Spring XML Databases Framework

by

Table of contents

1. Introduction

The Spring XMLDB framework is designed to ease the use of XML databases with the Spring Framework. It provides configurable generic beans for dataaccess to XML databases. It also includes the following features:

- Pooling of XML:DB connections to an XML Database which can be used similarily to JDBC connection pools
- Generic DAOs for querying with XPath, XQuery and XUpdate as well as retrieving/storing documents (binary and XML) and managing collections and databases themselves.
- Embedded startup of XML databases using Spring configured beans.
- Import/Export from file systems to XML databases using Spring resource abstraction framework.
- Generic Spring controllers for configuring queries on XML databases which can then be displayed through XSLT views or using JSTL tags.
- Generic Tiles controllers for integrating query results into tiles components, allowing content display to be controlled in definitions.
- XForms type form integration for binding form elements to XML documents in Spring called SPXForms.
- Schematron Validation for XML documents using the Spring Validator interface.
- XML Database Manager example application.
- Tiles based content website and simple CMS example application.

1.1. XML:DB

The Spring XMLDB Framework makes heavy use of XML:DB APIs. The XML:DB project is used to provide a common API to XML database products. Information on XML:DB can be found at http://xmldb-org.sourceforge.net/

1.2. Supported XML Databases

Currently only eXist and Xindice are supported. Hopefully i can add support for Tamino, XHive and Berkley XMLDB in the near future.

1.2.1. eXist

eXist is an Open Source native XML database featuring efficient, index-based XQuery

processing, automatic indexing, extensions for full-text search, XUpdate support and tight integration with existing XML development tools. The database implements the current XQuery 1.0 working draft as of November, 2003 (for the core syntax, some details already following later versions), with the exception of the XML schema related features.

The eXist XML database server is supported through its XML:DB API. More information can be found at http://exist.sourceforge.net/.

1.2.2. Xindice

Apache Xindice is a database designed from the ground up to store XML data or what is more commonly referred to as a native XML database.

The Apache Xindice database server is supported through its XML:DB API. More information can be found at http://xml.apache.org/xindice/

2. XML:DB DataSource

The class `uk.co.lakesidetech.springxmldb.datasource.BasicXMLDBDataSource` provides a basic implementation of the interface `uk.co.lakesidetech.springxmldb.datasource.XMLDBDataSource` to provide pooling connections to XML database collections.

The defintions of the XMLDBDataSource is as follows:

```
public interface XMLDBDataSource {

    /**
     * Return a XML:DB Collection connection to the XML:DB database. This
should return a connection
     * to the default set collection for a database. In most cases this
would be the root collection
     * /db
     * @return an XML:DB Collection object
     * @throws XMLDBException if the collection can not be obtained
     */
     public Collection getCollection() throws XMLDBException;

    /**
     * Return a XML:DB Collection connection to the XML:DB database for a
specified collection
     * given the collection path.
     * @param collectionPath the full collection path to the collection to
get for. i.e. /db/plays/shakespeare
     * @return an XML:DB Collection object
     * @throws XMLDBException
     */
```

```
     public Collection getCollection(String collectionPath) throws
XMLDBException;

    /**
      *  Return a XML:DB Collection connection to the XML:DB database for a
specified collection
      * given the collection path and user connection details username and
password
      * @param username the connection username
      * @param password the connection password
      * @param collectionPath
      * @return an XML:DB Collection object
      * @throws XMLDBException if the collection can not be obtained
      */
      public Collection getCollection(String username, String password,
String collectionPath) throws XMLDBException;

}
```

2.1. Configuration

The BasicXMLDBDataSource can then be configured in the Spring application context to create the datasource to a particular XML Database. The database can be embedded, running in the same JVM, or on a remote server as supported by the XML:DB driver for the database.

Embeded Exist Server:

```
<bean id="xmldbDataSource"
class="uk.co.lakesidetech.springxmldb.datasource.BasicXMLDBDataSource">
  <property
name="driverClassName"><value>org.exist.xmldb.DatabaseImpl</value></property>
  <property name="url"><value>xmldb:exist:///</value></property>
  <property name="username"><value></value></property>
  <property name="password"><value></value></property>
</bean>
```

Exist server running with XML-RPC access on localhost port 8080:

```
<bean id="xmldbDataSource"
class="uk.co.lakesidetech.springxmldb.datasource.BasicXMLDBDataSource">
  <property
name="driverClassName"><value>org.exist.xmldb.DatabaseImpl</value></property>
  <property
name="url"><value>xmldb:exist://localhost:8080/exist/xmlrpc</value></property>
  <property name="username"><value></value></property>
  <property name="password"><value></value></property>
</bean>
```

Embedded Xindice server:

```
<bean id="xmldbDataSource"
class="uk.co.lakesidetech.springxmldb.datasource.BasicXMLDBDataSource">
  <property
name="driverClassName"><value>org.apache.xindice.client.xmldb.DatabaseImpl</value></pro
  <property name="url"><value>xmldb:xindice-embed://</value></property>
```

```
  <property name="username"><value></value></property>
  <property name="password"><value></value></property>
</bean>
```

2.2. Usage

Once configured in a Spring context the datasource can then be used to supply XML:DB connections to collections on the configured XML database:

```
/** the database source for the xml database */
protected XMLDBDataSource dataSource;

/**
 * @return Returns the dataSource.
 */
public XMLDBDataSource getDataSource() {
  return dataSource;
}
/**
* @param dataSource The dataSource to set.
*/
public void setDataSource(XMLDBDataSource dataSource) {
  this.dataSource = dataSource;
}

public void doQuery(String collectionPath, String xpath) {

  Collection collection = null;
  try {
   collection = dataSource.getCollection(collectionPath);

  if (collection != null) {
    XPathQueryService service =
      (XPathQueryService) collection.getService(
       "XPathQueryService",
       "1.0");
     ResourceSet resultSet = service.query(xPathQuery);
   }

  } catch (XMLDBException e) {
    log.error(e);
    throw new XMLDBDataAccessException(e.getMessage(), e);
  } finally {
    try {
     if (collection != null)
       collection.close();
    } catch (XMLDBException e2) {
      log.error(e2);
    }
  }
}
```

Calling close() on the collection from the DataSource will not close the connection, instead it

will return it to the pool to be reused. The connection pool is keyed on the collection path string so will in fact hold a map of connection pools, one pool for each collection.

3. SpringXMLDB Data Access Objects

The SpringXMLDB Framework provides a generic set of data access objects for use with XML Databases. This gives an API to a set of common operations on storing, retrieving, querying and managing XML databases. The DAOs have a set of interfaces for accessing the XML database. Some DAOs have different implementations for different vendors of database due to differences in the XML:DB API implementations and support for certain functions.

- IXPathXMLDBDao - For querying XML Database collections with XPath.
- ICollectionManageXMLDBDao - For managing XML Database collections.
- IResourceManageXMLDBDao - For managing XML Database resources such as XML and binary files.
- IXQueryXMLDBDao - For querying XML Databases with XQuery.
- IXUpdateXMLDBDao - For updating XML Database resources with XUpdate.
- IDatabaseManageXMLDBDao - For managing the database instance.

3.1. AbstractXMLDBDao

All the XMLDB DAOs extend `AbstractXMLDBDao` which provides some base functionality for the dataaccess querying. All DAOs inherit this behaviour.

3.1.1. Namespace support

The XML:DB API requires that any namespaces used in the query should be registered with the XML:DB service being used. By setting the properties in the DAO object for any namespaces used, these namespaces will be registered on any service being used.

```
<bean id="xpathXMLDBDao"
class="uk.co.lakesidetech.springxmldb.dao.BaseXPathXMLDBDao">
  <property name="dataSource"><ref bean="xmldbDataSource"/></property>
  <property name="nameSpaces">
  <map>
    <entry key="dc">
       <value>http://purl.org/dc/elements/1.1/</value>
    </entry>
    <entry key="dct">
       <value>http://purl.org/dc/terms/1.0/</value>
    </entry>
  </map>
```

```
  </property>
</bean>
```

3.1.2. Exceptions

All Spring XMLDB dataaccess methods throw `uk.co.lakesidetech.springxmldb.exception.XMLDBDataAccessException` which are runtime exceptions extending the normal Spring exception DataAccessException.

3.2. IXPathXMLDBDao

There is one implementation of `IXPathXMLDBDao` which is `BaseXPathXMLDBDao`. This should work with all XML:DB databases.

```
    /**
     * Query a collection with xpath and return a map of results with keys
of
     * the resource IDs and values of w3c DOM documents of the results (not
necessarily the resource
     * documents, but the result of the query)
     * @param xPathQuery the xpath query to use
     * @param collectionPath the collection path to query (note some XML
databases will also
     * query child collections by default)
     * @return a Map of the resourceIDs and query results
     * @throws XMLDBDataAccessException if anything goes wrong with he
query
     */
public abstract Map queryWithXPathCollectionAsDOM(String xPathQuery,
   String collectionPath) throws XMLDBDataAccessException;

 /**
     * Query a collection with xpath and return a map of results with keys
of
     * the resource IDs and values of Strings of the results (not
necessarily the resource
     * documents, but the result of the query)
     * @param xPathQuery the xpath query to use
     * @param collectionPath the collection path to query (note some XML
databases will also
     * query child collections by default)
     * @return a Map of the resourceIDs and query results
     * @throws XMLDBDataAccessException if anything goes wrong with he
query
     */
public abstract Map queryWithXPathCollectionAsString(String xPathQuery,
   String collectionPath) throws XMLDBDataAccessException;

/**
     * Query a particular with xpath and return a map of results with keys
of
     * the resource IDs and values of w3c DOM documents of the results (not
```

```
necessarily the resource
     * documents, but the result of the query)
     * @param resourceId the resource to query
     * @param xPathQuery the xpath query to use
     * @param collectionPath the collection path to find the resource
     * @return a Map of the resourceIDs and query results
     * @throws XMLDBDataAccessException if anything goes wrong with he
query
     */
public abstract Map queryResourceWithXPathCollectionAsDOM(String
resourceId,
   String xPathQuery,
   String collectionPath) throws XMLDBDataAccessException;

/**
     * Query a particular with xpath and return a map of results with keys
of
     * the resource IDs and values of String xml of the results (not
necessarily the resource
     * documents, but the result of the query)
     * @param resourceId the resource to query
     * @param xPathQuery the xpath query to use
     * @param collectionPath the collection path to find the resource
     * @return a Map of the resourceIDs and query results
     * @throws XMLDBDataAccessException if anything goes wrong with he
query
     */
public abstract Map queryResourceWithXPathCollectionAsString(String
resourceId,
   String xPathQuery,
   String collectionPath) throws XMLDBDataAccessException;
```

3.2.1. BaseXPathXMLDBDao

`BaseXPathXMLDBDao` provides an implementation of `IXPathXMLDBDao` which should work on all XML:DB databases. This allows for collections to be queried by XPath.

This basically allows querying of a collection or just a single XML resource within a collection. The results are returned as a `Map` of results, this map is keyed with id of the resource returned and the result data of xpath query on that resource. The result data is either a `String` object or a `org.w3c.dom.Document` depending on the method used.

3.3. ICollectionManageXMLDBDao

The `ICollectionManageXMLDBDao` data access objects are used to retrieve information on collections and their contents and to add/remove collections from the XML database.

```
/**
 * Add a new child collection to an underlying existing collection
 * @param existingCollectionPath The full collection path of the existing
collection i.e. /db/test/content
```

```
 * @param newCollectionName The single name of the collection to create
i.e. newcol
 * @throws XMLDBDataAccessException problems accessing the XML database
 */
public void addCollection(String existingCollectionPath, String
newCollectionName) throws XMLDBDataAccessException;

/**
 * Add a new collection path to the XML database. This method should create
any non existing
 * collections on the path
 * @param collectionPath the full path of the collection to create i.e.
/db/testcontent/newcol
 * @throws XMLDBDataAccessException problems accessing the XML database
 * */
public void addCollectionPath(String collectionPath) throws
XMLDBDataAccessException;

/**
 * Remove a collection from the XML datbase given the full collection path
 * @param collectionPath the full path of the collection to remove i.e.
/db/testcontent/newcol
 * @throws XMLDBDataAccessException problems accessing the XML database
 */
public void removeCollection(String collectionPath) throws
XMLDBDataAccessException;

/**
 * Get the information on all the resources in a collection given the
collection path
 * This method should return a collection (in the java.util sense) of
<code>XMLDBResourceInfo</code>
 * objects
 * @param collectionPath the collection path to retrieve infomation on
 * @return a java.util.Collection of XMLDBResourceInfo resource information
 * @throws XMLDBDataAccessException problems accessing the XML database
 */
public java.util.Collection getCollectionResourceInfo(String
collectionPath) throws XMLDBDataAccessException;

/**
 * Evaluate if a specific collection exists at a collection path in the
database
 * @param collectionPath The collection path to check
 * @return true if the collection exists
 * @throws XMLDBDataAccessException problems accessing the XML database
 */
public boolean doesCollectionExist(String collectionPath) throws
XMLDBDataAccessException;

/**
 * Get the information on all child collections of a collection given its
collection path
 * This method should return a collection (in the java.util sense) of
```

```
<code>XMLDBCollectionInfo</code>
 * objects
 * @param collectionPath the collection path to retrieve infomation on
 * @return a java.util.Collection of XMLDBCollectionInfo resource
information
 * @throws XMLDBDataAccessException problems accessing the XML database
 */
public java.util.Collection getCollectionCollectionInfo(String
collectionPath) throws XMLDBDataAccessException;
```

The methods `getCollectionCollectionInfo(String collectionPath)` and `getCollectionResourceInfo(String collectionPath)` allow for returning information on the child collections and resources of a collection as simple JavaBean objects.

3.3.1. BaseCollectionManageXMLDBDao

The `BaseCollectionManageXMLDBDao` provides an implementation of `ICollectionManageXMLDBDao` just using XML:DB apis. This does not work for Xindice and eXist which require specific DAO implementations to provide the full functionality.

3.3.2. XindiceCollectionManageXMLDBDao

The `XindiceCollectionManageXMLDBDao` is a DAO specific for Xindice databases. This DAO allows for metadata information on resources to be returned along with the resource information in `uk.co.lakesidetech.springxmldb.data.xindice.XindiceXMLDBResourceInfo` objects. The DAO also allows collections to be created with a customised configuration which can be set in the DAO property **newColconfig**. The use of metadata in resource information be be changed with the property **useMetadata**. By default metadata is on.

3.3.3. ExistCollectionManageXMLDBDao

The `ExistCollectionManageXMLDBDao` is a DAO specific for eXist databases. It includes retrieving information for the last modified and created time of resources.

3.4. IResourceManageXMLDBDao

The `IResourceManageXMLDBDao` data access objects provides methods for the management of resources within XML database collections. These resources can be either XML resources or binary resources.

```
/**
  * Insert or update a XML document specfied in the String supplied in the
parameters.
  * If the supplied docID is null the dao will attempt to insert with a
```

```
null id relying on the
   * database to supply a generated ID. If given a docID the dao will
attempt to find an
   * existing document in the given collection path, if a document exists it
will update that document
   * otherwise it will insert a new document with the supplied id.
   * @param xmlDocument A String of the XML document to insert/update
   * @param docID The document ID to use
   * @param collectionPath The collection path to insert under
   * @return The ID of the document in the collection
   * @throws XMLDBDataAccessException if anything goes wrong in the data
access
   */
public abstract String insertUpdateXMLDocument(String xmlDocument,
    String docID, String collectionPath)
    throws XMLDBDataAccessException;

/**
   * Insert or update a file (either XML or binary) from a <code>File</code>
object
   * to the database. If the supplied docID is null the dao will attempt to
insert with a null id relying on the
   * database to supply a generated ID. If given a docID the dao will
attempt to find an
   * existing document in the given collection path, if a document exists it
will update that document
   * otherwise it will insert a new document with the supplied id.
   * @param file The File to insert/update
   * @param docID The document ID to use
   * @param collectionPath The collection path to insert under
   * @return The ID of the document in the collection
   * @throws XMLDBDataAccessException if anything goes wrong in the data
access
   */
public abstract String insertUpdateFile(File file,
    String docID, String collectionPath)
    throws XMLDBDataAccessException;

/**
   * Insert or update a XML document from a <code>File</code> object
   * to the database. If the supplied docID is null the dao will attempt to
insert with a null id relying on the
   * database to supply a generated ID. If given a docID the dao will
attempt to find an
   * existing document in the given collection path, if a document exists it
will update that document
   * otherwise it will insert a new document with the supplied id.
   * @param xmlDocument The File containing a XML document
   * @param docID The document ID to use
   * @param collectionPath The collection path to insert under
   * @return The ID of the document in the collection
   * @throws XMLDBDataAccessException if anything goes wrong in the data
access
   */
```

```
public abstract String insertUpdateXMLDocument(File xmlDocument,
String docID, String collectionPath)
   throws XMLDBDataAccessException;

/**
  * Insert or update a XML document from a supplied DOM object
  * to the database. If the supplied docID is null the dao will attempt to
insert with a null id relying on the
  * database to supply a generated ID. If given a docID the dao will
attempt to find an
  * existing document in the given collection path, if a document exists it
will update that document
  * otherwise it will insert a new document with the supplied id.
  * @param Document The File containing a XML document
  * @param docID The document ID to use
  * @param collectionPath The collection path to insert under
  * @return The ID of the document in the collection
  * @throws XMLDBDataAccessException if anything goes wrong in the data
access
  */
public abstract String insertUpdateXMLDocument(Document xmlDocument,
   String docID, String collectionPath)
   throws XMLDBDataAccessException;

/**
 * Remove a document (XML or binary) from a collection
 * @param docID The document ID to remove
 * @param collectionPath The collection path to find the document
 * @return true if resource removed
 * @throws XMLDBDataAccessException if anything goes wrong in the data
access
 */
public abstract boolean removeDocument(String docID, String collectionPath)
   throws XMLDBDataAccessException;

/**
  * Retrieve an XML document as a w3c DOM object from a collection in the
XML database
  * @param docID The document ID to retrieve
  * @param collectionPath The collection path to find the document
  * @return A W3C DOM document
  * @throws XMLDBDataAccessException if anything goes wrong in the data
access
  */
public abstract Document retrieveDocumentAsDOM(String docID, String
collectionPath)
  throws XMLDBDataAccessException;

/**
  * Retrieve an XML document as a String from a collection in the XML
database
  * @param docID The document ID to retrieve
  * @param collectionPath The collection path to find the document
  * @return A String with the XML content
```

```
  * @throws XMLDBDataAccessException if anything goes wrong in the data
access
 */
public abstract String retrieveDocumentAsString(String docID, String
collectionPath)
  throws XMLDBDataAccessException;

/**
 * Retrieve a XML:DB Resource which can be either a binary or XML document
from a collection in the XML database
 * @param docID The document ID to retrieve
 * @param collectionPath The collection path to find the document
 * @return A XML:DB Resource with either the String or byte[] binary
content of the file
 * @throws XMLDBDataAccessException if anything goes wrong in the data
access
 */
public abstract Resource retrieveDocumentAsResource(String docID, String
collectionPath)
  throws XMLDBDataAccessException;

/**
  * Insert or update a binary byte array into a XML database. If the byte[]
is a true XML resource
  * insert/update it as a XMLResource otherwise as a binary resource
  * to the database. If the supplied docID is null the dao will attempt to
insert with a null id relying on the
  * database to supply a generated ID. If given a docID the dao will
attempt to find an
  * existing document in the given collection path, if a document exists it
will update that document
  * otherwise it will insert a new document with the supplied id.
  * @param docBytes The byte array to insert or update
  * @param docID The document ID to use
  * @param collectionPath The collection path to insert under
  * @return The ID of the document in the collection
  * @throws XMLDBDataAccessException if anything goes wrong in the data
access
  */
public abstract String insertUpdateBinaryFile(byte[] docBytes, String
docID, String collectionPath)
  throws XMLDBDataAccessException ;

/**
 * Insert or update information in a inputstream into a XML database. If
the strem result is a true XML resource
 * insert/update it as a XMLResource otherwise as a binary resource
 * to the database. If the supplied docID is null the dao will attempt to
insert with a null id relying on the
 * database to supply a generated ID. If given a docID the dao will attempt
to find an
 * existing document in the given collection path, if a document exists it
will update that document
 * otherwise it will insert a new document with the supplied id.
```

```
 * @param stream The input stream to insert or update
 * @param docID The document ID to use
 * @param collectionPath The collection path to insert under
 * @return The ID of the document in the collection
 * @throws XMLDBDataAccessException if anything goes wrong in the data
access
 */
public abstract String insertUpdateInputStream(InputStream stream, String
docID, String collectionPath)
  throws XMLDBDataAccessException ;
```

The DAO implementations provide methods for the insertion and updating of XML resources. This can be as binary objects or as XML objects, the DAO implementations will automatically insert as an XML document if a binary resource or input stream can be parsed as an XML document, otherwise it will be inserted as a binary resource. If a docID is supplied the DAO will check to see if this resource exists in the given collection path, if it exists the resource will be updated, if it does not exist the resource will be created with that docID in the given collection. If a **null** docID is supplied the DAO will rely on generating a unique ID and storing under that. In all cases the ID of the resource created/updated is returned.

3.4.1. BaseResourceManageXMLDBDao

`BaseResourceManageXMLDBDao` is a DAO implementation of `IResourceManageXMLDBDao` for all XML:DB implementations including eXist and Xindice.

3.5. IXQueryXMLDBDao

The `IXQueryXMLDBDao` DAO implementations allow querying of collections and XML resources with the XQuery language.

```
    /**
     * Query a collection with XQuery and return a map of results with keys
of
     * the resource IDs and values of String xml of the results (not
necessarily the resource
     * documents, but the result of the query)
     * @param xQuery the xquery query to use
     * @param collectionPath the collection path to query (note some XML
databases will also
     * query child collections by default)
     * @return a Map of the resourceIDs and query results
     * @throws XMLDBDataAccessException if anything goes wrong with he
query
     */
public abstract Map queryWithXQueryCollectionAsString(String xQuery,
   String collectionPath) throws XMLDBDataAccessException;
```

```
    /**
     * Query a collection with XQuery and return a map of results with keys
of
     * the resource IDs and values of String xml of the results (not
necessarily the resource
     * documents, but the result of the query). Add a list of variables to
the query which can then
     * be used in the query syntax as a replacement.
     * @param xQuery the xquery query to use
     * @param collectionPath the collection path to query (note some XML
databases will also
     * query child collections by default)
     * @param variables map of key/values that can be used for replacement
in the xquery
     * @return a Map of the resourceIDs and query results
     * @throws XMLDBDataAccessException if anything goes wrong with he
query
     */
public abstract Map queryWithXQueryCollectionAsString(String xQuery,
   String collectionPath, Map variables) throws XMLDBDataAccessException;
```

The DAO allows queries to be made with an XQuery String and supply a list of variables to be used for substitution in the XQuery. This allows variable names such as $variable to be replaced with values in the end query. As with XPath DAOs the results are returned as a map of resource ID keys and the results of the XQuery as the result data. As XQuery allows parsing of results into various forms, the result may be unrelated to the data in the resource.

3.5.1. BaseXQueryXMLDBDao

The `BaseXQueryXMLDBDao` allows for XQuery to be made with any XML:DB drivers which support `XQueryService`. This does not include Xindice which has no current support for XQuery

3.5.2. ExistXQueryXMLDBDao

The `ExistXQueryXMLDBDao` allows for XQuery with older versions of eXist. Older versions used a different API structure to XML:DB so require a separate DAO instance. Newer versions of eXist use the XML:DB API so can use the normal `BaseXQueryXMLDBDao`.

3.6. IXUpdateXMLDBDao

The allows for XUpdates to be run on collections in an XML database or individual XML resources. Information on XUpdate can be found at http://xmldb-org.sourceforge.net/xupdate/index.html

```
/**
 * run a XUpdate set of commands on a collection in an XML database
```

```
 * @param xUpdate The xupdate string
 * @param collectionPath The collection path
 * @throws XMLDBDataAccessException if something goes wrong in the XMLDB
access
 */
public abstract void updateWithXUpdate(String xUpdate, String
collectionPath) throws XMLDBDataAccessException;

/**
 * run a XUpdate set of commands on a specific resource within a collection
in an XML database
 * @param xUpdate The xupdate string
 * @param collectionPath The collection path
 * @throws XMLDBDataAccessException if something goes wrong in the XMLDB
access
 */
public abstract void updateResourceWithXUpdate(String xUpdate, String
resourceId, String collectionPath) throws XMLDBDataAccessException;
```

3.6.1. BaseXUpdateXMLDBDao

The BaseXUpdateXMLDBDao is a DAO implementation using standard XML:DB API functions and can be used with either eXist or Xindice.

3.7. IDatabaseManageXMLDBDao

The IDatabaseManageXMLDBDao DAOs provide for methods to manage the actual XML database instance.

```
 /**
   * Shutdown the database at the URI given by the dao datasource
   * @throws XMLDBDataAccessException
   */
        public void shutdown() throws XMLDBDataAccessException;
```

3.7.1. ExistDatabaseManageXMLDBDao

Currently only an implementation for eXist for database management is supported.

3.8. XMLDB Facade

The interface `uk.co.lakesidetech.springxmldb.spring.IXMLDBFacade` aggregates many of the DAO functions into a facade layer on top of an XML database. It is this layer that the generic controllers and Tiles components use. The IXMLDBFacade is just an example of the usage of the DAO layer of the SpringXMLDB framework providing a simple implmentation for querying of the database. Most application business logic can access the DAO layer directly to perform functions for querying and persisting XML data.

```
public interface IXMLDBFacade {
```

```
 public abstract void xupdateXMLDocument(String xupdate, String docID,
String collectionPath);

 public abstract void xupdateCollection(String xupdate, String
collectionPath);

 public abstract String insertUpdateXMLDocument(String xmlDocument, String
docID, String collectionPath);

 public abstract boolean removeDocument(String docID, String
collectionPath);

 public abstract String retrieveDocumentAsString(String docID, String
collectionPath) ;

 public abstract Resource retrieveDocumentAsResource(String docID, String
collectionPath) ;

 public abstract Node queryWithXPathCollectionAsString(String xPathQuery,
String collectionPath)  throws XMLParsingException ;

 public abstract Node queryWithXPathCollectionAsString(String xPathQuery,
String collectionPath, int noResults)  throws XMLParsingException ;

 public abstract Node queryResourceWithXPathCollectionAsString(String
resourceId, String xPathQuery,
 String collectionPath)  throws XMLParsingException ;

 public abstract Node queryWithXQueryCollectionAsString(String xQuery,
String collectionPath)  throws XMLParsingException ;

 public abstract Node queryWithXQueryCollectionAsString(String xQuery,
String collectionPath, int noResults)  throws XMLParsingException ;

 public abstract Node queryWithXQueryCollectionAsString(String xQuery,
String collectionPath, Map variables)  throws XMLParsingException ;

 public abstract Node queryWithXQueryCollectionAsString(String xQuery,
String collectionPath, Map variables, int noResults)  throws
XMLParsingException ;
}
```

3.8.1. SimpleXMLDBFacade

The SimpleXMLDBFacade provides a simple implementation of the facade interface.The querying of results returns a complete XML document with the results. The results are aggregated using **results** as a root element, and each resources results are contained in a **result** element with a **resourceId** attribute. This allows the result set to be transformed for presentation with a single XSLT transform.

```
<results>
   <result resourceId="id1.xml">
```

```
    ....xml results....
   </result>

   <result resourceId="id2.xml">
     ......xml results...
   </result>
</results>
```

An example of a configured application context with the SimpleXMLDBFacade would be:

```
<bean id="xmldbFacade"
class="uk.co.lakesidetech.springxmldb.spring.SimpleXMLDBFacade">
  <property name="xpathDao"><ref bean="xpathXMLDBDao"/></property>
  <property name="xqueryDao"><ref bean="xqueryXMLDBDao"/></property>
  <property name="manageDao"><ref bean="xmlManageXMLDBDao"/></property>
  <property name="xupdateDao"><ref bean="xupdateXMLDBDao"/></property>
</bean>

<bean id="xmldbDataSource"
class="uk.co.lakesidetech.springxmldb.datasource.BasicXMLDBDataSource">
  <property
name="driverClassName"><value>org.exist.xmldb.DatabaseImpl</value></property>
  <property name="url"><value>xmldb:exist:///</value></property>
  <property name="username"><value></value></property>
  <property name="password"><value></value></property>
 </bean>

 <bean id="xpathXMLDBDao"
class="uk.co.lakesidetech.springxmldb.dao.BaseXPathXMLDBDao">
  <property name="dataSource"><ref bean="xmldbDataSource"/></property>
 </bean>

 <bean id="xmlManageXMLDBDao"
class="uk.co.lakesidetech.springxmldb.dao.BaseResourceManageXMLDBDao">
  <property name="dataSource"><ref bean="xmldbDataSource"/></property>
 </bean>

 <bean id="xqueryXMLDBDao"
class="uk.co.lakesidetech.springxmldb.dao.exist.ExistXQueryXMLDBDao">
  <property name="dataSource"><ref bean="xmldbDataSource"/></property>
 </bean>

 <bean id="xupdateXMLDBDao"
class="uk.co.lakesidetech.springxmldb.dao.BaseXUpdateXMLDBDao">
  <property name="dataSource"><ref bean="xmldbDataSource"/></property>
 </bean>
```

4. Startup of XML Databases

The Spring XMLDB framework allows for the startup of XML databases using the Spring application context configuration. This allows for easy deployment of XML database systems and testing.

4.1. Xindice

SpringXMLDB provides a spring-managed bean `uk.co.lakesidetech.springxmldb.startup.xindice.XindiceManagedServer` which can be used to start an embedded Xindice instance. Note this is for use in the same JVM only as it does not include an XML-RPC server or any of the features to allow it to be used remotely. If a seperate standalone Xindice database is needed it is best to deploy the full Xindice WAR.

```
<bean id="xmldatabase"
class="uk.co.lakesidetech.springxmldb.startup.xindice.XindiceManagedServer"
init-method="startup">
  <property name="databaseRootLocation"><value>db</value></property>
</bean>
```

The Spring attribute **init-method** allows for the database to be started up after the Spring application context is initialised.

Property	Description
databaseRootLocation	Spring Resource property indicating where the root of the database directory should be. If left out this will be set to a db directory in the same directory as the configuration file or in a db directory of the starting directory if no configuration file is specified. Resource types allow abstraction of the location of a File/Directory depending on how the Spring context was started.
configurationFile	Spring Resource property indicating the XML File of the Xindice configuration. If left out it will default to a configuration with a dbroot of ./db, using metadata, and with no auto indexing. Spring Resource types allow abstraction of the location of a File/Directory depending on how the Spring context was started.

4.2. eXist

SpringXMLDB provides 2 ways to start up a eXist XML database server, either as an embedded server using `uk.co.lakesidetech.springxmldb.startup.exist.ExistEmbeddedServer` or as a full server with HTTP and XML-RPC access using `uk.co.lakesidetech.springxmldb.startup.exist.ExistStandaloneServer`.

```
<bean id="xmldatabase"
```

```
class="uk.co.lakesidetech.springxmldb.startup.exist.ExistEmbeddedServer"
init-method="startup">
  <property name="databaseName"><value>exist</value></property>
  <property name="threads"><value>5</value></property>
  <property
name="configurationFile"><value>/resources/conf.xml</value></property>
  <property name="shutdownUser"><value>admin</value></property>
  <property name="shutdownPassword"><value></value></property>
</bean>
```

Property	**Description**
databaseName	String name of the database, to be used in the XML:DB connection string.
threads	The number of threads to use in the internal connections to the underlying file collections.
configurationFile	Required: Spring Resource location of the XML eXist configuration file.
shutdownUser	The eXist user that should be used to shutdown the server.
shutdownPassword	The eXist user password that should be used to shutdown the server.
shutdownListener	Optional org.exist.xmldb.ShutdownListener class that will be registered to look for shutdown events.

Property	**Description**
databaseName	as per ExistEmbeddedServer.
threads	as per ExistEmbeddedServer.
configurationFile	as per ExistEmbeddedServer.
shutdownUser	as per ExistEmbeddedServer.
shutdownPassword	as per ExistEmbeddedServer.
httpPort	The port to be used for listening for HTTP requests for the XML database.
xmlrpcPort	The port to be used for listening for XML-RPC requests for the XML database, this is used for remote XML:DB requests.

5. Import/Export XML Database Resources

The Spring XMLDB Framework allows easy import and export of XML Database content using the Spring Resource abstraction framework.

5.1. XMLDBImportProcessor

The class `uk.co.lakesidetech.springxmldb.processor.XMLDBImportProcessor` can be configured to set up importing to an XML database through the generic SpringXMLDB DAOs.

```
<bean id="uploadProcessor"
class="uk.co.lakesidetech.springxmldb.processor.XMLDBImportProcessor">
  <property name="xmldbManageDao"><ref
bean="xmlManageXMLDBDao"/></property>
  <property name="collectionManageDao"><ref
bean="collectionManageXMLDBDao"/></property>
</bean>
```

Method	Description
public String uploadFile(File file, String collectionPath)	Import a file on the local filesystem into the indicated collection path on the currently configured XMLDB datasource connection (through the configured DAOs). If the collection path does not exist it will be created. The filename will be used as the resource Id.
public Map uploadDirectory(File directory, String startingCollectionPath, boolean recursive)	Import a entire directroy on the local filesystem into the indicated collection path on the currently configured XMLDB datasource connection (through the configured DAOs). If the collection path does not exist it will be created. If the recursive flag is true then the directory will be walked for sub-directories. If a sub-directory is encountered a collection will be created for that directory in the set collection path and the contents of that sub-directory recursively imported. The filename will be used as the resource Id.

5.1.1. XMLDBAutoLoader

The class `uk.co.lakesidetech.springxmldb.processor.XMLDBAutoLoader` is an extension of `uk.co.lakesidetech.springxmldb.processor.XMLDBImportProcessor` that can be used to import a directory recursive when the application context is started. When

used in combination with embedded databases this can be very useful for loading test content.

```
<bean id="uploadProcessor"
class="uk.co.lakesidetech.springxmldb.processor.XMLDBAutoLoader"
init-method="init">
  <property name="xmldbManageDao"><ref
bean="xmlManageXMLDBDao"/></property>
  <property name="collectionManageDao"><ref
bean="collectionManageXMLDBDao"/></property>
  <property
name="initDirectory"><value>resources/content</value></property>
  <property name="collection"><value>/db/testcontent</value></property>
</bean>
```

Property	Description
initDirectory	Spring Resource property indicating the directory to start importing from. This directory will be imported recursively.
collection	The collection with which to start importing to. Defaults to /db

5.2. XMLDBExportProcessor

The class `uk.co.lakesidetech.springxmldb.processor.XMLDBExportProcessor` can be used to export XML database resources to a file system directory.

```
<bean id="exportProcessor"
class="uk.co.lakesidetech.springxmldb.processor.XMLDBExportProcessor">
  <property name="xmldbManageDao"><ref
bean="xmlManageXMLDBDao"/></property>
  <property name="collectionManageDao"><ref
bean="collectionManageXMLDBDao"/></property>
</bean>
```

Method	Description
public Map exportCollection(File directory, String collectionPath, boolean recursive)	Export the contents of a XML database collection to a local filesystem directory. If the recursive flag is set any sub-collections of the collection path will be outputted to the filesystem as directories and the contents of those directories recursively outputted to the new directory.

6. Spring XMLDB Controllers.

The Spring XMLDB web MVC controllers allow for some easy integration queries with

backend XML databases to be created for request handling in a web application. This allows for application dataaccess to be handled by adding and configuring Spring controllers without writing any dataaccess code. The Spring XMLDB controllers in general return XML results documents which can then be transformed with XSLT (or just rendered as XML), or as a standard JSP page where the XML document can be manipulated (usually with JSTL XML Tags).

All the Spring XMLDB controllers are extended from `uk.co.lakesidetech.springxmldb.spring.web.controller.AbstractXMLDBControl` The `AbstractXMLDBController` allows the collection to be accessed to be set from the bean properties or if the **acceptCollectionFromRequest** property is set to true then the collection can be passed in as a request parameter **collection**. In this if the bean property **collectionPrefix** is set this will be prefixed to the start of the collection. It is also possible to set the resolving view name and the result limit as properties.

Property	Description
xmldbFacade	A reference to the XML Facade implementing bean.
collection	The collection to query.
acceptCollectionFromRequest	If set to true the collection to query can be supplied as a request parameter collection.
collectionPrefix	If the collection is accepted as a request parameter then this property will be prefixed to the collection.
resultLimit	The maximum number of results to return and display.
view	The rendering view name to use.

6.1. SimpleXMLDBXsltView

`uk.co.lakesidetech.springxmldb.spring.web.view.SimpleXMLDBXsltView` is an implementation of `org.springframework.web.servlet.view.xslt.AbstractXsltView` used to aggregate multi results from the XMLDB controllers. In general the controller queries just set a model result of type Node which `AbstractXsltView` will render using the XSLT resource. Little known fact that if the xslt property is not set, then `AbstractXsltView` will just render the XML to the browser.

6.2. UrlFilenameXMLDBResourceController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.UrlFilenameXMLDBResc` retrieves a resource from the set collection (see AbstractXMLDBController) using the filename of the request URL as the id. The controller retrieves the resource and if it is a binary resource streams back the file to the browser, if it is an XML resource the xml document is added to the model return as a DOM Node to be rendered by the view. When used with a `AbstractXsltView` like `SimpleXMLDBXsltView` then the XML can be rendered or transformed. As a side effect of the way resources are rendered if the document includes a stylesheet or CSS reference the stylesheet/CSS can also be rendered from the XML database by this controller.

Property	Description
filenameSuffix	If set this suffix will be added to the request filename before looking up the resource ID. For instance a request **/example/xmlurl/test1** with a fileNameSuffix of **.xml** will become test1.xml.

```
<bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="content/*">contentController</prop>
    </props>
  </property>
</bean>

<bean id="contentController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.UrlFilenameXMLDBResourceCon
  <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
  <property name="view"><value>urlxml</value></property>
</bean>
```

6.3. AdvancedUrlFilenameXMLDBResourceController

Expands on `uk.co.lakesidetech.springxmldb.spring.web.controller.UrlFilenameXMLDBResc` to retrieve a resource using the filename on the request URL but also determining the collection to retrieve from using the pre-part of the URL. The **ignorePathForCollection** parameter is used to remove the first part of the URL to determine the collection to retrieve the resource from.

Property	Description
ignorePathForCollection	The controller will remove this string from the

	beginning of the URL (after the context and servlet name is removed) and use the rest of the URL to calculate the collection to query for the resource.

In the example bellow a URL of **http://localhost:8008/springxmldbmanage/a/content/db/samples/shakespeare/hamlet.xml** will result in the resource being retrieved from collection of **/db/samples/shakespeare** with a resource ID of **hamlet.xml.**

```
<bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="content/**">contentController</prop>
    </props>
  </property>
</bean>

<bean id="contentController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.AdvancedUrlFilenameXMLDBRes
  <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
  <property
name="ignorePathForCollection"><value>content/</value></property>
  <property name="view"><value>urlxml</value></property>
</bean>
```

6.4. SimpleXPathXMLDBController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.SimpleXPathXMLDBCont` is used to run an xpath query against a collection in an XML database where the xpath to run is supplied as a property of the object and can be set in the context configuration file. The results are returned as a XML Node aggregated as per SimpleXMLDBFacade, this can then be outputted by the view, transformed with XSLT or used in a JSP view layer (using JSTL XML tags).

Property	Description
xpath	The xpath to query the set collection with.

```
<bean id="listItemsController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.SimpleXPathXMLDBController"
    <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
    <property
name="collection"><value>/db/tilesnews/items</value></property>
    <property name="view"><value>listItems</value></property>
    <property name="xpath"><value>/item[metadata/type/text()='page'
    and metadata/subjects/subject/name/text()='Spring']</value></property>
</bean>
```

6.5. ParameterXPathXMLDBController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.ParameterXPathXMLDBC` is used to run an xpath query against a collection in an XML database where the xpath to run is supplied a request parameter (either through a form post or on the query string).

Property	Description
xpathParameter	The request parameter which will supply the xpath query.

```
<bean id="xpathParameterController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.ParameterXPathXMLDBControll
    <property
name="acceptCollectionFromRequest"><value>true</value></property>
    <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
    <property name="view"><value>urlxml</value></property>
    <property name="xpathParameter"><value>xpath</value></property>
</bean>
```

6.6. MultiXPathXMLDBController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.MultiXPathXMLDBContr` is used to run multiple xpath queriesagainst a collection in an XML database and aggregate the results into a Map of query results in XML against the query name. The queries to run are set using a map property of the controller with a query name and the xpath query.

Property	Description
xpathQueries	A map of the xpath queries to run on the set collection with the key as the query name and the value as the XPath query.

```
<bean id="xpathMutliQueriesController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.MultiXPathXMLDBController">
    <property
name="acceptCollectionFromRequest"><value>false</value></property>
    <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
    <property name="view"><value>urlxml</value></property>
    <property name="xpathQueries">
          <map>
               <entry key="query1">
                   <value>//item[metadata/type/text()='page']</value>
               </entry>
               <entry key="query2">
                   <value>//item[metadata/type/text()='news']</value>
               </entry>
            </map>
```

```
    </property>
</bean>
```

Using the SimpleXMLDBXsltView to aggregate the results would result in a XML file of:

```
<?xml version="1.0"?>
<root>
  <query1>
    <results>
      <result resourceId="test1.xml">
        .....
      </result>
      <result resourceId="test2.xml">
        .....
      </result>
    </results>
  </query1>
  <query2>
     <results>
      <result resourceId="test5.xml">
        .....
      </result>
      <result resourceId="test8.xml">
        .....\
      </result>
    </results>
  </query2>
</root>
```

6.7. XPathRequestParamSubXMLDBController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.XPathRequestParamSub` is used to run an xpath query against a collection in an XML database where the xpath to run is supplied as a property of the object and can be set in the context configuration file. The controller will also replace any values of **$variable** in the xpath query with the value of request parameters of that name allowing query/form parameters to be inserted in the query string.

Property	Description
xpath	The xpath to query the set collection with.

```
<bean id="listItemsController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.XPathRequestParamSubXMLDBCo
    <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
    <property
name="collection"><value>/db/tilesnews/items</value></property>
    <property name="view"><value>listItems</value></property>
    <property name="xpath"><value>/item[metadata/type/text()='page'
    and
metadata/subjects/subject/name/text()='$page_subject']</value></property>
</bean>
```

6.8. SimpleXQueryXMLDBController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.SimpleXQueryXMLDBCon` is used to run an XQuery (http://www.w3.org/TR/xquery/) query against a collection in an XML database where the XQuery to run is supplied as a property of the object and can be set in the context configuration file.

Property	Description
xquery	A XQuery string to run on the collection.

```
<bean id="listItemsController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.SimpleXQueryXMLDBController
    <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
    <property
name="collection"><value>/db/tilesnews/items</value></property>
    <property name="view"><value>listItems</value></property>
    <property name="xquery"><value>for $item in //item order by
xs:dateTime($item/metadata/created)
     descending return $item</value></property>
</bean>
```

6.9. XQueryRequestParametersXMLDBController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.XQueryRequestParamet` is used to run an XQuery (http://www.w3.org/TR/xquery/) query against a collection in an XML database where the xquery to run is supplied as a property of the object and can be set in the context configuration file. The controller will also pass all request parameters as a map to be used for parameter replacement in the xquery. This allows parameters **$variable** to be replaced with values from form or query string input.

Property	Description
xquery	A XQuery string to run on the collection or individual resource.

6.10. FileXQueryXMLDBController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.FileXQueryXMLDBContr` runs an XQuery stored in a File resource against an XML database collection. This is useful for large XQueries. The file is loaded using Spring's resource abstraction framework. On reading the file the controller caches the result and the modification time of the query, if file is subsequently modified the query is reloaded.

Property	Description
xqueryFile	A Spring Resource file to use to load the XQuery from.

```
<bean id="listItemsController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.FileXQueryXMLDBController">
    <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
    <property
name="collection"><value>/db/tilesnews/items</value></property>
    <property name="view"><value>listItems</value></property>
    <property
name="xqueryFile"><value>/WEB-INF/resources/listItems.xql</value></property>
</bean>
```

6.11. RemoveXMLDBResourceController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.RemoveXMLDBResourceC` will remove a resource from a collection given a request parameter **id** of the resourceId to remove from the collection.

6.12. XUpdateXMLDBController

The `uk.co.lakesidetech.springxmldb.spring.web.controller.xupdate.XUpdateXMLDB` can be used to run XUpdate (http://xmldb-org.sourceforge.net/xupdate/) queries on either an entire collection or a single resource on a collection. The XUpdate query to run is supplied as a JavaBean property in the context configuration. If a request parameter **id** is supplied then the XUpdate query will be run only against that resourceId in the collection otherwise it will be run against the entire collection.

Property	Description
xupdate	The XUpdate query to run

```
<bean id="xupdateCommentsController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.xupdate.XUpdateXMLDBControl
  <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
  <property name="collection"><value>/db/tilesnews/items</value></property>
  <property name="view"><value>redirect:index.html</value></property>
  <property name="xupdate"><value><![CDATA[
<xupdate:modifications version="1.0"
      xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:append select="/addresses/address[@id = 1]/phone[@type='work']"
>
      <xupdate:attribute name="extension">223</xupdate:attribute>
   </xupdate:append>
</xupdate:modifications>
```

```
  ]]></value></property>
</bean>
```

6.13. XUpdateRequestParamSubXMLDBController

The
`uk.co.lakesidetech.springxmldb.spring.web.controller.xupdate.XUpdateReque`
works the same as
`uk.co.lakesidetech.springxmldb.spring.web.controller.xupdate.XUpdateXMLDB`
except it will replace instances of **$variable** with values from the request parameters supplied either by a form or on the query string.

Property	Description
xupdate	The XUpdate query to run

```
<bean id="xupdateCommentsController"
class="uk.co.lakesidetech.springxmldb.spring.web.controller.xupdate.XUpdateRequestParam
  <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
  <property name="collection"><value>/db/tilesnews/items</value></property>
  <property name="view"><value>redirect:index.html</value></property>
  <property name="xupdate"><value><![CDATA[
<xupdate:modifications version="1.0"
xmlns:xupdate="http://www.xmldb.org/xupdate" >
 <xupdate:append select="//item/comments" child="last()">
   <xupdate:element name="comment">
     <name>$name</name>
     <email>$email</email>
     <url>$url</url>
     <commentText>$commentText</commentText>
   </xupdate:element>
  </xupdate:append>
</xupdate:modifications>
  ]]></value></property>
</bean>
```

7. Spring XMLDB Tiles Components

The Spring XMLDB Tiles Components provide very similar functionality to the controllers except can be used as Tiles component controllers to be configured in Tiles definition files and deliver content XML to different parts of the page. The example Tiles news site shows these components in action using JSTL xml:transform tag to transform the XML results with XSLT.

7.1. AbstractXMLDBTilesComponent

The
`uk.co.lakesidetech.springxmldb.spring.web.tiles.AbstractXMLDBTilesCompone`

a base class for all the tiles components. All other SpringXMLDB Tiles components use the same attributes. All the SpringXMLDB components use the XMLDBFacade which is looked up in the application context. The bean name of the facade is passed in as attribute to the tiles component.

Tiles Attribute	Description
facadeName	The Spring bean name of the XMLDB Facade, this is used to lookup the facade bean from the ApplicationContext.
collection	The collection to run the query against.
noResults	The maximum number of results to return

All the tiles components expose the XML results as a attribute **results**. Using the tiles attribute inheritance of definitions it is easier to define a base definition for SpringXMLDB components which can then be overridden if necessary:

```
<definition name="xmldbbase">
  <put name="facadeName" value="xmldbFacade"/>
  <put name="noResults" value="10"/>
  <put name="collection" value="/db/tilesnews/items"/>
</definition>
```

7.2. SimpleXPathXMLDBTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.SimpleXPathXMLDBTilesComp` works much like `uk.co.lakesidetech.springxmldb.spring.web.controller.SimpleXPathXMLDBCont` and makes an xpath query on the given collection and sets the results as the attribute **results**.

Tiles Attribute	Description
xpath	The XPath query to make on the XML database collection.

```
<definition name="news" extends="xmldbbase"
page="/WEB-INF/jsp/xsltresults.jsp"
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.SimpleXPathXMLDBTilesC
  <put name="noResults" value="10"/>
  <put name="xslt" value="/xslt/news.xsl"/>
  <put name="xpath" value="//item[metadata/type/text()='news']"/>
</definition>
```

In this example the jsp file xsltresults.jsp looks like this.

```
<%@ taglib prefix="tiles" uri="http://jakarta.apache.org/struts/tags-tiles"
%>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
```

```
<%@ taglib prefix="x" uri="http://java.sun.com/jstl/xml" %>

<tiles:useAttribute id="idSylesheet" name="xslt" ignore="true"/>
<c:import var="xslt" url="${idSylesheet}" />
<tiles:importAttribute name="results"/>
<x:transform xml="${results}" xslt="${xslt}">
</x:transform>
```

This makes an XSLT transform of the XML results for this Tiles components and allows the XSLT used to be set in the definition file. Using this method it is possible to select the content and form a presentation for display all controlled by the Tiles configuration file.

7.3. XPathRequestParamSubTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.XPathRequestParamSubTiles` works much like `uk.co.lakesidetech.springxmldb.spring.web.controller.`XPathRequestParamSubXMI and makes an xpath query on the given collection and sets the results as the attribute **results**, but it also subsititutes any request parameters given **$request-param-name** in the xpath.

Tiles Attribute	Description
xpath	The XPath query to make on the XML database collection.

```
<definition name="news" extends="xmldbbase"
page="/WEB-INF/jsp/xsltresults.jsp"
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.XPathRequestParamSubTi
  <put name="noResults" value="10"/>
  <put name="xslt" value="/xslt/news.xsl"/>
  <put name="xpath" value="//item[metadata/type/text()='$type']"/>
</definition>
```

7.4. XPathAttributesSubTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.XPathAttributesSubTilesCo` works much like `uuk.co.lakesidetech.springxmldb.spring.web.tiles.XPathRequestParamSubTile` and makes an xpath query on the given collection and sets the results as the attribute **results**, but it also subsititutes any attributes in the tiles configuration given**$attribute-name** in the xpath. This allows for overriding definitions to alter the xpath slightly.

Tiles Attribute	Description
xpath	The XPath query to make on the XML database collection.

```
<definition name="contentbase" extends="xmldbbase"
page="/WEB-INF/jsp/xsltresults.jsp"
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.XPathAttributesSubTile
  <put name="noResults" value="10"/>
  <put name="xpath" value="//item[metadata/type/text()='$type']"/>
</definition>

<definition name="news" extends="contentbase">
  <put name="type" value="news"/>
  <put name="xslt" value="/xslt/news.xsl"/>
</definition>

<definition name="features" extends="contentbase">
  <put name="type" value="feature"/>
  <put name="xslt" value="/xslt/feature.xsl"/>
</definition>
```

7.5. XMLResourceXMLDBTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.XMLResourceXMLDBTilesComp` allows for a single XML resource to be returned and outputed in the Tiles Component. The resource is selected from the collection using the Tiles attribute id.

Tiles Attribute	Description
id	The id of the resource to retrieve.

```
<definition name="mainstory" extends="xmldbbase"
page="/WEB-INF/jsp/xsltresults.jsp"
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.XMLResourceXMLDBTilesC
  <put name="xslt" value="/xslt/displayarticle.xsl"/>
  <put name="collection" value="/db/articles"/>
  <put name="id" value="bigstory.xml"/>
</definition>
```

7.6. XMLResourceRequestParamXMLDBTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.XMLResourceRequestParamXM` allows for a single XML resource to be returned and outputed in the Tiles Component. The resource is selected using a resourceId supplied as a request parameter, either as a form element or request parameter. The default request parameter name to use is *id* but this can be changed with the Tiles attribute **idParamName**.

Tiles Attribute	Description
idParamName	The request parameter name to get the id of the resource to retrieve. Defaults to id.

```
<definition name="mainstory" extends="xmldbbase"
```

```
page="/WEB-INF/jsp/xsltresults.jsp"
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.XMLResourceRequestPara
  <put name="xslt" value="/xslt/displayarticle.xsl"/>
  <put name="collection" value="/db/articles"/>
  <put name="idParamName value="articleid"/>
</definition>
```

7.7. SimpleXQueryXMLDBTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.SimpleXQueryXMLDBTilesCom` works much like `uk.co.lakesidetech.springxmldb.spring.web.controller.SimpleXQueryXMLDBCon` and makes an XQuery query on the given collection and sets the results as the attribute **results**.

Tiles Attribute	Description
xquery	The XQuery query to make on the XML database collection.

```
<definition name="journalitem" extends="xmldb"
page="/WEB-INF/jsp/xsltresults.jsp"
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.SimpleXQueryXMLDBTiles
  <put name="xslt" value="/xslt/journal.xsl"/>
  <put name="xquery" value="for $item in
//item[metadata/type/text()='journal']
    where some $m in $item/metadata satisfies (xs:dateTime($m/issued) &lt;
fn:current-dateTime()
    and xs:dateTime($m/valid) &gt; fn:current-dateTime()) order by
xs:dateTime($item/metadata/created)
    descending return $item"/>
</definition>
```

7.8. XQueryRequestParametersTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.XQueryRequestParametersTi` works much like `uk.co.lakesidetech.springxmldb.spring.web.controller.XQueryRequestParamet` and makes an XQuery query on the given collection and sets the results as the attribute **results**. The component also sets XQuery variables for all request parameters.

Tiles Attribute	Description
xquery	The XQuery query to make on the XML database collection.

```
<definition name="journalitem" extends="xmldb"
page="/WEB-INF/jsp/xsltresults.jsp"
```

```
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.XQueryRequestParameter
  <put name="xslt" value="/xslt/journal.xsl"/>
  <put name="xquery" value="for $item in
//item[metadata/type/text()='journal']
    where some $m in $item/metadata satisfies (xs:dateTime($m/issued) &lt;
fn:current-dateTime()
    and xs:dateTime($m/valid) &gt; fn:current-dateTime()
    and $m/subject/name/text()=$subject) order by
xs:dateTime($item/metadata/created)
    descending return $item"/>
</definition>
```

7.9. XQueryAttributesTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.XQueryRequestParametersTi` works much like `uk.co.lakesidetech.springxmldb.spring.web.tiles.XQueryRequestParametersTi` and makes an XQuery query on the given collection and sets the results as the attribute **results**. The component also sets XQuery variables for all Tiles attributes. This allows for overriding definitions to alter the XQuery slightly.

Tiles Attribute	Description
xquery	The XQuery query to make on the XML database collection.

```
<definition name="journalitem" extends="xmldb"
page="/WEB-INF/jsp/xsltresults.jsp"
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.XQueryAttributesTilesC
  <put name="xslt" value="/xslt/journal.xsl"/>
  <put name="xquery" value="for $item in
//item[metadata/type/text()='journal']
    where some $m in $item/metadata satisfies (xs:dateTime($m/issued) &lt;
fn:current-dateTime()
    and xs:dateTime($m/valid) &gt; fn:current-dateTime()
    and $m/subject/name/text()=$subject) order by
xs:dateTime($item/metadata/created)
    descending return $item"/>
</definition>

<definition name="springjournalitem" extends="journalitem">
  <put name="subject" value="Spring"/>
</definition>

<definition name="londonjournalitem" extends="journalitem">
  <put name="subject" value="London"/>
</definition>
```

7.10. FileXQueryTilesComponent

The `uk.co.lakesidetech.springxmldb.spring.web.tiles.FileXQueryTilesComponent` works much like `uk.co.lakesidetech.springxmldb.spring.web.controller.FileXQueryXMLDBContr it` runs an XQuery stored in a File resource against an XML database collection. This is useful for large XQueries. The file is loaded from the servlet context, the filelocation should therefore be relative to the root of the context. The component makes an XQuery query on the given collection and sets the results as the attribute **results**. The component also sets XQuery variables for all Tiles attributes. This allows for overriding definitions to alter the XQuery slightly.

Tiles Attribute	Description
xqueryfileLocation	The file location of the xquery relative to the root of the web application context.

```
<definition name="journalitem" extends="xmldb"
page="/WEB-INF/jsp/xsltresults.jsp"
controllerClass="uk.co.lakesidetech.springxmldb.spring.web.tiles.FileXQueryTilesCompone
  <put name="xslt" value="/xslt/journal.xsl"/>
  <put name="xqueryfileLocation" value="/WEB-INF/query/journal.xql"/>
</definition>
```

8. SPXForms

The *SPXForms* implementation allows for binding of form elements to XML documents in a similar way to Spring's Form Controllers work. An *SPXForm* controller can be used to load an XML document, edit it on a form and have the submission rebound to the XML document before validating and processing. The *SPXForm* can be used to create new XML documents or edit existing ones. It is not necessary to use XML databases with *SPXForms*, the implementation is only concerned with the binding of XML documents.

SPXForms is an XForms (http://www.w3.org/MarkUp/Forms/) -like implementation. It is very difficult to build a fully compliant XForms engine using serverside technology as the original specification was designed to be implemented as a clientside application. Currently serverside XForms implementations of some form exist at Chiba (http://chiba.sourceforge.net/) and OXF (http://www.orbeon.com/) . *SPXForms* currently only supports a small subset of XForms functionality.

SPXForms consists of three main elements.

- A serverside binding framework. Similar to the Spring binding framework, this binds request parameters to the command object XML document and also runs xforms actions submited to the serverside.
- A base `AbstractSpXFormsController` which should be extended to create Form

Controllers for processing XML documents.

- A JSP tag library for binding form elements to the XML document and for XForms tags such as *triggers*, *actions*, *groups*, *repeats* and *submits*.

8.1. Creating a new SPXForm Controller.

The base class `uk.co.lakesidetech.spxforms.web.servlet.mvc.AbstractSpXFormsController` provides a base class for form controllers to use the SPXForms binding and tag framework.

A concrete form controller should extend and provide an implementation of the abstract method:

```
protected abstract Object formXFormsInstance(HttpServletRequest request)
```

This is used to get the Object instance used for the XForms binding, this can be used to get an existing XML document, create a new one or create a document with default values. Currently the following binding objects are supported:

- `org.w3c.dom.Document`
- `org.jdom.Document`
- JavaBeans (not yet but soon)

The form submission can be handled by overriding the method :

```
protected ModelAndView processXFormSubmission(
            HttpServletRequest request, HttpServletResponse response,
Object command, BindException errors)
            throws Exception {
```

Alternatively any other method name can be used for different submissions using a tag `<spxform:submit submission="methodName"/>` . This allows for different submission methods to be used depending on what submission button was used, as per the XForms specification (http://www.w3.org/TR/xforms/slice11.html) .

```
protected ModelAndView methodName(
            HttpServletRequest request, HttpServletResponse response,
Object command, BindException errors)
            throws Exception {
```

Property	Description
commandName	The name of the command object, the XML document, to be set on the request attribute
commandClass	The class type of the command object (currently only `org.w3c.dom.Document` or `org.jdom.Document` is supported).
validators	An array of Validator objects to validate the XML

	input. (Schema validation to be supported soon)
validateOnBinding	Flag to validate on the binding of the form elements
messageCodesResolver	
bindOnNewForm	Use request parameters to bind initial values on a new form.
successView	The view to render on successful form submission.
formView	The view to render displaying the form itself.

8.1.1. Example

The following is an example of an SPXFormController. The uk.co.lakesidetech.spxforms.web.servlet.mvc.EditXMLFileSpXFormsController can be used to edit an XML file saved on disc and referenced through the Spring resource abstraction framework.

```
public class EditXMLFileSpXFormsController extends
AbstractSpXFormsController {

    private Resource xmlFile;

    /** logging class */
    private static Log log =
LogFactory.getLog(EditXMLFileSpXFormsController.class);

    /**
     * @param xmlFile The xmlFile to set.
     */
    public void setXmlFile(Resource xmlFile) {
        this.xmlFile = xmlFile;
    }

    protected Object formXFormsInstance(HttpServletRequest request) {
//      Create a factory object for creating DOM parsers
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
        factory.setValidating(false);
//         Now use the factory to create a DOM parser (a.k.a. a
DocumentBuilder)
        DocumentBuilder parser;

        try {
            parser = factory.newDocumentBuilder();
//          Parse the file and build a Document tree to represent its
content
```

```
            return parser.parse(xmlFile.getFile());
        } catch (ParserConfigurationException e) {
            log.error(e);
            throw new NestableRuntimeException(e);
        } catch (SAXException e) {
            log.error(e);
            throw new NestableRuntimeException(e);
        } catch (IOException e) {
            log.error(e);
            throw new NestableRuntimeException(e);
        }

    }

    protected ModelAndView processXFormSubmission(
            HttpServletRequest request, HttpServletResponse response,
Object command, BindException errors)
            throws Exception {
        return save(request,response,command, errors);
    }

    protected ModelAndView save(
            HttpServletRequest request, HttpServletResponse response,
Object command, BindException errors)
            throws Exception {
        Document doc = (Document)command;
//      Prepare the DOM document for writing
        Source source = new DOMSource(doc);

        // Prepare the output file
        File file = xmlFile.getFile();
        Result result = new StreamResult(file);

        // Write the DOM document to the file
        Transformer xformer =
TransformerFactory.newInstance().newTransformer();
        xformer.transform(source, result);

        return new ModelAndView(getSuccessView(), errors.getModel());
    }
}
```

8.2. SPXForm JSP Tags

The binding of form elements to the XML document is supported by a similar tag to `<spring:bind>`. It also has other tags based on the XForms specification for manipulating the XML instance document.

8.2.1. <spxform:bind>

The `spxform:bind` tag provides support for evaluating a node of the XML document, as

per the normal spring:bind tag the status object contains the value of node and the expression used for binding a form element to that XML node. The xml nodes are expressed as **xpath** expressions. The `spxform:bind` uses a **ref** attribute which can either start with the name of the command object (set in the form controller property) followed by the xpath expression of the node to bind to or just the xpath expression to bind to if the command name is the default, nor overidden by the FormController property. The `spxform:bind` tag currently replaces any of the presentation agnostic tags for elements in xforms such as input, secret, textarea, select, select1.

It is not neccessary for the xpath paths in the document to exist when they are referenced, instead the Binding on the serverside will create these paths when the binds are made. This allows for creation of new document structures directly from the form elements.

As an example given the XML document as the instance document with command name *instance*:

```
<person>
  <firstname>Joe</firstname>
  <surname>Smith</surname>
  <address>
    <street1>224 Latimer Road</street1>
    <street2></street2>
    <province>Shepards Bush</province>
    <city>London</city>
    <county>London</county>
    <postcode>W5 3ER</postcode>
    <country>United Kingdom</country>
  </address>
</person>
```

You could use the following bind tag to bind to a form

```
<%@ taglib prefix="spxform"
uri="http://www.lakesidetech.co.uk/spxforms/tags" %>
.....
<spxform:bind ref="instance/person/address/street1">
  street1: <input type="text" name="<c:out value='${status.expression}'/>"
size="125" value="<c:out value='${status.value}'/>" />
</spxform:bind>
<br/>
<spxform:bind ref="instance/person/address/street2">
  street2: <input type="text" name="<c:out value='${status.expression}'/>"
size="25" value="<c:out value='${status.value}'/>" />
</spxform:bind>
<br/>
<spxform:bind ref="instance/person/address/province">
  province: <input type="text" name="<c:out value='${status.expression}'/>"
size="25" value="<c:out value='${status.value}'/>" />
</spxform:bind>
```

Property	Description

ref	The ref to bind to on the command object in the form of *commandname/xpath-to-node* (except where a previous group object has set the command name). The attribute ref is used instead of path to conform with XForms specification.
id	Optional id of the bind. Used to conform with the XForms specification.
htmlEscape	If the output of the status object should be escaped for HTML.
Property	**Description**
status	`uk.co.lakesidetech.spxforms.web.servlet.support.S` object which contains a value of the selected node and expression for binding the request parameter

8.2.2. <spxform:group>

The `spxform:group` tag is used much like the spring nestedPath tag is used as a container for defining a hierarchy of form controls. Groups can be nested to create complex hierarchies. The ref used in the group tag is then applied to all the binds used within it. Using the previous XML file as an example a form implementation could be.

```
<%@ taglib prefix="spxform"
uri="http://www.lakesidetech.co.uk/spxforms/tags" %>
.....
<spxform:group ref="instance/person">
  <spxform:bind ref="firstname">
    firstname: <input type="text" name="<c:out
value='${status.expression}'/>" size="10" value="<c:out
value='${status.value}'/>" />
  </spxform:bind>
  <br/>
  <spxform:bind ref="surname">
    surname: <input type="text" name="<c:out
value='${status.expression}'/>" size="10" value="<c:out
value='${status.value}'/>" />
  </spxform:bind>
  <br/>
  <spxform:group ref="address">
   <spxform:bind ref="street1">
     street1: <input type="text" name="<c:out
value='${status.expression}'/>" size="125" value="<c:out
value='${status.value}'/>" />
   </spxform:bind>
   <br/>
```

```
   <spxform:bind ref="street2">
    street2: <input type="text" name="<c:out
value='${status.expression}'/>" size="25" value="<c:out
value='${status.value}'/>" />
   </spxform:bind>
   <br/>
   <spxform:bind ref="province">
    province: <input type="text" name="<c:out
value='${status.expression}'/>" size="25" value="<c:out
value='${status.value}'/>" />
   </spxform:bind>
  </spxform:group>:
</spxform:group>
```

Property	**Description**
ref	The ref to bind to on the command object in the form of *commandname/xpath-to-node* for the containing group of bind elements.The attribute ref is used instead of path to conform with XForms specification.
id	Optional id of the group. Used to conform with the XForms specification.
Property	**Description**
nestedRef	String of the current nested ref which will be used by the `spxform:bind` tags.

8.2.3. <spxform:repeat>

The spxform:repeat is used to iterate over a set of nodes. This element defines a UI mapping over a homogeneous collection selected by Node Set Binding Attributes. This node-set must consist of contiguous child element nodes, with the same local name and namespace name of a common parent node. The nodeset is defined by the attribute nodeset using an xpath expression as in the bind tag. The tag will iterate over existing members of the collection creating a nested reference for each node Using the following example XML.

```
<person>
  <firstname>Joe</firstname>
  <surname>Smith</surname>
  <address>
    <street1>224 Latimer Road</street1>
    <street2></street2>
    <province>Shepards Bush</province>
    <city>London</city>
    <county>London</county>
    <postcode>W5 3ER</postcode>
    <country>United Kingdom</country>
  </address>
```

```
  <telephonenumbers>
       <number type="home">
           <fullnumber>020755683934</fullnumber>
           <extension></extension>
       </number>
       <number type="office">
           <fullnumber>02087005000</fullnumber>
           <extension>345</extension>
       </number>
       <number type="mobile">
           <fullnumber>07945348865</fullnumber>
           <extension></extension>
       </number>
</person>
```

We can use the following repeat document to edit the telephonenumbers:

```
<%@ taglib prefix="spxform"
uri="http://www.lakesidetech.co.uk/spxforms/tags" %>
.....
<spxform:group ref="instance/person">
  <spxform:repeat nodeset="telephonenumbers/number">
    type: <spxform:bind ref="@type">
    <select name="<c:out value='${status.expression}'/>">
     <option value="home" <c:if test="${status.value == 'home'}">
selected="selected" </c:if> >Home</option>
     <option value="office" <c:if test="${status.value ==
'office'}">selected="selected"</c:if> >Office</option>
     <option value="mobile" <c:if test="${status.value ==
'mobile'}">selected="selected"</c:if> >Mobile</option>
    </select>
    </spxform:bind>
    <br/>
    <spxform:bind ref="fullnumber">
      fullnumber: <input type="text" name="<c:out
value='${status.expression}'/>" size="12" value="<c:out
value='${status.value}'/>" />
    </spxform:bind>
    <br/>
    <spxform:bind ref="extension">
      extension: <input type="text" name="<c:out
value='${status.expression}'/>" size="5" value="<c:out
value='${status.value}'/>" />
    </spxform:bind>
  </spxform:repeat>
</spxform:group>
```

Property	**Description**
nodeset	The ref to bind to on the command object in the form of *commandname/xpath-to-node* for the containing group of bind elements.The attribute ref is used instead of path to conform with XForms specification.

id	Optional id of the repeat. Used to conform with the XForms specification.
Property	**Description**
nestedRef	String of the current nested ref of the node being iterated over (for example `instance/person/telephonenumbers/number[1]`) which will be used by the `spxform:bind` tags.
currentIndex	The current index of the node in the homogeneous nodeset.

8.2.4. <spxform:trigger>

The trigger tag is used to allow user-triggered actions to take place on the XML document. Any actions contained by the trigger tag will be run on the serverside before returning the form. This allows for buttons to be added to the page to perform inserts, deletes and setvalues. The trigger tag creates a variable **spxform_element** that should be used to bind to a submit HTML element name such as a `<input type="submit"/>` or `<input type="image"/>`. Any actions inside the trigger will then be performed serverside when the trigger is submitted.

```
<%@ taglib prefix="spxform"
uri="http://www.lakesidetech.co.uk/spxforms/tags" %>
.....
<spxform:group ref="instance/person">
  <spxform:repeat nodeset="telephonenumbers/number">
    type: <spxform:bind ref="@type">
    <select name="<c:out value='${status.expression}'/>">
     <option value="home" <c:if test="${status.value == 'home'}">
selected="selected" </c:if> >Home</option>
     <option value="office" <c:if test="${status.value ==
'office'}">selected="selected"</c:if> >Office</option>
     <option value="mobile" <c:if test="${status.value ==
'mobile'}">selected="selected"</c:if> >Mobile</option>
    </select>
    </spxform:bind>
    <br/>
    <spxform:bind ref="fullnumber">
      fullnumber: <input type="text" name="<c:out
value='${status.expression}'/>" size="12" value="<c:out
value='${status.value}'/>" />
    </spxform:bind>
    <br/>
    <spxform:bind ref="extension">
      extension: <input type="text" name="<c:out
value='${status.expression}'/>" size="5" value="<c:out
value='${status.value}'/>" />
    </spxform:bind>
```

```
  </spxform:repeat>
  <spx:trigger>
   <input type="submit" name="<c:out value='${spxform_element}'/>"
value="Insert New Number After This"/>
   <spx:insert nodeset="/person/telephonenumbers/number"
at="${currentIndex}" position="after"/>
  </spx:trigger>
  <spx:trigger>
     <input type="submit" name="<c:out value='${spxform_element}'/>"
value="Delete This Number"/>
     <spx:delete nodeset="/person/telephonenumbers/number"
at="${currentIndex}"/>
  </spx:trigger>
</spxform:group>
```

Property	Description
id	Optional id of the trigger. The id is used to determine the actions contained within, a unique id will be generated for the page if one is not supplied. Used to conform with the XForms specification.
Property	**Description**
spxform_element	Variable holds the name of input element to be used to cause this trigger to run.

8.2.5. <spxform:delete>

The `<spxform:delete>` tag is an action tag which can be used with triggers to manipulate the document, specifically removing a node with a homogenous nodeset. The **nodeset** attribute defines the homogenous nodeset to delete from while the **at** attribute defines which node to delete. As per the XForms specification these indexes are actually **1** based not 0 based.

```
<spx:trigger>
     <input type="submit" name="<c:out value='${spxform_element}'/>"
value="Delete This Number"/>
     <spx:delete nodeset="/person/telephonenumbers/number"
at="${currentIndex}"/>
</spx:trigger>
```

Property	Description
id	Optional id of the action. The id is used to run the actions contained within, a unique id will be generated for the page if one is not supplied. Used to conform with the XForms specification.
nodeset	A full xpath evaluation to the nodeset to delete a node from (without any commandName).

at	The 1 based index at which to delete the node.
Property	**Description**
spxform_element	Variable holds the name of input element to be used to cause this action to run. Should not be needed in normal behaviour as this is outputed automatically as a hidden element.

8.2.6. <spxform:insert>

The `<spxform:insert>` tag is an action tag which can be used with triggers to manipulate the document, specifically inserting a node within a homogenous nodeset. The **nodeset** attribute defines the homogenous nodeset to insert into, the **at** attribute defines where the node should be inserted and the **position** indicates if the node should be inserted before or after the node indicated by the **at** attribute . As per the XForms specification these indexes are actually 1 based not 0 based.

```
<spx:trigger>
   <input type="submit" name="<c:out value='${spxform_element}'/>"
value="Insert New Number After This"/>
   <spx:insert nodeset="/person/telephonenumbers/number"
at="${currentIndex}" position="after"/>
</spx:trigger>
```

Property	**Description**
id	Optional id of the action. The id is used to run the actions contained within, a unique id will be generated for the page if one is not supplied. Used to conform with the XForms specification.
nodeset	A full xpath evaluation to the nodeset to insert a node into (without any commandName).
at	The 1 based index at which to insert the node.
position	Indicates if the node should be inserted before or after the node indicated by the at attribute. Valid contents are *before* or *after*
Property	**Description**
spxform_element	Variable holds the name of input element to be used to cause this action to run. Should not be needed in normal behaviour as this is outputed automatically as a hidden element.

8.2.7. <spxform:setvalue>

The <spxform:setvalue> tag is an action tag which can be used with triggers to manipulate the document, specifically setting the value of a xpath reference within a document to a certain value. This can be used with a insert action for instance to set some default values of newly created nodes.

```
<spx:trigger>
   <input type="submit" name="<c:out value='${spxform_element}'/>"
value="Insert New Number After This"/>
   <spx:insert nodeset="/person/telephonenumbers/number"
at="${currentIndex}" position="after"/>
   <spx:setvalue
ref="/person/telephonenumbers/number[${currentIndex+1}]/@type"
value="mobile"/>
</spx:trigger>
```

Property	Description
id	Optional id of the action. The id is used to run the actions contained within, a unique id will be generated for the page if one is not supplied. Used to conform with the XForms specification.
ref	A full xpath evaluation to the node for which to set a value.
value	The value to set the node to.
Property	**Description**
spxform_element	Variable holds the name of input element to be used to cause this action to run. Should not be needed in normal behaviour as this is outputed automatically as a hidden element.

8.2.8. <spxform:submit>

The <spxform:submit> tag is used for indicating a user-triggered XML document submission. While the SPXFormControllers use regular form submissions for running actions the <spxform:submit> indicates the form should be processed. The <spxform:submit> tag also allows multiple submission methods to be made available. The submission element in the tag indicates the name of the method to run in the SPXFormController.

```
protected ModelAndView methodName(
            HttpServletRequest request, HttpServletResponse response,
Object command, BindException errors)
            throws Exception {
```

The submi tag creates a variable **spxform_element** that should be used to bind to a submit HTML element name such as a <input type="submit"/> or <input

type="image"/>. Any actions inside the submit will then be performed serverside when the submission is submitted.

```
<spx:submit submission="saveOrUpdateXML">
   <input type="submit" alignment="center" name="<c:out
value='${spxform_element}'/>" value="submit">
</spx:submit>
<spx:submit submission="cancel">
   <input type="submit" alignment="center" name="<c:out
value='${spxform_element}'/>" value="cancel">
</spx:submit>
```

For example pressing the saveOrUpdateXML submission will run the method:

```
protected ModelAndView saveOrUpdateXML(
            HttpServletRequest request, HttpServletResponse response,
Object command, BindException errors)
            throws Exception {
```

If the method does not exist the form controller will try to run processXFormSubmission instead.

Property	**Description**
id	Optional id of the action. The id is used to run the actions contained within, a unique id will be generated for the page if one is not supplied. Used to conform with the XForms specification.
submission	The name of the method to run in the form controller.
Property	**Description**
spxform_element	Variable holds the name of input element to be used to cause this action to run. Should not be needed in normal behaviour as this is outputed automatically as a hidden element.

9. Validation

The SPX framework includes validating methods for validating data in XML documents. This uses the Spring Validator interface which can be used to validate XML models independently but integrates with the SPXForms MVC layer for validating XML form input. Currently the following validators are supported.

- SchematronValidator

9.1. Schematron Validation

Schematron is a XML validation language using patterns in trees. The schematron homepage can be found at http://xml.ascc.net/resource/schematron/schematron.html. The Schematron allows you to develop and mix two kinds of schemas:

- Report elements allow you to diagnose which variant of a language you are dealing with.
- Assert elements allow you to confirm that the document conforms to a particular schema.

The Schematron is based on a simple action:

- First, find a context nodes in the document (typically an element) based on XPath path criteria;
- Then, check to see if some other XPath expressions are true, for each of those nodes.

9.1.1. SchematronValidator

The class `uk.co.lakesidetech.spxforms.validation.schematron.SchematronValidator` is an implementation of the Spring Validator interface. The schematron validator can be used by specifiying the resource property schematronResource to point to the XML definition file of the schematron. Within the schematron file are a serious of patterns, rules with asserts and reports.

Simply the validator will check all the rules given an xpath context in the document. Within the rules are either **asserts** or **reports** which have xpath tests. If the test evaluates to false on a assert an error is added with the assert message (within the assert element), if a test evaluates to true on a report then an error is added with the report message (within the report element). The test can also just attempt to evaluate a value, if the value is not found it evaluates to false. The message can either be the key to text in a messages file or just the actual default text message to use. The field name used for the error object is the context of the rule.

Property	Description
schematronResource	The path to the XML definition of the schematron to use in the validation. This uses the Spring Resouce abstraction so can be used in a web application or independently.

```
<bean id="itemXmlValidator"
class="uk.co.lakesidetech.spxforms.validation.schematron.SchematronValidator">
  <property
name="schematronResource"><value>/resources/schematron/itemschematron.xml</value></prop
</bean>

<bean id="createEditItemXFormController"
class="uk.co.lakesidetech.springxmldb.examples.tilesnews.web.spring.CreateEditItemXForm
```

```
     <property name="validators">
            <list>
                <ref local="itemXmlValidator"/>
            </list>
        </property>
      <property name="sessionForm"><value>true</value></property>
   <property name="xmldbFacade"><ref bean="xmldbFacade"/></property>
   <property name="commandName"><value>item</value></property>
   <property
name="collection"><value>/db/tilesnews/items</value></property>
   <property name="formView"><value>createEditItemXForm</value></property>
   <property
name="successView"><value>createEditItemConfirmation</value></property>
</bean>
```

9.1.1.1. Example Schematron

The following is an example schematron:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron" >
<title>tilesnews item schematron</title>
<pattern name="Test Context">
  <rule context="/">
            <assert test="item">.schematron.item.required</assert>
         </rule>
  <rule context="/item">
            <assert
test="metadata">.schematron.metadata.required</assert>
        </rule>
        <rule context="/item">
            <assert test="content">.schematron.content.required</assert>
        </rule>
 </pattern>
     <pattern name="Test Metadata">
     <rule context="/item/metadata">
            <assert test="title">.schematron.title.required</assert>
         </rule>
   <rule context="/item/metadata">
            <assert test="type">.schematron.type.required</assert>
         </rule>
         <rule context="/item/metadata/title">
            <assert test="string-length(.) &gt;
0">.schematron.title.required</assert>
         </rule>
    <rule context="/item/metadata/type">
            <assert test="string-length(.) &gt;
0">.schematron.type.required</assert>
         </rule>
  <rule context="/item/metadata/subjects">
            <assert test="count(subject) &lt;
11">.schematron.subjects.max</assert>
         </rule>
     </pattern>
```

```
<pattern name="Test Content">
     <rule context="/item/content">
               <assert
test="headline">.schematron.headline.required</assert>
          </rule>
          <rule context="/item/content/headline">
               <assert test="string-length(.) &gt;
0">.schematron.headline.required</assert>
          </rule>
     </pattern>
</schema>
```

When used with an SPXForm and a <spx:bind> tag this can then display errors as normal for instance:

```
<spx:bind ref="/item/metadata/title">
  <input type="text" name="<c:out value='${status.expression}'/>" size="75"
value="<c:out value='${status.value}'/>" /> *
  <c:forEach items="${status.errorMessages}" var="error"><font
color="red"><c:out value="${error}"/></font></c:forEach>
</spx:bind>
```

10. Example Applications

There are two example applications which contain all of the features listed above. A Spring controller based XML database manager application and a Spring and Tiles based news site.

10.1. Spring XML Database Manager

The Spring XML Database Manager is a simple example of functions for manipulating XML databases. It has the following features.

- Browse Collections, showing child collections, resources and resource dates.
- View Resources (binary and XML).
- Remove Resources.
- Add Collections.
- Remove Collections.
- Upload Resources.
- Query Collections with XPath

The Spring XML Database Manager in CVS is configured to start up a embedded Xindice server and load some example content on startup. The configuration can be changed to use an embedded eXist server by changing web.xml from:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  param-value>/WEB-INF/dataaccessContext-xmldb.xml
```

```
/WEB-INF/applicationContext.xml
/WEB-INF/xindiceStartupContext.xml</param-value>
</context-param>

to

<context-param>
  <param-name>contextConfigLocation</param-name>
  param-value>/WEB-INF/dataaccessContext-xmldb.xml
/WEB-INF/applicationContext.xml
/WEB-INF/existStartupContext.xml</param-value>
</context-param>
```

The Spring XML Database Manager can also be changed to manage a remote database by changing the DataSource properties to point to a remote server.

10.2. SpringXMLDB Tiles News Site

The SpringXMLDB Tiles News Site is an example using the SpringXMLDB Tiles components to deliver a content managed, dynamic news website. The CMS section of the application allows for content items to be listed, created, edited and deleted using a SPXForms page for content creation and deletion. The dynamic delivery site uses SpringXMLDB Tiles components to deliver content to different components of the page configured by the Tiles Definition XML file, the XML content is then XSLT transformed for the page.

The SpringXMLDB Tiles News Site uses the XQuery support of eXist XML database to provide content ordering and better query features. Concequently it will not run on Xindice.