Open Geospatial Consortium Inc.

Date: 22 March 2007

Reference number of this OGC® Project Document: **OGC 07-012**

Version: 1.0

Category: OpenGIS® Discussion Paper

Editors: Jennifer Marcus & Chuck Morris

Compliance Test Engine Interoperability Program Report

Document type:	OpenGIS® Discussion Paper
Document stage:	Approved
Document language:	English

Contents

i. Preface

This document provides an overview of the open source OGC Web Services Compliance Test Engine developed during the OWS4 project.

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained on the OGC website.

This is a draft OGC IPR for review by OGC members and other interested parties. It is a working draft document and may be updated, replaced by other documents at any time. It is inappropriate to use OGC Draft IPRs (DIPRs) as reference material or to cite them as other than "work in progress." This is work in progress and does not imply endorsement by the OGC membership.

This document was developed by the Compliance and Interoperability, Test and Evaluation (CITE) group as part of the OGC Interoperability Program OGC Web Services-4 initiative. The authors of this document are OWS4/CITE participants.

ii. Submitting organizations

This Interoperability Program Report is being submitted to the OGC Interoperability Program by the following organizations:

Northrop Grumman IT, TASC

Galdos

The Open Planning Project

lat/lon

iii. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

CONTACT	COMPANY
Jennifer Marcus	Northrop Grumman IT, TASC
Chuck Morris	Northrop Grumman IT, TASC
Richard Martell	Galdos
Chris Holmes	The Open Planning Project
Jens Fitze	lat/lon

iv. Revision history

Date	Release	Author	Paragraph modified	Description
22 March 2007	1.0	Jennifer Marcus & Chuck Morris	N/A	Initial Release

v. Future Work

N/A

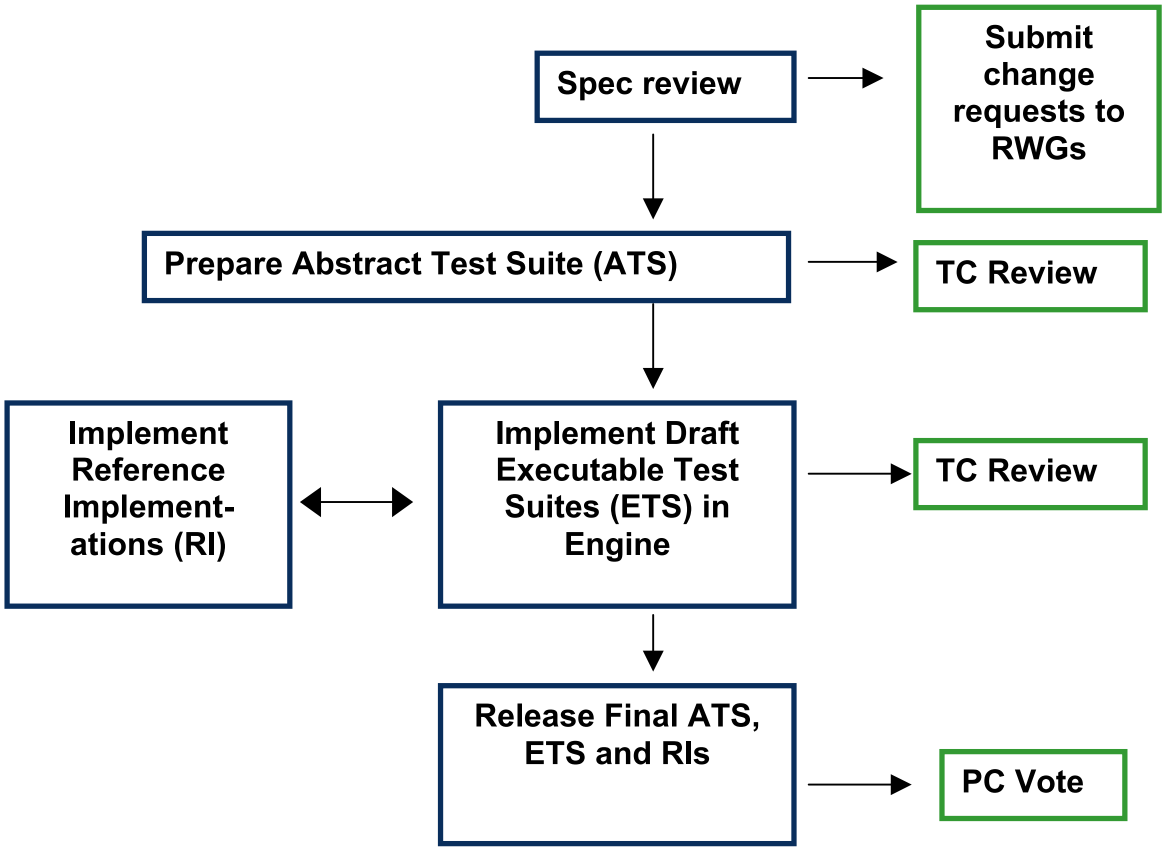
Introduction

As a work item in the OWS4/Conformance and Interoperability Test and Evaluation (CITE) project, Northrop Grumman Information Technology (NGIT) provided an open source web services compliance engine. NGIT refers to this engine as the Test Evaluation And Measurement (TEAM) Engine. It executes scripts written in Compliance Test Language (CTL), a grammar also developed by NGIT. This IPR describes TEAM Engine in detail and provides information on how it was used in OWS-4/CITE.

One of our goals in OWS4/CITE was to compare the behavior of TEAM engine with the behavior of the currently baselined and implemented OGC compliance test engine, developed by The Open Group. To accomplish this, NGIT developed a translator to convert the currently approved Executable Test Suites (ETS), or test scripts, for WMS 1.1.1 and WFS 1.0 into CTL. We installed these test suites in a public installation of TEAM engine and held a beta testing period where we invited OGC members to try them out and provide feedback. This IPR also describes the results of this comparison.

1 Compliance Testing Overview

OGC Compliance Tests are developed via OGC initiatives typically funded by outside sponsors. A participant is selected to develop a compliance test for a specific OGC implementation specification. The participant first documents an Abstract Test Suite (ATS) to indicate what elements of the implementation specification will be tested. The ATS is reviewed by the OGC Technical Committee (TC) community and feedback is incorporated into the ATS. In addition, any ambiguities in the specification are indicated and change proposals against those implementation specifications are developed and worked through the OGC process as needed. From the ATS, an Executable Test Suite (ETS) is developed. The ETS runs within the OGC Compliance Test Engine. The ETS is reviewed by the OGC TC and feedback or appeals from the TC are reviewed. Any required changes are incorporated into the ETS. Reference Implementations (RI) are open source implementations of an OGC Web Service which is 100% compliant with the associated compliance tests. RIs are developed in tight conjunction with development of the ETS. The RI development team collaborates closely with the ETS developer to ensure that the RI is fully compliant to the ETS. At the point at which all feedback has been incorporated into the ETS and the RI is 100% compliant, the compliance package (including the ATS, ETS and RI) are submitted to the OGC Planning Committee (PC) for approval as official OGC compliance tests. The following is a diagram of the previously described process.


Spec review
Submit change requests to RWGs
Prepare Abstract Test Suite (ATS)
TC Review
Implement Reference Implement-ations (RI)
Implement Draft Executable Test Suites (ETS) in Engine
TC Review
Release Final ATS, ETS and RIs
PC Vote

Upon approval by the PC, compliance tests become part of the official OGC Compliance Test Program. The purpose of the OGC Compliance Testing Program is to permit vendors and users to take advantage of the standards that OGC has created. The program provides a process for testing compliance of products to OpenGIS® Implementation Specifications.

This goal of the compliance testing process is to determine that a product implementation of a particular Implementation Specification fulfills all mandatory elements as specified and that these elements are operable. Compliance testing may become more stringent over time, especially as a particular Implementation Specification matures.

OGC compliance tests are available for some but not all OpenGIS Implementation Specifications. OGC hosts tests suites for the following specifications:

AVAILABLE FOR DOWNLOAD	AVAILABLE FOR ONLINE TESTING
Catalogue Service Interface 1.0	**Web Coverage Service 1.0.0**
Coordinate Transformation 1.0	**Web Feature Service 1.0.0**
Gridded Coverages 1.0	**Web Map Service 1.1.1**
Simple Features SQL 1.1	**Catalog Service/Web 2.0.1 (will seek approval Apr 2007)**
Simple Features COM 1.1	**Web Feature Service 1.1 with GML 3.1.1 Simple Features Profile 1.0 validation (will seek approval Apr 2007)**
Simple Features CORBA 1.0	**Web Map Service 1.3 (will seek approval Apr 2007)**
	GeoRSS Validator (will seek approval Apr 2007)
	Web Map Context 1.1.1 (will seek approval Apr 2007)

A vender may use OGC's marks (trademarks or certification marks) to indicate to their customers that they have achieved compliance with OpenGIS Implementation Specifications after completing the following steps:

1. Submitting a Candidate Product to OGC’s Compliance Testing Program,
2. Successfully passing compliance testing,
3. Receiving a certificate stating such success,
4. Receiving a fully executed license from OGC to use the trademark,
5. Paying the trademark license fee, and
6. Providing a fully functional, licensed copy of the tested software to OGC.

Compliance testing, in its present phase does not ensure, or even test, interoperability of software products. However, as the specifications mature the likelihood of interoperability will be higher.

2 Compliance Testing Components

The following are the elements of the OGC Web Services Compliance Test Program

COMPONENT	DESCRIPTION
Compliance Test Engine	Open Source Test Evaluation And Measurement (TEAM) engine
Abstract Test Suite	Testable assertions extracted from specification document; defined as mandatory or optional; test cases are specified independently of any particular test procedure (ISO 19105, 4.4); may be used to create an ETS for a particular test harness
Executable Test Suite	Compliance test which is executed by the Compliance Test Engine
Compliance Test Language (CTL)	Compliance Test Language is an XML grammar for documenting and scripting suites of compliance tests for verifying that an implementation of a specification complies with the specification. A suite of CTL files is installed in the compliance test engine, which executes the scripts and determines whether the implementation being tested passes or fails.
Reference Implementation	Open source implementation of an OGC Web Service which is 100% compliant with the associated compliance tests.
Test Data	Static dataset provided through OGC test program; loaded into service implementation to be tested; this data may be necessary for the ETS to test the service implementation
ETS Translator	Converts OGC compliance tests originally developed for The Open Group Engine into CTL and produces wrappers to enable use of TEAM engine custom functions and translators

The following sections describe the major software components involved in running compliance tests.

2.1 Compliance Test Engine

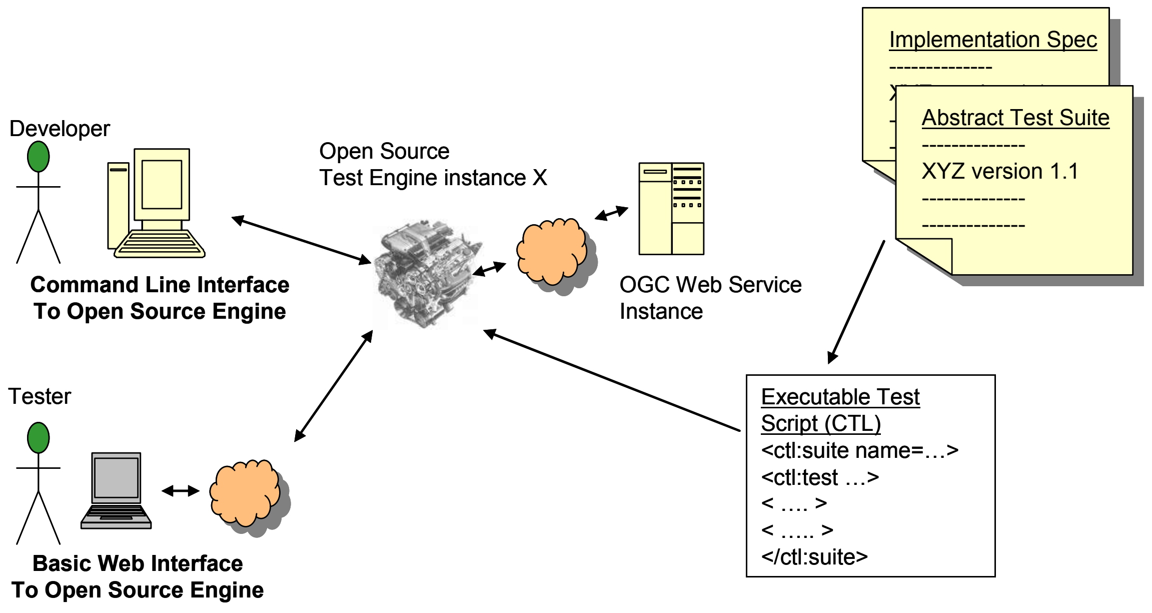
As a work item in the OWS4/Conformance and Interoperability Test and Evaluation (CITE) project, Northrop Grumman Information Technology (NGIT) provided an open source web services compliance engine. NGIT refers to this engine as the Test Evaluation And Measurement (TEAM) Engine.

2.1.1 Compliance Test Engine Technical Description

TEAM Engine is a cross-platform application that runs in a Java Virtual Machine. The core application is easy to install and requires little or no configuration, but it has a command line interface that is more appropriate for developers than end users. However, an optional TEAM Engine Manager application is also available. It is more difficult to install and configure since it runs in the Apache Tomcat servlet container, but it provides an easy-to-use environment for executing test suites over the web. We anticipate that OGC will use this interface to put test suites online.

This engine is an open source software product, available for download from SourceForge at http://sourceforge.net/projects/teamengine.

The following graphic depicts the compliance test engine and its interface to executable tests and services to be tested.


Developer
Command Line Interface To Open Source Engine
Open Source Test Engine instance X
OGC Web Service Instance
Implementation Spec
Abstract Test Suite
XYZ version 1.1
Tester
Basic Web Interface To Open Source Engine
Executable Test Script (CTL)
<ctl:suite name=…>
<ctl:test …>
< …. >
< ….. >
</ctl:suite>

2.1.2 Compliance Test Engine Licensing

TEAM Engine is open source software, released under the Mozilla Public License. No license key is required and the source code is publicly available. Test Script developers can modify the engine if necessary when developing new Executable Test Scripts. Configuration management of the source code will be managed by the OGC through SourceForge.

2.1.3 Compliance Test Script Notation Overview

Compliance Test Language (CTL) is the TEAM Engine script notation. CTL is based on XSL, which allows the use of a full range of looping and decision constructs and variable declarations. Extension functions can be written in XSL or Java using integrated Java datatypes and standard W3C DOM XML datatypes. Tests can be arranged in a hierarchy, allowing reuse of generic tests and reducing the number of times requests must be made to the service. These advantages should lead to a flexible, intuitive, open and easy-to-maintain test scripts.

A detailed description of the CTL was released as a Discussion paper, OGC document 06-126. It can be downloaded from the OGC Network at http://portal.opengeospatial.org/files/?artifact_id=16860.

2.1.4 Core Engine Installation

We recommend that script developers start with the core command line application. It is available from the teamengine-bin archive on SourceForge. Installation and use instructions can be found at http://teamengine.svn.sourceforge.net/viewvc/*checkout*/teamengine/branches/ows4/docs/teamengine.html. The same information is also included in Annex A of this document.

For some sets of tests you may need to increase the heap space allocated to the Java VM before running the tests. You can do this by adding "-Xmx256m" to the JAVA_OPTS variable in setenv.bat or setenv.sh in the teamengine/bin directory.

2.1.5 Adding an ETS to the Compliance Engine

An ETS consists of CTL Scripts, any Java resource files they may depend on, and related files to be installed online with the web version of the engine. Each ETS should be contained in its own subdirectory under the scripts directory of the engine distribution. An ETS should be distributed as an archive that may be installed by unpacking it into the scripts directory.

Some suites may depend on additional components that contain Java extensions to the engine. Components should be contained in a subdirectory under the components directory of the engine distribution. A component should be distributed as an archive that may be installed by unpacking it into the components directory.

Details describing how to package an ETS can be found in Annex C.

2.1.6 Manager Web Application Installation

The TEAM Engine Manager web application is not distributed in binary form since the binary form is not useful without components and scripts installed. To set up the application, you should assemble the components you need and rebuild from source.

To build from source, unpack the teamengine-bin, teamengine-src, and teamengine-manager-src archives from SourceForge, all into the same directory. Then unpack the scripts and components you want to install into the teamengine directory. Rebuild with Apache Ant, using the build.xml file in the root teamengine directory. This will create teamengine-manager-bin archive, with a configuration file tailored with the ETS's that you have installed. See Appendix D for more details on building releases.

To install the teamengine-manager-bin archive, see http://teamengine.svn.sourceforge.net/viewvc/*checkout*/teamengine/branches/ows4/docs/manager.txt . The same information is also included in Annex B of this document.

2.1.7 Configuration Management

The source code for TEAM Engine is controlled in a Subversion repository at SourceForge (http://teamengine.svn.sourceforge.net). The SourceForge project is administered by TASC. Upon request, TASC can grant access to other developers who want to contribute to the project.

The development during the OWS-4 project took place in the ows4 branch in the repository. To make building and executing the OGC test suites easier, this branch contains external properties that reference the script files in the OGC Subversion repository. This means that when you check out the engine code, you will get the script files as well, installed at the proper location so they will be visible to the engine.

More information on how the engine and script files are structured for OGC development is detailed in Annex D below.

2.2 Executable Test Suites

The following is a list of executable test suites developed as part of the OWS4 project, and the organizations responsible for their development.

- WMS 1.3.0 (TASC)
- WFS 1.1.0 (Galdos)
- CS/W 2.0.1 (Galdos)
- WMC 1.1.0 (TASC)

In addition, the following test suites originally written for the Open Group engine were translated into CTL for use with TEAM Engine.

- WMS 1.1.1
- WFS 1.0.0

The executable test suites are available from the OGC Subversion repository (account required) at https://svn.opengeospatial.org:8443/ogc-projects/cite/trunk/.

Also, OWS4/CITE participant lat/lon developed an application which automatically derives a large ETS for intensive WMS testing from a capabilities document. Here, CTL and the engine were found to be flexible enough to allow easy loading and execution this random test scripts. This could be used for automatic testing of service instances.

2.3 Reference Implementations

The development of compliance test suites plays a crucial role in the implementation of interoperable Spatial Data Infrastructures. Open Source Reference Implementations support this task by "testing the tester" and working as freely available example implementations for all kinds of implementers.

OGC has defined a reference implementation as an open source, fully functional implementation of a specification in reference to which other implementations can be evaluated. The OGC provides open source reference implementations for every implementation specification that has a compliance test. This is to ensure maximum transparency of its specifications for both vendors and customers.

Several reference implementations were developed as part of the OWS4 project. They were tested using TEAM Engine with the Executable Test Suites listed in section 2.2.

The following is a list of reference implementations developed as part of the OWS4 project.

- **WMS 1.3**
    - Developed by: lat/lon
    - Product Name: deegree
    - Protected by: GNU Lesser General Public License (GNU LGPL)
    - The deegree WMS is configured to be both a 1.1.1 and a 1.3.0 WMS. The respective GetCapabilities requests are:

        - http://havola.lat-lon.de/deegreeows4wms/services?request=GetCapabilities&version=1.1.1&service=WMS
        - http://havola.lat-lon.de/deegreeows4wms/services?request=GetCapabilities&version=1.3.0&service=WMS
- **WFS 1.1**
    - Developed by: The Open Planning Project (TOPP)
    - Product Name: GeoServer
    - Protected by: GPL 2.0 license
    - http://geo.openplans.org:8080/geoserver/wfs?request=GetCapabilities&version=1.1.0&service=WFS
- **CS/W 2.0.1**
    - Developed by: United Nations Food and Agriculture Organization (UN FAO)
    - Product Name: GeoNetwork
    - Protected by:GNU GPL
    - The GeoNetwork CSW is available online for review and online testing at: http://www.crisalis-tech.com:8081/geonetwork/srv/en/csw?request=GetCapabilities&service=CSW&version=2.0.1

The following describes OGC's officially approved reference implementations prior to development in OWS4:

- **WMS 1.0 & WCS 1.0.0**
    - Developed by: lat/lon
    - Product Name: deegree
    - Protected by: GNU Lesser General Public License (GNU LGPL)
    - http://deegree.sourceforge.net/src/demos.html
- **WFS 1.0**
    - Developed by: The Open Planning Project
    - Product Name: GeoServer

- Protected by: GPL 2.0 license
- http://geoserver.org

3 Engine Comparison

The OGC is currently using a test engine purchased from the Open Group for testing WMS 1.1.1, WFS 1.0.0, and WCS 1.0.0 products. This section examines how this existing engine compares with TEAM Engine.

3.1 Engine Capabilities

Both the Open Group engine and TEAM Engine can be used to test web services, but they use different notations for defining tests. The richness of the script notation determines what script developers are capable of programming the engine to test. TASC, the TEAM Engine developer, submitted the following comparison.

> The CTL script notation used by TEAM Engine is based on XSL, which allows the use of a full range of looping and decision constructs and complex variables which are not supported in the Open Group engine. Extension functions for TEAM Engine can be written in XSL or Java using integrated Java datatypes and standard W3C DOM XML datatypes. Extension functions can be written for the Open Group engine also, but they must be written in Java as Jaxen extension functions, which requires knowledge of the Jaxen XPath interpreter. In TEAM Engine, tests can be arranged in a hierarchy, allowing reuse of generic tests and reducing the number of times requests must be made to the service. This should lead to more flexible, intuitive, open and easier-to-maintain test scripts.

3.2 Behavior Comparison Approach

At the beginning of the OWS4/CITE project, a testing period was established to examine how the translated ETS for WMS 1.1.1 and WFS 1.0 running in the TEAM engine behaved in comparison to these ETS running in the existing Open Group engine. An email was sent out to the TC and PC lists inviting the community to execute the WMS1.1.1 and WFS 1.0 tests from the TEAM engine against services that had previously passed the compliance tests in The Open Group engine. The goal was to evaluate the degree to which the TEAM engine produced the same results as The Open Group engine when the same compliance test was run in each engine. The test period was open to the community from 7-21 July 2006.

3.3 Behavior Comparison Results

Results were tracked and reported in an online issue tracker hosted on the OGC portal. We also received some direct emails including information on issues. All email and issue tracker issues are available online for reference. The issues ranged on the following topics:

1. Browser incompatibilities for Firefox users
2. Configuration mistakes on the TEAM engine
3. Fixes to The Open Group engine that were not implemented correctly and/or existing problems with that engine

4. Tests required fixes to XPath expressions

These issues were overall straightforward to resolve. A brief discussion on each follows:

1. The browser incompatibilities with Firefox were fixed within days and those users could proceed with testing. These problems were not related to inconsistencies between engines.

2. Configuration errors were also remedied quickly so that testing could proceed. These problems were not related to inconsistencies between engines.

3. Two outstanding bugs with the ETS running in The Open Group Engine were brought to light. The test process was a beneficial check and balance for the current state of the OGC compliance test engine which helped the group to resolve some outstanding problems. In addition, this exercise made it evident that a clear policy on maintaining the OGC Compliance Test Engine and ETS is needed. These specific issues are discussed in detail in Section 3.2.

4. A few of the tests which were translated to the new engine did not compile because the XPath expressions in the existing tests were not well formed. The XPath expressions were corrected during translation to the TEAM Engine. It did not appear that these incorrect expressions affected the pass/fail status of the tests in The Open Group Engine.

3.4 Differences in Behavior between Engines

There are two issues where the TEAM engine gives different test results than The Open Group engine.

1. It was found that The Open Group engine does not appear to be validating XML correctly in some cases.

2. The other bug indicated a problem with a production test script. A change proposal was written and approved, but never implemented. This is a case where the production script is too permissive and passes where TEAM engine fails it. This particular test is in the production engine, but wasn't turned on because of an error (an attribute set to false instead of true). The translator translated the script but didn't look at the attribute that turned it off so code was turned on in the translated script. There was a change proposal against The Open Group Engine to turn the attribute to true, but it was not implemented correctly.

3. There was one other issue that indicated a need for a change proposal for an existing WMS 1.1.1 test. This is not an issue where we get a different result in the two engines, but it is an issue that would need to be fixed in both engines.

Overall, the translation of ETS from those running in The Open Group Engine into CTL to execute in the TEAM engine was successful. The resultant recommendation is to move to the TEAM engine as the official OGC Compliance Test Engine. It is also recommended that a prior pass status from The Open Group Engine would be honoured until the time at which an organization needs to renew their compliance test certificate. Any tests run through the new OGC Compliance Test Engine would need to pass all tests in order to gain a certificate of compliance.

4 Summary: Recommendations for OGC Compliance Test Program

RECOMMENDATION	JUSTIFICATION & NEXT STEPS
Migrate to TEAM Engine & Maintain ETS only in TEAM Engine	• Comparison completed and documented • Test developers and testers have seen success and improved usability with TEAM Engine • Open source licensing ensures availability to new test developers • Some errors were discovered in The Open Group engine and there is limited ability to fix problems with TOG Engine within the OGC community • OGC PC should vote to adopt this engine as the official Compliance Test Engine to which all new ETS are developed against • WCS test should be translated and reviewed as soon as feasible
Refine OGC Compliance Test program dispute/resolution process (appeal process) for upgrading and replacing versions of tests and engine	• Successful compliance test program depends on mechanism to mature tests and engine regularly • The policy should • define a process for appealing disputed tests or submitting other change requests to an existing ETS • define procedures for processing change requests and upgrading the ETS if necessary • define whether and when certified products must be retested after a test suite has been upgraded • define a policy for maintenance on test suites for deprecated specifications
Host interoperability event (plug fest)	• Interoperability Events would further ensure the usability and ease of implementation for OGC web services specifications • Successfully passed compliance tests should be entry requirement for Interoperability Events • Implementers and spec editors (RWG members) should both be equally involved • Feedback from Interoperability Events should be fed directly into the OGC Compliance Testing maintenance program

5 Documentation

5.1 OGC Compliance Test Program Documentation

The OGC Compliance Test program documentation can be found at the following URL:

http://www.opengeospatial.org/resource/testing

5.2 OGC OWS4 Compliance Test Thread Documentation

Documentation related to the OWS4 compliance testing efforts, including development of the new open source compliance test engine and test scripts for WFS 1.1, WMS 1.3 and CS/W 2.0.1 can be found via the OGC portal (account required) at the following URL:

https://portal.opengeospatial.org/wiki/twiki/bin/view/OWS4/OWSComplianceTest#Compliance_Test_Engine

These documents are initially draft Interoperability Program Reports. They will be submitted through the OGC document process and request approval as official Interoperability Program Reports (IPRs) at the April 2007 Technical/Planning committee meetings. After they have been approved, the documentation will be available from the OGC web site (www.opengeospatial.org).

5.3 Appeals to OGC Compliance Test Program

Issues or appeals to the OGC compliance tests and the compliance test engine are registered and tracked via OGC's online issue tracker. The issue tracker can be accessed (account required) via the following URL:

http://portal.opengeospatial.org/index.php?m=projects&a=view&project_id=85&tab=6

5.4 Compliance Test Language Documentation

A detailed description of the CTL was released as a Discussion paper, OGC document 06-126 can be downloaded from the OGC Network at http://portal.opengeospatial.org/files/?artifact_id=16860.

Annex A - TEAM Engine Core Installation and Usage

This annex is a copy of the teamengine.html file included in the teamengine-bin archive, available from http://sourceforge.net/projects/teamengine.

1 TEAM Engine

1.1 Overview

The Test, Evaluation, And Measurement (TEAM) Engine is a test script interpreter. It executes test scripts written in Compliance Test Language (CTL) to verify that an implementation of an OGC specification complies with the specification.

1.2 Installation

1. **Install JRE 1.5.0** - You will need a copy of the Java Runtime Environment (JRE), version 1.5.0 or greater (also called JRE 5.0). It is part of the Java 2 Platform Standard Edition (J2SE) 5.0 package. If not already on your system, download this from http://java.sun.com and install it.
2. **Unpack the archive** - Download the TEAM binary distribution archive and unpack it to the location of your choice.
3. **Set environment variables** - The TEAM commands use the JAVA_HOME environment variable to determine the location of the JRE. If it is not already set to the location of your copy of JRE 1.5.0 or greater, edit the file setenv.bat (Windows) or setenv.sh (Unix/Linux) to set the JAVA_HOME variable. It should be set to the JRE directory, which should contain a bin directory that contains the java executable. You may also set a JAVA_OPTS variable to pass other arguments to the Java Virtual Machine (JVM), such as proxy parameters.

1.3 Test Tool

1.3.1 Synopsis

Use the test tool (test.bat for Windows or test.sh for Unix/Linux) to execute CTL scripts. Executing the tool with no command-line parameters will provide a synopsis as shown below:

```
Test mode:
  Use to start a test session.

  test [-mode=test] -source={ctlfile|dir} [-source={ctlfile|dir}] ...
```

```
    [-suite=[{namespace_uri,|prefix:}]suite_name] [-logdir=dir] [-
session=session]

Resume mode:
  Use to resume a test session that was interrupted before completion.

  test -mode=resume -source={ctlfile|dir} [-source={ctlfile|dir}] ...
    -logdir=dir -session=session

Retest mode:
  Use to reexecute individual tests.

  test -mode=retest -source={ctlfile|dir} [-source={ctlfile|dir}] ...
    -logdir=dir test1 [test2] ...

Doc mode:
  Use to generate a list of assertions.

  test -mode=doc -source={ctlfile|dir} [-source={ctlfile|dir}] ...
    [-suite=[{namespace_uri,|prefix:}]suite_name]
```

In this synopsis, optional items are enclosed in brackets ([]). Braces ({}) are used to indicate options when several values are allowed, and a vertical bar (|) is used to separate the options. Option names and values are case sensitive.

1.3.2 Test Mode

Use test mode to start a new test session. You may specify test mode using parameter -mode=test, or you may omit the -mode parameter since it defaults to test mode.

Supply the source CTL file(s) to execute using a -source parameter. Its value may specify a CTL file, or may specify a directory containing multiple CTL files. If it specifies a directory, each file in the directory with a .ctl or a .xml extension is assumed to be a CTL file. Be aware that for each of these files the test tool will create intermediate stylesheet files in the source directory with either .txsl (test, resume, and retest modes) or .dxsl (doc mode) extensions.

You may supply more than one -source parameter. If there are tests in multiple sources with the same qualified name, the tests in the later source location will take precedence.

In test mode, execution begins with one of the test suite elements in the source CTL. If there is just one suite in the CTL sources, the test tool will start there, but if there are multiple suites, you will need to supply a -suite parameter. The suite name may be specified using the suite element□s unqualified name, its prefix qualified name, or its namespace qualified name. For the prefix qualified form, use the same namespace prefix used in the CTL file, followed by a colon, followed by the base suite name. For the namespace qualified form, use the full namespace URI, followed by a comma, followed by the base suite name.

It is not necessary to log your test session, but doing so will allow you to use the log viewer to see details on any tests that failed, recover if the session is interrupted, or reexecute failed tests. To log the session, supply a -logdir parameter. Its value should be a directory on the local file system. If the directory does not exist, it is created.

Inside the log directory, the system will create a session directory where it will store the log files for the new test session. You may supply a name for this directory using the -session parameter or the system will create one automatically. If the session directory already exists, its contents will be overridden.

1.3.3 Resume Mode

Occasionally, a test session may get interrupted. This could happen if the JVM process dies or a test produces a Java exception that halts the test process. If this happens to a test session that is being logged, you can run the test tool again using parameter -mode=resume to continue the process where it left off.

In resume mode, you must supply the same -source parameter(s) and -logdir parameter that were used when the session was started in test mode. You must also identify the session to be resumed with the -session parameter.

1.3.4 Retest Mode

If for any reason you would like to reexecute any of the tests in a logged test session, run the test tool again using parameter -mode=retest. Supply the same -source parameter(s) and -logdir parameter that were used when the session was started in test mode.

Following the named parameters, supply a list of the test(s) to be reexecuted. The tests should be identified by their logged test path, which starts with the session identifier. This is the value shown in parenthesis following the test name in the test results or test summary log.

1.3.5 Doc Mode

Doc mode does not actually execute any of the CTL code. It produces a list of the assertions in a test suite for documentation purposes.

To run the tool in doc mode, specify parameter -mode=doc. Supply -source parameter(s) and, if necessary, a -suite parameter as in test mode.

1.4 Log Viewer

Use the log viewer (viewlog.bat for Windows or viewlog.sh for Unix/Linux) to view test results for logged sessions. Executing the tool with no command-line parameters will provide a synopsis as shown below:

```
To list sessions in a log directory:
  viewlog -logdir=dir

To list tests in a session:
  viewlog -logdir=dir -session=session

To view detailed results for tests:
  viewlog -logdir=dir test1 [test2] ...
```

The parameters for the log viewer are similar to the parameters for the test tool. For a list the sessions that exist in a log directory, supply the -logdir parameter. For a hierarchical list the tests in a test session and their results, supply both the -logdir parameter and the -session parameter. For a detailed result log, including HTTP requests and their results and any other logged messages, supply the -logdir parameter and a list of the test(s) to view, identified by their test path.

1.5 Developer

TEAM engine was developed by Northrop Grumman Corporation, jointly with The National Technology Alliance and is Copyright (C) 2005-2006, Northrop Grumman Corporation. All Rights Reserved.

1.6 License

The TEAM Engine is open source software provided to you under the terms and conditions of the Mozilla Public License, version 1.1. A copy of the license is included in the distribution archive in the text file license.txt. An HTML version of the license is available online at http://www.mozilla.org/MPL/MPL-1.1.html.

TEAM Engine depends on several other open source software packages, which are included in the distribution archive. These dependant packages and their licenses are listed below.

Saxon 8.6.1	Mozilla Public License, Version 1.0
Xerces J 2.8.0	Apache Software License, Version 2.0

Annex B – Installing the TEAM Engine Manager Web Application

This annex is a copy of the manager.txt document included in the teamengine-manager-src archive, available from http://sourceforge.net/projects/teamengine. It describes how to install the teamengine-manager-bin archive, which is created when TEAM Engine is built from source.

```
Installing the TEAM Engine Manager web application.

1) Install JDK 1.5.0 or greater

2) Install Tomcat 5.x or greater

3) Copy the UserFilesRealm.jar file from the setup directory in the
teamengine-manager binary archive into the Tomcat server/lib directory

4) Copy teamengine.war from the setup directory in the teamengine-
manager binary archive into the Tomcat webapps directory.  Tomcat
should expand the archive and create a teamengine directory.

5) Create a directory to store user files.  For example, you might
create a directory called users in the webapps/teamengine/WEB-INF
directory.

6) Update the teamengine context file.  When Tomcat expanded the war
file, it should have created a teamengine.xml context file in the
conf/catalina/localhost directory.  Edit this file and change the root
attribute on the Realm element to the fully qualified path name of the
directory created in step 5.

7) Create a configuration file called config.xml in the
webapps/teamengine/WEB-INF/classes/com/occamlab/te/web directory, using
config.xml.example as a template.  Set the home element to the home URL
for the application.  Set the usersdir element to the directory you
created in step 5.  Create a sources element for each set of source
scripts you have.  Give it a unique id attribute and enclose source
element(s) for each script file and/or directory in the collection.

8) Restart Tomcat.  The TEAM Engine Manager web application should now
be running at http://localhost:8080/teamengine
```

Annex C - Packaging an executable test suite

This annex summarizes the basic packaging conventions for preparing an executable test suite (ETS). An ETS is hierarchically structured as shown below.

```
ets
  +-- /ctl                  -> CTL scripts
  +-- /resources            -> supporting schemas
  +-- /web
        +-- /data           -> test data archive
        +-- /docs           -> supporting XHTML documentation
```

The CTL test scripts are bundled within ctl:package elements that constitute distinct test groups. Test groups correspond to defined capability sets; for example, each conformance class (if defined) should have a corresponding package.

There is always a main package—called main.ctl—that provides a single point of entry for running an ETS. It contains a ctl:suite declaration and the starting test definition. Any package may include other packages as needed using XInclude elements[1] to modularize the test suite:

```
<ctl:package xmlns:ctl="http://www.occamlab.com/ctl">

  <xi:include href="package-1.ctl"/>
  <xi:include href="package-2.ctl"/>
  <!-- etc. -->
</ctl:package>
```

It is generally a good idea to run a few basic readiness tests before starting to run the complete test suite. Such tests should verify the capabilities document and check that the implementation under test is ready to undergo conformance testing (e.g., it has been correctly configured with test data if necessary).

If applicable, test data sets are provided as archives in the web/data directory; these may be downloaded via the web interface as needed. An (X)HTML document in the web/docs directory should briefly describe the test suite and summarize what capabilities are covered.

The WFS 1.1 test suite exemplifies many of these conventions and provides a rich source of examples for test suite developers.

[1] XML Inclusions (XInclude) Version 1.0 (W3C Recommendation): <http://www.w3.org/TR/xinclude/>.

Annex D - Compliance Engine Configuration Management

Test Suite Directories

Test suites should be created in the cite project on OGC's Subversion repository (account required):

https://svn.opengeospatial.org:8443/ogc-projects/cite/

Development may begin directly in the trunk subdirectory. A directory should be created for each specification, and a subdirectory for each version, and an ets subdirectory containing the executable test suite files. The ets directory should contain three sub directories:

ctl	CTL scripts
resources	Resources they depend on
web	Files to add to the web distribution

For example, here is the structure of the WFS 1.1.0 test suite:

trunk/wfs	Specification directory
trunk/wfs/1.1.0	Version subdirectory
trunk/wfs/1.1.0/docs	Documentation (ats, CRs, etc.)
trunk/wfs/1.1.0/ets	Executable Test Suite
trunk/wfs/1.1.0/ets/ctl	CTL scripts
trunk/wfs/1.1.0/ets/resources	Java resources
trunk/wfs/1.1.0/ets/web	Web files
trunk/wfs/1.1.0/ets/web/data/data-wfs-1.1.0.zip	Test dataset archive
trunk/wfs/1.1.0/ets/web/docs/wfs/1.1.0/index.html	Test suite summary web page

Component Directories

It may sometimes be necessary to create Java components to use as custom functions or parsers to extend CTL. It may be appropriate to put generic components into SourceForge, but OGC-specific components should be created in the cite project on OGC's Subversion repository. A new directory should be created for each component package. It should contain an ant build file (build.xml) plus lib, resources, and src subdirectories. For example, here is the structure for the cite1 component in the OGC SVN repository, which contains legacy components from the CITE1 project and is a dependency of the WFS 1.0.0 and WMS 1.1.1 test suites:

trunk/components/	Components directory
trunk/components/cite1	CITE1 component
trunk/components/cite1/build.xml	Ant build file
trunk/components/cite1/lib	Dependant jars

trunk/components/cite1/resources	Java resources
trunk/components/cite1/src	Source code

TEAM Engine Directories

The TEAM Engine code is housed in an SVN repository on SourceForge (http://sourceforge.net/projects/teamengine). If any modifications to this code are necessary for OGC development, a new branch should be created, and development should proceed along that branch. Development during the OWS-4 project has taken place in branches/ows4.

The directory structure is based on the Sun java project conventions, and looks like this:

```
apps
  engine
    lib
    src
      java
    resources
    build.xml
  manager
    lib
    src
      java
    web
      index.jsp
      xxx.jsp
      xxx.html
      WEB-INF
        web.xml
    build.xml
bin
  xxx.bat
  xxx.sh
components
docs
scripts
setup
build.xml
build.properties.example
LICENSE.txt (MPL)
```

After an OGC-specific branch has been set up, it is helpful to link to the OGC-specific components and scripts by setting up externals definitions on the components and scripts directories. Use “svn propset” to set the svn:externals property on the directories to set up the links.

Releases

Building distribution files is an easy, highly automated process.

1. Install a copy of Apache Ant (from http://ant.apache.org/).
2. Checkout the branch you want to build from the teamengine project on SourceForge.
3. Install custom components and scripts, if necessary (May have been done automatically through external links to the OGC SVN repository).
4. Set user-specific properties. Create a build.properties file, using build.properties.example from the SVN repository as an example. You may set the following properties:

Property	**Required**	**Description**
version.stamp	No	Version number used in teamengine directory and .zip archives.
Usersdir	No	Users directory location to be used by manager web app (teamengine.war).
tomcat.dir	Yes, for manager and realm	Location of a local tomcat installation

5. Run ant on the main build file. The following distribution files are generated.
   zips/teamengine{version}-bin.zip
   zips/teamengine{version}-src.zip
   zips/teamengine-manager{version}-src.zip
   zips/teamengine-manager{version}-bin.zip (Binary with scripts already installed)
   zips/ccc.zip (An archive is created for each installed component)
   zips/sss.zip (An archive is created for each installed script)

ANNEX E - TEAM Engine/CTL FAQ

These questions were asked during the OWS4/CITE development phase. They are updated on the CITE wiki on the OGC portal (https://portal.opengeospatial.org/wiki/twiki/bin/view/OWS4/ComplianceEngine)

1. Can an XMLValidatingParser be set up with a predefined set of schemas?

   Yes. Here is an example parser declaration.

```
<parser name="example:XMLValidatingParser">
   <java class="com.occamlab.te.parsers.XMLValidatingParser"
         method="parse" initialized="true">
      <with-param name="schemas_links">
         <parsers:schemas>
            <parsers:schema type="url">
   http://schemas.opengis.net/wms/1.3.0/exceptions_1_3_0.xsd
            </parsers:schema>
            <!-- more schema elements here ... -->
         </parsers:schemas>
      </with-param>
   </java>
</parser>
```

2. Is there a tool to convert the scripts into HTML documentation?

CTL was designed with documentation in mind. Descriptions should be filled in the body of param, context, assertion, comments elements in the test element, and link elements pointing back to a location in an HTML version of the specification should be provided. The assertion element may include references to contextual labels, and if so, label attributes should be supplied on for-each statements and on with-param elements in call-test statements.

We don't have a tool for creating complete HTML documentation from the test scripts yet, but the engine can generate a list of assertions from the test scripts. To generate assertions, run the test utility with the -mode=doc option.

For example, in Annex A of the CTL document, the assertions listed in section A.1 were generated from the tests in section A.2. Notice how text from the label attributes in the code gets substituted into the generated assertion list.

3. Can you evaluate XPath expressions stored in variables?

With straight XPath you can't evaluate expressions stored in variables, but Saxon (the XSLT interpreter TEAM Engine uses) does have an evaluate function you could make use of. For example:

```
<xsl:for-each select="$cap-doc">
    <xsl:value-of select="saxon:evaluate($expression)"
                  xmlns:saxon="http://saxon.sf.net/">
```

```
</xsl:for-each>
```

Strictly speaking, the saxon:evaluate function is not part of the CTL spec, so if someone else were to write a CTL interpreter, code using the evalute function would not work. It should work fine with TEAM Engine though.

4. What is the anchor element in the request instruction supposed to do

The anchor element was an idea that may be misnamed, was never implemented, and may not be very useful anyway. The idea was that it would append a # followed by the contents of the element to the url. It will probably go away unless someone expresses a desire for it.

5. How does the initialized="true" attribute on a java element work?

When you define a parser with initialized="true", it calls the constructor once on initialization, before calling the first test. When the parser is used in a test, it calls the method on the existing initialized object instance rather than creating a new object and initializing it each time.