abz reporting

ABRA

Documentation

ABZ Reporting GmbH
Berliner Allee 58
64295 Darmstadt / Germany

Table of Contents

1. About ABRA

1.1. XBRL

XBRL (eXtensible Business Reporting Language) is an international standard for financial reporting. Besides the electronic data exchange, the XBRL standard supports the creation, the distribution, the publication and the analysis of business reports. Thus, XBRL offers all the prerequisites to improve the quality of the entire information supply chain and make it more cost efficient.

1.2. Notes About the Current Version: ABRA 3

ABRA is a framework designed for the processing of XBRL documents. The current version of ABRA addresses primarily software developers with the need to process business reporting documents which are compliant to the XBRL 2.1 standard. Besides basic XML skills XBRL knowledge is required to develop XBRL applications with ABRA.

Starting with version 3.4.2 ABRA TE is certified by XBRL International as an XBRL compliant software.

ABRA version 3 has an improved performance and offers many new features:

- Java & .NET implementation
- Instance collections
- Taxonomy cache
- Incremental loading of taxonomy extensions
- XML database connector for storing taxonomies
- In-memory extensions for XML database taxonomies
- XBRL Dimensions 1.0
- XBRL Generic Links 1.0
- XBRL Formula 1.0
- Table Linkbase 1.0

- Taxonomy Packages 1.0
- Fast SAX based DTS discovery
- XML Schema validation
- XBRL 2.1 taxonomy & instance validation
- XBRL Dimensions 1.0 taxonomy & instance validation
- XBRL Formula 1.0 instance validation
- Scalable server integration (parallelizable)
- High level taxonomy information based on the XML Schema infoset
- Configurable instance visualisation (HTML and Excel)

1.3. Design Goals and Architecture

ABRA is conformant to the current XBRL version (2.1). If you are not familiar with XBRL, please have a look at the links section which provides references about tutorials and other XBRL resources. Particularly because of its data model, XBRL is complex but also a very expressive EDI (electronic data interchange) standard. It is based on several XML standards which are provided by the W3C. Hence, lots of XML processing tools exist, like XSLT or XQuery, which can be applied to XBRL documents. However, the manifold XML tools do not suffice for a proper processing of the complex XBRL data model.

The architecture of ABRA was designed to provide ease of use with flexibility to software developers. Thus, Java and C# XML APIs (DOM and SAX) are used. Current version of ABRA also supports XSLT, but its use is highly discouraged since it is planned for removal.

ABRA is designed with memory constraints in mind. We have two implementations for taxonomy loading. One we call "in-memory" which keeps most information in memory while the "persistent" solution persists taxonomy information on disk. Additionally the second time you load a taxonomy as a persistent taxonomy the speedup is significant. Our tests show that the persistent taxonomy implementation only uses 33% of the memory that the "in-memory" implementation does.

1.4. Included Software Components

Please note that Java JDK 8 or higher is required to run ABRA. Furthermore, ABRA utilizes the Apache XML parser Xerces which is included within the distribution. During parse time some XBRL specific DOM Level annotations are added. These annotations significantly speed up the processing time of XBRL documents.

The ABRA software components are developed solely in Java, but equal .NET version is also maintained.

2. Java HowTo

In this section you can find a quick introduction to the basic steps that can be done using ABRA from Java. The code for those examples can be found under the `examples/abraExample` directory. Example project, `abraExample` is easy to setup and have running. You just need to run `gradlew initAbra`, afterwards copy your `abz-abra.lic` license file into directory `license` and finally either `gradlew eclipse` or `gradlew idea` depending on your IDE. After that, you can easily import project into your IDE and execute examples which are written as junit tests.

2.1. Installing a license

Installation of license must happen prior to any other ABRA code execution in order for ABRA library to work. License can be installed in few ways. Installing license as a file

```
File licenseFile = new File(getLicenseFilePath());
LicenseManager licenseManager = LicenseManager.getInstance();
licenseManager.setLicenseFile(licenseFile);
```

Figure 1. Installing license file

installing raw license, note that byte array can be embedded into code directly.

```
File licenseFile = new File(getLicenseFilePath());
FileInputStream fis = new FileInputStream(licenseFile);
byte[] rawLicense = new byte[(int) licenseFile.length()];
try {
    fis.read(rawLicense);
} finally {
    fis.close();
}
LicenseManager licenseManager = LicenseManager.getInstance();
licenseManager.setRawLicense(rawLicense);
```

Figure 2. Installing raw license

Also, classLoader can be provided to look for `abz-abra.lic` license file in classpath. Note, this will only be taken into account if neither license file nor raw license are set.

```
ClassLoader classLoader = this.getClass().getClassLoader();
licenseManager.setClassLoader(classLoader);
```

Figure 3. Setting ClassLoader for license

2.2. Loading a Taxonomy

To load a taxonomy you need to obtain a TaxonomyBuilder and provide the entry files for discovering all taxonomy files.

```
File gaapSchemaFile = new File(getTaxonomiesDirPath(),
        -"de-gaap-ci-2014-04-02/de-gaap-ci-2014-04-02-shell-fiscal.xsd");
File gcdSchemaFile = new File(getTaxonomiesDirPath(), -"de-gcd-2014-04-02/de-gcd-2014-04-02-
shell.xsd");
TaxonomyBuilderFactory taxonomyBuilderFactory = new TaxonomyBuilderFactory();
TaxonomyBuilder taxonomyBuilder = taxonomyBuilderFactory.createInMemoryTaxonomyBuilder();
Taxonomy tax = taxonomyBuilder.withSchema(gaapSchemaFile).withSchema(gcdSchemaFile).build();
```

Figure 4. Taxonomy loading

If taxonomy files are located localy and you want to use them instead of downloading them from the Internet, you can specify catalog file to be used for mapping remote URIs to files on file system.

```
File catalog = new File(getTaxonomiesDirPath(), -"corep-finrep-bbk-2014-03-31/catalog.xml")
        -.getAbsoluteFile();
System.setProperty("xml.catalog.files", catalog.getAbsolutePath());
```

Figure 5. Setting catalog file

You also need to set `CatalogResolver` as `URIResolver` in `TaxonomyBuilder`. In order to use `CatalogResolver`, you will need `xml-resolver.jar` which can be found here.

```
taxonomyBuilder.withURIResolver(new CatalogResolver());
```

Figure 6. Setting CatalogResolver as URIResolver

If you like to validate a taxonomy to conform to XBRL 2.1 rules, use the TaxonomyValidator.

Note

Have in mind that API of taxonomy validator will change in future to return Java objects as result instead of XML.

```
TaxonomyValidatorFactory valFactory = TaxonomyValidatorFactory.newInstance();
TaxonomyValidator validator = valFactory.taxonomyValidator();
Document doc = DOMHelper.newDocument();
validator.validate(tax, new DOMResult(doc));
NodeList errorList = doc.getDocumentElement().getElementsByTagName("error");
System.out.println("Taxonomy validation: -");
if (errorList.getLength() == 0) {
    System.out.println("There are no errors.-");
} else {
    System.out.println("Errors are: -");
    for (int i = 0; i < errorList.getLength(); i++) {
        Node error = errorList.item(i);
        System.out.println(DOMHelper.toString(error));
    -}
}
```

Figure 7. Taxonomy validation

2.3. Reading the Presentation Linkbase Structure

Reading and moving through the structure of taxonomy can be accomplished with class RelationNavigator. First you need to get all the roles of the presentation linkbase of taxonomy.

```
System.out.println("Presentation roles for the taxonomy are:");
ElementFilter presentationFilter = new ElementFilterName(XBRLUtil.XBRL_LINKBASE_NS_URI,
        XBRLUtil.LINK_PRESENTATION_LINK_LN);
List<ElementDocContainer> extendedLinks = tax.getNavigator().-
getExtendedLinks(presentationFilter);
Set<String> roles = new HashSet<String>();
for (ElementDocContainer link -: extendedLinks) {
    String role = link.getElement().getAttributeNS(XBRLUtil.XLINK_NS_URI, XBRLUtil.-
XLINK_ROLE_ATTR);
    roles.add(role);
}
System.out.println(roles);
```

Figure 8. Acquiring all roles from presentation linkbase

Now, for each role you can get root concepts of that role.

```
String roleTcct = -"http://www.xbrl.de/taxonomies/de-gaap-ci/" + -"role/
transfersCommercialCodeToTax";
System.out.println("Root concepts for role \"" + roleTcct + -"\" are:");
RelationElementFilter presentationRootFilter = new ElementFilterNameParentRole(XBRLUtil.-
XBRL_LINKBASE_NS_URI,
        XBRLUtil.LINK_PRESENTATION_ARC_LN, roleTcct);
RelationNavigator nav = tax.getNavigator();
List<ElementDocContainer> rootConcepts = nav.getRootConcepts(nav.-
getRelations(presentationRootFilter));

ArrayList<String> rootNames = new ArrayList<String>();
for (ElementDocContainer concept -: rootConcepts) {
    rootNames.add(tax.getConceptInfoset(concept).getLocalname());
}
System.out.println(rootNames);
```

Figure 9. Acquiring root concepts for specified role

For each concept in a role you can get list of his child concepts. In the code example below, it is shown how to get a list of child concept names for a given role.

```
String conceptLocalName = -"bs.ass";
String roleBS = -"http://www.xbrl.de/taxonomies/" + -"de-gaap-ci/role/balanceSheet";
// de-gaap-ci scheme namespace
String conceptNamespace = -"http://www.xbrl.de/taxonomies/de-gaap-ci-2014-04-02";
System.out.println("Child concepts for concept \"" + conceptLocalName + -"\", role \"" + roleBS
        + -"\" and namespace \"" + conceptNamespace + -" are:");

RelationElementFilter presentationChildConceptsFilter = new -
ElementFilterNameParentRole(XBRLUtil.XBRL_LINKBASE_NS_URI,
        XBRLUtil.LINK_PRESENTATION_ARC_LN, roleBS);
ElementDocContainer rootConcept = tax.getConcept(conceptNamespace, conceptLocalName);
List<ElementDocContainer> children = tax.getNavigator().endpointToEndpoint(rootConcept,
        presentationChildConceptsFilter, Relation.BY_ORDER_ATTRIBUTE, null);
ArrayList<String> childNames = new ArrayList<String>();
for (ElementDocContainer concept -: children)
    childNames.add(tax.getConceptInfoset(concept).getLocalname());

System.out.println(childNames);
```

Figure 10. Acquiring child concepts for specified root concept

At the end, with combination of these RelationNavigator methods you can read the complete concepts tree structure of the presentation linkbase. In this example the tree structure is printed to `out.txt` file.

```
private String printChildConceptsRecursively(Taxonomy tax, RelationElementFilter roleFilter,
        ConceptInfoset parentConcept, String tabulator) {
    String retVal = -"";
    RelationNavigator nav = tax.getNavigator();
    List<ElementDocContainer> children = nav.endpointToEndpoint(parentConcept.getConcept(), -
roleFilter,
            Relation.BY_ORDER_ATTRIBUTE, null);

    for (ElementDocContainer concept -: children) {
        ConceptInfoset conceptInfo = tax.getConceptInfoset(concept);
        String conceptName = conceptInfo.getLocalname();
        retVal += tabulator + conceptName + -"\n";
        retVal += printChildConceptsRecursively(tax, roleFilter, conceptInfo, tabulator -
+ -"\t");
    -}
    return retVal;
}
```

Figure 11. Reading of complete concepts tree structure

2.4. Accessing Concept Properties via ConceptInfoset Interface

This is an example for reading concept properties. See ConceptInfoset's JavaDoc for more information on concept properties.

```
String gaapNS = -"http://www.xbrl.de/taxonomies/de-gaap-ci-2014-04-02";
String conceptLocalName = -"bs.eqLiab.equity.netIncome";
ElementDocContainer concept = tax.getConcept(gaapNS, conceptLocalName);
ConceptInfoset conceptInfo = tax.getConceptInfoset(concept);
System.out.println("ConceptInfoset properties -" + -"for concept with localname: -" + -
conceptInfo.getLocalname()
        + -", and namespace: -" + conceptInfo.getNamespace() + -", are: -");
System.out.println("Balance Type: -" + conceptInfo.getBalance());
System.out.println("Period Type: -" + conceptInfo.getPeriodType());
System.out.println("Xbrl Data Type: -" + conceptInfo.getXbrlBasicType());
System.out.println("Is Abstract: -" + conceptInfo.isAbstract());
System.out.println("Is Duration: -" + conceptInfo.isDuration());
System.out.println("Is Instant: -" + conceptInfo.isInstant());
System.out.println("Is Item: -" + conceptInfo.isItem());
System.out.println("Is Nillable: -" + conceptInfo.isNillable());
System.out.println("Is Numeric: -" + conceptInfo.isNumeric());
```

Figure 12. Reading concept properties

Getting concept's labels can be accomplished in different ways. One way is using ConceptInfoset's method. The method has a String parameter that specifies the language of the returned labels.

```
System.out.println("Labels for english language are -: -");
int i = 1;
for (String label -: conceptInfo.getLabels("en").values()) {
    System.out.println("Label -" + i + -". -: -" + label);
    i++;
}
```

Figure 13. Getting concept's labels per language

Getting all concept labels in all languages can be accomplished using RelationNavigator.

```
System.out.println("All labels in all languages for concept -" + conceptInfo.getLocalname() -
+ -" are: -");
RelationElementFilter presentationFilter = new ElementFilterName(XBRLUtil.XBRL_LINKBASE_NS_URI,
        XBRLUtil.LINK_LABEL_ARC_LN);
RelationNavigator nav = tax.getNavigator();
List<ElementDocContainer> children = nav.endpointToEndpoint(concept, presentationFilter, -
Relation.BY_ORDER_ATTRIBUTE, null);
for (ElementDocContainer element -: children) {
    System.out.println(DOMHelper.getTextContent(element.getElement()));
}
```

Figure 14. Getting concept's labels in all languages

2.5. Instance Document Creation

When creating an instance of document, there are two classes that play together. First InstanceBuilder is used to collect facts and to generate an instance document. FactFactory is used to create the facts that will be added to the InstanceBuilder.

```
XBRLInstances instance = null;
InstanceBuilder builder = new InstanceBuilder(tax);
FactFactory factory = builder.getFactFactory();
```

Figure 15. InstanceBuilder and FactFactory

Before creating facts for instance, you need to set up couple more things. Every fact has a period type, so first it is required to make instant and duration context for facts.

```
DurationContext duration = factory.createDurationContext("http://example.com/
test", -"example1",
        XBRLUtil.convertDateFromXML("2012-01-01"), XBRLUtil.convertDateFromXML("2012-12-31"));
Context instant = factory.createInstantContextFromEndDate(duration);
```

Figure 16. Creating duration and instance context

Unit needs to be defined when creating numeric facts. For example, for a monetary concept you need to create a numeric fact with monetary unit.

```
Unit pureUnit = factory.createPureUnit();
Unit eurUnit = factory.createEuroMonataryUnit();
```

Figure 17. Creating unit

Now you can create facts for the instance document. For every fact you will need to specify concept, period type and value. For numeric facts, unit is also required. This example shows how to create facts for several xbrl data types.

```
ConceptInfoset conceptInfo = tax.getConceptInfoset(gaapNS, -"is.netIncome.otherTaxes");
NumericFact monetaryFact = factory.createNumericFact(conceptInfo, new BigDecimal(123.24), -
duration, eurUnit);

conceptInfo = tax.getConceptInfoset(gaapNS, -"nt.employeesEffectiveDate");
NumericFact decimalFact = factory.createNumericFact(conceptInfo, new BigDecimal(555.55), -
instant, pureUnit);

conceptInfo = tax.getConceptInfoset(gaapNS, -"nt.taxReport");
NonNumericFact stringFact = factory.createNonNumericFact(conceptInfo, -"Some tax report", -
instant);

conceptInfo = tax.getConceptInfoset(gaapNS, -"nt.employeesAveragePeriod");
ItemFact numericNilFact = factory.createNilFact(conceptInfo, duration, pureUnit);

conceptInfo = tax.getConceptInfoset(gaapNS, -"bs.eqLiab.defIncome.comment");
Fact nonNumericNilFact = factory.createNilFact(conceptInfo, instant, null);

conceptInfo = tax.getConceptInfoset(gaapNS, -"nt.directorsSign.date");
CalendarFact dateFact = factory
        -.createDateFact(conceptInfo, XBRLUtil.convertDateFromXML("2012-01-01"), duration);

conceptInfo = tax.getConceptInfoset(gcdNS, -"genInfo.report.autoNumbering");
NonNumericFact boolFact = factory.createBooleanFact(conceptInfo, true, duration);

conceptInfo = tax.getConceptInfoset(gaapNS, -"hbst.transfer.bsAss.name");
QNameFact qnameFact = factory.createQNameFact(conceptInfo, new QName("http://www.namespace.-
com", -"localPart"),
        duration);
```

Figure 18. Creating facts

In case when creating a compound fact (tuple), it is required that all facts that belong to the tuple are already created. Then it is possible to create the compound fact with all child facts as parameter.

```
// creating facts for tuple concept
// nt.relationships.ShareholdersKapCoSpec

conceptInfo = tax.getConceptInfoset(gaapNS, -"nt.relationships.ShareholdersKapCoSpec.-
NameSeat");
NonNumericFact stringTupleFact = factory.createNonNumericFact(conceptInfo, -"name", duration);

conceptInfo = tax.getConceptInfoset(gaapNS, -"nt.relationships.ShareholdersKapCoSpec.-
IssuedKap");
NumericFact monetaryTupleFact = factory.createNumericFact(conceptInfo, new BigDecimal(555.-
55), duration,
        eurUnit);

// making tupleFact
conceptInfo = tax.getConceptInfoset(gaapNS, -"nt.relationships.ShareholdersKapCoSpec");
TupleFact tupleFact = factory.createTupleFact(conceptInfo, stringTupleFact, monetaryTupleFact);
```

Figure 19. Creating tuple fact

For each fact you can add a footnote.

```
// creating footnotes for facts
String textFootnoteVal = -"first value";
String xhtmlFootnoteVal = -"<h1>second value</h1>";
Footnote textFootnote = factory.createFootnote(textFootnoteVal, false, -"en");
Footnote xhtmlFootnote = factory.createFootnote(xhtmlFootnoteVal, true, -"en");

// adding footnotes to facts
monetaryFact.setFootnotes(Arrays.asList(xhtmlFootnote));
stringFact.setFootnotes(Arrays.asList(textFootnote));
```

Figure 20. Adding footnote to fact

As you can see, when adding a xhtml value for a footnote, the second parameter parseAsXml is set to true. Also, you can add same footnote for more than one fact and more than one footnote for a single fact.

At the end you need to add all facts to InstanceBuilder and create the instance document. When calling buildInstance method with all parameters set to null, you will get an instance document that has unnecessary schemaRefs. To avoid this you need to implement an URIReplacer.

```
builder.addFact(monetaryFact);
builder.addFact(decimalFact);
builder.addFact(stringFact);
builder.addFact(numericNilFact);
builder.addFact(nonNumericNilFact);
builder.addFact(dateFact);
builder.addFact(qnameFact);
builder.addFact(boolFact);
builder.addFact(tupleFact);
// implementation of URIReplacer to remove unnecessary
// schemaRefs from document instance
URIReplacer replacer = new URIReplacer() {
    public String replaceURI(String arg0) {
        String gaapEntry = -"de-gaap-ci-2014-04-02-shell-fiscal.xsd";
        String gcdEntry = -"de-gcd-2014-04-02-shell.xsd";
        if (!arg0.contains(gaapEntry) && -!arg0.contains(gcdEntry))
            return null;
        else
            return arg0;
    -}
};

instance = builder.buildInstance(null, null, replacer);

System.out.println("Example instance document is: -");
System.out.println(DOMHelper.toString(instance.getDocument()));
```

Figure 21. Adding all facts and building instance

2.6. Load an Instance Document

To load an instance document for an existing taxonomy, use the already loaded taxonomy. Instance can be loaded from its file.

```
XBRLInstances instance = tax.createInstance(new StreamSource(instanceFile), false, null);
```

Figure 22. Creating instance

2.7. Read Information from an Instance Document

To read information from an already loaded instance use the class InstanceFactProvider. It provides access to the instance facts via the ABRA fact model.

```
InstanceFactProvider instanceFactProvider = new InstanceFactProvider(tax, instance);
```

Figure 23. Accessing instance facts through the ABRA model

To access all facts from the instance use getRootFacts. There is an optional ElementFilter argument, that is used to limit the returned facts.

```
Collection<Fact> facts = instanceFactProvider.getRootFacts(null);
```

Figure 24. Accessing instance facts through the ABRA model

For each fact you can access its properties, like value.

```
printStream.print(fact.getValue());
```

Figure 25. Accessing fact's value

Or you can access the context.

```
Context context = fact.getContext();
printContext(context, level + 1);
```

Figure 26. Accessing fact's contextl

Or you can access the unit - only for numeric facts.

```
if (fact.isNumeric()) {
    Unit unit = ((ImmutableNumericFact)fact).getUnit();
    printUnit(unit, level + 1);
}
```

Figure 27. Accessing fact's unit

2.8. Instance Document Visualization

To visualize an instance document, InstanceVisualisation class is used.

```
InstanceVisualisationFactory visualisationFactory = InstanceVisualisationFactory.newInstance();
InstanceVisualisation instanceVisualisation = visualisationFactory.newVisualiser();
Document out = DOMHelper.newDocument();
instanceVisualisation.visualise(tax, instance, -"de", new DOMResult(out));
String path = -"out.html";
PrintWriter o = new PrintWriter(new OutputStreamWriter(new FileOutputStream(path), -"UTF8"));
System.out.println("Visalisation of document -" + -"instace is located at \"" + path + -"\"");
o.write(DOMHelper.toString(out));
o.close();
```

Figure 28. Instance visualization

This generates a HTML representation of the xbrl instance document.

2.9. Instance Document Validation

To validate an xbrl instance document, InstanceValidator class is used.

```
InstanceValidator validator = tax.newInstanceValidatorBuilder().ignoreTaxonomyRefsChecks()
        -.ignoreDuplicatesChecks().build();
// Activate or deactivate some validation features
InstanceValidationResult result = validator.validate(instance);
System.out.println("Instance document validation: -");
if (result.getInstanceElementErrors().isEmpty()) {
    System.out.println("There are no errors.-");
} else {
    System.out.println("Errors are: -");
    for (InstanceElementError err -: result.getInstanceElementErrors()) {
        System.out.println(err.getCode() + -":" + err.getMessage());
    -}
}
```

Figure 29. Instance validation

This validates the given xbrl file according to the xbrl 2.1 standard.

Also, for xbrl formula 1.0 validation, parameters can be set to instance validator.

```
Map<QName, XdmValue> params = new HashMap<QName, XdmValue>();
params.put(new QName("http://www.eurofiling.info/xbrl/ext/filing-indicators", -"tC_00.01"), -
new XdmAtomicValue(true));
params.put(new QName("ReportingLevel"), new XdmAtomicValue("con"));
ValueProvider<XdmValue> paramValueProvider = new MapValueProvider<XdmValue>(params);
validatorBuilder.withFormulaParameters(paramValueProvider);
```

Figure 30. Setting formula parameters for validation

To list all parameters which are present in taxonomy, use ParameterRepository.

```
ParameterRepository parameterRepository = tax.getParameterRepository();
for(Parameter p -: parameterRepository.list()) {
        System.out.println("Paramter: -" + p.getGlobalName() + -" of type: -" + p.getType() -
+ -" is -"
        + (p.isRequired() -? -"mandatory.-" -: -"not mandatory.-"));
        if(!p.isRequired()) {
            System.out.println("Expression is: -" + p.getValueExpression().getExpression());
        -}
}
```

Figure 31. Listing all parameters from taxonomy

2.10. Fact Aggregation

Abra offers a way for calculating sum or difference for a whole financial statement by utilizing the taxonomy's calculation linkbase. This can be accomplished with the class FactAggregator.

First, initialize FactAggregator and reset calculation roles. The order of the calculation roles is significant as the FactAggregator calculates one role after the another in the given order.

```
String rolePrefix = -"http://www.xbrl.de/taxonomies/de-gaap-ci/role";
String[] roles = new String[] { rolePrefix + -"/notes", rolePrefix + -"/OtherReportElements",
        rolePrefix + -"/managementReport", rolePrefix + -"/contingentLiabilities",
        rolePrefix + -"/changesEquityAccounts", rolePrefix + -"/changesEquityStatement",
        rolePrefix + -"/incomeStatement", rolePrefix + -"/fiscalCalculations", rolePrefix + -"/
balanceSheet",
        rolePrefix + -"/appropriationProfits", rolePrefix + -"/cashFlowStatement" -};
FactAggregator<NumericFact> aggregator = new FactAggregatorImpl<NumericFact>();
aggregator.setCalculationRolesAndReset(tax, roles);
```

Figure 32. FactAggregator initialization

After that, you need to set FactAccesor implementation for the FactAggregator. FactAccesor is used by FactAggregator to access facts that are aggregated and to abstract from the current fact model. So you need to create a FactAccesor and set its unit, instant context and duration context properties.

```
InstanceBuilder builder = new InstanceBuilder(tax);
FactFactory factory = builder.getFactFactory();
FactAccessorImpl acc = new FactAccessorImpl(factory);
DurationContext duration = factory.createDurationContext("http://example.com/
test", -"example1",
        XBRLUtil.convertDateFromXML("2010-01-01"), XBRLUtil.convertDateFromXML("2010-12-31"));
InstantContext instant = factory.createInstantContextFromEndDate(duration);
Unit eurUnit = factory.createEuroMonataryUnit();
acc.setDurationContext(duration);
acc.setInstantContext(instant);
acc.setUnit(eurUnit);
aggregator.setFactAccessor(acc);
```

Figure 33. Assigning FactAccesor to FactAggregator

For other calculations that exist and that are not expressed by the taxonomy's calculation linkbase rules, FactAggregatedListener exists. You can write your own implementation of this listener for any calculation purpose you have. This listener is called whenever a fact value is calculated, even when the value did not change.

```
ElementDocContainer isNetIncome = tax.getConcept(gaapNS, -"is.netIncome");
ElementDocContainer bsNetIncome = tax.getConcept(gaapNS, -"bs.eqLiab.equity.netIncome");
aggregator.addFactAggregatedListener(new TransferFactValueImpl<NumericFact>(isNetIncome, -
bsNetIncome));
```

Figure 34. Adding listener to FactAggregator

In this case, it uses transferFactValue implementation of FactAggregatedListener. It creates a new Fact for concept "Net income/net loss for the financial year"(bs.eqLiab.equity.netIncome)

with the value from fact for concept "Net loss for the year"(is.netIncome) as soon as "Net loss for the year" is calculated. Now you can add some facts to aggregator to see what is calculated.

```
ConceptInfoset conceptInfo = tax.getConceptInfoset(gaapNS,
        -"is.netIncome.regular.fin.netInterest.expenses.valueDiscount");
NumericFact fact = factory.createNumericFact(conceptInfo, new BigDecimal("100.00"),
        conceptInfo.isInstant() -? instant -: duration, eurUnit);
aggregator.addFact(fact);
conceptInfo = tax.getConceptInfoset(gaapNS, -"is.netIncome.regular.fin.netInterest.expenses.-
regularInterest");
fact = factory.createNumericFact(conceptInfo, new BigDecimal("100.00"), conceptInfo.-
isInstant() -? instant
        -: duration, eurUnit);
aggregator.addFact(fact);
conceptInfo = tax.getConceptInfoset(gaapNS, -"is.netIncome.regular.fin.netParticipation.-
earnings");
fact = factory.createNumericFact(conceptInfo, new BigDecimal("1000.00"), conceptInfo.-
isInstant() -? instant
        -: duration, eurUnit);
aggregator.addFact(fact);
conceptInfo = tax.getConceptInfoset(gaapNS, -"is.netIncome.regular.fin.netParticipation.-
earningSecurities");
fact = factory.createNumericFact(conceptInfo, new BigDecimal("1000.00"), conceptInfo.-
isInstant() -? instant
        -: duration, eurUnit);
aggregator.addFact(fact);
conceptInfo = tax.getConceptInfoset(gaapNS, -"bs.ass.currAss.receiv.shareholders");
fact = factory.createNumericFact(conceptInfo, new BigDecimal("2000.00"), conceptInfo.-
isInstant() -? instant
        -: duration, eurUnit);
aggregator.addFact(fact);
conceptInfo = tax.getConceptInfoset(gaapNS, -"bs.eqLiab.equity.capRes");
fact = factory.createNumericFact(conceptInfo, new BigDecimal("4000.00"), conceptInfo.-
isInstant() -? instant
        -: duration, eurUnit);
aggregator.addFact(fact);
aggregator.aggregate();

System.out.println("All aggregated fact are: -");
Collection<NumericFact> aggregatedFacts = aggregator.getFacts();
for (NumericFact nf -: aggregatedFacts) {
    System.out.println("\n\nConcept: -" + nf.getLocalname() + -"\n\tNumeric value: -" + nf.-
getNumericValue());
}
```

Figure 35. Adding facts and calculating

For every fact that was added to aggregator, it calculates and creates parent fact. Then for its parent fact, aggregator creates its parent fact and so on to the root concept of that role for the whole calculation tree. It calculates these facts over calculation relationships in taxonomy. The weight attribute that can be "1" or "-1" is also taken into account. Also, as you can see, when aggregator calculated these facts fact for "bs.eqLiab.equity.netIncome" is increased by sum of fact for "is.netIncome".

2.11. Sax Discovery

With use of the class SAXDiscovery it is possible to see what documents are referenced from one or more starting documents.

```
String baseURI = XBRLURIResolver.expandURI(getTaxonomiesDirPath() + -"/de-gaap-
ci-2014-04-02/", null);
List<String> uris = new ArrayList<String>();
uris.add("de-gaap-ci-2014-04-02.xsd");
List<DTSDocumentInfo> result = SAXDiscovery.processDTSDiscovery(new StandardURIResolver(), -
null, uris, baseURI);
System.out.println("Referenced docs are: -");
for (DTSDocumentInfo doc -: result) {
    System.out.println(doc);
}
```

Figure 36. Sax discovery

2.12. XBRL Formula Processing

A XBRL formula specifies validations based on XBRL instance facts and generation of derived data as output XBRL instance facts.

This leads to two differnet processing types:

- Assertions
  - Value Assertions
  - Existence Assertions
  - Consistency Assertions
- Formulas

2.12.1. Value Assertions

Value assertions often are the most used formula linkbase feature, providing a way to check input XBRL instance facts against an expression. After loading the taxonomy you can access the repository (ValueAssertionRepository) that contains all available value assertions.

```
ValueAssertionRepository repo = tax.getValueAssertionRepository();
```

Figure 37. Acquiring repository of value assertions

Next, you can get a builder that is able to prepare a value assertion for execution.

```
ExecutableValueAssertionBuilder builder = tax.getValueAssertionBuilder();
```

Figure 38. Acquiring value assertion builder

And third, the processor that executes one or more value assertions on an instance.

```
ValueAssertionProcessor processor = tax.getValueAssertionProcessor();
```

Figure 39. Acquiring value assertion processor

First, determine which value assertions from repository to execute. These have to be prepared by the builder for execution. The builder returns ExecutableValueAssertions. The ExecutableValueAssertions can be used on multiple instances with multiple threads.

```
for (ValueAssertion va -: repo.listValueAssertions())
    valueAssertions.add(builder.newInstance(va));
```

Figure 40. Creating executable value assertions

You can also set values to particular parameters and set those to be used by processor when executing particular ExecutableValueAssertion. Parameters are set per ExecutableValueAssertion.

```
Map<QName, XdmValue> params = new HashMap<QName, XdmValue>();
params.put(new QName("http://www.eurofiling.info/xbrl/ext/filing-indicators", -"tC_00.01"), -
new XdmAtomicValue(true));
params.put(new QName("ReportingLevel"), new XdmAtomicValue("con"));
ValueProvider<XdmValue> paramValueProvider = new MapValueProvider<XdmValue>(params);
for(ExecutableValueAssertion eva -: valueAssertions)
    eva.setParameterValueProvider(paramValueProvider);
```

Figure 41. Setting parameters for executable value assertions

The ExecutableValueAssertions together with a loaded XBRL instance document are passed to the ValueAssertionProcessor, that generates AssertionResult objects.

```
Map<ExecutableValueAssertion, Collection<AssertionResult<ExecutableValueAssertion>>> result = -
processor
        -.process(valueAssertions, instance, null);
```

Figure 42. Processing value assertions

Each value assertion can produce multiple results. You can process the results and get access to the validation messages.

```
for (ValidationMessage validationMessage -: assertionResult.getValidationMessages()) {
    print(validationMessage);
}
print(assertionResult.getExecutionContext());
```

Figure 43. Validation message processing

2.12.2. Formulas

After loading the taxonomy you can access the repository (FormulaRepository) that contains all available formulas.

```
FormulaRepository repo = tax.getFormulaRepository();
```

Figure 44. Acquiring repository of formulas

Next, you can get a builder that is able to prepare a formula for execution.

```
ExecutableFormulaBuilder builder = tax.getFormulaBuilder();
```

Figure 45. Acquiring formula builder

And third, the processor that executes one or more formulas on an instance.

```
FormulaProcessor processor = tax.getFormulaProcessor();
```

Figure 46. Acquiring formula processor

First, determine which formulas from repository to execute. These have to be prepared by the builder for execution. The builder returns ExecutableFormulas. The ExecutableFormulas can be used on multiple instances with multiple threads.

```
for (Formula f -: repo.listFormulas())
    formulas.add(builder.newInstance(f));
```

Figure 47. Creating executable formulas

You can also set values to particular parameters and set those to be used by processor when executing particular ExecutableFormula. Parameters are set per ExecutableFormula.

```
Map<QName, XdmValue> params = new HashMap<QName, XdmValue>();
params.put(new QName("http://www.eurofiling.info/xbrl/ext/filing-indicators", -"tC_00.01"), -
new XdmAtomicValue(true));
params.put(new QName("ReportingLevel"), new XdmAtomicValue("con"));
ValueProvider<XdmValue> paramValueProvider = new MapValueProvider<XdmValue>(params);
for(ExecutableFormula ef -: formulas)
    ef.setParameterValueProvider(paramValueProvider);
```

Figure 48. Setting parameters for executable formulas

The ExecutableFormulas together with a loaded XBRL instance document are passed to the FormulaProcessor, that generates XBRLInstances object.

```
XBRLInstances resultInstance = processor.process(formulas, instance, -
(ErrorHandler<ExecutableFormula>) null);
```

Figure 49. Processing formulas

2.12.3. Known limitations

In cases when variable set has large number of variables and when variables have more than one value, number of intermediate results for evaluations (which are calculated as cross product of all variable values) might become large enough to exceed `java.lang.Integer.MAX_VALUE`. In that case exception will be thrown. This situation is usually a sign that either particular variable set (value assertion, formula or existence assertion) is not written optimally or instance that is being processed has fact duplicates. You can control fact duplicates with two following properties `IdenticalFactDuplicatesAllowance` and `NonIdenticalFactDuplicatesAllowance`. Those properties are part of `TaxonomyConfig`.

2.13. Table Linkbase 1.0

After loading the taxonomy you can access the repository (TableLinkbaseRepository) that contains all available table definitions and list them.

```
List<Table> tables = tax.getTableRepository().listTables();
```

Figure 50. Acquiring Table Linkbase repository of available table definitions

Next, you prepare the visualisation either with or without an instance

```
PreparedVisualization preparedVisualization = tax.getTableVisualizer().prepare(tables, -
inputInstance);
```

Figure 51. Prepare Table Linkbase visualisation including instance data

```
PreparedVisualization preparedVisualization = tax.getTableVisualizer().prepare(tables);
```

Figure 52. Prepare Table Linkbase visualisation without instance data

Thereafter, you can prepare visualisation parameters that influence the visualisation. e.g. You can set the language.

```
VisualizationParameters parameters = new VisualizationParameters();
parameters.setLanguage("en");
parameters.setShrinkOptionsDecision(new ShrinkOptionsDecision() {
    @Override
    public ShrinkOptions shrink(VisualizationContext context) {
        return ShrinkOptions.NONE;
    -}
});

parameters.setGenerateMissingLabelsDecision(new GenerateMissingLabelsDecision() {
    @Override
    public boolean generateMissingLabels(VisualizationContext context) {
        return true;
    -}
});
```

Figure 53. Defining Table Linkbase visualisation parameters

Finally you can visualise the tables to an OutputStream e.g as HTML using your defined parameters.

```
preparedVisualization.visualize(VisualizationFormat.HTML, outputStream, parameters);
```

Figure 54. Visualise Table Linkbase tables

3. .NET HowTo

In this section you can find a quick introduction to the basic steps that can be done using ABRA from .NET. The code for those examples can be found under the `examples/abraExample` directory. Example project, `abraExample` is easy to setup and have running. You just need to run `gradlew initAbra`, afterwards copy your `abz-abra.lic` license file into directory `license` and afterwards you can simply open solution in Visual Studio. After that, you can easily build solution from Visual Studio and execute dll in NUnit runner, since examples are written as Nunit tests.

3.1. Installing a license

Installation of license must happen prior to any other ABRA code execution in order for ABRA library to work. License can be installed in few ways. Installing license as a file

```
java.io.File licenseFile = new java.io.File(getLicenseFilePath());
LicenseManager licenseManager = LicenseManager.getInstance();
licenseManager.setLicenseFile(licenseFile);
```

Figure 55. Installing license file

installing raw license, note that byte array can be embedded into code directly.

```
byte[] rawLicense = File.ReadAllBytes(getLicenseFilePath());
LicenseManager licenseManager = LicenseManager.getInstance();
licenseManager.setRawLicense(rawLicense);
```

Figure 56. Installing raw license

Also, license file named `abz-abra.lic` can be put next to the ABRA dll while executing, and license will be recognized and installed automatically. Note, this will only be taken into account if neither license file nor raw license are set.

3.2. Loading a Taxonomy

To load a taxonomy you need to obtain a TaxonomyBuilder and provide the entry files for discovering all taxonomy files.

```
java.io.File gaapSchemaFile = new java.io.File(getTaxonomyDirPath(), -"de-gaap-ci-2014-04-02/
de-gaap-ci-2014-04-02-shell-fiscal.xsd");
java.io.File gcdSchemaFile = new java.io.File(getTaxonomyDirPath(), -"de-gcd-2014-04-02/de-
gcd-2014-04-02-shell.xsd");
TaxonomyBuilderFactory taxonomyBuilderFactory = new TaxonomyBuilderFactory();
TaxonomyBuilder taxonomyBuilder = taxonomyBuilderFactory.createInMemoryTaxonomyBuilder();
Taxonomy tax = taxonomyBuilder.withSchema(gaapSchemaFile).withSchema(gcdSchemaFile).build();
```

Figure 57. Taxonomy loading

If taxonomy files are located localy and you want to use them instead of downloading them from the Internet, you can specify catalog file to be used for mapping remote URIs to files on file system.

```
java.io.File catalog = new java.io.File(taxonomyDirPath, -"corep-finrep-bbk-2014-03-31/
catalog.xml").getAbsoluteFile();
java.lang.System.setProperty("xml.catalog.files", catalog.getAbsolutePath());
```

Figure 58. Setting catalog file

You also need to set `CatalogResolver` as `URIResolver` in `TaxonomyBuilder`.

```
taxonomyBuilder.withURIResolver(new CatalogResolver());
```

Figure 59. Setting CatalogResolver as URIResolver

If you like to validate a taxonomy to conform to XBRL 2.1 rules, use the TaxonomyValidator.

Note

Have in mind that API of taxonomy validator will change in future to return .NET objects as result instead of XML.

```
TaxonomyValidatorFactory valFactory = TaxonomyValidatorFactory.newInstance();
TaxonomyValidator validator = valFactory.taxonomyValidator();
Document doc = DOMHelper.newDocument();
SAX2DOMImpl sax2Dom = new SAX2DOMImpl(doc, doc);
SAXResult saxResult = new SAXResult();
saxResult.setHandler(sax2Dom);
saxResult.setLexicalHandler(sax2Dom);
validator.validate(tax, saxResult);
NodeList errorList = doc.getDocumentElement().getElementsByTagName("error");
Console.WriteLine("Taxonomy validation: -");
if (errorList.getLength() == 0)
{
    Console.WriteLine("There are no errors.-");
}
else
{
    Console.WriteLine("Errors are: -");
    for (int i = 0; i < errorList.getLength(); i++)
    {
        org.w3c.dom.Node error = errorList.item(i);
        Console.WriteLine(DOMHelper.toString(error));
    -}
}
```

Figure 60. Taxonomy validation

3.3. Reading the Presentation Linkbase Structure

Reading and moving through the structure of taxonomy can be accomplished with class RelationNavigator. First you need to get all the roles of the presentation linkbase of taxonomy.

```
Console.WriteLine("Presentation roles for the taxonomy are:");
RelationElementFilter roleFilter = new ElementFilterName(XBRLUtil.XBRL_LINKBASE_NS_URI, -
XBRLUtil.LINK_PRESENTATION_LINK_LN);
IList<ElementDocContainer> extendedLinks = tax.getNavigator().getExtendedLinks(roleFilter);

HashSet<string> roles = new HashSet<string>();
foreach (ElementDocContainer link in extendedLinks)
{
    string role = link.getElement().getAttributeNS(XBRLUtil.XLINK_NS_URI, XBRLUtil.-
XLINK_ROLE_ATTR);
    roles.Add(role);
}
foreach (string role in roles)
{
    Console.WriteLine(role);
}
```

Figure 61. Acquiring all roles from presentation linkbase

Now, for each role you can get root concepts of that role.

```
String roleTcct = -"http://www.xbrl.de/taxonomies/de-gaap-ci/" + -"role/
transfersCommercialCodeToTax";
Console.WriteLine("Root concepts for role \"" + roleTcct + -"\" are:");
RelationElementFilter presentationRootFilter = new ElementFilterNameParentRole(XBRLUtil.-
XBRL_LINKBASE_NS_URI, XBRLUtil.LINK_PRESENTATION_ARC_LN, roleTcct);
RelationNavigator nav = tax.getNavigator();
IList<ElementDocContainer> rootConcepts = nav.getRootConcepts(nav.-
getRelations(presentationRootFilter));

foreach (ElementDocContainer concept in rootConcepts)
{
    Console.WriteLine(tax.getConceptInfoset(concept).getLocalname());
}
```

Figure 62. Acquiring root concepts for specified role

For each concept in a role you can get list of his child concepts. In the code example below, it is shown how to get a list of child concept names for a given role.

```
String conceptLocalName = -"bs.ass";
String roleBS = -"http://www.xbrl.de/taxonomies/" + -"de-gaap-ci/role/balanceSheet";
// de-gaap-ci scheme namespace
String conceptNamespace = -"http://www.xbrl.de/taxonomies/de-gaap-ci-2014-04-02";
Console.WriteLine("Child concepts for concept \"{0}\", role \"{1}\" and namespace \"{2}\" -
are:", conceptLocalName, roleBS, conceptNamespace);

RelationElementFilter presentationChildConceptsFilter = new -
ElementFilterNameParentRole(XBRLUtil.XBRL_LINKBASE_NS_URI, XBRLUtil.LINK_PRESENTATION_ARC_LN, -
roleBS);
Comparator arcComparator = Relation.BY_ORDER_ATTRIBUTE;
ElementDocContainer rootConcept = tax.getConcept(conceptNamespace, conceptLocalName);
IList<ElementDocContainer> children = tax.getNavigator().endpointToEndpoint(rootConcept, -
presentationChildConceptsFilter, arcComparator, null);
foreach (ElementDocContainer child in children)
{
    Console.WriteLine(tax.getConceptInfoset(child).getLocalname());
}
```

Figure 63. Acquiring child concepts for specified root concept

At the end, with combination of these RelationNavigator methods you can read the complete concepts tree structure of the presentation linkbase. In this example the tree structure is printed to `out.txt` file.

```
private string printChildConceptsRecursively(Taxonomy tax, RelationElementFilter roleFilter, -
ConceptInfoset parentConcept, String tabulator)
{
    string retVal = -"";
    Comparator arcComparator = Relation.BY_ORDER_ATTRIBUTE;
    RelationNavigator nav = tax.getNavigator();
    IList<ElementDocContainer> children = nav.endpointToEndpoint(parentConcept.getConcept(), -
roleFilter, arcComparator, null);

    foreach (ElementDocContainer concept in children)
    {
        com.abzreporting.abra.taxonomy.ConceptInfoset conceptInfo = tax.-
getConceptInfoset(concept);
        string conceptName = conceptInfo.getLocalname();
        retVal += tabulator + conceptName + -"\n";
        retVal += printChildConceptsRecursively(tax, roleFilter, conceptInfo, tabulator -
+ -"\t");
    -}
    return retVal;
}
```

Figure 64. Reading of complete concepts tree structure

3.4. Accessing Concept Properties via ConceptInfoset Interface

This is an example for reading concept properties. See ConceptInfoset's documentation for more information on concept properties.

```
string gaapNS = -"http://www.xbrl.de/taxonomies/de-gaap-ci-2014-04-02";
string conceptLocalName = -"bs.eqLiab.equity.netIncome";
ElementDocContainer concept = tax.getConcept(gaapNS, conceptLocalName);
ConceptInfoset ci = tax.getConceptInfoset(concept);
Console.WriteLine("ConceptInfoset properties for concept with localname: {0}, and namespace: -
{1} are:",
    ci.getLocalname(), ci.getNamespace());
Console.WriteLine("Balance Type: {0}", ci.getBalance());
Console.WriteLine("Period Type: {0}", ci.getPeriodType());
Console.WriteLine("Xbrl Data Type: {0}", ci.getXbrlBasicType());
Console.WriteLine("Is Abstract: {0}", ci.isAbstract());
Console.WriteLine("Is Duration: {0}", ci.isDuration());
Console.WriteLine("Is Instant: {0}", ci.isInstant());
Console.WriteLine("Is Item: {0}", ci.isItem());
Console.WriteLine("Is Nillable: {0}", ci.isNillable());
Console.WriteLine("Is Numeric: {0}", ci.isNumeric());
```

Figure 65. Reading concept properties

Getting concept's labels can be accomplished in different ways. One way is using ConceptInfoset's method. The method has a String parameter that specifies the language of the returned labels.

```
Console.WriteLine("Labels for english language are -: -");
Collection labelValues = ci.getLabels("en").values();
int i = 1;
for (Iterator labelIt = labelValues.iterator(); labelIt.hasNext(); -)
{
    string label = (string)labelIt.next();
    Console.WriteLine("Label {0}. -: {1}", i, label);
    i++;
}
```

Figure 66. Getting concept's labels per language

Getting all concept labels in all languages can be accomplished using RelationNavigator.

```
Console.WriteLine("All labels in all languages for concept {0} are: -", ci.getLocalname());
RelationElementFilter presentationFilter = new ElementFilterName(XBRLUtil.-
XBRL_LINKBASE_NS_URI, XBRLUtil.LINK_LABEL_ARC_LN);
Comparator arcComparator = Relation.BY_ORDER_ATTRIBUTE;
RelationNavigator nav = tax.getNavigator();
IList<ElementDocContainer> children = nav.endpointToEndpoint(concept, presentationFilter, -
arcComparator, null);
foreach (ElementDocContainer element in children)
{
    Console.WriteLine(element.getElement().getTextContent());
}
```

Figure 67. Getting concept's labels in all languages

3.5. Instance Document Creation

When creating an instance of document, there are two classes that play together. First Instance-Builder is used to collect facts and to generate an instance document. FactFactory is used to create the facts that will be added to the InstanceBuilder.

```
XBRLInstances instance = null;
InstanceBuilder builder = new InstanceBuilder(tax);
FactFactory factory = builder.getFactFactory();
```

Figure 68. InstanceBuilder and FactFactory

Before creating facts for instance, you need to set up couple more things. Every fact has a period type, so first it is required to make instant and duration context for facts.

```
DurationContext duration = factory.createDurationContext("http://example.com/
test", -"example1",
    XBRLUtil.convertDateFromXML("2012-01-01"), XBRLUtil.convertDateFromXML("2012-12-31"));
Context instant = factory.createInstantContextFromEndDate(duration);
```

Figure 69. Creating duration and instance context

Unit needs to be defined when creating numeric facts. For example, for a monetary concept you need to create a numeric fact with monetary unit.

```
Unit pureUnit = factory.createPureUnit();
Unit eurUnit = factory.createEuroMonataryUnit();
```

Figure 70. Creating unit

Now you can create facts for the instance document. For every fact you will need to specify concept, period type and value. For numeric facts, unit is also required. This example shows how to create facts for several xbrl data types.

```
ConceptInfoset ci = tax.getConceptInfoset(gaapNS, -"is.netIncome.otherTaxes");
NumericFact monetaryFact = factory.createNumericFact(ci, new BigDecimal(123.24), duration, -
eurUnit);

ci = tax.getConceptInfoset(gaapNS, -"nt.employeesEffectiveDate");
NumericFact decimalFact = factory.createNumericFact(ci, new BigDecimal(555.55), instant, -
pureUnit);

ci = tax.getConceptInfoset(gaapNS, -"nt.taxReport");
NonNumericFact stringFact = factory.createNonNumericFact(ci, -"Some tax report", instant);

ci = tax.getConceptInfoset(gaapNS, -"nt.employeesAveragePeriod");
ItemFact numericNilFact = factory.createNilFact(ci, duration, pureUnit);

ci = tax.getConceptInfoset(gaapNS, -"bs.eqLiab.defIncome.comment");
Fact nonNumericNilFact = factory.createNilFact(ci, instant, null);

ci = tax.getConceptInfoset(gaapNS, -"nt.directorsSign.date");
CalendarFact dateFact = factory.createDateFact(ci, XBRLUtil.convertDateFromXML("2012-01-01"), -
duration);

ci = tax.getConceptInfoset(gcdNS, -"genInfo.report.autoNumbering");
NonNumericFact boolFact = factory.createBooleanFact(ci, new java.lang.Boolean(true), duration);

ci = tax.getConceptInfoset(gaapNS, -"hbst.transfer.bsAss.name");
QNameFact qnameFact = factory.createQNameFact(ci, new QName("http://www.namespace.-
com", -"localPart"), duration);
```

Figure 71. Creating facts

In case when creating a compound fact (tuple), it is required that all facts that belong to the tuple are already created. Then it is possible to create the compound fact with all child facts as parameter.

```
// creating facts for tuple concept nt.relationships.ShareholdersKapCoSpec
ci = tax.getConceptInfoset(gaapNS, -"nt.relationships.ShareholdersKapCoSpec.NameSeat");
NonNumericFact stringTupleFact = factory.createNonNumericFact(ci, -"name", duration);

ci = tax.getConceptInfoset(gaapNS, -"nt.relationships.ShareholdersKapCoSpec.IssuedKap");
NumericFact monetaryTupleFact = factory.createNumericFact(ci, new BigDecimal(555.55), -
duration, eurUnit);

// making tupleFact
ci = tax.getConceptInfoset(gaapNS, -"nt.relationships.ShareholdersKapCoSpec");
TupleFact tupleFact = factory.createTupleFact(ci, stringTupleFact, monetaryTupleFact);
```

Figure 72. Creating tuple fact

For each fact you can add a footnote.

```
// creating footnotes for facts
string textFootnoteVal = -"first value";
string xhtmlFootnoteVal = -"<h1>second value</h1>";
Footnote textFootnote = factory.createFootnote(textFootnoteVal, false, -"en");
Footnote xhtmlFootnote = factory.createFootnote(xhtmlFootnoteVal, true, -"en");

// adding footnotes to facts
monetaryFact.setFootnotes(Arrays.asList(xhtmlFootnote));
stringFact.setFootnotes(Arrays.asList(textFootnote));
```

Figure 73. Adding footnote to fact

As you can see, when adding a xhtml value for a footnote, the second parameter parseAsXml is set to true. Also, you can add same footnote for more than one fact and more than one footnote for a single fact.

At the end you need to add all facts to InstanceBuilder and create the instance document. When calling buildInstance method with all parameters set to null, you will get an instance document that has unnecessary schemaRefs. To avoid this you need to implement an URIReplacer.

```
builder.addFact(monetaryFact);
builder.addFact(decimalFact);
builder.addFact(stringFact);
builder.addFact(numericNilFact);
builder.addFact(nonNumericNilFact);
builder.addFact(dateFact);
builder.addFact(qnameFact);
builder.addFact(boolFact);
builder.addFact(tupleFact);
// implementation of URIReplacer to remove unnecessary
// schemaRefs from document instance
URIReplacer replacer = new CustomURIReplacer();

instance = builder.buildInstance(null, null, replacer);

Console.WriteLine("Example instance document is: -");
Console.WriteLine(DOMHelper.toString(instance.getDocument()));
```

Figure 74. Adding all facts and building instance

3.6. Load an Instance Document

To load an instance document for an existing taxonomy, use the already loaded taxonomy. Instance can be loaded from its file.

```
XBRLInstances instance = tax.createInstance(new StreamSource(instanceFile), false, null);
```

Figure 75. Creating instance

3.7. Read Information from an Instance Document

To read information from an already loaded instance use the class InstanceFactProvider. It provides access to the instance facts via the ABRA fact model.

```
InstanceFactProvider instanceFactProvider = new InstanceFactProvider(tax, instance);
```

Figure 76. Accessing instance facts through the ABRA model

To access all facts from the instance use getRootFacts. There is an optional ElementFilter argument, that is used to limit the returned facts.

```
IList<Fact> facts = instanceFactProvider.getRootFacts(null);
```

Figure 77. Accessing instance facts through the ABRA model

For each fact you can access its properties, like value.

```
Console.Write(fact.getValue());
```

Figure 78. Accessing fact's value

Or you can access the context.

```
Context context = fact.getContext();
printContext(context, level + 1);
```

Figure 79. Accessing fact's contextl

Or you can access the unit - only for numeric facts.

```
if (fact.isNumeric()) {
    Unit unit = ((ImmutableNumericFact)fact).getUnit();
    printUnit(unit, level + 1);
}
```

Figure 80. Accessing fact's unit

3.8. Instance Document Visualization

To visualize an instance document, InstanceVisualisation class is used.

```
InstanceVisualisationFactory visualisationFactory = InstanceVisualisationFactory.newInstance();
InstanceVisualisation instanceVisualisation = visualisationFactory.newVisualiser();
Document doc = DOMHelper.newDocument();
SAX2DOMImpl sax2Dom = new SAX2DOMImpl(doc, doc);
SAXResult saxResult = new SAXResult();
saxResult.setHandler(sax2Dom);
saxResult.setLexicalHandler(sax2Dom);
instanceVisualisation.visualise(tax, instance, -"de", saxResult);
Console.WriteLine("Visalisation of document instace is located at \"out.html\"");
File.WriteAllText("out.html", DOMHelper.toString(doc));
```

Figure 81. Instance visualization

This generates a HTML representation of the xbrl instance document.

3.9. Instance Document Validation

To validate an xbrl instance document, InstanceValidator class is used.

```
InstanceValidator validator = tax.newInstanceValidatorBuilder().ignoreTaxonomyRefsChecks().-
ignoreDuplicatesChecks().build();
// Activate or deactivate some validation features
InstanceValidationResult result = validator.validate(instance);
Console.WriteLine("Instance document validation: -");
if (result.getInstanceElementErrors().Count == 0)
{
    Console.WriteLine("There are no errors.-");
}
else
{
    Console.WriteLine("Errors are: -");
    foreach (InstanceElementError error in result.getInstanceElementErrors())
    {
        Console.WriteLine("{0}:{1}", error.getCode(), error.getMessage());
    -}
}
```

Figure 82. Instance validation

This validates the given xbrl file according to the xbrl 2.1 standard.

Also, for xbrl formula 1.0 validation, parameters can be set to instance validator.

```
IDictionary<QName, net.sf.saxon.s9api.XdmValue> parameters = new Dictionary<QName, net.sf.-
saxon.s9api.XdmValue>();
parameters.Add(new QName("http://www.eurofiling.info/xbrl/ext/filing-indicators", -"tC_00.-
01"), new net.sf.saxon.s9api.XdmAtomicValue(true));
parameters.Add(new QName("ReportingLevel"), new net.sf.saxon.s9api.XdmAtomicValue("con"));
ValueProvider<net.sf.saxon.s9api.XdmValue> paramValueProvider = new -
DictionaryValueProvider<net.sf.saxon.s9api.XdmValue>(parameters);
validatorBuilder.withFormulaParameters(paramValueProvider);
```

Figure 83. Setting formula parameters for validation

To list all parameters which are present in taxonomy, use ParameterRepository.

```
ParameterRepository parameterRepository = tax.getParameterRepository();
foreach(Parameter p in parameterRepository.list())
{
    Console.WriteLine("Paramter: -" + p.getGlobalName() + -" of type: -" + p.getType() + -" -
is -"
        + (p.isRequired() -? -"mandatory.-" -: -"not mandatory.-"));
        if(!p.isRequired())
        {
            Console.WriteLine("Expression is: -" + p.getValueExpression().getExpression());
        -}
}
```

Figure 84. Listing all parameters from taxonomy

3.10. Fact Aggregation

Abra offers a way for calculating sum or difference for a whole financial statement by utilizing the taxonomy's calculation linkbase. This can be accomplished with the class FactAggregator. First, initialize FactAggregator and reset calculation roles. The order of the calculation roles is significant as the FactAggregator calculates one role after the another in the given order.

```
string rolePrefix = -"http://www.xbrl.de/taxonomies/de-gaap-ci/role";
string[] roles = new string[]
{
    rolePrefix + -"/notes",
    rolePrefix + -"/OtherReportElements",
    rolePrefix + -"/managementReport",
    rolePrefix + -"/contingentLiabilities",
    rolePrefix + -"/changesEquityAccounts",
    rolePrefix + -"/changesEquityStatement",
    rolePrefix + -"/incomeStatement",
    rolePrefix + -"/fiscalCalculations",
    rolePrefix + -"/balanceSheet",
    rolePrefix + -"/appropriationProfits",
    rolePrefix + -"/cashFlowStatement"
};
FactAggregator<NumericFact> aggregator = new FactAggregatorImpl<NumericFact>();
aggregator.setCalculationRolesAndReset(tax, roles);
```

Figure 85. FactAggregator initialization

After that, you need to set FactAccesor implementation for the FactAggregator. FactAccesor is used by FactAggregator to access facts that are aggregated and to abstract from the current fact model. So you need to create a FactAccesor and set its unit, instant context and duration context properties.

```
InstanceBuilder builder = new InstanceBuilder(tax);
FactFactory factory = builder.getFactFactory();
FactAccessorImpl<NumericFact> acc = new FactAccessorImpl<NumericFact>(factory);
DurationContext duration = factory.createDurationContext("http://example.com/
test", -"example1",
    XBRLUtil.convertDateFromXML("2010-01-01"), XBRLUtil.convertDateFromXML("2010-12-31"));
InstantContext instant = factory.createInstantContextFromEndDate(duration);
Unit eurUnit = factory.createEuroMonataryUnit();
acc.setDurationContext(duration);
acc.setInstantContext(instant);
acc.setUnit(eurUnit);
aggregator.setFactAccessor(acc);
```

Figure 86. Assigning FactAccesor to FactAggregator

For other calculations that exist and that are not expressed by the taxonomy's calculation linkbase rules, FactAggregatedListener exists. You can write your own implementation of this listener for any calculation purpose you have. This listener is called whenever a fact value is calculated, even when the value did not change.

```
ElementDocContainer isNetIncome = tax.getConcept(gaapNS, -"is.netIncome");
ElementDocContainer bsNetIncome = tax.getConcept(gaapNS, -"bs.eqLiab.equity.netIncome");
aggregator.addFactAggregatedListener(new TransferFactValueImpl<NumericFact>(isNetIncome, -
bsNetIncome));
```

Figure 87. Adding listener to FactAggregator

In this case, it uses transferFactValue implementation of FactAggregatedListener. It creates a new Fact for concept "Net income/net loss for the financial year"(bs.eqLiab.equity.netIncome)

with the value from fact for concept "Net loss for the year"(is.netIncome) as soon as "Net loss for the year" is calculated. Now you can add some facts to aggregator to see what is calculated.

```
ConceptInfoset ci = tax.getConceptInfoset(gaapNS, -"is.netIncome.regular.fin.netInterest.-
expenses.valueDiscount");
NumericFact fact = factory.createNumericFact(ci, new BigDecimal("100.00"), ci.isInstant() -? -
(Context)instant -: duration, eurUnit);
aggregator.addFact(fact);

ci = tax.getConceptInfoset(gaapNS, -"is.netIncome.regular.fin.netInterest.expenses.-
regularInterest");
fact = factory.createNumericFact(ci, new BigDecimal("100.00"), ci.isInstant() -? -
(Context)instant -: duration, eurUnit);
aggregator.addFact(fact);

ci = tax.getConceptInfoset(gaapNS, -"is.netIncome.regular.fin.netParticipation.earnings");
fact = factory.createNumericFact(ci, new BigDecimal("1000.00"), ci.isInstant() -? -
(Context)instant -: duration, eurUnit);
aggregator.addFact(fact);

ci = tax.getConceptInfoset(gaapNS, -"is.netIncome.regular.fin.netParticipation.-
earningSecurities");
fact = factory.createNumericFact(ci, new BigDecimal("1000.00"), ci.isInstant() -? -
(Context)instant -: duration, eurUnit);
aggregator.addFact(fact);

ci = tax.getConceptInfoset(gaapNS, -"bs.ass.currAss.receiv.shareholders");
fact = factory.createNumericFact(ci, new BigDecimal("2000.00"), ci.isInstant() -? -
(Context)instant -: duration, eurUnit);
aggregator.addFact(fact);

ci = tax.getConceptInfoset(gaapNS, -"bs.eqLiab.equity.capRes");
fact = factory.createNumericFact(ci, new BigDecimal("4000.00"), ci.isInstant() -? -
(Context)instant -: duration, eurUnit);
aggregator.addFact(fact);

aggregator.aggregate();

Console.WriteLine("All aggregated fact are: -");
IList<NumericFact> aggregatedFacts = aggregator.getFacts();
foreach (NumericFact nf in aggregatedFacts)
{
    Console.WriteLine("\n\nConcept: {0}\n\tNumeric value: {1}", nf.getLocalname(), nf.-
getNumericValue());
}
```

Figure 88. Adding facts and calculating

For every fact that was added to aggregator, it calculates and creates parent fact. Then for its parent fact, aggregator creates its parent fact and so on to the root concept of that role for the whole calculation tree. It calculates these facts over calculation relationships in taxonomy. The weight attribute that can be "1" or "-1" is also taken into account. Also, as you can see, when aggregator calculated these facts fact for "bs.eqLiab.equity.netIncome" is increased by sum of fact for "is.netIncome".

3.11. Sax Discovery

With use of the class SAXDiscovery it is possible to see what documents are referenced from one or more starting documents.

```
string baseURI = XBRLURIResolver.expandURI("de-gaap-ci-2014-04-02/", getTaxonomyDirPath());
java.util.List uris = new ArrayList();
uris.add("de-gaap-ci-2014-04-02.xsd");
java.util.List result = SAXDiscovery.processDTSDiscovery(new StandardURIResolver(), null, -
uris, baseURI);
Assert.Greater(result.size(), 0);
Console.WriteLine("Referenced docs are: -");
for (Iterator resultIt = result.iterator(); resultIt.hasNext(); -)
{
    DTSDocumentInfo doc = (DTSDocumentInfo)resultIt.next();
    Console.WriteLine(doc);
}
```

Figure 89. Sax discovery

3.12. XBRL Formula Processing

A XBRL formula specifies validations based on XBRL instance facts and generation of derived data as output XBRL instance facts.

This leads to two differnet processing types:

- Assertions
  - Value Assertions
  - Existence Assertions
  - Consistency Assertions
- Formulas

3.12.1. Value Assertions

Value assertions often are the most used formula linkbase feature, providing a way to check input XBRL instance facts against an expression. After loading the taxonomy you can access the repository (ValueAssertionRepository) that contains all available value assertions.

```
ValueAssertionRepository repo = tax.getValueAssertionRepository();
```

Figure 90. Acquiring repository of value assertions

Next, you can get a builder that is able to prepare a value assertion for execution.

```
ExecutableValueAssertionBuilder builder = tax.getValueAssertionBuilder();
```

Figure 91. Acquiring value assertion builder

And third, the processor that executes one or more value assertions on an instance.

```
ValueAssertionProcessor processor = tax.getValueAssertionProcessor();
```

Figure 92. Acquiring value assertion processor

First, determine which value assertions from repository to execute. These have to be prepared by the builder for execution. The builder returns ExecutableValueAssertions. The ExecutableValueAssertions can be used on multiple instances with multiple threads.

```
foreach (ValueAssertion va in repo.listValueAssertions())
{
    valueAssertions.Add(builder.newInstance(va));
}
```

Figure 93. Creating executable value assertions

You can also set values to particular parameters and set those to be used by processor when executing particular ExecutableValueAssertion. Parameters are set per ExecutableValueAssertion.

```
IDictionary<QName, net.sf.saxon.s9api.XdmValue> parameters = new Dictionary<QName, net.sf.-
saxon.s9api.XdmValue>();
parameters.Add(new QName("http://www.eurofiling.info/xbrl/ext/filing-indicators", -"tC_00.-
01"), new net.sf.saxon.s9api.XdmAtomicValue(true));
parameters.Add(new QName("ReportingLevel"), new net.sf.saxon.s9api.XdmAtomicValue("con"));
ValueProvider<net.sf.saxon.s9api.XdmValue> paramValueProvider = new -
DictionaryValueProvider<net.sf.saxon.s9api.XdmValue>(parameters);
foreach (ExecutableValueAssertion eea in valueAssertions)
    eea.setParameterValueProvider(paramValueProvider);
```

Figure 94. Setting parameters for executable value assertions

The ExecutableValueAssertions together with a loaded XBRL instance document are passed to the ValueAssertionProcessor, that generates AssertionResult objects.

```
IDictionary<ExecutableValueAssertion, IList<AssertionResult<ExecutableValueAssertion>>> -
result = processor.process(valueAssertions, instance, null);
```

Figure 95. Processing value assertions

Each value assertion can produce multiple results. You can process the results and get access to the validation messages.

```
InstanceFactProvider instanceFactProvider = new InstanceFactProvider(tax, instance);
foreach (KeyValuePair<ExecutableValueAssertion, -
IList<AssertionResult<ExecutableValueAssertion>>> keyValue in result)
{
    ExecutableValueAssertion exeValAssertion = keyValue.Key;
    IList<AssertionResult<ExecutableValueAssertion>> assertionResults = keyValue.Value;
    print(exeValAssertion, assertionResults, instanceFactProvider);
}
```

Figure 96. Validation message processing

3.12.2. Formulas

After loading the taxonomy you can access the repository (FormulaRepository) that contains all available formulas.

```
FormulaRepository repo = tax.getFormulaRepository();
```

Figure 97. Acquiring repository of formulas

Next, you can get a builder that is able to prepare a formula for execution.

```
ExecutableFormulaBuilder builder = tax.getFormulaBuilder();
```

Figure 98. Acquiring formula builder

And third, the processor that executes one or more formulas on an instance.

```
FormulaProcessor processor = tax.getFormulaProcessor();
```

Figure 99. Acquiring formula processor

First, determine which formulas from repository to execute. These have to be prepared by the builder for execution. The builder returns ExecutableFormulas. The ExecutableFormulas can be used on multiple instances with multiple threads.

```
foreach (Formula f in repo.listFormulas())
{
    formulas.Add(builder.newInstance(f));
}
```

Figure 100. Creating executable formulas

You can also set values to particular parameters and set those to be used by processor when executing particular ExecutableFormula. Parameters are set per ExecutableFormula.

```
IDictionary<QName, net.sf.saxon.s9api.XdmValue> parameters = new Dictionary<QName, net.sf.-
saxon.s9api.XdmValue>();
parameters.Add(new QName("http://www.eurofiling.info/xbrl/ext/filing-indicators", -"tC_00.-
01"), new net.sf.saxon.s9api.XdmAtomicValue(true));
parameters.Add(new QName("ReportingLevel"), new net.sf.saxon.s9api.XdmAtomicValue("con"));
ValueProvider<net.sf.saxon.s9api.XdmValue> paramValueProvider = new -
DictionaryValueProvider<net.sf.saxon.s9api.XdmValue>(parameters);
foreach (ExecutableFormula ef in formulas)
    ef.setParameterValueProvider(paramValueProvider);
```

Figure 101. Setting parameters for executable formulas

The ExecutableFormulas together with a loaded XBRL instance document are passed to the FormulaProcessor, that generates XBRLInstances object.

```
XBRLInstances resultInstance = processor.process(formulas, instance, null);
```

Figure 102. Processing formulas

3.12.3. Known limitations

In cases when variable set has large number of variables and when variables have more than one value, number of intermediate results for evaluations (which are calculated as cross product

of all variable values) might become large enough to exceed `java.lang.Integer.MAX_VALUE`. In that case exception will be thrown. This situation is usually a sign that either particular variable set (value assertion, formula or existence assertion) is not written optimally or instance that is being processed has fact duplicates. You can control fact duplicates with two following properties `IdenticalFactDuplicatesAllowance` and `NonIdenticalFactDuplicatesAllowance`. Those properties are part of `TaxonomyConfig`.

3.13. Table Linkbase 1.0

Currently Table Linkbase visualisation is only supported in the Java release.

4. Migration from ABRA 2 to ABRA 3

ABRA 3 comes with new improved API. It should be much easier to use ABRA than before, however, for people who are using ABRA 2 this is change, and it includes some effort in adjusting code. This page contains examples of old codes and examples written with new API which correspond to old code.

4.1. License file installation

Instead of old way of installation of license file:

```
byte[] licenseInfo = -.-.-.-;
Repository.setLicense(licenseInfo, true);
```

Here is a new way:

```
java.io.File licenseFile = new java.io.File(getLicenseFilePath());
LicenseManager licenseManager = LicenseManager.getInstance();
licenseManager.setLicenseFile(licenseFile);
```

Figure 103. Installing license file

4.2. "abra-db-dir" system property (deprecated with version 3.5)

Previously, property for setting location of database could be set as a `abra-db-dir` system property. This way of setting the location is now deprecated, and is be set when creating `TaxonomyBuilderFactory` using constructor:

```
TaxonomyBuilderFactory(boolean removeOldLockFile, String persistentDir)
```

4.3. "removeOldLockFile" system property

Instead of the old way of setting `removeOldLockFile`

```
TaxonomyFactoryPDOM.setRemoveOldLockFile(boolean removeOldLockFile)
```

this property should be set during creation of `TaxonomyBuilderFactory`:

```
new TaxonomyBuilderFactory(boolean removeOldLockFile)
new TaxonomyBuilderFactory(boolean removeOldLockFile, String persistentDir)
```

4.4. Renaming

In ABRA 3 suffix 'I' was removed from the following interfaces:

Table 1.

Name 2	Name 3
`TaxonomyI`	`Taxonomy`
`XBRLInstancesI`	`XBRLInstances`

`XBRLSchemaI`	`XBRLSchema`
`XBRLLinkbaseI`	`XBRLLinkbase`
`InstanceCollectionI`	`InstanceCollection`

The listed interfaces kept their method signatures. The only minor change is present in `TaxonomyI` interface where register parameter was removed from `createInstance` methods.

4.5. Taxonomy Loading

New classes have replaced `TaxonomyFactory` and `TaxonomyCreatorI` components. Instead of old way of creating taxonomy:

```
File gaapSchemaFile = new File(getTaxonomiesDirPath(),
    -"de-gaap-ci-2014-04-02/de-gaap-ci-2014-04-02-shell-fiscal.xsd");
File gcdSchemaFile = new File(getTaxonomiesDirPath(),
    -"de-gcd-2014-04-02/de-gcd-2014-04-02-shell.xsd");
String gaapSchemaURI = XBRLURIResolver.expandURI(gaapSchemaFile.getAbsoluteFile().toURI().-
toString(), null);
String gcdSchemaURI = XBRLURIResolver.expandURI(gcdSchemaFile.getAbsoluteFile().toURI().-
toString(), null);
TaxonomyCreatorI taxCreator = Repository.newTaxonomyCreator(new TaxonomyConfig(), true);
taxCreator.insertSchema(gaapSchemaURI);
taxCreator.insertSchema(gcdSchemaURI);
TaxonomyI tax = taxCreator.createTaxonomy();
String taxKey = -"taxKey";
Repository.insertTaxonomy(taxKey, tax);
```

there is a new, simplier way to do it:

```
File gaapSchemaFile = new File(getTaxonomiesDirPath(),
        -"de-gaap-ci-2014-04-02/de-gaap-ci-2014-04-02-shell-fiscal.xsd");
File gcdSchemaFile = new File(getTaxonomiesDirPath(), -"de-gcd-2014-04-02/de-gcd-2014-04-02-
shell.xsd");
TaxonomyBuilderFactory taxonomyBuilderFactory = new TaxonomyBuilderFactory();
TaxonomyBuilder taxonomyBuilder = taxonomyBuilderFactory.createInMemoryTaxonomyBuilder();
Taxonomy tax = taxonomyBuilder.withSchema(gaapSchemaFile).withSchema(gcdSchemaFile).build();
// no need to register taxonomy with key
```

Figure 104. New, modified way to create a taxonomy

As you can see, new API uses `TaxonomyBuilder` instead of `TaxonomyCreatorI` and its API is fluent, which allows inline constructing with multiple method calls.

4.6. Fact Model Change

Classes and interfaces that existed in packages

```
com.abzreporting.abra.creation.dom
com.abzreporting.abra.creation.impl
com.abzreporting.abra.creation.model
```

were moved to packages

```
com.abzreporting.abra.model.dom
com.abzreporting.abra.model.impl
```


```
com.abzreporting.abra.model
```

4.7. Builder package

New package `com.abzreporting.abra.builder` is created. The following classes from package `com.abzreporting.abra.creation` were moved to this new package:

```
CompoundFactValueException
ConceptNotFoundException
ConceptTypeException
ContextValueException
FactFactory
FactForAbstractConceptException
FactValidator
FactWithoutContextException
InstanceBuilder
InstanceIdProvider
ItemTypeException
MonetaryFactUnitException
NillFactForNotNillableConceptException
NonNumericConceptException
NumericConceptException
NumericFactWithoutUnitException
SharesFactUnitException
SimpleFactValueException
WrongContextPeriodException
XBRLCreationException
XMLEmitter
```

4.8. Repository class

`Repository` class is completely removed from system, code such as: `Repository.getFactory()` or `Repository.newTaxonomyCreator(...)` do not exist any more. Also, you don't need to insert or remove taxonomy from `Repository` class as before.

4.9. Old factory type and system properties

In ABRA 2, choice on which factory to use was done by setting system property `taxonomy-factory` to `com.abzreporting.abra.taxonomy.pdom.TaxonomyFactoryPDOM`. If property was not set, default factory implementation was used. Now in ABRA 3, this is not possible. In order to choose which factory you would like to use, you need to explicitly create particular type of `TaxonomyBuilder`. That can be accomplished by calling one of two possible groups of methods on `TaxonomyBuilderFactory`. Either `createInMemoryTaxonomyBuilder(...)` or `createPersistentTaxonomyBuilder(...)`.

4.10. Instance Validation

Interface `InstanceValidator` and class `InstanceValidatorImpl` are removed and instead of it now there is only class `InstanceValidator`. In previous version it was acquired using `InstanceValidatorFactory`:

```
InstanceValidatorFactory validatorFactory = InstanceValidatorFactory.newInstance();
InstanceValidator validator = validatorFactory.instanceValidator(taxonomyKey);
Document doc = DOMHelper.newDocument();
validator.validate(instance, new DOMResult(doc));
```


now, `Taxonomy` is used to get the `InstanceValidator`

```
InstanceValidator validator = taxonomy.newInstanceValidatorBuilder().build();
InstanceValidationResult result = validator.validate(inputXBRLInstances)
```

4.11. Dimensional validation

Dimensional validation is done as part of the instance validation. It had to be explicitly turned on in previous version:

```
InstanceValidator validator = validatorFactory.instanceValidator(taxonomyKey);
validator.setFeature(InstanceValidator.VALIDATE_DIMENSIONS, true);
```

Now it is it turned on by default. You can disable it during creation of `InstanceValidator`:

```
InstanceValidator validator = taxonomy.newInstanceValidatorBuilder()
    -.ignoreDimensionsChecks()
    -.build();
```

4.12. XSLT Execution

Usage of XSLT in ABRA 3 is highly discouraged since XSLT will e removed in some of following minor versions. However, it is still possible to use it. Since XSLT code used `Repository` static methods in execution, and `Repository` class is completely removed. New class, `XsltExecutionContext` is introduced. In order to execute XSLT, setting of `TaxonomyBuilderFactory, TaxonomyBuilderType, Taxonomy` and `XBRLDocumentManager` is needed, depending on type of XSLT. For this purpose, methods are provided:

```
public static void setTaxonomyBuilderFactory(TaxonomyBuilderFactory taxonomyBuilderFactory)
public static TaxonomyBuilderFactory getTaxonomyBuilderFactory()
public static void setTaxonomyBuilderType(TaxonomyBuilderType taxonomyBuilderType)
public static TaxonomyBuilderType getTaxonomyBuilderType()
public static void setXbrlDocumentManager(XBRLDocumentManager xbrlDocumentManager)
public static XBRLDocumentManager getXbrlDocumentManager()
public static void insertTaxonomy(String key, Taxonomy taxonomy)
public static Taxonomy getTaxonomy(String key)
public static Taxonomy removeTaxonomy(String key)
public static boolean containsTaxonomy(String key)
public static String getTaxonomyKey(Taxonomy tax)
public static Set<String> getTaxonomyKeySet()
public static void clear()
```

If you use XSLT and set some properties through `XsltExecutionContext`, you will need to clean them up by calling `XsltExecutionContext.clear()`; Not doing so, may cause memory leaks.

The class `XSLTUtil` was moved from package `com.abzreporting.abra.util` to package `com.abzreporting.abra.xslt`

4.13. Additional .NET Changes

In .NET ABRA version 3, we will have .NET wrapper around ABRA, which will use `System.Collection.Generics` collections from .NET library, instead of java collections. This is documented in .NET section.

4.14. Planned changes (to be implemented)

In the future system property `abra.config` will be removed, and instead database settings will be explicitly set.

5. ABRA Standards Conformance

5.1. XBRL 2.1

ABRA is an XBRL processor for documents that are conformant to the XBRL version 2.1. Former versions of the XBRL standard are not supported and it is very unlikely that they will be supported in future versions. We recommend converting prior XBRL documents to version 2.1.

5.2. XBRL Formula 1.0

ABRA is also fully conformant to the XBRL Formula 1.0.

5.3. XSLT 2.0

The Saxon XSLT processor implements most features of the XSL Transformations (XSLT) Version 2.0 W3C Recommendation. The included Saxon library is conformant to the basic conformance level of XSLT 2.0.

5.4. DOM Level 3

The conformance to the DOM Level 3 standard of the W3C is an important aspect for the ABRA architecture, because the XLink relations of XBRL documents are annotated as user-defined data in the DOM tree.

5.5. Java

ABRA requires Java JDK 8. It does not work with earlier JDK versions! The JAXP packages are included in JDK 8.

5.6. .NET

ABRA is also available as a .NET version 3.5 compatible package. For XSLT it produces the same output as the Java version.

6. XBRL Applications / Links

The page contains some basic information for XBRL starters and a list to XBRL tutorials and other important XBRL sources.

6.1. XBRL Application Areas

XBRL enables business companies to purify their data ones and use it repeatedly, for instance publications, analysis and consolidation. In future, XBRL will fulfill the reporting requirements towards institutions like supervisory authorities, business registers and economic research institutes. Today, XBRL is mainly used for financial accounting. There exist several XBRL formats (taxonomies) for national and international accounting methods, i.e. IAS, US GAAP and GermanAP.

6.2. XBRL Documents/Tutorials

A taxonomy defines a format of business reports. The structure and the typing of the components of a business report are defined with the XML Schema language. The relationships between components of XBRL documents are described with XLinks. The XML Schema and the collection of XLinks (linkbases) are the metadata. The instance documents contain the data (facts) of business reports. Their content of instance documents can be validated according to its meta data description.

6.3. Links

- XBRL Home Page
- An Introduction to XBRL
- XBRL Getting Started for Developers
- XBRL Tutorial at XML.com
- XBRL Technology Report at XML Coverpages