XML reports are to provide a machine-readable store the test data, DuckHawk also produces HTML reports. HTML reports also include a summary report as well as reports for each test class. The reports for the test classes are linked from summary report.

### Log Files

During execution of the WFSValidator test suite a log-file is generated. By default it is stored under the name WFSValidator.log in the directory in which the test suite had been started. The default configuration is to backup and reset this log file once it reaches 200KB in size. The name and the maximum size of the log file can be adjusted in the configuration file log4j.properties, which is located in the same directory.

## Adding and Adjusting WFSValidator Tests

The test suite provides a set of example tests and Schematron rules. This will most likely be sufficient for initial testing, however in most cases it will be necessary to adjust and extend the test suite.

### Adjusting Existing Tests

Existing tests can be adjusted by editing the configuration file for that test. The configuration file for each test defines the parameters specific to that test.

The only parameter required for a test in the WFSValidator module is a WFS-request. Each line of the file, except of the first line which defines the name of the parameters, contains one request. The request must be in one single line without any line breaks.

### Adding New Tests

New tests can be added simply by extending WFSValidatorGenericTest.csv. A new, alternative configuration file can also be created. The parameter *testsConfigurationFile* in the WFSValidator.properties controls which configuration file to use.

### Adding Schematron Rules

The test suite verifies each response to a request against the Schematron rules defined in all files with the extension defined by the parameter *schematronFilesExtension* in the Schematron folder defined by the parameter *schematronFolder*.

To add new rules simply add a file containing the rules with the correct extension to the Schematron folder. Responses will be verified against these new rules in the next test run.

# References

The webpage of the DuckHawk Project:

[**http://xircles.codehaus.org/projects/duckhawk**](http://xircles.codehaus.org/projects/duckhawk)

DuckHawk’s Source Repository:

**http://svn.codehaus.org/duckhawk/**