RELAXML User Manual

Steffen Ulsø Knudsen Christian Thomsen

May, 2005

Contents

Chapter 1

User Manual

In this manual, we briefly describe how to use RELAXML. First we consider the XML files used for defining options, concepts and structure definitions. The Schemas for these files are given in Appendix A. Then, we consider how to perform an export, how to perform an import and finally how to perform a deletion. A complete example will not be given here, since the following chapter is devoted to a longer example.

Notice that to run RELAXML, a JRE (version 1.4.2 or higher) and Xerces2 (available from http://xml.apache.org/xerces2-j/download.cgi) will be needed. Further, a JDBC driver used to connect to the DBMS is required.

1.1 Options XML Files

An options XML file is used for specifying user and site specific settings. It thus holds informations about the database to use. An options file is required both when importing and exporting.

An example of an options file is shown below.

```
<?xml version="1.0"?>
<Options
        xmlns="http://relaxml.com/ns-0.2"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://relaxml.com/ns-0.2 OptionsSchema.xsd">

  <Driver>org.postgresql.Driver</Driver>
  <Url>jdbc:postgresql://localhost/username</Url>
  <User>username</User>
  <Password>password</Password>
  <Catalog>null</Catalog>
  <Schema>null</Schema>
  <Separator1>#</Separator1>
  <Separator2>$</Separator2>
  <TouchedTable Create="Yes">RELAXML_TOUCHED</TouchedTable>
  <TouchedPKSeparator>$</TouchedPKSeparator>
  <SortTable>RELAXML_SORT</SortTable>
  <MaxVarcharLength>4000</MaxVarcharLength>
  <TypeMapper>com.relaxml.xml.TypeMapping</TypeMapper>
  <SystemCase>lower</SystemCase>
  <MaxRunsResolveDeadLinks>10</MaxRunsResolveDeadLinks>
  <CommitInterval>0</CommitInterval>
```

```
</Options>
```

Listing 1.1: An options file

Inside the Driver element, the JDBC driver to use is specified. The Url element is used for specifying which database to connect to. The format of this string is dependent of the used DBMS and JDBC driver.

The user name and password to the DBMS are specified inside the User and Password elements. It is also necessary to define which catalog and schema to use. These informations are given inside the Catalog and Schema elements. Notice that the string null is converted to the value `null`.

Inside the Separator1 element, a single character must be given. This character is used between the concept name and table name when a long name in the three-part naming schema is constructed. Similarly, the separator character that is used between the table name and the column name is given inside the Separator2 element. The character given in the Separator1 element must be different from the character given in the Separator2 element.

When importing, RELAXML needs access to the table specified in the element TouchedTable. By default this table is created by RELAXML when required and dropped when it is not needed anymore. However, the user should ensure that the given name is always valid, i.e., that another table with the same name does not exist. Therefore, on a multiuser site every user should have an options file with a unique name given in the TouchedTable element.

To ensure compatibility with DBMSs that do not support temporary tables, RELAXML does not create this table as a temporary table. If the used DBMS supports temporary tables and the user wants to exploit this, it is possible to turn the automatic creation of this table off.

If RELAXML should not create the table the attribute Create="No" must be given with the TouchedTable element. The user will then have to create the table before RELAXML is used. The table should have the columns T_TABLE, T_ATTRIBUTE and T_PRIMKEYVALUE that all should be of type varchar (or similar). It is recommended that the table is created as a temporary table as shown below since RELAXML does not attempt to empty the table when not used anymore.

```
CREATE GLOBAL TEMPORARY TABLE tablename
  (T_TABLE VARCHAR(255),
   T_ATTRIBUTE VARCHAR(255),
   T_PRIMKEYVALUE VARCHAR(255))
ON COMMIT PRESERVE ROWS;
```

Further, if the table is declared as a temporary table, multiple users can use the temporary table at a time but such that each of them only uses his own data.

Notice that the length of the varchars should be long enough to hold any of the used table names, any of the used column names or any of the used (composite) primary keys, respectively. When composite primary keys are present in an import their values will be concatenated and temporarily stored in this table. When the values are concatenated the character specified inside the Touched-

PKSeparator is used. This character should not be present in any of the values for composite primary keys.

Notice that the performance of RELAXML is significantly better when an index is created for the table descirbed above. An index can be created by the command

```
CREATE INDEX indexname
  ON tablename(T_PRIMKEYVALUE, T_ATTRIBUTE, T_TABLE);
```

When performing an export where grouping is used, RELAXML will create a table used for sorting. The name of this table is specified inside the element SortTable. This name should be unique to every running instance of RELAXML. The table will hold columns of type varchar for which the length is set in the MaxVarcharLength element.

The type mapper between values declared in `java.sql.Types` and Schema types is defined in the TypeMapper element. `com.relaxml.xml.TypeMapping` is shipped with RELAXML, but this might be extended by the user to adjust to specific needs. The class has three methods. `getTypeName(int)` which given a value from `java.sql.Types` must return a `String` holding the name to use in the generated Schema, `getTypeMax(int)` and `getTypeMin(int)` that given a type must return a `String` holding the minimum/maximum value allowed for this type. If no such values exist, `null` should be returned.

Inside the element SystemCase lower, upper or mixed can be entered. This decides how identifiers entered by the user are treated. If lower or upper is specified, all identifiers are converted to upper case or lower case, respectively. If mixed is specified, no identifiers will be converted.

Inside the element MaxResolveDeadLinks, a number deciding the maximum attempts of recursive applications of the method to remove dead links can be given. If this number is 0 there is no limit for the number of attempts.

Inside the element CommitInterval it is specified how often RELAXML should commit when importing. When this value is set to 0 RELAXML will only commit when all data in the XML document to import has been imported. If the value is set to some positive value x, RELAXML will commit whenever x data rows have been read from the XML and imported to the database.

Notice that if the used DBMS supports deferrable foreign key constraints these will only be utilized by RELAXML if the commit interval is set to 0.

When the options file has been created it is possible to get various informations on the JDBC driver and test if a connection can be established by using the command

```
java com.relaxml.RelaXML -options:Options.rxo
 -jdbcdriverprofile
```

1.2 Concept XML Files

A concept is also specified in an XML file. Such a file should have the extension ".rxc". Its structure is as shown below.

```
<?xml version="1.0"?>

<Concept
        xmlns="http://relaxml.com/ns-0.2"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://relaxml.com/ns-0.2 ConceptSchema.xsd">

  <Caption>MyConcept</Caption>

  <Parents>
    <Parent>parent1</Parent>
      ...
    <Parent>parentN</Parent>
  </Parents>

  <Data>
    <Relation>
      ...
    </Relation>
  </Data>

  <Columns>
    <Column>column1</Column>
      ...
    <Column Updatable="No">columnN</Column>
  </Columns>

  <RowFilter>SQL condition</RowFilter>

  <Transformations>
    <Transformation>transformation1</Transformation>
      ...
    <Transformation>transformationN</Transformation>
  </Transformations>

  <DeletionOrder>
    <Run>
      <DeleteFrom>relation1</DeleteFrom>
        ...
      <DeleteFrom>relationN</DeleteFrom>
    </Run>
      ...
    <Run>
      ...
    </Run>
  </DeletionOrder>
</Concept>
```

Listing 1.2: A concept file

Inside the Caption element, the name of the root element in the generated XML is specified. After this follows the Parents element in which concepts to inherit from can be given.

Inside the Data element, a Relation element is given. In this Relation element the data to extract is defined. A Relation either consists of a Join element that is given two Relation elements representing relations to join (by means of a join specified by the user) or a BaseRel element that holds the name of a relation in the database. Since a Join element holds two Relation elements it is possible to nest Joins as in the following example.

```
    <Relation>
     <Join Type="theta" Column1="Classes#CLASSES$TID"
```

```
          Operator="EQ" Column2="Classes#TEACHERS$TID">
    <Relation>
     <Join Type="theta" Column1="Classes#STUDENTS$SID"
           Operator="EQ" Column2="Classes#ENROLLMENTS$SID">
       <Relation>
         <Join Type="theta" Column1="Classes#ENROLLMENTS$CID"
               Operator="EQ" Column2="Classes#CLASSES$CID">
           <Relation>
             <BaseRel>ENROLLMENTS</BaseRel>
           </Relation>
           <Relation>
             <BaseRel>CLASSES</BaseRel>
           </Relation>
         </Join>
       </Relation>
       <Relation>
         <BaseRel>STUDENTS</BaseRel>
       </Relation>
     </Join>
    </Relation>
    <Relation>
      <BaseRel>TEACHERS</BaseRel>
    </Relation>
   </Join>
</Relation>
```

Listing 1.3: A Relation element

For further details the reader is referred to Appendix A.

Inside the Columns element, a number of Column elements can be given. Each of these holds the (SQL) name of a column to include from the relation found in the Data element. If the attribute Updatable="No" is given, RELAXML will not change the column from the XML when importing. It is also possible to give the attribute Updatable="Yes". This is the default.

The Columns element is optionally followed by the RowFilter element. In the RowFilter element an SQL condition restricting the set of exported rows can be given.

After the RowFilter element comes the Transformations element in which a number of transformations to apply to the relation found in the Data element can be specified. Note that the order of these transformations reflects the order in which they are applied.

After the Transformations a DeletionOrder element can optionally follow. Inside this element an order for how to delete from used base relations can be given. Multiple Run elements can be given here and each Run element can hold multiple DeleteFrom elements in each of which a name of a base relation must be given. When deleting RELAXML will parse the XML once for each Run element. For each base relation listed in the Run element being considered in the current parse, RELAXML will try to delete the read data from that relation. If no DeletionOrder element is present, RELAXML attempts to find one automatically. Notice that deletion orders are not inherited from parents.

1.3 Structure Definition XML Files

A structure definition file defines how the structure of the generated XML will be. A structure definition should define a position in the XML for each column

in the transformed derived table which the used concept gives rise to. To see which columns are available from a given concept the following command can be used.

```
java com.relaxml.RelaXML -info -options:Options.rxo
 -concept:Concept.rxc
```

An example of a structure definition file is shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Example of an structure definition XML file -->

<StructureDefinition
      xmlns="http://relaxml.com/ns-0.2"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://relaxml.com/ns-0.2 StructureDefinitionSchema.xsd
          ">

  <Encoding>ISO-8859-1</Encoding>
  <Comment>This is a comment.</Comment>
  <Comment>This is another comment</Comment>

  <NullSubstitute>n/a</NullSubstitute>
  <Indention>Yes</Indention>
  <GenerateSchema>Yes</GenerateSchema>
  <SchemaFile>ClassesSchema.xsd</SchemaFile>

  <Schema>
    <Container TagName="CLASS" GroupBy="Yes">
      <Attribute Name="Classes#CLASSES$NAME"/>
      <Attribute Name="Classes#CLASSES$CID" TagName="CLASSID"/>
      <Element Name="Classes#TEACHERS$NAME" TagName="TEACHER" GroupBy="Yes">
        <Attribute Name="Classes#TEACHERS$TID" TagName="TEACHERID"/>
      </Element>
      <Container TagName="STUDENTS" GroupBy="Yes">
        <Element Name="Classes#STUDENTS$NAME" TagName="STUDENT" GroupBy="No">
          <Attribute Name="Classes#STUDENTS$SID" TagName="ID"/>
        </Element>
      </Container>
    </Container>
  </Schema>
</StructureDefinition>
```

Listing 1.4: A structure definition file

In the Encoding element, a string that defines the encoding of the generated XML is given. This encoding must be one supported by the local Java installation. Typical values are ISO-8859-1, UTF-8 and UTF-16.

After the Encoding element, any number of Comment elements can follow. A string inside a Comment element is inserted in the generated XML as a comment (by means of <!- - ... - ->).

In the data to export there might be NULL values. These cannot be written directly to the XML. So in the NullSubstitute element a string is given which is placed in the XML instead of NULL. Notice that when importing, any value identical to this string will be treated as NULL.

In the Indention element either “Yes” or “No” can be specified. If “Yes” is specified, the XML will be pretty-printed such that nested elements have whitespaces in front of them. This will make the XML easier to read for humans, but make the size of the document grow.

The GenerateSchema element decides whether a Schema file should be generated for the XML document to create. The legal values are "Yes" and "No".

In the SchemaFile element, the name of the Schema file which the generated XML document should link is specified.

In the Schema element, the actual structure of the XML to generate is specified. Inside the Schema element, it is possible to specify different kinds of elements to place in the XML. A Container element will create elements that hold other elements and/or attributes. For a Container element a TagName attribute must be specified. This dictates the name that the elements will be given. Further a GroupBy attribute (that may have the value "Yes" or "No") can be given. This dictates if the generated XML should be grouped by this element type. If a GroupBy attribute is not given, it will default to "No".

Elements that hold data and some numbers of attributes (perhaps 0) are declared by the Element tag. An Element tag must be given a Name attribute that decides which column in the transformed derived table the data should be read from. Further it can be given a TagName attribute to decide the name of the element in the XML. If a TagName is not given, a default value will be found from the Name attribute. As for Container elements a GroupBy attribute can also be specified.

Attributes for elements (declared by Element or Container elements) are declared by the Attribute element. As for the Element elements, a Name attribute must be given and a TagName can be given. However, a GroupBy attribute cannot be given since this is decided by means of the element that should hold the attribute being declared.

Notice that the content of the Schema element does not have to describe a tree, but may also describe a forest. The generated XML will under all circumstances be a tree since every element declared in the structure definition will be inserted with the root element as an ancestor.

1.4 Performing an Export

When an options file, a concept file and a structure definition file are present we are ready to perform an export. The export can be started with the following command.

```
java com.relaxml.RelaXML -export -options:Options.rxo
 -concept:Concept.rxc -structure:StructureDefinition.rxs
```

This will print the generated XML to the standard output stream. If the XML instead should be printed to the file data.xml, the argument `-file:data.xml` should also be given. If informations about what is happening should be printed to the standard error stream as the export goes on `-v` could be specified to make RELAXML verbose or `-vv` to make RELAXML very verbose.

By default RELAXML will detect if the data to export contains dead links. If dead links are present, the user will be asked if the export should be performed anyway or cancelled. If the argument `-resolvedeadlinks` is given,

RELAXML will attempt to resolve the dead links. Since this in principle might take very many iterations, the number of iterations is limited by the MaxRunsResolveDeadLinks in the options file. If the argument `-ignoredeadlinks` is given, dead links will neither be detected nor resolved.

1.5 Performing an Import

The insertion of an XML file to the database can be performed by the following command.

```
java com.relaml.RelaXML -insert -options:Options.rxo
 -file:data.xml
```

Here `-insert` could have been replaced by `-update` or `-merge`. Also when importing, it is possible to specify `-v` or `-vv` to make RELAXML verbose or very verbose.

By default the read XML file is validated against its Schema. The validation can, however, be turned off by giving the argument `-novalidation`.

1.6 Performing a Deletion

To delete the data in the file data.xml from the database (if possible) the following command should be given.

```
java com.relaml.RelaXML -delete -options:Options.rxo
 -file:data.xml
```

The given data file should be an XML file in the same format as those generated by RELAXML. Thus, the root element must contain `concept` and `structure` attributes referencing a concept file and a structure definition file, respectively.

Also when deleting, validation of the XML document against its Schema is performed, unless the `-novalidation` parameter is given.

1.7 Using the Transformation Framework

All transformations must in some way extend the Java class `Transformation`. If concepts that use a transformation should allow import operations, the class `TransformationWithInverse` (which itself is an extension of `Transformation`) must be extended. Both of these classes are in the package `com.relaxml.transformations`.

A class extending `Tranformation` should at least define the method `transform(DataRow)`. In addtion to this method, a class extending `TransformationWithInverse` must also define the method `inverseTransform(`

DataRow). These methods will be invoked for each row being exported/imported by RELAXML.

Transformations cannot directly add or delete cells from a given DataRow object. Instead, the constructor of the transformation must register which cells to add and delete. RELAXML will then automatically add the cells to add before transform(DataRow) is invoked and delete the cells to delete after the invocation of transform(DataRow). That is, even though a specific cell will be deleted by a transformation, it is possible to read the value of the cell from the transformation. When inverseTransform(DataRow) is invoked (i.e., when importing) the opposite happens. Thus cells that were registered to be deleted are recreated before inverseTransform is invoked (but without any values) and cells that were registered to be created are deleted after the invocation of inverseTransform.

To register addition and deletion of cells, the methods registerCellAddition(String name, int type) and registerCellDeletion(String name) should be applied. The type parameter given to registerCellAddition should be taken from java.sql.Types where symbolic names are available for SQL types or from com.relaxml.Transformation.JavaToSQLTypes where symbolic names for Java types are available.

A transformation should only register the addition/deletion of a specific cell name once. Therefore, this should be done from the constructor of the transformation. After these registrations, the constructor should invoke the method initialize(). The initialize() method must always be invoked exactly once by a transformation. An example of a typical transformation with an inverse is shown below.

```
import com.relaxml.transformations.TransformationWithInverse;

public class MyTransformation extends TransformationWithInverse {
    public MyTransformation() {
        registerCellAddition("SomeCell", java.sql.Types.INTEGER);
        registerCellDeletion("AntoherCell");

        initialize();
    }

    public void transform(DataRow row) {
        // Perform the transformation here...
    }

    public void inverseTransform(DataRow row) {
        // Perform the inverse transformation here...
    }

}
```

Listing 1.5: The constructor of a transformation

Inside the transform and inverseTransform methods, the real work of the transformation goes on. To get access to a specific cell from a DataRow object, the method getCell(String name) is used. This method will return a Cell object which can be modified. When Cell objects are used, it is important to know the type of the value held by the cell. The type will influence which get and set methods to use. For example, if the Cell object holds a string value, the methods getString() and setString(String) should be used. The type of a cell can be obtained by means of the getType() method which

returns a value from `java.sql.Types`. No matter which type is used, the value can be set to `null` by means of the `setNull()` method.

It might be required for a transformation to change the type held by a cell. To do this, the constructor of the transformation must invoke the method `registerCellConversion(String name, int newType)`. After this, the transformation can still use the get method for the old type, but only the set method for the new type. Assume for example that the cell with name A should have its type converted from integer to float. Then the method `registerCellConversion("A", java.sql.Types.FLOAT)` must be invoked from the constructor of the transformation. Inside the `transform(DataRow)` method it is then possible to use the `getInt()` method (since in the given `DataRow`, the `Cell` with name A holds an integer). However, when a set method is used, the `setFloat(float)` should be used. RELAXML automatically checks that all cell types in a `DataRow` are all right after each invocation of the `transform` and `inverseTransform` methods.

Chapter 2

Example

In this chapter, we demonstrate how RELAXML can be used for generating XML files with data from a relational database. We consider a small database with fictive data for a university. The database has the following schema.

$$\text{Students} = \{\underline{\text{SID}} : \text{Integer}, \text{Name} : \text{Varchar(30)}, \text{Address} : \text{Varchar(30)}\},$$

$$\text{Teachers} = \{\underline{\text{TID}} : \text{Integer}, \text{Name} : \text{Varchar(30)}, \text{Address} : \text{Varchar(30)}\},$$

$$\text{Classes} = \{\underline{\text{CID}} : \text{Integer}, \text{Name} : \text{Varchar(30)}, \text{TID} : \text{Integer}\},$$

$$\text{Enrollments} = \{\underline{\text{SID}} : \text{Integer}, \underline{\text{CID}} : \text{Integer}\},$$

where

Classes(TID) is a foreign key referencing Teachers(TID),

Enrollments(SID) is a foreign key referencing Students(SID) and

Enrollments(CID) is a foreign key referencing Classes(CID).

As seen, the database holds information on names and addresses of students and teachers, names of classes and which teachers are giving them and which classes students are enrolled into. The tables hold the data shown below.

SID	**Name**	**Address**
1	Angelina Prodi	Maribyrnong
2	Arthur Smith	Maribyrnong
3	Peter Chang	Maribyrnong
4	Sandra Nicholson	Collingwood

Students

TID	**Name**	**Address**
1	Donald Johnson	Williamstown
2	John Holmes	Footscray
3	Anand Ghandi	Heidelberg
4	Ann Smith	Carlton

Teachers

CID	Name	TID
1	Math1	1
2	Multimedia	3
3	Networked Multimedia	3
4	Java	2
5	Internet Programming	2
6	Databases	4
7	Simulation	1

Classes

SID	CID
1	4
1	6
1	5
2	4
2	7
3	1
4	4
4	5
4	6
1	1

Enrollments

The concept that we consider extracts informations about each class, the teacher giving it and the students enrolled into it. Thus the attributes shown below are included.

- SID and Name from the Students relation
- TID and Name from the Teachers relation
- CID, Name and TID from the Classes relation
- SID and CID from the Enrollments relation.

To extract meaningful data we use the following join conditions.

- Enrollments.SID = Students.SID
- Enrollments.CID = Classes.CID
- Teachers.TID = Classes.TID.

The (still not transformed) derived table is shown on the next page. Notice that to save space only the last parts of the column names are shown. Because of the join conditions it of course holds that there are three pairs of redundant columns.

Students$SID	Students$Name	Teachers$TID	Classes$TID	Teachers$Name	Classes$CID	Classes$Name	Enrollments$SID	Enrollments$CID
1	Angelina Prodi	1	1	Donald Johnson	1	Math1	1	1
1	Angelina Prodi	2	2	John Holmes	4	Java	1	4
1	Angelina Prodi	2	2	John Holmes	5	Internet Programming	1	5
1	Angelina Prodi	4	4	Ann Smith	6	Databases	1	6
2	Arthur Smith	2	2	John Holmes	4	Java	2	4
2	Arthur Smith	1	1	Donald Johnson	7	Simulation	2	7
3	Peter Chang	1	1	Donald Johnson	1	Math1	3	1
4	Sandra Nicholson	2	2	John Holmes	4	Java	4	4
4	Sandra Nicholson	2	2	John Holmes	5	Internet Programming	4	5
4	Sandra Nicholson	4	4	Ann Smith	6	Databases	4	6

To remove the redundancy, we create the class `ClassesRedundancyRemover` which is an extension of `RedundancyRemover`. All we have to do is to specify the pairs of redundant columns. The first column in each pair will be kept while the second will be deleted when exporting and recreated when importing.

```
import com.relaxml.transformations.RedundancyRemover;

public class ClassesRedundancyRemover extends RedundancyRemover {
    public ClassesRedundancyRemover() {
        registerRedundancy("TEACHERS$TID", "CLASSES$TID");
        registerRedundancy("CLASSES$CID", "ENROLLMENTS$CID");
        registerRedundancy("STUDENTS$SID", "ENROLLMENTS$SID");

        initialize();
    }
}
```

Listing 2.1: The transformation used in the example

The concept file, Classes.rxc, is shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<Concept
       xmlns="http://relaxml.com/ns-0.2"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://relaxml.com/ns-0.2 ConceptSchema.xsd">

  <Caption>Classes</Caption>

  <Parents>
  </Parents>

  <Data>
    <Relation>
     <Join Type="theta" Column1="Classes#CLASSES$TID"
                Operator="EQ" Column2="Classes#TEACHERS$TID">
       <Relation>
        <Join Type="theta" Column1="Classes#STUDENTS$SID"
                   Operator="EQ" Column2="Classes#ENROLLMENTS$SID">
          <Relation>
            <Join Type="theta" Column1="Classes#ENROLLMENTS$CID"
                   Operator="EQ" Column2="Classes#CLASSES$CID">
              <Relation>
                <BaseRel>ENROLLMENTS</BaseRel>
              </Relation>
              <Relation>
                <BaseRel>CLASSES</BaseRel>
              </Relation>
            </Join>
          </Relation>
          <Relation>
            <BaseRel>STUDENTS</BaseRel>
          </Relation>
        </Join>
       </Relation>
       <Relation>
         <BaseRel>TEACHERS</BaseRel>
       </Relation>
      </Join>
    </Relation>
  </Data>

  <Columns>
    <Column>STUDENTS.SID</Column>
    <Column>STUDENTS.NAME</Column>
    <Column>CLASSES.NAME</Column>
    <Column>CLASSES.CID</Column>
    <Column>CLASSES.TID</Column>
    <Column>TEACHERS.TID</Column>
    <Column>TEACHERS.NAME</Column>
    <Column>ENROLLMENTS.CID</Column>
```

```
    <Column>ENROLLMENTS.SID</Column>
  </Columns>

  <Transformations>
    <Transformation>ClassesRedundancyRemover</Transformation>
  </Transformations>
</Concept>
```

Listing 2.2: The concept used in the example

Now we have to give the structure definition for the XML. The structure definition file, Classes.rxs, is shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Example of an structure definition XML file -->

<StructureDefinition
      xmlns="http://relaxml.com/ns-0.2"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://relaxml.com/ns-0.2 StructureDefinitionSchema.xsd
          ">

  <Encoding>ISO-8859-1</Encoding>

  <Comment>This is an example.</Comment>
  <Comment>The shown data is fictive.</Comment>

  <NullSubstitute>n/a</NullSubstitute>
  <Indention>Yes</Indention>
  <GenerateSchema>Yes</GenerateSchema>
  <SchemaFile>ClassesSchema.xsd</SchemaFile>

  <Schema>
    <Container TagName="CLASS" GroupBy="Yes">
      <Attribute Name="Classes#CLASSES$NAME"/>
      <Attribute Name="Classes#CLASSES$CID" TagName="CLASSID"/>
      <Element Name="Classes#TEACHERS$NAME" TagName="TEACHER" GroupBy="Yes">
        <Attribute Name="Classes#TEACHERS$TID" TagName="TEACHERID"/>
      </Element>
      <Container TagName="STUDENTS" GroupBy="Yes">
        <Element Name="Classes#STUDENTS$NAME" TagName="STUDENT">
          <Attribute Name="Classes#STUDENTS$SID" TagName="ID"/>
        </Element>
      </Container>
    </Container>
  </Schema>

</StructureDefinition>
```

Listing 2.3: The structure definition used in the example

Notice that we group by the container CLASS (such that each class is only listed once) and TEACHER (such that the teacher who gives a class is only listed once under the class) and the container STUDENTS (such that under a specific class all its enrolled students are listed inside one STUDENTS element).

We do not list the options file, Options.rxo, since it depends on the used DBMS. To create the XML file Classes.xml we type

```
java com.relaxml.RelaXML -export -concept:Classes.rxc
  -structure:Classes.rxs -options:Options.rxo
  -file:Classes.xml
```

After this, Classes.xml is as shown below.

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!-- XML generated by RelaXML Fri Apr 08 10:09:50 MEST 2005 -->
<!-- This is an example. -->
<!-- The shown data is fictive. -->

<Classes concept='Classes.rxc' structure='Classes.rxs'
     xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
     xmlns='http://relaxml.com/ns-0.2'
     xs:schemaLocation='http://relaxml.com/ns-0.2 ClassesSchema.xsd'>
    <CLASS NAME='Databases' CLASSID='6'>
        <TEACHER TEACHERID='4'>Ann Smith</TEACHER>
        <STUDENTS>
            <STUDENT ID='1'>Angelina Prodi</STUDENT>
            <STUDENT ID='4'>Sandra Nicholson</STUDENT>
        </STUDENTS>
    </CLASS>
    <CLASS NAME='Internet Programming' CLASSID='5'>
        <TEACHER TEACHERID='2'>John Holmes</TEACHER>
        <STUDENTS>
            <STUDENT ID='1'>Angelina Prodi</STUDENT>
            <STUDENT ID='4'>Sandra Nicholson</STUDENT>
        </STUDENTS>
    </CLASS>
    <CLASS NAME='Java' CLASSID='4'>
        <TEACHER TEACHERID='2'>John Holmes</TEACHER>
        <STUDENTS>
            <STUDENT ID='1'>Angelina Prodi</STUDENT>
            <STUDENT ID='2'>Arthur Smith</STUDENT>
            <STUDENT ID='4'>Sandra Nicholson</STUDENT>
        </STUDENTS>
    </CLASS>
    <CLASS NAME='Math1' CLASSID='1'>
        <TEACHER TEACHERID='1'>Donald Johnson</TEACHER>
        <STUDENTS>
            <STUDENT ID='1'>Angelina Prodi</STUDENT>
            <STUDENT ID='3'>Peter Chang</STUDENT>
        </STUDENTS>
    </CLASS>
    <CLASS NAME='Simulation' CLASSID='7'>
        <TEACHER TEACHERID='1'>Donald Johnson</TEACHER>
        <STUDENTS>
            <STUDENT ID='2'>Arthur Smith</STUDENT>
        </STUDENTS>
    </CLASS>
</Classes>
```

Listing 2.4: The XML file generated in the example

The generated Schema, ClassesSchema.xsd, is as shown below.

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!-- XML Schema for RelaXML Data File -->
<!-- Schema generated by RelaXML Fri Apr 08 10:09:50 MEST 2005 -->

<xs:schema
  xmlns='http://relaxml.com/ns-0.2'
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  xmlns:rx='http://www.relaxml.com/ns-0.2'
  targetNamespace='http://relaxml.com/ns-0.2'
  elementFormDefault='qualified'>


  <!-- Data type for CLASSES#STUDENTS$SID -->
  <xs:simpleType name='dataType0'>
    <xs:restriction base='xs:integer'>
    </xs:restriction>
  </xs:simpleType>

  <!-- Data type for CLASSES#STUDENTS$NAME -->
  <xs:simpleType name='dataType1'>
```

```
  <xs:union>
    <xs:simpleType>
      <xs:restriction base='xs:string'>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base='xs:string'>
        <xs:enumeration value='n/a'/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>

<!-- Data type for CLASSES#CLASSES$NAME -->
<xs:simpleType name='dataType2'>
  <xs:union>
    <xs:simpleType>
      <xs:restriction base='xs:string'>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base='xs:string'>
        <xs:enumeration value='n/a'/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>

<!-- Data type for CLASSES#CLASSES$CID -->
<xs:simpleType name='dataType3'>
  <xs:restriction base='xs:integer'>
  </xs:restriction>
</xs:simpleType>

<!-- Data type for CLASSES#TEACHERS$NAME -->
<xs:simpleType name='dataType4'>
  <xs:union>
    <xs:simpleType>
      <xs:restriction base='xs:string'>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base='xs:string'>
        <xs:enumeration value='n/a'/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>

<!-- Data type for CLASSES#TEACHERS$TID -->
<xs:simpleType name='dataType5'>
  <xs:restriction base='xs:integer'>
  </xs:restriction>
</xs:simpleType>



 <!-- Element declarations -->
<xs:element name='Classes'>
  <xs:complexType>
    <xs:sequence maxOccurs='unbounded'>
      <xs:sequence maxOccurs='unbounded'>
        <xs:element name='CLASS'>
          <xs:complexType>
            <xs:sequence maxOccurs='unbounded'>
              <xs:element name='TEACHER'>
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base='dataType4'>
                      <xs:attribute name='TEACHERID' type='dataType5'/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>  <!-- TEACHER -->
```

```
                <xs:sequence maxOccurs='unbounded'>
                  <xs:element name='STUDENTS'>
                    <xs:complexType>
                      <xs:sequence maxOccurs='unbounded'>
                        <xs:element name='STUDENT'>
                          <xs:complexType>
                            <xs:simpleContent>
                              <xs:extension base='dataType1'>
                                <xs:attribute name='ID' type='dataType0'/>
                              </xs:extension>
                            </xs:simpleContent>
                          </xs:complexType>
                        </xs:element> <!-- STUDENT -->
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element> <!-- STUDENTS -->
                </xs:sequence>
              </xs:sequence>
              <xs:attribute name='NAME' type='dataType2'/>
              <xs:attribute name='CLASSID' type='dataType3'/>
            </xs:complexType>
          </xs:element> <!-- CLASS -->
        </xs:sequence>
      </xs:sequence>
      <xs:attribute name='concept'>
        <xs:simpleType>
          <xs:restriction base='xs:normalizedString'/>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name='structure'>
        <xs:simpleType>
          <xs:restriction base='xs:normalizedString'/>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 2.5: The Schema file generated in the example

This file can be difficult for humans to read. However, the helping comments shown in the file are automatically added by RELAXML.

Notice that in the generated XML file, Classes.xml, the values for CLASSID, TEACHERID and ID (for a STUDENT) should never be changed since their values originate from primary keys. Therefore a checksum should be used for these values. To keep the example relatively simple we did not use that. But checksums could have been added with the following transformation.

```
import com.relaxml.transformations.*;

public class PKChecksums extends ChecksumTransformation {
    public PKChecksums() {
        registerChecksum("Classes#STUDENTS$SID", "CS_SID");
        registerChecksum("Classes#CLASSES$CID", "CS_CID");
        registerChecksum("Classes#TEACHERS$TID", "CS_TID");
        initialize();
    }
}
```

Listing 2.6: A transformation that adds checksums

The structure definition would then have to be changed to also decide the location of CS_SID, CS_CID and CS_TID.

Appendix A

XML Schemas for Setup Files

A.1 Options XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- RelaXML                                                -->
<!-- Copyright (C) 2004                                     -->
<!-- Steffen Ulsø Knudsen and Christian Thomsen             -->
<!-- {steffen,chr}@relaxml.com                              -->

<!-- Concept XML Schema -->

<xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:rx="http://relaxml.com/ns-0.2"
        targetNamespace="http://relaxml.com/ns-0.2"
        elementFormDefault="qualified">

  <xs:element name="Options">
    <xs:complexType>
      <xs:all>
        <xs:element name="Driver" type="xs:string"/>
        <xs:element name="Url" type="xs:string"/>

        <xs:element name="User" type="xs:string" />
        <xs:element name="Password" type="xs:string" />

        <xs:element name="Catalog" type="xs:string"/>
        <xs:element name="Schema" type="xs:string"/>
        <xs:element name="Separator1" type="rx:SeparatorType"/>
        <xs:element name="Separator2" type="rx:SeparatorType"/>
        <xs:element name="TouchedTable">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="Create" type="rx:YesNoType" default="Yes"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="TouchedPKSeparator" type="rx:SeparatorType"/>
        <xs:element name="SortTable" type="xs:string"/>
        <xs:element name="MaxVarcharLength" type="xs:integer"/>
        <xs:element name="TypeMapper" type="xs:string"/>
        <xs:element name="SystemCase" type="rx:SystemCaseType"/>
        <xs:element name="MaxRunsResolveDeadLinks" type="xs:nonNegativeInteger"/>
        <xs:element name="CommitInterval" type="xs:nonNegativeInteger" minOccurs
             ="0" maxOccurs="1"/>
      </xs:all>
```

```
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="YesNoType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Yes"/>
      <xs:enumeration value="No"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="SeparatorType">
     <xs:restriction base="xs:string">
       <xs:length value="1" fixed="true"/>
     </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="SystemCaseType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="upper"/>
      <xs:enumeration value="lower"/>
      <xs:enumeration value="mixed"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Listing A.1: Options XML Schema

A.2 Concept XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- RelaXML                                                -->
<!-- Copyright (C) 2004                                     -->
<!-- Steffen Ulsø Knudsen and Christian Thomsen             -->
<!-- {steffen,chr}@relaxml.com                              -->

<!-- Concept XML Schema -->

<xs:schema
        xmlns="http://relaxml.com/ns-0.2"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:rx="http://www.relaxml.com/ns-0.2"
        targetNamespace="http://relaxml.com/ns-0.2"
        elementFormDefault="qualified">

  <xs:element name="Concept">
    <xs:complexType>
      <xs:all>
        <xs:element name="Caption" type="xs:string"/>

        <xs:element name="Parents">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Parent" type="xs:string" minOccurs="0" maxOccurs="
                    unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="Data">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Relation" type="RelationType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="Columns">
```

```
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Column" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="Updateable" type="YesNoType" default="
                          Yes"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="RowFilter" type="xs:string" minOccurs="0"/>

        <xs:element name="Transformations">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Transformation" type="xs:string" minOccurs="0"
                  maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="DeletionOrder" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Run" minOccurs="1" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="DeleteFrom" type="xs:string" minOccurs="1"
                        maxOccurs="unbounded"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

      </xs:all>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="RelationType">
    <xs:choice>
      <xs:element name="BaseRel" type="xs:string"/>
      <xs:element name="ConceptRel" type="xs:string"/>
      <xs:element name="Join" type="JoinRelType"/>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="JoinRelType">
    <xs:sequence>
      <xs:element name="Relation" type="RelationType"/>
      <xs:element name="Relation" type="RelationType"/>
    </xs:sequence>

    <xs:attribute name="Type" type="xs:string"/>
    <xs:attribute name="Column1" type="xs:string"/>
    <xs:attribute name="Operator" type="xs:string"/>
    <xs:attribute name="Column2" type="xs:string"/>
  </xs:complexType>

  <xs:simpleType name="YesNoType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Yes"/>
      <xs:enumeration value="No"/>
    </xs:restriction>
  </xs:simpleType>
```

```
</xs:schema>
```

Listing A.2: Concept XML Schema

A.3 Structure Definition XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- RelaXML Structure Definition Schema          -->
<!-- Copyright (C) 2004                           -->
<!-- Steffen Ulsø Knudsen and Christian Thomsen   -->
<!-- {steffen,chr}@relaxml.com                    -->

<!-- Structure Definition XML Schema -->



<xs:schema
        xmlns="http://relaxml.com/ns-0.2"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:rx="http://www.relaxml.com/ns-0.2"
        targetNamespace="http://relaxml.com/ns-0.2"
        elementFormDefault="qualified">

  <xs:element name="StructureDefinition">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Encoding" type="EncodingType" minOccurs="0"
          maxOccurs="1"/>
        <xs:element name="Comment" type="xs:string" minOccurs="0"
          maxOccurs="unbounded"/>
        <xs:element name="NullSubstitute" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
        <xs:element name="Indention" type="YesNoType" minOccurs="0"
          maxOccurs="1"/>
        <xs:element name="GenerateSchema" type="YesNoType" minOccurs="0"
          maxOccurs="1"/>
        <xs:element name="SchemaFile" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
        <xs:element name="Schema" type="SchemaType" minOccurs="1"
          maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="EncodingType">
    <xs:restriction base="xs:string">
      <!-- Enumerations may be added -->
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="SchemaType">
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Container" type="ContainerTagType"/>
        <xs:element name="Element" type="ElementTagType"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ContainerTagType">
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Attribute" type="AttributeTagType"/>
        <xs:element name="Element" type ="ElementTagType"/>
        <xs:element name="Container" type="ContainerTagType"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="TagName" type="xs:string" use="optional"/>
```

```
    <xs:attribute name="GroupBy" type="YesNoType" default="No"/>
  </xs:complexType>

  <xs:complexType name="ElementTagType">
    <xs:sequence>
      <xs:element name="Attribute" type="AttributeTagType" minOccurs="0"
       maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="TagName" type="xs:string" use="optional"/>
    <xs:attribute name="GroupBy" type="YesNoType" default="No"/>
  </xs:complexType>

  <xs:complexType name="AttributeTagType">
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="TagName" type="xs:string" use="optional"/>
  </xs:complexType>

  <xs:simpleType name="YesNoType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Yes"/>
      <xs:enumeration value="No"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Listing A.3: Structure Definition XML Schema

Appendix B

License

or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or

for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS