Learning

This first section contains a tour through XMLUnit's features, the next sections will cover them in more detail. Note that it has a strong focus on using the XMLTestCase class which is one option to use XMLUnit, but not the only one. XMLUnit's features can be fully used without any dependency on JUnit at all. XMLUnit enables JUnit-style assertions to be made about the content and structure of XML1. It is an open source project hosted at http://xmlunit.sourceforge.net/ that grew out of a need to test a system that generated and received custom XML messages. The problem that we faced was how to verify that the system generated the correct message from a known set of inputs. Obviously we could use a DTD or a schema to validate the message output, but this approach wouldn't allow us to distinguish between valid XML with correct content and valid XML with incorrect content. What we really wanted was an assertXMLEqual() method, so we could compare the message that we expected the system to generate and the message that the system actually generated. And that was the beginning of XMLUnit. XMLUnit provides a single JUnit extension class, XMLTestCase, and a set of supporting classes that allow assertions to be made about: The differences between two pieces of XML (via Diff and DetailedDiff classes).The validity of a piece of XML (via Validator class).The outcome of transforming a piece [1] of XML using XSLT (via Transform class).The evaluation of an XPath expression on a piece of XML (via classes implementing the XpathEngine interface).Individual nodes in a piece of XML that are exposed by DOM Traversal (via NodeTest class)XMLUnit can also treat HTML content, even badly-formed HTML, as valid XML to allow these assertions to be made about web pages (via the HTMLDocumentBuilder class).

As with many projects some words in XMLUnit have particular meanings so here is a quick overview. A piece of XML is a DOM Document, a String containing marked-up content, or a Source or Reader that allows access to marked-up content within some resource. XMLUnit compares the expected control XML to some actual test XML. The comparison can reveal that two pieces of XML are identical, similar or different. The unit of measurement used by the comparison is a difference, and differences can be either recoverable or unrecoverable. Two pieces of XML are identical if there are no differences between them, similar if there are only recoverable differences between them, and different if there are any unrecoverable differences between them.

XMLUnit requires a JAXP compliant XML parser virtually everywhere. Several features of XMLUnit also require a JAXP compliant XSLT transformer. If it is available, a JAXP compliant XPath engine will be used for XPath tests. To build XMLUnit at least JAXP 1.2 is required, this is the version provided by the Java class library in JDK 1.4. The JAXP 1.3 (i.e. Java5 and above) XPath engine can only be built when JAXP 1.3 is available. As long as you don't require support for XML Namespaces or XML Schema, any JAXP 1.1 compliant implementations should work at runtime. For namespace and schema support you will need a parser that complies to JAXP 1.2 and supports the required feature. The XML parser shipping with JDK 1.4 (a version of Apache Crimson) for example is compliant to JAXP 1.2 but doesn't support Schema validation. XMLUnit is supposed to build and run on any Java version after 1.3 (at least no new hard JDK 1.4 dependencies have been added in XMLUnit 1.1), but it has only been tested on JDK 1.4.2 and above. To build XMLUnit JUnit 3.x (only tested with JUnit 3.8.x) is required. It is not required at runtime unless you intend to use the XMLTestCase or XMLAssert classes. XMLUnit consists of a few classes all living in the org.custommonkey.xmlunit package. You can use these classes directly from your code, no matter whether you are writing a unit test or want to use XMLUnit's features for any other purpose. This section provides a few hints of where to start if you want to use a certain feature of XMLUnit, more details can be found in the more specific sections later in this document. Heart and soul of XMLUnit's comparison engine is DifferenceEngine but most of the time you will use it indirectly via the Diff class. You can influence the engine by providing (custom) implementations for various interfaces and by setting a couple of options on the XMLUnit class. More information is available in Section 3. All validation happens in the Validator class. The default is to validate against a DTD, but XML Schema validation can be enabled by an option (see Validator.useXMLSchema). Several options of the XMLUnit class affect validation. More information is available in Section 4. The Transform class provides an easy to use layer on top of JAXP's transformations. An instance of this class is initialized with the source document and a stylesheet and the result of the transformation can be retrieved as a String or DOM Document. The output of Transform can be used as input to comparisons, validations, XPath tests and so on. There is no detailed sections on transformations since they are really only a different way to create input for the rest of XMLUnit [2] machinery. Examples can be found in Section 1.6. It is possible to provide a custom [3] javax.xml.transform.URIResolver via the XMLUnit.setURIResolver method. You can access the underlying XSLT transformer via XMLUnit.getTransformerFactory. The central piece of XMLUnit's XPath support is the XpathEngine interface. Currently two implementations of the interface exist, SimpleXpathEngine and org.custommonkey.xmlunit.jaxp13.Jaxp13XpathEngine. SimpleXpathEngine is a very basic implementation that uses your XSLT transformer under the covers. This also means it will expose you to the bugs found in your transformer like the transformer claiming a stylesheet couldn't be compiled for very basic XPath expressions. This has been reported to be the case for JDK 1.5. org.custommonkey.xmlunit.jaxp13.Jaxp13XpathEngine uses JAXP 1.3's javax.xml.xpath package and seems to work more reliable, stable and performant than SimpleXpathEngine. You use the XMLUnit.newXpathEngine method to obtain an instance of the XpathEngine. As of XMLUnit 1.1 this will try to use JAXP 1.3 if it is available and fall back to SimpleXpathEngine. Instances of XpathEngine can return the results of XPath queries either as DOM NodeList or

plain Strings. More information is available in Section 5. To test pieces of XML by traversing the DOM tree you use the NodeTester class. Each DOM Node will be passed to a NodeTester implementation you provide. The AbstractNodeTester class is provided as a NullObject Pattern base class for implementations of your own. More information is available in Section 6. Initially XMLUnit was tightly coupled to JUnit and the recommended approach was to write unit tests by inheriting from the XMLTestCase class. XMLTestCase provides a pretty long list of assert... methods that may simplify your interaction with XMLUnit's internals in many common cases. The XMLAssert class provides the same set of assert...s as static methods. Use XMLAssert instead of XMLTestCase for your unit tests if you can't or don't want to inherit from XMLTestCase. All power of XMLUnit is available whether you use XMLTestCase and/or XMLAssert or the underlying API directly. If you are using JUnit 3.x then using the specific classes may prove to be more convenient.

If you are using a JDK 1.4 or later, your Java class library already contains the required XML parsers and XSLT transformers. Still you may want to use a different parser/transformer than the one of your JDK - in particular since the versions shipping with some JDKs are known to contain serious bugs. As described in Section 1.4 there are two main approaches to choose the XML parser of XSLT transformer: System properties and setters in the XMLUnit class. If you use system properties you have the advantage that your choice affects the whole JAXP system, whether it is used inside of XMLUnit or not. If you are using JDK 1.4 or later you may also want to review the Endorsed Standards Override Mechanism to use a different parser/transformer than the one shipping with your JDK. The second option - using the XMLUnit class - allows you to use different parsers for control and test documents, it even allows you to use different parsers for different test cases, if you really want to stretch it that far. It may also work for JDK 1.4 and above, even if you don't override the endorsed standards libraries. You can access the underlying JAXP parser by XMLUnit.newControlParser, XMLUnit.newTestParser, XMLUnit. getControlDocumentBuilderFactory, XMLUnit.getTestDocumentBuilderFactory and XMLUnit.getSAXParserFactory (used by Validator). Note that all these methods return factories or parsers that are namespace aware. The various build... methods in XMLUnit provide convenience layers for building DOM Documents using the configured parsers. You can also set the class name for the XPathFactory to use when using JAXP 1.3 by passing the class name to XMLUnit- .setXPathFactory.

At the center of XMLUnit's support for comparisons is the DifferenceEngine class. In practice you rarely deal with it directly but rather use it via instances of Diff or DetailedDiff classes (see Section 3.5). The DifferenceEngine walks two trees of DOM Nodes, the control and the test tree, and compares the nodes. Whenever it detects a difference, it sends a message to a configured DifferenceListener (see Section 3.3) and asks a Comparison- Controller (see Section 3.2) whether the current comparison should be halted. In some cases the order of elements in two pieces of XML may not be significant. If this is true, the DifferenceEngine needs help to determine which Elements to compare. This is the job of an ElementQualifier (see Section 3.4). The types of differences DifferenceEngine can detect are enumerated in the DifferenceConstants interface and represented by instances of the Difference class. A Difference can be recoverable; recoverable Differences make the Diff class consider two pieces of XML similar while non-recoverable Differences render the two pieces different. The types of Differences that are currently detected are listed in Table 1 to Table 4 (the first two columns refer to the DifferenceConstants class). Note that some of the differences listed may be ignored by the DifferenceEngine if certain configuration options have been specified. See Section 3.8 for details. DifferenceEngine passes differences found around as instances of the Difference class. In addition to the type of of difference this class also holds information on the nodes that have been found to be different. The nodes are described by NodeDetail instances that encapsulate the DOM Node instance as well as the XPath expression that locates the Node inside the given piece of XML. NodeDetail also contains a ″value″ that provides more information on the actual values that have been found to be different, the concrete interpretation depends on the type of difference as can be seen in Table 5. As said in the first paragraph you won't deal with DifferenceEngine directly in most cases. In cases where Diff or DetailedDiff don't provide what you need you'd create an instance of DifferenceEngine passing a ComparisonController in the constructor and invoke compare with your DOM trees to compare as well as a DifferenceListener and ElementQualifier. The listener will be called on any differences while the control method is executing.

IgnoreTextAndAttributeValuesDifferenceListener doesn't do anything in skippedComparison. It ″downgrades″ Differences of type ATTR_VALUE, ATTR_VALUE_EXPLICITLY_SPECIFIED and TEXT_VALUE to recoverable differences. This means if instances of IgnoreTextAndAttributeValuesDifferenceListener are used together with Diff then two pieces of XML will be considered similar if they have the same basic structure. They are not considered identical, though. Note that the list of ignored differences doesn't cover all textual differences. You should configure XMLUnit to ignore comments and whitespace and to consider CDATA sections and text nodes to be the same (see Section 3.8) in order to cover COMMENT_- VALUE and CDATA_VALUE as well.

Only Elements with the same name - and Namespace URI if present - qualify. In Example 3.4 this means control node 1 will be compared to test node 2. Then control node 2 will be compared to test node 3 because DifferenceEn-

gine will start to search for the matching test Element at the second test node, the same sequence number the control node is at. Control node 3 is compared to test node 3 as well and control node 4 to test node 4.

MultiLevelElementNameAndTextQualifier has in a way been the predecessor of Section 3.4.4. It also matches element names and those of nested child elements until it finds matches, but unlike RecursiveElementNameAndTextQualifier, you must tell MultiLevelElementNameAndTextQualifier at which nesting level it should expect the nested text. MultiLevelElementNameAndTextQualifier's constructor expects a single argument which is the nesting level of the expected text. If you use an argument of 1, MultiLevelElementNameAndTextQualifier is identical to ElementNameAndTextQualifier. In Example 3.5 a value of 2 would be needed. By default MultiLevelElementNameAndTextQualifier will not ignore whitespace between the elements leading up to the nested text. If your piece of XML contains this sort of whitespace (like Example 3.5 which contains a newline and several space characters between) you can either instruct XMLUnit to ignore whitespace completely (see Section 3.8.1) or use the two-arg constructor of MultiLevelElementNameAndTextQualifier introduced with XMLUnit 1.2 and set the ignoreEmptyTexts argument to true. In general RecursiveElementNameAndTextQualifier requires less knowledge upfront and its whitespace-handling is more intuitive.

Diff and DetailedDiff provide simplified access to DifferenceEngine by implementing the ComparisonController and DifferenceListener interfaces themselves. They cover the two most common use cases for comparing two pieces of XML: checking whether the pieces are different (this is what Diff does) and finding all differences between them (this is what DetailedDiff does). DetailedDiff is a subclass of Diff and can only be constructed by creating a Diff instance first. The major difference between them is their implementation of the ComparisonController interface: DetailedDiff will never stop the comparison since it wants to collect all differences. Diff in turn will halt the comparison as soon as the first Difference is found that is not recoverable. In addition DetailedDiff collects all Differences in a list and provides access to it. By default Diff will consider two pieces of XML as identical if no differences have been found at all, similar if all differences that have been found have been recoverable (see Table 1 to Table 4) and different as soon as any non-recoverable difference has been found. It is possible to specify a DifferenceListener to Diff using the overrideDifferenceListener method. In this case each Difference will be evaluated by the passed in DifferenceListener. By returning RETURN_IGNORE- _DIFFERENCE_NODES_IDENTICAL the custom listener can make Diff ignore the difference completely. Likewise any Difference for which the custom listener returns RETURN_IGNORE_DIFFERENCE_NODES_SIMILAR will be treated as if the Difference was recoverable. There are several overloads of the Diff constructor that allow you to specify your piece of XML in many ways. There are overloads that accept additional DifferenceEngine and ElementQualifier arguments. Passing in a DifferenceEngine of your own is the only way to use a ComparisonController other than Diff. Note that Diff and DetailedDiff use ElementNameQualifier as their default ElementQualifier. This is different from DifferenceEngine which defaults to no ElementQualifier at all. To use a custom ElementQualifier you can also use the overrideElementQualifier method. Use this with an argument of null to unset the default ElementQualifier as well.

ANNOTATIONS

1. My first annotation
2. My second annotation
3. My third annotation

XML Units

This first section contains a tour through XMLUnits features [1], the next sections will cover [2] them in more detail. Note that it has a strong focus on using the XMLTestCase class which [3] is one option to use XMLUnit, but not the only one. XMLUnits features can be fully used without any dependency on JUnit at all. XMLUnit enables JUnit-style assertions to be made about the content and structure of XML1. It is an open source project hosted at http://xmlunit.sourceforge.net/ that grew out of a need to test a system that generated and received custom XML messages. The problem that we faced was how to verify that the system generated the correct message from a known set of inputs. Obviously we could use a DTD or a schema to validate the message output, but this approach wouldnt allow us to distinguish between valid XML with correct content and valid XML with incorrect content. What we really wanted was an assertXMLEqual() method, so we could compare the message that we expected the system to generate and the message that the system actually generated. And that was the beginning of XMLUnit. XMLUnit provides a single JUnit extension class, XMLTestCase, and a set of supporting classes that allow assertions to be made about: The differences between two pieces of XML (via Diff and DetailedDiff classes).The validity of a piece of XML (via Validator class).The outcome of transforming a piece of XML using XSLT (via Transform class).The evaluation of an XPath expression on a piece of XML (via classes implementing the XpathEngine interface).Individual nodes in a piece of XML that are exposed by DOM Traversal (via NodeTest class)XMLUnit can also treat HTML content, even badly-formed HTML, as valid XML to allow these assertions to be made about web pages (via the HTMLDocumentBuilder class).

As with many projects some words in XMLUnit have particular meanings so here is a quick overview. A piece of XML is a DOM Document, a String containing marked-up content, or a Source or Reader that allows access to marked-up content within some resource. XMLUnit compares the expected control XML to some actual test XML. The comparison can reveal that two pieces of XML are identical, similar or different. The unit of measurement used by the comparison is a difference, and differences can be either recoverable or unrecoverable. Two pieces of XML are identical if there are no differences between them, similar if there are only recoverable differences between them, and different if there are any unrecoverable differences between them.

XMLUnit requires a JAXP compliant XML parser virtually everywhere. Several features of XMLUnit also require a JAXP compliant XSLT transformer. If it is available, a JAXP compliant XPath engine will be used for XPath tests. To build XMLUnit at least JAXP 1.2 is required, this is the version provided by the Java class library in JDK 1.4. The JAXP 1.3 (i.e. Java5 and above) XPath engine can only be built when JAXP 1.3 is available. As long as you dont require support for XML Namespaces or XML Schema, any JAXP 1.1 compliant implementations should work at runtime. For namespace and schema support you will need a parser that complies to JAXP 1.2 and supports the required feature. The XML parser shipping with JDK 1.4 (a version of Apache Crimson) for example is compliant to JAXP 1.2 but doesnt support Schema validation. XMLUnit is supposed to build and run on any Java version after 1.3 (at least no new hard JDK 1.4 dependencies have been added in XMLUnit 1.1), but it has only been tested on JDK 1.4.2 and above. To build XMLUnit JUnit 3.x (only tested with JUnit 3.8.x) is required. It is not required at runtime unless you intend to use the XMLTestCase or XMLAssert classes. XMLUnit consists of a few classes all living in the org.custommonkey.xmlunit package. You can use these classes directly from your code, no matter whether you are writing a unit test or want to use XMLUnits features for any other purpose. This section provides a few hints of where to start if you want to use a certain feature of XMLUnit, more details can be found in the more specific sections later in this document. Heart and soul of XMLUnits comparison engine is DifferenceEngine but most of the time you will use it indirectly via the Diff class. You can influence the engine by providing (custom) implementations for various interfaces and by setting a couple of options on the XMLUnit class. More information is available in Section 3. All validation happens in the Validator class. The default is to validate against a DTD, but XML Schema validation can be enabled by an option (see Validator.useXMLSchema). Several options of the XMLUnit class affect validation. More information is available in Section 4. The Transform class provides an easy to use layer on top of JAXPs transformations. An instance of this class is initialized with the source document and a stylesheet and the result of the transformation can be retrieved as a String or DOM Document. The output of Transform can be used as input to comparisons, validations, XPath tests and so on. There is no detailed sections on transformations since they are really only a different way to create input for the rest of XMLUnits machinery. Examples can be found in Section 1.6. It is possible to provide a custom javax.xml.transform.URIResolver via the XMLUnit-.setURIResolver method. You can access the underlying XSLT transformer via XMLUnit.getTransformerFactory. The central piece of XMLUnits XPath support is the XpathEngine interface. Currently two implementations of the interface exist, SimpleXpathEngine and org.custommonkey.xmlunit.jaxp13.Jaxp13XpathEngine. SimpleXpathEngine is a very basic implementation that uses your XSLT transformer under the covers. This also means it will expose you to the bugs found in your transformer like the transformer claiming a stylesheet couldnt be compiled for very basic XPath expressions. This has been reported to be the case for JDK 1.5. org.custommonkey.xmlunit.jaxp13.Jaxp13XpathEngine uses JAXP 1.3s javax.xml.xpath package and seems to work more reliable, stable and performant than SimpleXpathEngine. You use the XMLUnit.newXpathEngine method to obtain an instance of the XpathEngine. As of XMLUnit 1.1 this will try to use JAXP 1.3 if it is available and fall back to SimpleXpathEngine. Instances of XpathEngine can return the results of XPath queries either as DOM NodeList or plain Strings. More information is available in Section 5. To test pieces of XML by traversing the

DOM tree you use the NodeTester class. Each DOM Node will be passed to a NodeTester implementation you provide. The AbstractNodeTester class is provided as a NullObject Pattern base class for implementations of your own. More information is available in Section 6. Initially XMLUnit was tightly coupled to JUnit and the recommended approach was to write unit tests by inheriting from the XMLTestCase class. XMLTestCase provides a pretty long list of assert... methods that may simplify your interaction with XMLUnits internals in many common cases. The XMLAssert class provides the same set of assert...s as static methods. Use XMLAssert instead of XMLTestCase for your unit tests if you cant or dont want to inherit from XMLTestCase. All power of XMLUnit is available whether you use XMLTestCase and/or XMLAssert or the underlying API directly. If you are using JUnit 3.x then using the specific classes may prove to be more convenient.

If you are using a JDK 1.4 or later, your Java class library already contains the required XML parsers and XSLT transformers. Still you may want to use a different parser/transformer than the one of your JDK - in particular since the versions shipping with some JDKs are known to contain serious bugs. As described in Section 1.4 there are two main approaches to choose the XML parser of XSLT transformer: System properties and setters in the XMLUnit class. If you use system properties you have the advantage that your choice affects the whole JAXP system, whether it is used inside of XMLUnit or not. If you are using JDK 1.4 or later you may also want to review the Endorsed Standards Override Mechanism to use a different parser/transformer than the one shipping with your JDK. The second option - using the XMLUnit class - allows you to use different parsers for control and test documents, it even allows you to use different parsers for different test cases, if you really want to stretch it that far. It may also work for JDK 1.4 and above, even if you dont override the endorsed standards libraries. You can access the underlying JAXP parser by XMLUnit.newControlParser, XMLUnit.newTestParser, XMLUnit. getControlDocumentBuilderFactory, XMLUnit.getTestDocumentBuilderFactory and XMLUnit.getSAXParserFactory (used by Validator). Note that all these methods return factories or parsers that are namespace aware. The various build... methods in XMLUnit provide convenience layers for building DOM Documents using the configured parsers. You can also set the class name for the XPathFactory to use when using JAXP 1.3 by passing the class name to XMLUnit- .setXPathFactory.

At the center of XMLUnits support for comparisons is the DifferenceEngine class. In practice you rarely deal with it directly but rather use it via instances of Diff or DetailedDiff classes (see Section 3.5). The DifferenceEngine walks two trees of DOM Nodes, the control and the test tree, and compares the nodes. Whenever it detects a difference, it sends a message to a configured DifferenceListener (see Section 3.3) and asks a Comparison- Controller (see Section 3.2) whether the current comparison should be halted. In some cases the order of elements in two pieces of XML may not be significant. If this is true, the DifferenceEngine needs help to determine which Elements to compare. This is the job of an ElementQualifier (see Section 3.4). The types of differences DifferenceEngine can detect are enumerated in the DifferenceConstants interface and represented by instances of the Difference class. A Difference can be recoverable; recoverable Differences make the Diff class consider two pieces of XML similar while non-recoverable Differences render the two pieces different. The types of Differences that are currently detected are listed in Table 1 to Table 4 (the first two columns refer to the DifferenceConstants class). Note that some of the differences listed may be ignored by the DifferenceEngine if certain configuration options have been specified. See Section 3.8 for details. DifferenceEngine passes differences found around as instances of the Difference class. In addition to the type of of difference this class also holds information on the nodes that have been found to be different. The nodes are described by NodeDetail instances that encapsulate the DOM Node instance as well as the XPath expression that locates the Node inside the given piece of XML. NodeDetail also contains a ″value″ that provides more information on the actual values that have been found to be different, the concrete interpretation depends on the type of difference as can be seen in Table 5. As said in the first paragraph you wont deal with DifferenceEngine directly in most cases. In cases where Diff or DetailedDiff dont provide what you need youd create an instance of DifferenceEngine passing a ComparisonController in the constructor and invoke compare with your DOM trees to compare as well as a DifferenceListener and ElementQualifier. The listener will be called on any differences while the control method is executing.

IgnoreTextAndAttributeValuesDifferenceListener doesnt do anything in skippedComparison. It ″downgrades″ Differences of type ATTR_VALUE, ATTR_VALUE_EXPLICITLY_SPECIFIED and TEXT_VALUE to recoverable differences. This means if instances of IgnoreTextAndAttributeValuesDifferenceListener are used together with Diff then two pieces of XML will be considered similar if they have the same basic structure. They are not considered identical, though. Note that the list of ignored differences doesnt cover all textual differences. You should configure XMLUnit to ignore comments and whitespace and to consider CDATA sections and text nodes to be the same (see Section 3.8) in order to cover COMMENT_- VALUE and CDATA_VALUE as well.

Only Elements with the same name - and Namespace URI if present - qualify. In Example 3.4 this means control node 1 will be compared to test node 2. Then control node 2 will be compared to test node 3 because DifferenceEn-

gine will start to search for the matching test Element at the second test node, the same sequence number the control node is at. Control node 3 is compared to test node 3 as well and control node 4 to test node 4.

MultiLevelElementNameAndTextQualifier has in a way been the predecessor of Section 3.4.4. It also matches element names and those of nested child elements until it finds matches, but unlike RecursiveElementNameAndTextQualifier, you must tell MultiLevelElementNameAndTextQualifier at which nesting level it should expect the nested text. MultiLevelElementNameAndTextQualifiers constructor expects a single argument which is the nesting level of the expected text. If you use an argument of 1, MultiLevelElementNameAndTextQualifier is identical to ElementNameAndTextQualifier. In Example 3.5 a value of 2 would be needed. By default MultiLevelElementNameAndTextQualifier will not ignore whitespace between the elements leading up to the nested text. If your piece of XML contains this sort of whitespace (like Example 3.5 which contains a newline and several space characters between) you can either instruct XMLUnit to ignore whitespace completely (see Section 3.8.1) or use the two-arg constructor of MultiLevelElementNameAndTextQualifier introduced with XMLUnit 1.2 and set the ignoreEmptyTexts argument to true. In general RecursiveElementNameAndTextQualifier requires less knowledge upfront and its whitespace-handling is more intuitive.

Diff and DetailedDiff provide simplified access to DifferenceEngine by implementing the ComparisonController and DifferenceListener interfaces themselves. They cover the two most common use cases for comparing two pieces of XML: checking whether the pieces are different (this is what Diff does) and finding all differences between them (this is what DetailedDiff does). DetailedDiff is a subclass of Diff and can only be constructed by creating a Diff instance first. The major difference between them is their implementation of the ComparisonController interface: DetailedDiff will never stop the comparison since it wants to collect all differences. Diff in turn will halt the comparison as soon as the first Difference is found that is not recoverable. In addition DetailedDiff collects all Differences in a list and provides access to it. By default Diff will consider two pieces of XML as identical if no differences have been found at all, similar if all differences that have been found have been recoverable (see Table 1 to Table 4) and different as soon as any non-recoverable difference has been found. It is possible to specify a DifferenceListener to Diff using the overrideDifferenceListener method. In this case each Difference will be evaluated by the passed in DifferenceListener. By returning RETURN_IGNORE- _DIFFERENCE_NODES_IDENTICAL the custom listener can make Diff ignore the difference completely. Likewise any Difference for which the custom listener returns RETURN_IGNORE_DIFFERENCE_NODES_SIMILAR will be treated as if the Difference was recoverable. There are several overloads of the Diff constructor that allow you to specify your piece of XML in many ways. There are overloads that accept additional DifferenceEngine and ElementQualifier arguments. Passing in a DifferenceEngine of your own is the only way to use a ComparisonController other than Diff. Note that Diff and DetailedDiff use ElementNameQualifier as their default ElementQualifier. This is different from DifferenceEngine which defaults to no ElementQualifier at all. To use a custom ElementQualifier you can also use the overrideElementQualifier method. Use this with an argument of null to unset the default ElementQualifier as well.

ANNOTATIONS

1. My first annotation
2. My second annotation
3. My third annotation