Open Geospatial Consortium Inc.

Date: 2006-09-06

Reference number of this OGC® document: OGC 06-126

Version: 0.4

Category: OGC® Discussion Paper

Editor: Chuck Morris

Compliance Test Language (CTL) Discussion Paper

Document type:	Candidate OGC® Implementation Specification
Document subtype:	
Document stage:	Public Discussion Paper
Document language:	English

Contents

i. Preface

Suggested additions, changes, and comments on this draft document are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

ii. Document terms and definitions

This document uses the specification terms defined in Subclause 5.3 of [OGC 05-008], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this specification

iii. Submitting organizations

The following organizations submitted this document to the Open Geospatial Consortium Inc.

Northrop Grumman Information Technology, TASC

iv. Document contributor contact points

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

Name	**Organization**
Chuck Morris	Northrop Grumman IT, TASC

v. Revision history

Date	Release	Editor	Primary clauses modified	Description
9/7/06	0.4	Chuck Morris		Posted to OGC Pending Documents page

vi. Changes to the OGC Abstract Specification

The OGC® Abstract Specification does not require changes to accommodate the technical contents of this document.

Foreword

As part of the Temporal Assessment and Evaluation program sponsored by the National Technology Alliance (NTA), Northrop Grumman developed both a script language for writing tests to verify that an implementation of a specification complies with the specification and a test engine to execute the test scripts. This CTL specification documents the language for writing test scripts. The test engine has been released under an open source license, and is available for free download from SourceForge at http://sourceforge.net/projects/teamengine.

The Open Geospatial Consortium is using CTL to develop tests for several specifications as part of the Compliance Testing thread in the OGC Web Services, Phase 4 (OWS-4) Initiative. This document should give implementers a better understanding of how the tests work. Hopefully, this document will lead to earlier and more complete definition and implementation of test suites for other specifications as well.

Introduction

Compliance Test Language is an XML grammar for documenting and scripting suites of tests for verifying that an implementation of a specification complies with the specification.

A suite of CTL files is typically installed in a compliance test engine, which executes the scripts and determines whether the implementation being tested passes or fails. The CTL files can also be used to generate user documentation for the tests in the suite.

Compliance Test Language (CTL)

1 Scope

This document establishes Compliance Test Language, an XML grammar for documenting and scripting suites of tests for verifying that an implementation of a specification complies with the specification.

This document does:

- Define the structure of a test suite and tests in the test suite.
- Define the elements that can be used to form the test script code.
- Define interfaces for producing Java classes to extend the basic CTL functionality.
- Provide informative examples test scripts and sample script output.
- Provide guidelines on terminology used to properly document tests.

It does **not**:

- Define an interface for submitting tests to a test engine
- Define a normative format for logging output from a test engine
- Define a normative format for documentation generated from the test suite

2 Conformance

This specification defines a test suite to check files written in CTL for conformance. A CTL file is conformant if all of the assertions listed in Annex A, section A.1 are true. Code to test the assertions is described in Annex A, section A.2.

3 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document.

W3C REC-html401-19991224, *HTML 4.01 Specification,*
http://www.w3.org/TR/1999/REC-html401-19991224

W3C REC-xhtml-basic-20001219, *XHTML™ Basic*, http://www.w3.org/TR/2000/REC-xhtml-basic-20001219

W3C REC-xslt-19991116, *XSL Transformations (XSLT) Version 1.0*, http://www.w3.org/TR/1999/REC-xslt-19991116

These documents may also contain a list of normative references that are also applicable.

4 Terms and definitions

No special terms or definitions have been identified.

5 Conventions

5.1 Abbreviated terms

CGI	Common Gateway Interface
CTL	Compliance Test Language
HTML	HyperText Markup Language
URL	Universal Resource Locator
W3C	World Wide Web Consortium
XPath	XML Path Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

5.2 Encoding Listings

This document describes the encoding of XML elements using three column tables. The Element/Attribute column presents the elements and attributes in a typical indented XML structure, with a single element or attribute per line. In this column, text in italics indicates text that should be replaced with some other value. A vertical bar (|) is used to separate options when several values are allowed. Bold text indicates a default option.

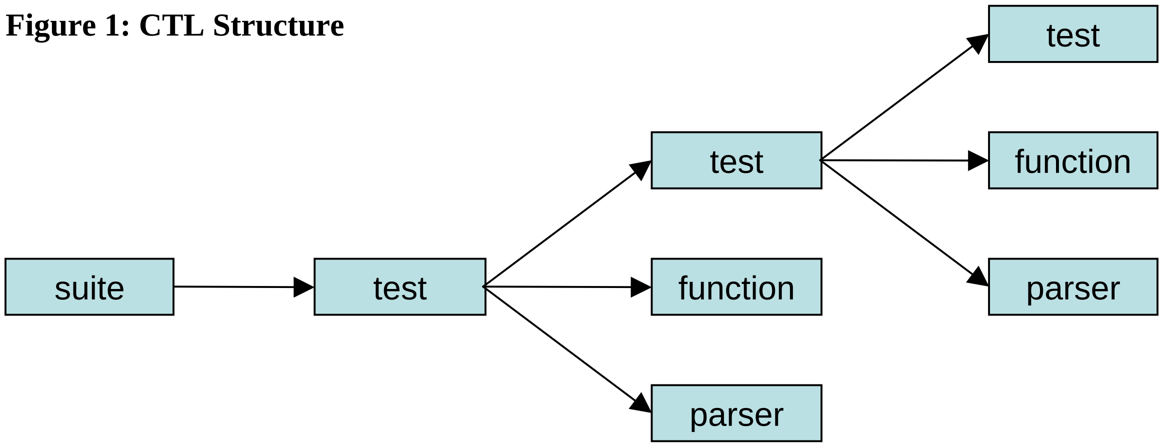
The Usage column indicates whether the element or attribute is required or optional. If other usage constraints apply, they are indicated by footnotes in this column.

The description column gives a brief description of the element or attribute. More detailed descriptions are provided in subsequent sections of the specification.

6 Overview of CTL

A test suite in CTL consists of a set of objects. The initial object is a suite, which identifies a starting test. The starting test contains instructions that may call other tests or use functions and parsers. See Figure 1 below.

Figure 1: CTL Structure



A CTL file is an XML file that contains a CTL object or a package element as its root element. The package element, which is a container for multiple CTL objects, is described in detail in section 7. Each CTL object is identified by a unique, namespace qualified name, so the set of objects in a test suite may span several files. The CTL objects are described in section 8.

Test objects contain programmatic code that consists of XSL instructions and/or CTL instructions. The CTL instructions are described in section 9.

Some parser objects are built-in to CTL, and may be used without declaring a parser object. The built-in parsers are described in section 10.

7 Package

7.1 Introduction

The **<package>** element is a container for CTL objects or traditional XSL templates. Each CTL object is complete on its own and may be placed in its own file, or several objects may be placed in a package for the convenience of grouping them in the same file.

7.2 Encoding

Element/Attribute	**Usage**	**Description**
<package>	Required	Establishes a package
<*suite*|*test*|*function*|*parser*|*xsl:template*/>	Optional*	CTL Objects
</package>		

Usage Notes:
[*] May be repeated.

8 CTL Objects

8.1 Suite

8.1.1 Introduction

The **<suite>** element establishes a set of tests. It names a test suite, describes its purpose, and identifies the starting test where processing begins when the test suite is executed.

8.1.2 Encoding

Element/Attribute	**Usage**	**Description**
<suite	Required	Establishes a test suite
name="*QName*">	Required	Test suite name
<title>*string*</title>	Required	Test suite title
<description>*string*</description>	Optional	Describes the test suite
<starting-test>*QName*</starting-test>	Required	Identifies the first test in the test suite
</suite>		

8.1.3 Example

```
<ctl:suite name="example:suite">
    <ctl:title>Example Test Suite</ctl:title>
    <ctl:description>
        Tests compliance to the Example Service
        Implementation Specification, version 1.0
    </ctl:description>
    <ctl:starting-test>example:main</ctl:starting-test>
</ctl:suite>
```

8.2 Test

8.2.1 Introduction

The **<test>** element establishes an individual test. It states an assertion and contains program code to test the assertion.

8.2.2 Encoding

Element/Attribute	**Usage**	**Description**
<test	Required	Establishes a test
name="*QName*">	Required	Test name
<param	Optional[*]	Test parameters

name="*string*">	Required	Parameter name
string	Optional	Parameter description
</param>		
<context>*string*</context>	Optional	Describes the calling context
<assertion>*string*</assertion>	Required	States the test assertion
<comments>*string*</comments>	Optional*	Comments about the test
<link>*string*</link>	Optional*	Link to a corresponding clause in the specification
<code>	Required	The test code
<*instruction*>	Required*	Test code instructions
</code>		
</test>		

Usage Notes:
* May be repeated.

8.2.3 Element Descriptions

8.2.3.1 Param

Tests may have zero or more parameters specified as **<param>** elements, named with a **name** attribute. The content of the element should describe the parameter.

8.2.3.2 Context

When a test is executed, there is always an XML tree in memory, and a current node within the tree. This is known as the context. In some cases, the context in which a test is called is important, while in other cases in may not matter. If the test code depends on the context, the test should describe the proper context state with a **<context>** element.

8.2.3.3 Assertion

The **<assertion>** element specifies the test assertion. The assertion is a statement derived from the specification being tested which is true for a valid implementation.

The assertion may contain references to the labels of the parameters or the context by using {$*name*} notation. For example, {$context} refers to the current context label, and {$p1} refers to the label of the parameter named "p1". The value of the label may evaluate to the content of the <context> or <param> element, or the value of the label or label-expr attribute on the <for-each> or <with-param> elements, depending on whether documentation is being generated or whether the test is actually being executed.

In some cases, the assertion text may be vague if read in isolation, but its meaning should be clear when viewed in the context of the assertion of the calling test. For example, the assertion "A summary is present" is vague, but if the assertion of the parent test is "HTML tables are properly formatted", then it is clear that the test looks for a summary on an HTML table.

8.2.3.4 Comments

The **<comments>** element allows documentation of additional comments on the purpose of the test or method of verification. The comments should apply to the test as a whole. Of course, XML comments inside <!-- --> tags may also be used, but this comment tag is more formal and may be used in documentation.

8.2.3.5 Link

Zero or more **<link>** elements may be used to provide links back to the place or places in the specification where the assertion was derived from.

8.2.3.6 Code

The **<code>** element contains the programmatic code used to verify the assertion. Any XML is allowed in the code element. It should contain CTL instructions and/or XSL instructions for evaluation by a test engine. The test will pass unless one or more **<fail>** instructions are executed.

8.2.4 Example

```
<ctl:test name="example:zulu">
    <ctl:param name="time">time string</ctl:param>
    <ctl:assertion>
        If the hours field is included in {$time},
        the suffix Z (for zulu) is required.
    </ctl:assertion>
    <ctl:link>wms:B.2.1</ctl:link>
    <ctl:code>
        <xsl:if test="contains($time, 'T')">
            <xsl:variable name="len" select="string-length($time)"/>
            <xsl:if test="not(substring($time, $len) = 'Z')">
                <ctl:fail/>
            </xsl:if>
        </xsl:if>
    </ctl:code>
</ctl:test>
```

8.3 Function

8.3.1 Introduction

The **<function>** element is used to declare user-defined functions or external java functions. These functions may be called as XPath functions in instructions that use XPath expressions, or they may be called directly with the <call-function> instruction.

8.3.2 Encoding

Element/Attribute	**Usage**	**Description**
<function	Required	Establishes a function
name="*QName*">	Required	Function name
<description>*string*</description>	Optional	Describes the function

<param	Optional[*]	Function parameters
name="string">	Optional	Parameter name
string	Optional	Parameter description
</param>		
<var-params	Optional[B]	Indicates the function may be called with a variable number of parameters
min="*int*"	Required	Minimum number of parameters
max="*int*"/>	Required	Maximum number of parameters
<context>*string*</context>	Optional	Describes the calling context
<return>*string*</return>	Optional	Describes what the function returns
<comments>*string*</comments>	Optional[*]	Comments about the function
<code>	Optional[C]	Function is implemented as CTL
<instruction/>	Required[*]	Function code instructions
</code>		
<java	Optional[C]	Function is implemented as Java
class="*string*"	Required	Fully qualified Java class name
method="*string*"	Required	A method in the Java class
initialized="true|**false**">	Optional	Indicates whether the class is initialized. Default is **false**.
<with-param	Optional[D*]	Initialization parameter
name="*string*"	Optional	Parameter name
select="*XPath*">	Optional	Parameter value
XML node	Optional[E]	Parameter value
</with-param>		
</java>		
</function>		

Usage Notes:
[*] May be repeated.
[B] Allowed only if the <java> element is supplied.
[C] Choose between a <code> element or a <java> element.
[D] Allowed only if initialized="true".
[E] If there is a select attribute, the content of the <with-param> element should be empty.

8.3.3 Element Descriptions

8.3.3.1 Param

Functions may have zero or more parameters specified as **<param>** elements, named with a **name** attribute. The content of the element should describe the parameter.

8.3.3.2 Var-params

Some functions implemented as Java methods may be called with a variable number of parameters. This optional element indicates that a variable number of parameters is allowed. It has attributes **min** and **max**, indicating the maximum and minimum number of parameters allowed

8.3.3.3 Context

When a function is executed, there is always an XML tree in memory, and a current node within the tree. This is known as the context. In some cases, the context in which a function is called is important, while in other cases in may not matter. If the function code depends on the context, it should describe the proper context state with a **<context>** element.

8.3.3.4 Return

The optional **<return>** element should be used to provide a brief description of what the function returns.

8.3.3.5 Description

The optional **<description>** element is used to describe the function.

8.3.3.6 Comment

Functions may contain zero or more **<comment>** elements.

8.3.3.7 Code

The **<code>** element is required unless a <java> element is supplied. Any XML is allowed in the code element. It should contain CTL instructions and/or XSL instructions implementing the function.

8.3.3.8 Java

The **<java>** element is required unless a <code> element is supplied. It indicates the function is implemented as an external Java method. It contains **class** and **method** attributes that identify the Java class and method.

For a java method to be used, it must be implemented as a public method. If the method is not static, this must be indicated by setting the **initialized** attribute to “true”. This will initialize the class, calling the class constructor (which also must be public) by passing it the parameter values supplied by the optional <with-param> elements.

There are restrictions on the parameter types that may be used in the java method and the class constructor. They must be one of Java’s built-in primitive types (boolean, char, byte, short, int, long, float, double), String, org.w3c.dom.Node, or org.w3c.dom.NodeList.

If there is a <context> element, it must be the first parameter in the method and must be of type org.w3c.dom.Node or org.w3c.dom.NodeList.

The method may return a primitive type value, or a value of type String, org.w3c.dom.Node, or org.w3c.dom.NodeList.

8.3.4 Examples

```
<function name="example:add">
    <description>Adds two numbers</description>
    <param name="num1">First Number</param>
    <param name="num2">Second Number</param>
    <return>num1 + num2</return>
    <code>
        <xsl:value-of select="$num1 + $num2"/>
    </code>
</function>

<function name="example:sqrt">
    <description>Calculates a square root</description>
    <param name="num"/>
    <return>the square root of num</return>
    <java class="java.lang.Math" method="sqrt"/>
</function>
```

8.4 Parser

8.4.1 Introduction

The **<parser>** element is used to declare a parser and link it to its external java implementation. Parsers are used in the <request> instruction to convert the response from a web service into well-formed XML for processing.

8.4.2 Encoding

Element/Attribute	**Usage**	**Description**
<parser	Required	Establishes a parser
name="*QName*">	Required	Parser name
<description>*string*</description>	Optional	Describes the function
<comments>*string*</comments>	Optional[*]	Comments about the function
<java	Required	Function is implemented as Java
class="*string*"	Required	Fully qualified Java class name
method="*string*"	Required	A method in the Java class
initialized="true|**false**">	Optional	Indicates whether the class is initialized. Default is **false**.
<with-param	Optional[D*]	Initialization parameter
name="*string*"	Optional	Parameter name
select="*XPath*">	Optional	Parameter value
XML node	Optional[E]	Parameter value
</with-param>		

```
  </java>
</parser>
```

Usage Notes:

* May be repeated.

D Allowed only if initialized="true".

E If there is a select attribute, the content of the <with-param> element should be empty.

8.4.3 Element Descriptions

8.4.3.1 Description

The optional **<description>** element is used to describe the parser.

8.4.3.2 Comment

Parsers may contain zero or more **<comment>** elements.

8.4.3.3 Java

The required **<java>** element contains **class** and **method** attributes that identify the Java class and method.

The java method used must be implemented as a public method. If the method is not static, this must be indicated by setting the **initialized** attribute to “true”. This will initialize the class, calling the class constructor (which also must be public) by passing it the parameter values supplied by the optional <with-param> elements.

Parser methods must contain three parameters. The first is of type java.net.URLConnection and contains the URLConnection object used to make the web service request. The second is of type org.w3c.Element, and contains an a copy of the parser element. The third is of type java.io.PrintWriter and can be used to write messages to the log. The parser method should return an XML document of type org.w3c.Document containing parsed XML, a String, or null if it encounters an error.

9 Instructions

9.1 Form

9.1.1 Introduction

The **<form>** instruction is used to retrieve user input. An XHTML form is generated and presented to the user. The user may the fill in the fields on the form and press a submit button. The instruction returns the values of the form fields and the button that was pressed.

9.1.2 Encoding

The form instruction may contain a combination of literal XML elements and/or other CTL or XSL instructions. When any enclosed instructions have been evaluated, the resulting XML shall conform to this encoding.

Element/Attribute	Usage	Description
<form	Required	The form instruction
height="*integer*"	Optional	Desired form height in pixels
width="*integer*">	Optional	Desired form width in pixels
<*XHTML*/>	Optional[*]	XHTML Basic elements
</form>		

Usage Notes:
[*] May be repeated.

9.1.3 Element and Attribute Descriptions

9.1.3.1 Height and Width

The form element contains height and width attributes, used to specify the desired height and width of the form in pixels. The user agent should regard these as hints on the size of window required, but does not have to follow these guidelines.

9.1.3.2 XHTML Basic elements

The content of the form should evaluate into XHTML Basic elements. All of the XHTML Basic elements valid inside the XHTML Basic <form> element are supported, with the exception of the <object> element.

Of particular interest are the XHTML Basic elements that define form controls: <input>, <select>, and <textarea>. These elements can be used to define user options that the form will return. The form must contain at least one <input type=”submit”> element to allow the user to submit the form.

9.1.4 Results

When the user presses a submit button on the form, the instruction completes. It returns the user input as a set of key-value pairs, where the key corresponds to a control name and the value corresponds to the control value. A key-value pair is generated only for successful controls, as formally defined in section 17.13.2 of the HTML 4 specification, the relevant portions of which are presented below.

A successful control must have a control name.

However:

- Controls that are disabled cannot be successful.

- If a form contains more than one submit button, only the activated submit button is successful.
- All "on" checkboxes may be successful.
- For radio buttons that share the same value of the name attribute, only the "on" radio button may be successful.
- For menus, the control name is provided by a SELECT element and values are provided by OPTION elements. Only selected options may be successful. When no options are selected, the control is not successful and neither the name nor any values are submitted to the server when the form is submitted.

If a control doesn't have a current value when the form is submitted, user agents are not required to treat it as a successful control.

Furthermore, user agents should not consider reset buttons successful.

The results shall conform to this encoding:

Element/Attribute	**Usage**	**Description**
<values>	Required	Container for key/value pairs
<value	Optional*	Key-value pairs
key="*string*">	Required	Key name for the pair
string	Required	Value for the pair
</value>		
</values>		

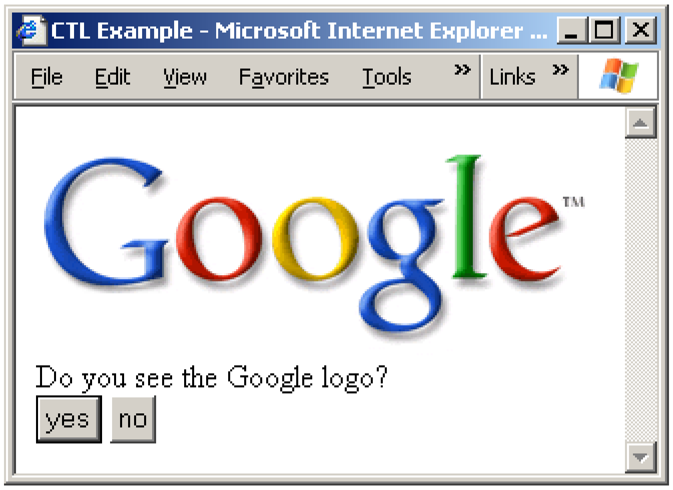
Usage Notes:
* May be repeated.

9.1.5 Example

This form:

```
<ctl:form>
   <p>
      <img src="http://www.google.com/intl/en/images/logo.gif"/><br/>
      Do you see the Google logo?<br/>
      <input type="submit" name="answer" value="yes"/>
      <input type="submit" name="answer" value="no"/>
   </p>
</ctl:form>
```

Will be displayed to the user graphically:


CTL Example - Microsoft Internet Explorer ...
File Edit View Favorites Tools Links
Google™
Do you see the Google logo?
yes no

If the user presses the Yes button, the results will look like this:

```
<values>
   <value key="answer">yes</value>
</values>
```

9.2 Request

9.2.1 Introduction

The **<request>** element submits an HTTP request to a web service or other resource, and returns an XML representation of the response.

9.2.2 Encoding

The request instruction may contain a combination of literal XML elements and/or other CTL or XSL instructions. When any enclosed instructions have been evaluated, the resulting XML shall conform to this encoding.

Element/Attribute	**Usage**	**Description**
<request>	Required	The request instruction
<url>*URL*</url>	Required	Requested web resource
<method>get|post|head</method>	Required	HTTP method for the request
<param	Optional[*]	CGI Parameters to add to the URL
name="*string*">	Required	Parameter name
string	Required	Parameter value
</param>		
<body>*XML*|*string*</body>	Optional[A]	Body for HTTP post requests
<*parser*/>	Optional	Parser instruction for converting the response into XML
</request>		

Usage Notes:
[*] May be repeated.
[A] Only allowed if method is post.

9.2.3 Element and Attribute Descriptions

9.2.3.1 URL

The required <url> element indicates the web service or other HTTP resource where the request will be submitted.

9.2.3.2 Method

The required <method> element indicates the HTTP method that will be used to make the request. Supported method types are get, post, and head.

9.2.3.3 Param

Optional <param> elements may be included to supply CGI parameters for the request. The CGI parameters are added to the URL before the request is sent.

9.2.3.4 Body

The <body> element is used only for requests where the method is set to "post". It contains the data to post when the request is sent. The element may contain character data, which is passed on directly, or XML which is serialized and then passed on.

9.2.3.5 Parser

A parser may be used to instruct the engine how to convert the response from the request into an XML representation. If present, the name of this element should correspond to the name attribute of a parser element (see section 8.4). Parsers may contain various required or optional attributes or elements, depending on the parser implementation. See section 10 for the correct encoding for built-in parsers.

9.2.4 Results

The request element returns an XML representation of the response, as generated by the parser. If no parser element is present and the response is well-formed XML, the XML is returned. Otherwise, nothing is returned.

9.2.5 Example

```
<request>
   <url>http://www.somewms.com</url>
   <method>get</method>
   <param name="SERVICE">WMS</param>
   <param name="REQUEST">GetCapabilities</param>
   <param name="VeRsIoN">1.1.1</param>
</request>
```

9.3 Call-Test

9.3.1 Introduction

The call-test instruction executes a subtest.

9.3.2 Encoding

Element/Attribute	**Usage**	**Description**
<call-test	Required	The call-test instruction
name="*QName*">	Required	Name of subtest to execute
<with-param	Optional[*]	Parameters
name="*string*"	Required	Parameter name
label="*string*"	Optional	Parameter label
label-expr="*XPath*"	Optional	Parameter label expression
select="*XPath*">	Optional	Parameter value
XML node	Optional[A]	Parameter value
</with-param>		
</call-test>		

Usage Notes:
[*] May be repeated.
[A] Should be empty unless the select attribute is supplied

9.3.3 Element and Attribute Descriptions

9.3.3.1 Name

The name attribute identifies the test to execute, and corresponds to the name of a <test> element as described in section 8.2.

9.3.3.2 With-param

If the test to be called contains parameters, values for the parameters may be supplied using <with-param> elements. It is similar to the <xsl:with-param> instruction, but it supports two additional attributes. The optional **label** attribute is used for documentation purposes, to describe what is being passed to the subtest in the current instance of the call-test instruction. The optional **label-expr** attribute is an XPath expression that is calculated at run time to generate a label for the parameter. This value is substituted into the assertion text if the assertion text contains a reference to the parameter.

9.3.4 Results

This instruction does not generate any content, but it will pass on failures from the subtest. In other words, if the subtest fails, the failure bubbles up the call stack, and parent tests will also fail.

9.3.5 Examples

```
<call-test name="example:subtest">
   <with-param name="param1" select=".//table[1]" label="The first table"
               label-expr="concat('Table titled ', caption)"/>
</call-test>

<xsl:for-each select=".//table">
   <call-test name="example:subtest">
      <with-param name="param1" label="Each table"
                  label-expr="concat('Table titled ', caption)">
         <xsl:value-of select="."/>
      </with-param>
   </call-test>
</xsl:for-each>
```

9.4 For-each

9.4.1 Introduction

The for-each instruction loops through each node in a node-set, making it the current node and processing the instructions in the for-each instruction body. It is identical to the xsl:for-each instruction, except that it allows setting labels with the label and label-expr attributes.

9.4.2 Encoding

Element/Attribute	**Usage**	**Description**
<for-each	Required	The for-each instruction
select="*XPath*"	Required	Name of subtest to execute
label="*string*"	Optional	Parameter label
label-expr="*XPath*">	Optional	Parameter label expression
<*instruction/*>	Required*	Parameter value
</for-each>		

Usage Notes:
* May be repeated.

9.4.3 Element and Attribute Descriptions

9.4.3.1 Select

The required select attribute shall specify an XPath expression that evaluates to a node-set.

9.4.3.2 Label

The optional **label** attribute is used for documentation purposes. It describes the context for any enclosed call-test instructions.

9.4.3.3 Label-expr

The optional **label-expr** attribute is an XPath expression that is calculated at run time to generate a label for the context. This value is substituted into the assertion text of any subtests that contains a reference to the context.

9.4.3.4 Instructions

The body of the for-each instruction shall contain CTL instructions and/or XSL instructions. These instructions are processed for each node in the node-set specified by the select attribute, using that node as the current node.

9.4.4 Results

Returns any XML returned by processing the enclosed instructions.

9.4.5 Example

```
<for-each select=".//table" label="Each table">
          label-expr="concat('Table titled ', caption)">
   <call-test name="example:subtest"/>
</for-each>
```

9.5 Message

9.5.1 Introduction

The message instruction presents a message to the user. For instance, it may be used to describe what is wrong when a test fails. It may also be useful for debugging purposes when tests are in the development stage.

The message may be supplied as an XPath expression using the select attribute or it may be provided as the body of the element. In either case, the message presented to the user will be the string value of the selected nodes.

9.5.2 Encoding

Element/Attribute	**Usage**	**Description**
<message	Required	The message instruction
select="*XPath*">	Optional	Name of subtest to execute
message	Optional[A]	Parameter value
</message>		

Usage Notes:
[A] Allowed only if no select attribute is provided.

9.5.3 Element and Attribute Descriptions

9.5.3.1 Select

The optional select attribute specifies an XPath expression. If present, the message will be generated by evaluating the expression and converting it to a string.

9.5.3.2 Message

If there is no select attribute, the content of the message instruction specifies the message. It may contain mixed content, including character data and other instructions. When any enclosed instructions have been evaluated, the result is converted to a string to generate the message.

9.5.4 Results

This instruction does not generate any content.

9.5.5 Examples

These two examples are equivalent:

```
<message select="concat('Table ', caption)"/>
```

```
<message>Table <xsl:value-of select="caption"/></message>
```

9.6 Fail

9.6.1 Introduction

The fail instruction is used to indicate that a test has failed.

9.6.2 Encoding

Element/Attribute	Usage	Description
<fail/>	Required	The fail instruction

9.6.3 Element and Attribute Descriptions

The fail instruction shall not contain any elements or attributes.

9.6.4 Results

When the test processor encounters the fail instruction, it indicates that the test has failed. However, this does not halt test execution. The rest of the instructions will continue to be processed, including any subsequent calls to subtests, but regardless of their results the current test will fail.

The instruction does not generate any content.

9.6.5 Examples

```
<xsl:if test="not(2 + 2 = 4)">
  <message>The laws of addition have failed</message>
  <fail/>
</xsl:if>
```

9.7 Call-function

9.7.1 Introduction

The call-function instruction executes a named function.

9.7.2 Encoding

Element/Attribute	**Usage**	**Description**
<call-function	Required	The call-function instruction
name="*QName*">	Required	Name of function to execute
<with-param	Optional[*]	Parameters
name="*string*"	Optional	Parameter name
select="*XPath*">	Optional	Parameter value
XML node	Optional[A]	Parameter value
</with-param>		
</call-test>		

Usage Notes:
[*] May be repeated.
[A] Should be empty unless the select attribute is supplied

9.7.3 Element and Attribute Descriptions

9.7.3.1 Name

The name attribute identifies the function to execute, and corresponds to the name of a <function> element as described in section 8.3.

9.7.3.2 With-param

If the test to be called contains parameters, values for the parameters may be supplied using <with-param> elements. Parameters must be passed in order. If the optional name attribute is specified, it must match the parameter in the corresponding position in the function definition is named, the names must match. As with the <xsl:with-param> instruction, the parameter value may be specified as an XPath expression using the select attribute or it may be specified using the content of the with-param element.

9.7.4 Results

This instruction returns the value of the function call.

9.7.5 Example

```
<call-function name="example:myfunction">
   <with-param select=”param1”/>
   <with-param><param2/></with-param>
</call-function>
```

10 Built-in parsers

10.1 Introduction

This section describes several parser objects that are built-in to Compliance Test Language and shall be supported by test processors. Tests can use these parsers without declaring them.

10.2 CDataParser

10.2.1 Introduction

CDataParser is a parser that parses the content into character data.

10.2.2 Encoding

Element/Attribute	**Usage**	**Description**
<parsers:CDataParser/>	Required	The parser element

10.3 HTTPParser

10.3.1 Introduction

HTTPParser is a parser that returns HTTP header information uses other parser(s) to parse the content. It supports multipart messages.

10.3.2 Encoding

Element/Attribute	**Usage**	**Description**
<parsers:HTTPParser>	Required	The parser element
<parsers:parse	Optional*	Container for schema elements
part="*integer*"	Optional	Part number parser applies to, for multipart responses
mime="*mimeType*">	Optional	MIME type parser applies to
<*parser*/>	Optional	Parser instruction for converting A content part into XML
</parsers:parse>		
</parsers:HTTPParser>		

Usage Notes:

[*] May be repeated.

10.3.3 Results

10.3.3.1 Standard MIME-type Results

If the Content-Type header of the message is not multipart, the results shall conform to this encoding:

Element/Attribute	**Usage**	**Description**
<response>	Required	Container for the response
<protocol></protocol>		
<status	Required	HTTP status of the response
protocol="*string*"	Required	Protocol/version
code="*integer*">	Required	Status code
string	Required	Status message text
</status>		
<headers>	Required	HTTP headers returned
<header	Optional[*]	HTTP header
name="*string*">	Required	Header name
string	Required	Header value
</header>		
</headers>		
<content>	Optional	Content returned by the selected parser
<xml/>	Optional	Parsed XML node
</content>		
</response>		

Usage Notes:
[*] May be repeated.

10.3.3.2 Multipart MIME-type Results

If the Content-Type header of the message is multipart, the results shall conform to this encoding:

Element/Attribute	**Usage**	**Description**
<multipart-response>	Required	Container for the response
<status	Required	HTTP status of the response
protocol="*string*"	Required	Protocol/version
code="*integer*">	Required	Status code
string	Required	Status message text
</status>		
<headers>	Required	HTTP headers returned
<header	Optional[*]	HTTP header

name="*string*">	Required	Header name
string	Required	Header value
</header>		
</headers>		
<part	Optional *	Container for the response
num="*integer*">	Required	HTTP headers returned
<headers>	Required	HTTP headers returned
<header	Optional*	HTTP header
name="*string*">	Required	Header name
string	Required	Header value
</header>		
</headers>		
<content>	Optional	Content returned by the selected parser
<element/>	Optional	Parsed XML element
</content>		
</part>		
</multipart-response>		

Usage Notes:
* May be repeated.

10.4 XMLValidatingParser

10.4.1 Introduction

XMLValidatingParser is a parser that parses data as XML and validates it against one or more XML Schemas.

10.4.2 Encoding

Element/Attribute	**Usage**	**Description**
<parsers:XMLValidatingParser	Required	The parser element
ignoreErrors="true|**false**"	Optional	Determines behavior if there are validation errors
ignoreWarnings="**true**|false">	Optional	Determines behavior if there are validation warnings
<parsers:schemas>	Optional	Container for schema elements
<parsers:schema	Required*	Schema to validate against
type="url|file|resource">	Optional	Schema type
string	Optional[A]	Schema location
</parsers:schema>		
</parsers:schemas>		

</parsers:XMLValidatingParser>

Usage Notes:
* May be repeated.
A Required if the type attribute is present.

10.4.3 Results

The XMLValidatingParser attempts to parse the data stream returned by a request into XML, and validates the XML against each of the XML Schemas. Any validation errors and warnings are presented to the user. If there are no errors or warnings, or if errors and warning are ignored, the parser returns the XML. If the data stream cannot be parsed or there are errors or warnings that are not ignored, no content is returned.

10.4.4 Example

```
<xsl:variable name="results">
    <request>
        <url>http://www.example.com/example.xml</url>
        <method>get</method>
        <parsers:XMLValidatingParser>
            <parsers:schemas>
                <parsers:schema type="url">
                 http://www.example.com/example.xsd
                </parsers:schema>
            </parsers:schemas>
        </parsers:XMLValidatingParser>
    </request>
</xsl:variable>
<xsl:if test="not($results/*)">
    <message>Parsing or validation failed.</message>
</xsl:if>
```

Annex A
(normative)

Test suite

A.1 Assertions

For a CTL instance document to be conformant, these assertions must be true:

1. **schema-validation**: The CTL instance document validates against the CTL schema file.

2. **validate-function**: Each function is valid.

   2.1. **var-params**: If the function supports a variable number of parameters it is implemented as a java function.

   2.2. **init-params**: If the element is implemented using a java class with initialization parameters its inititialized attribute is set to true.

   2.3. **select**: For each initialization parameter, if the parameter contains a select attribute it does not contain content.

3. **validate-parser**: Each parser is valid.

   3.1. **init-params**: If the element is implemented using a java class with initialization parameters its inititialized attribute is set to true.

   3.2. **select**: For each initialization parameter, if the parameter contains a select attribute it does not contain content.

A.2 Test Code

This specification includes normative test suite code, written in CTL for testing a CTL file for conformance.

These tests are also available online at the URL http://cite.occamlab.com/ctl/0.4/ctl.xml.

```
<package
 xmlns:ct="http://www.occamlab.com/ctl/1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:ctl="http://www.occamlab.com/ctl"
 xmlns:parsers="http://www.occamlab.com/te/parsers"
 xmlns="http://www.occamlab.com/ctl"
>
   <test name="ct:schema-validation">
```

```
       <param name="ctl-doc">A CTL document</param>
       <assertion>{$ctl-doc} validates against the CTL schema
file.</assertion>
       <code>
          <xsl:variable name="reparsed-ctl">
             <parse>
                <content>
                   <xsl:value-of select="ctl-doc"/>
                </content>
                <parsers:XMLValidatingParser>
                   <parsers:schemas>
                      <parsers:schema
type="resource">com/occamlab/te/schemas/ctl.xsd</parsers:schema>
                   </parsers:schemas>
                </parsers:XMLValidatingParser>
             </parse>
          </xsl:variable>
          <xsl:if test="not($reparsed-ctl/*)">
             <ctl:fail/>
          </xsl:if>
       </code>
    </test>

    <test name="ct:validate-function">
       <context>A CTL Function element</context>
       <assertion>{$context} is valid.</assertion>
       <code>
          <call-test name="ct:var-params"/>
          <call-test name="ct:init-params"/>
          <ctl:for-each select="ctl:java/ctl:with-param" label="each
initialization parameter" label-expr="concat('Parameter ', @name)">
             <call-test name="ct:select"/>
          </ctl:for-each>
       </code>
    </test>

    <test name="ct:validate-parser">
       <context>A CTL Parser element</context>
       <assertion>{$context} is valid.</assertion>
       <code>
          <call-test name="ct:init-params"/>
          <ctl:for-each select="ctl:java/ctl:with-param" label="each
initialization parameter" label-expr="concat('Parameter ', @name)">
             <call-test name="ct:select"/>
          </ctl:for-each>
       </code>
    </test>

    <test name="ct:var-params">
       <context>a CTL Function element</context>
       <assertion>If the function supports a variable number of
parameters it is implemented as a java function.</assertion>
       <code>
          <xsl:if test="ctl:var-params and not(ctl:java)">
             <ctl:fail/>
          </xsl:if>
       </code>
    </test>

    <test name="ct:init-params">
       <context>a CTL Function or Parser element</context>
       <assertion>If the element is implemented using a java class with
initialization parameters its inititialized attribute is set to
true.</assertion>
       <code>
          <xsl:if test="ctl:java[ctl:with-param and
not(@initialized='true')]">
             <ctl:fail/>
          </xsl:if>
       </code>
   </test>

   <test name="ct:select">
       <context>A java class initialization parameter</context>
       <assertion>For {$context}, if the parameter contains a select
attribute it does not contain content.</assertion>
       <code>
          <xsl:if test="@select and *">
             <ctl:fail/>
          </xsl:if>
       </code>
   </test>

   <test name="ct:main">
       <assertion>A CTL 1.0 instance document is valid.</assertion>
       <code>
          <xsl:variable name="form-values">
             <ctl:form xmlns="">
                Enter the URL of the CTL 1.0 instance document:<br/>
                <input type="text" name="url"/>
                <br/>
                <input type="submit" value="OK"/>
             </ctl:form>
          </xsl:variable>
          <xsl:variable name="ctl-doc">
             <request>
                <url><xsl:value-of select="$form-
values/values/value[@key='url']"/></url>
                <method>get</method>
             </request>
          </xsl:variable>
          <call-test name="ct:schema-validation">
             <with-param name="ctl-doc" select="$ctl-doc" label="The
CTL instance document" label-expr="'The CTL instance document'"/>
          </call-test>
          <for-each select="$ctl-doc//ctl:function" label="Each
function" label-expr="concat('Function ', @name)">
             <call-test name="ct:validate-function"/>
          </for-each>
          <for-each select="$ctl-doc//ctl:parser" label="Each parser"
label-expr="concat('Parser ', @name)">
             <call-test name="ct:validate-parser"/>
          </for-each>
       </code>
   </test>

   <suite name="ct:validator">
       <title>CTL 1.0 Validator</title>
       <description>Validates a CTL instance document.</description>
       <starting-test>ct:main</starting-test>
    </suite>
</package>
```

Annex B
(normative)

XML schemas

This specification includes a normative XML Schema file, included below. This file is also available online at the URL http://cite.occamlab.com/ctl/0.4/ctl.xsd.

```
<xs:schema
 targetNamespace="http://www.occamlab.com/ctl"
 xmlns:ctl="http://www.occamlab.com/ctl"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 elementFormDefault="qualified">

   <xs:include schemaLocation="ctl_instructions.xsd"/>
   <!-- CTL supports XSLT 1.0 instructions, but since the W3C has not
published an XSLT 1.0 schema
        and 1.0 instructions are forwards-compatible with 2.0
instructions, include the XSLT 2.0 schema -->
   <xs:import namespace="http://www.w3.org/1999/XSL/Transform"
schemaLocation="xslt20.xsd"/>

   <xs:element name="package" type="ctl:packageType"/>
   <xs:complexType name="packageType">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
         <xs:choice>
            <xs:element ref="xsl:template"/>
            <xs:element ref="ctl:function"/>
            <xs:element ref="ctl:parser"/>
            <xs:element ref="ctl:package"/>
            <xs:element ref="ctl:test"/>
            <xs:element ref="ctl:suite"/>
         </xs:choice>
      </xs:sequence>
   </xs:complexType>

   <xs:element name="function" type="ctl:functionType"/>
   <xs:complexType name="functionType">
      <xs:sequence>
         <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:simpleContent>
                  <xs:extension base="xs:string">
                     <xs:attribute name="name" type="xs:string"
                      use="optional"/>
                  </xs:extension>
               </xs:simpleContent>
            </xs:complexType>
         </xs:element>
         <xs:element name="var-params" minOccurs="0">
            <xs:complexType>
```

```
                <xs:simpleContent>
                    <xs:extension base="xs:string">
                        <xs:attribute name="min" type="xs:integer"
                         use="required"/>
                        <xs:attribute name="max" type="xs:integer"
                         use="required"/>
                    </xs:extension>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="context" type="xs:string"
         minOccurs="0"/>
        <xs:element name="return" type="xs:string"
         minOccurs="0"/>
        <xs:element name="description" type="xs:string"
         minOccurs="0"/>
        <xs:element name="comment" type="xs:string"
         minOccurs="0" maxOccurs="unbounded"/>
        <xs:choice>
            <xs:element name="java" type="ctl:javaType"/>
            <xs:element name="code" type="ctl:codeType"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="xs:QName"
     use="required"/>
</xs:complexType>

<xs:element name="parser" type="ctl:functionType"/>

<xs:element name="test" type="ctl:testType"/>
<xs:complexType name="testType">
    <xs:sequence>
        <xs:element name="param" type="ctl:paramType"
         minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="context" type="xs:string"
         minOccurs="0"/>
        <xs:element name="assertion" type="xs:string"/>
        <xs:element name="comment" type="xs:string"
         minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="link" type="xs:string"
         minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="code" type="ctl:codeType"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:QName"
     use="required"/>
</xs:complexType>

<xs:element name="suite" type="ctl:suiteType"/>
<xs:complexType name="suiteType">
    <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="description" type="xs:string"
         minOccurs="0"/>
        <xs:element name="starting-test" type="xs:QName"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:QName"
     use="required"/>
</xs:complexType>

<xs:complexType name="paramType">
    <xs:simpleContent>
```

```
            <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:string"
                 use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

    <xs:complexType name="javaType">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="with-param">
                <xs:complexType mixed="true">
                    <xs:sequence>
                        <xs:any minOccurs="0" processContents="lax"/>
                    </xs:sequence>
                    <xs:attribute name="name" type="xs:string"
                     use="optional"/>
                    <xs:attribute name="select" type="xs:string"
                     use="optional"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="class" type="xs:string"
         use="required"/>
        <xs:attribute name="method" type="xs:string"
         use="required"/>
        <xs:attribute name="initialized" type="ctl:boolean"
         use="optional"/>
    </xs:complexType>

    <xs:complexType name="codeType" mixed="true">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:any processContents="lax"/>
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="boolean">
        <xs:restriction base="xs:string">
            <xs:enumeration value="true"/>
            <xs:enumeration value="false"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```