JUDIE – Functional Specification

Thomas Kestler, OnOffshoreSoft GmbH

V 1.0 – Milestone 1
2008-03-22, last update 2008-06-13

JUDIE – Functional Specification 1
Overview 2
  Existing Tools and Standards 2
Platform 2
Architecture 2
Implementation 3
    CVS 3
    Packages 3
    XML Schema 4
    Functionality 5
    Schema object metadata 7
    Metadata is optional 7
    Partial import 8
    CLOB and BLOB support 8
    Limitations 8
    Logging 8
    Auto Increment 8
    Referential constraints 8
    Coding Conventions 9
    Javadoc 9
    Manual/Tutorial 9
  Testing 9
  The command line tool 9
    Options 10
    File 11
    Examples 11
  JUDIEclipse - The Eclipse Plugin 12
    JUDIEclipse Wizards 14
Features postponed 16
  Pre-/post import SQL 16
  SQL dialects 16
Appendix A – DTD/ XML Schema 17
  DTD 19
Appendix B - Ant Task Description 20
  Required 20
  Sample ant files 20
    Sample 1 – export two tables 20
    Sample 2 – export by custom SQL 21
    Sample 3 – import all from XML file 21
    Sample 4 – import some tables from XML file 21

Overview

JUDIE stands for „Java Universal Database Import Export“ and is an OpenSource tool for exporting some or all data from JDBC databases to XML and vice versa. JUDIE can be used as Eclipse Plugin, command line application or even Web service. It's main purpose is to serve as technology template and for advertising the name of OnOffshoreSoft GmbH. JUDIE will be published under LGPL license and available at sourceforge.net.

This document is the functional specification for JUDIE. Development for JUDIE is planned in several milestones:

- Milestone 1 – basic functionality
- Milestone 2 – advanced functionality

This document is only about Milestone 1 which will result in releases with version number 1.X.

Existing Tools and Standards

The idea is not very new, there are already other tools that provide import/export features. The QuantumDB Eclipse plugin also offers similar features, but it has limitations (and doesn't seem to be maintained).

There are several XML schemas for database data export like SUN JDBC Rowset, Oracle XSU, Apache DB/torque and others. Because of the nature of XML it is easy to convert to other schemas using XSLT, so there is no real disadvantage in using our own schema.

There is another tool, Jailer, on sourceforge.net with a similar approach, but not based on XML. It could be used for inspiration.

Platform

JUDIE requires at least JAVA 5 (because of generics) and JDBC 2.0. Backports to JAVA 1.4 and earlier version could be easily done by removing the generics from code, but this is the task of the user itself or external maintainers.

Architecture

JUDIE is based on JDBC 2.0 API. The goal is that nearly all data can be imported and exported from and to any database which supports JDBC. But, practice shows, that there always slight differences between databases that even JDBC cannot bridge. The goal is not to have the all-capable tool monster that tries to handle every single difference, instead the XML files should give a basis for manual corrections to solve such problems.

JUDIE exports data of a query, one or more tables or all tables into XML which is described by a XML schema later in this document. This XML schema is designed to hold the maximum of information for further processing. If the user wants to have only a subset of this data or convert (say date format to German format) then he can apply XSLT templates to achieve this. Some basic XSLT stylesheets for converting to

ORACLE XSU format or to convert date formats should be implemented with JUDIE as examples. Also meta information (about table structure like columns, datatypes and so on) will be part of the XML and could be stripped using XSLT.

JUDIE has a layered architecture. The main part is the service layer which does the import or export. The applications use this service layer by well defined interface. The applications built upon this service could be servlet, command line tool, Web service and so on. The service interface only knows streams for input file when importing and output when exporting.

The interface to the JUDIE service must be flexible enough to be used in totally different applications: Web service, Eclipse plugin, standlone application, servlet and so on.

Therefore the interface for JUDIEService looks like this (javadoc comments not shown here):

```
public interface JUDIEService {

      void exportData(Connection connection, JUDIEParams params,
OutputStream out, Writer msgWriter)
            throws IllegalArgumentException, JUDIEException;


      void importData(Connection connection, JUDIEParams params,
InputStream in, Writer msgWriter)
            throws IllegalArgumentException, JUDIEException;

}
```

Implementation

JUDIE will be published as OpenSource under LGPL license and be made available also under Eclipse Plugin Central.

CVS

The code will be maintained at sourceforge.net project JUDIE.

Packages

The main JAVA package is `de.onoffshoresoft.judie`

The service components including the basic processing will reside in `de.onoffshoresoft.judie.service`. The code from this package will be packed into a common deliverable `judie.jar`.

The command line application will reside in package `de.onoffshoresoft.judie.cmd` The command line tool will also be packaged into `judie.jar`.

The eclipse plugin will reside in `de.onoffshoresoft.judie.plugin`

The plugin will be packed together with `judie.jar` as an eclipse plugin according to plugin requirements.

Also an additional ant task will be available under `de.onoffshoresoft.judie.ant.JUDIETask`

XML Schema

The XML schema has a root element `dbexport` which holds the table data. Each table is introduced by a `dbtable` element with `name` and `dbschema` attributes. If schema metadata is exported then it is contained in tableinfo element which holds `colinfo` elements with column information like `name`, `type`, `sqltype`, `scale`, `precision` and `nullable`.

After the optional `tableinfo` element the `rowset` element holds the table rows. The `column` elements hold the column data in String representation. And here the trouble begins: CHAR columns can have leading or trailing spaces or even line breaks. These must be preserved. Number data types could be formatted in different formats, similar to date data types. JUDIE uses only one format for numbers and dates: US English. If the user wants to convert date format in exported XML file, he has to use XSLT.

NULL-values are expressed by an attribute `isnull="true"` in the `column` element. This only has to be set when value is null and avoids ambiguity with empty string, which also would result in empty element.

Each table can hold additional information about foreign key constraints in the `<constraint>` Element after the `<colinfo>` Elements (multiple constraint Elements are allowed).

```
<?xml version="1.0" encoding="UTF-8"?>
<dbexport>
 <dbtable name="BID" dbschema="">

  <tableinfo>
   <colinfo name="ID" type="NUMBER" sqltype="2" scale="0" precision="19">
</colinfo>
   <colinfo name="ISBUYNOW" type="CHAR" sqltype="1" scale="0" precision="1">
</colinfo>
   <colinfo name="ITEM" type="NUMBER" sqltype="2" scale="0" precision="19">
</colinfo>
   <colinfo name="AMOUNT" type="NUMBER" sqltype="2" scale="-127" precision="126">
</colinfo>
   <colinfo name="datetime" type="DATE" sqltype="91" scale="0" precision="0">
</colinfo>
   <colinfo name="BIDDER" type="NUMBER" sqltype="2" scale="0" precision="19">
</colinfo>
  </tableinfo>


  <rowset>
   <row>
    <column name="ID">6</column>
    <column name="ISBUYNOW">N</column>
    <column name="ITEM">5</column>
    <column name="AMOUNT">0</column>
    <column name="datetime" format="yyyy-MM-dd">2007-05-17</column>
    <column name="BIDDER">2</column>
   </row>
...
```

The <constraint> Element looks like this:

```
<constraint type="FK" name="FK_1234567" delete="cascade">
      <col order="1">CUST_ID</col>
      <reftable>CUSTOMER</reftable>
      <refcol order="1">ID</refcol>
</constraint>
```

Multiple `col/refcol` Elements possible (composite key), optional `<refcatalog>` and `<refschema>` Elements. See Limitations below. Maybe `<constraint>` Element not even part of Milestone 1.

A full DTD/SML Schema can be found in the Appendix B below.

Validation of XML file will only be done on import (can be deactivated).

Functionality

The functionality is divided into two main parts:

- Export
- Import

When exporting data the user can decide, which table(s) or data should be exported:

- Single or multiple tables
- All tables of a schema/user/catalog
- All tables in database
- Specific SQL statement

In addition to tables, user can also export from views, but views are handled like tables, that is, XML file meta information shows table definition and when importing (with create schema objects option on), a table will be created.

Because the list of tables in a database can be very long and gathering tables take very long, the user must choose

When importing, the user can decide which data should be imported:

- Single or multiple tables from XML file
- All tables of a schema/user/catalog in XML file
- All tables in XML file

and if schema objects should be created:

1. Do not create table, error if target is missing in database
2. Create table, error if target exists in database
3. Create table, ignore existing target table (and just import) – the default.
4. Force re-creation (drop existing table, create new). This is a dangerous operation and it is recommended that calling application prompts an extra confirmation dialog “You have chosen to force table creation. This results in dropping existing tables in database and loss of table data” before doing the import

Processing will stop when an error occurs, in case of export the generated file may by incomplete and not well formed. The user is responsible to clean it up. Actual transaction will be rolled back, see commit mode below.

When importing the user can choose the commit mode:

- Autocommit mode (default, slow)
- Commit after N rows inserted (or table finished)
- Whole import in one transaction (by setting N to very high number)

Also for import and export the user can control the number of rows to be exported from each table (while this often is not very useful when there are foreign key constraints, but maybe useful for set-up of test database without constraints).

Data type mapping

Different databases have different data types. Even with JDBC standard there are ambiguities left. For example, MySQL has three types matching the JDBC SQL type 8: `FLOAT`, `DOUBLE` and `DOUBLE_PRECISION`. So the JDBC SQL type to database type mapping is not unique. JUDIE uses the following rule:

If both sql type and data type name in exported SQL match one of the rows returned from `DatabaseMetaData.getTypeInfo()` then the column gets created with the data type name. Otherwise read the mapping (sql type to database type name) from

properties file `datatypes.properties`. This file contains a mapping for different databases. If the user wants to import into a database not listed in this property file, he can simply add one section (and contribute to community). If no mapping can be found in properties file, JUDIE uses the first matching data type name from `getTypeInfo()`.

Connecting to database

The user must connect to database before any export or import can be done. The database connection must be set-up (and opened) by the application calling the JUDIE service. So the plugin could use for example the DataTools framework to connect to a database already defined in Eclipse. The command line tool would open connection direct via JDBC DriverManager. The JUDIE service is free to change connection settings like commit mode, so it is highly recommended that the application closes and frees the connection after executing JUDIE service.

The Eclipse project also has a sub project called Data Tools Project (DTP) for generic handling of database connections to all plugins, but unfortunately it is totally misdesigned and JUDIE will not support DTP in milestone 1.

XML Character encoding

The XML file will be in UTF-8 encoding. This should be sufficient to hold data from international applications including Asian character sets.

All characters with special meanings in XML like “<”, “>” or “&” will be encoded as entities (>: and so on) and decoded on import.

Performance / Data Volume

JUDIE must be able to export/import reasonable amounts of data. Volumes up to one million rows must be handled with good performance. So the XML handling must take care of this and DOM must not be used for generating or parsing XML data. XML parsing and writing must be done in streaming mode (it is not allowed to read in whole XML into memory for parsing or build XML tree in memory for exporting).

Schema object metadata

The export XML file may contain metadata about the table exported describing the columns of this table and their data types. Metadata does not include any information about physical layout like tablespace, clustering and so on. If user wants to control physical storage of tables and indexes, he must first create the tables manually by SQL and then run the import.

If metadata is exported, it is taken from ResultSet.getMetaData() (not by browsing catalog information). So also export from custom SQL or views works the same way.

Metadata is optional

The export of metadata into XML file is optional, so JUDIE does not know whether string data has been exported from VARCHAR or CLOB column. On import JUDIE will analyze the target table and decide how to import the data (using CLOB API or setString()). Same is true for other data types: in source database (numeric) id could

come from VARCHAR column while in target database this column is NUMBER. JUDIE tries to insert the exported data anyway (again using setString()).

Partial import

It is also possible that import data has fewer columns than target table (because source database had fewer columns). No matter at all, the import statement is build from the information in the `rows` element only.

CLOB and BLOB support

CLOB and BLOB data will be supported. The exported XML File will contain BLOB data in Base64 coded form to avoid problems. CLOB data will be exported as string data. Limitations may exist with older database or JDBC driver versions, see below.

Limitations

Not all vendor specific features can be supported by JUDIE. Moreover the common base among all database systems is fairly small. Even things like auto-increment columns (IDENTITY in SQLServer) are no available in other databases. So JUDIE can only support creating theses columns as appropriate (INTEGER) data type.

CLOB/BLOB handling can be difficult with older JDBC drivers of older database versions. In some combinations export/import of CLOB/BLOB may fail.

Supported databases for Milestone 1 are ORACLE (9i to 11g), MySQL (4 and higher) and SQL Server (2000 and higher, jTDS driver). JUDIE should work also on most other databases, but maybe with limitations.

Handling large amount of data could result in larger memory footprint, so user must set JVM options like –Xms and –Xmx to assign more memory (default is 64m, not much).

Structured data types are not supported in Milestone 1.

Logging

JUDIEService will log debug, info and error messages using JAVA logging subsystem. The user can decide which log level by configuration.

Auto Increment

If a table with auto increment column already exists and JUDIE should import data then the database would refuse to insert values into this table. A possible workaround would be to issue a PRE_INSERT SQL before insert (like SET IDENTITY_INSERT [TableName] ON in SQL Server) and a POST-INSERT SQL after the insert. This is not part of Milestone 1.

Referential constraints

Most relational databases are heavily using referential constraints. While exporting table data is not affected by this, the import is. If the database contains the tables CUSTOMER and ORDER and ORDER references CUSTOMER than you cannot

insert ORDER rows before the CUSTOMER rows have been inserted. To solve the problem JUDIE will export the tables in an order that allows import into a existing schema with referential constraints. Therefore JUDIE analyses the referential constraints between the tables selected for export and orders them accordingly: build a tree of relations an order from leaf to root. This information can be queried using `getImportedKeys`() from JDBC API, but it takes some time to loop over all (selected) tables and their columns to find all referential constraints. Therefore the option for ordered export can be disabled (default is on). Views will get exported after any table without any special order.

Schema information also holds information about foreign keys, so the CREATE TABLE statements could contain foreign key constraints. Unfortunately there are differences in SQL dialects which make this difficult to do, so in Milestone 1 there will be no FK constraints in CREATE TABLE statements issued by JUDIE.

Also views can be exported, but view have no referential constraints. JUDIE will not analyze the constraints of the tables behind the view. Views will be exported into `dbtable` elements like tables.

Coding Conventions

Actual official SUN JAVA coding conventions have to be applied. See http://java.sun.com/docs/codeconv/CodeConventions.pdf

Javadoc

All JAVA classes must be well commented with useful Javadoc comments describing input parameters, return values, exceptions and noticeable details. Default behavior must be described in comments. Comments are written in English.

Manual/Tutorial

The goal is to have a high quality manual and/or tutorial at the end of the project. The priority for this is lower than for coding, but at least a basic manual should be created.

Testing

Software testing is mainly done by unit testing. Because of budget constraints no higher level testing will be done automatically. The module test suite should be done using JUnit. Test scripts should pre-create the test databases, fill with test data, run the unit tests and check the data. Test code and scripts will be checked in into version control system and must me maintained all the time, so that they run after checkout without manual interception.

The command line tool

The command line tool enables administrators or developers to easily export/import data. The syntax is similar to other Unix style commands:

```
judie.sh {options} {file}
judie.bat {options} {file}
```

The script just calls the main class `de.onoffshoresoft.judie.cmd.JudieCmd` with the parameters provided to the script.

Options

`-i` **i**mport
`-e` **e**xport

one of both must be supplied

`-d driver` **d**river class like `oracle.jdbc.driver.OracleDriver`, this class or JAR must be included into classpath.

`-c url` JDB**C** Url like `jdbc:oracle:thin:@localhost:1521:XE`

`-u user` **u**ser name

`-p password` **p**assword, if not present, the command reads the password from standard input after prompting (“enter password: “).

`-t` … tables export and import: This can be one **t**able name or a list of tables, separated by comma. Tables names can be fully qualified names like catalog.schema.table or simple names. This just depends on use case. Normally you access tables from your (implicit) user schema by simple names. If you want to dump tables from other schemas or catalogs, you use FQN. If missing on import, then **all** tables in XML file will be imported into target database!

`-s sql` export: **S**QL to extract data.

`-S alias` export: alia**s** table name for the SQL result, used as table name in XML file. If not supplied the default alias is “sql”.

`-o` export: **o**mit metadata. Normally table metadata is exported. This option turns off export of table metadata.

`-r N` export: limit number of rows exported per table, import: commit after N **r**ows. N must be a positive number > 0, but below MAXINT (2^31-1). Per default auto commit is on, which is similar to N=1. The row counter is incremented over all tables, so setting it to a very high number would result in one large transaction for the whole import.

-n import: validato**n**, activates XML validation on import (on export never any validation is done). By default validation is off. XSD file is referrenced via http URL `http://judie.sourceforge.net/judie.xsd`

`-m 1|2|3|4` import: error handling **m**ode, see above. Default is 3.

-x export: do not order tables for export according to referential constraints, default is on.

-z export: **z**ip the output stream, import: unzip import stream. Default: off. If exported to file, the file of the ZIP archive is that supplied by the user. It is recommended to use the extension .xml.zip. The created ZIP archive will contain one entry `judie.xml` with the exported data. On import the program will open the given file and presume it is a ZIP archive and will read the first entry as the XML file with exported data.

-f format export: custom **f**ormat for date (dt), time(ti) or timestamp (ts) followed by colon and a JAVA format pattern. This option can appear multiple times.

Example 1: …`–f "ts:dd.MM.yyyy HH:mm:ss" –f "dt:dd.MM.yy" –f "ti: hh:mm"`
Example 2: …`–f "ts:dd.MM.yy HH:mm" "ti:hh:mm"`

If supplied this format string will be stored in `format` attribute of `column` element in exported XML. On import this attribute will be recognized and used for conversion. This is for users convenience to get exported data in XML in his native format.

-v schema import: **o**verride schema – by default (without this option) schema gets preserved when importing tables, that is, JUDIE tries to import table in same schema for where it was exported. If EMP was exported from SCOTT, JUDIE will import into table SCOTT. Using " …–v TEST …" will override schema information and import into TEST.EMP. Same is true for table creation.

-b SQL import: issue this SQL **b**efore importing each table, Milestone 2.

-a SQL import: issue this SQL **a**fter importing each table, Milestone 2

File

Normally user provides the name of the XML file to export into or to read from. If file is missing, export goes to `System.out` or import reads from `System.in`. This is usual Unix behaviour.

Examples

Export table EMP from Oracle database schema SCOTT to file emp.xml:

```
judie.sh –e –t EMP –d oracle.jdbc.driver.OracleDriver –c
jdbc:oracle:thin:@localhost:1521:XE –u scott –p tiger emp.xml

judie.sh –e –t SCOTT.EMP,SCOTT.DEPT –d oracle.jdbc.driver.OracleDriver –c
jdbc:oracle:thin:@localhost:1521:XE –u scott –p tiger emp_and_dept.xml

judie.sh –e –s "SELECT * FROM DEPT WHERE location = 'NY'" –S DEPT –d
oracle.jdbc.driver.OracleDriver –c jdbc:oracle:thin:@localhost:1521:XE –u
scott –p tiger dept.xml

judie.sh –i –d oracle.jdbc.driver.OracleDriver –c
jdbc:oracle:thin:@localhost:1521:XE –u scott –p tiger emp.xml
```

```
judie.sh -i -t SCOTT.EMP -d oracle.jdbc.driver.OracleDriver -c
jdbc:oracle:thin:@localhost:1521:XE -u scott -p tiger emp_and_dept.xml
```

JUDIEclipse - The Eclipse Plugin

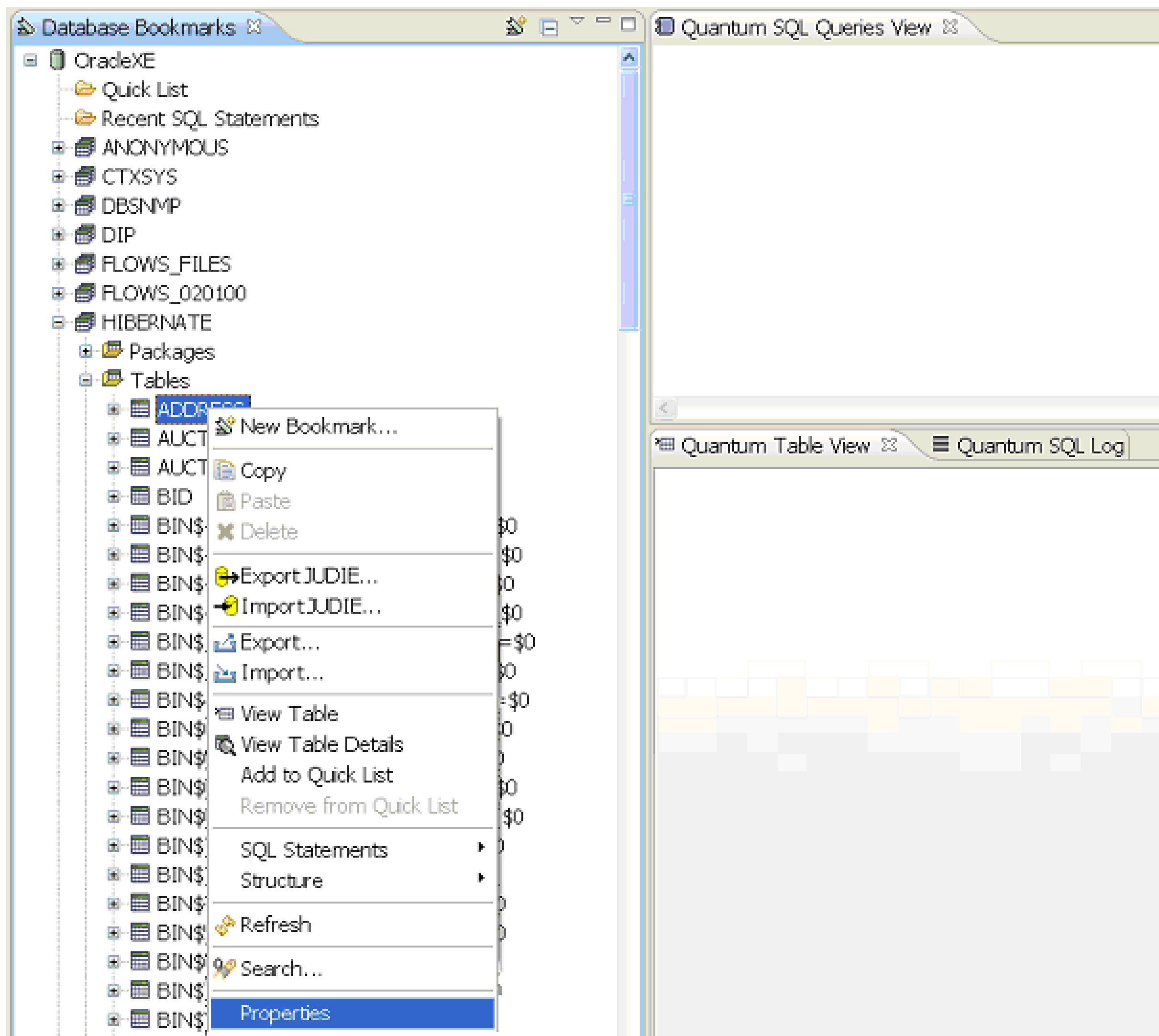
Eclipse itself is very popular and therefor an Eclipse Plugin would increase acceptance of JUDIE. We will name it JUDIEclipse. There are already a number of database tools in Eclipse and our plugin just extends an existing plugin to add the export/import feature. For the first release we plan to extend the QuantumDB plugin. That means that someone who wants to use JUDIEclipse has to install QuantumDB (it will be a prerequiste and installed automatically if user accepts it).

Supported platform: Eclipse 3.3 or higher.

JUDIEclipse consists of three layers:

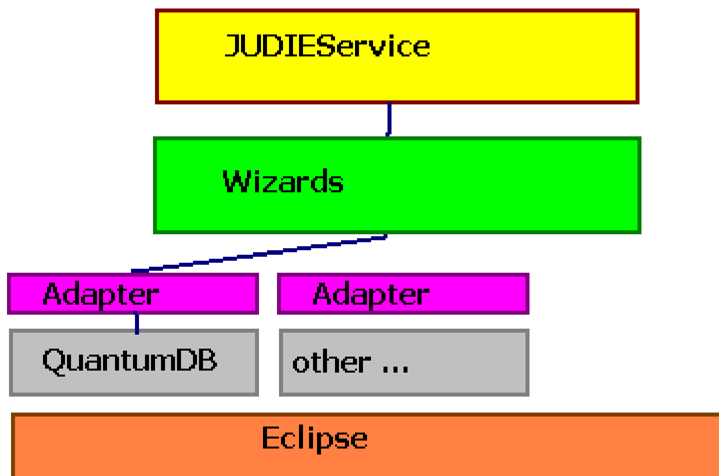
- the adapter connecting to database pulgin like QuantumDB
- export and import wizard
- service layer (JUDIEService as described above)

The integration is done via menu items (preferred via view contribution):



The user selects one or more tables and chooses either Export JUDIE or Import JUDIE menu item. The appropriate wizard dialog will appear and the user can set some options and select the XML file. If the user clicks OK in the wizard the export or import will be done. Errors will be notified by Error Dialog (and log entry in Eclipse log). Tables can be from different schemas, also views can get selected. Export from or import more than one connection is not possible (error message “cannot export from or import to more than one database”).

The following picture shows the architecture of JUDIEclipse:



The picture shows that there could be more than one adapter for several databse plugins (there are many existing database plugins like EclipseSQL, EclipseDB, PowerSQL, JfaceDbc, and so on. http://www.eclipseplugincentral.com/Web_Links-index-req-viewcatlink-cid-3.html lists 38 plugins).

The adapter retrieves the necessary data like table names, schema names, connection and so on an calls the export or import wizard. The wizard displays the UI to the user and finally calls JUDIEService to do the export or import.

JUDIEclipse Wizards

The following screen roughly describe the layout of the two plugin wizards of JUDIEclipse. The layout is not perfect, it's done using Paint, SWT and Eclipse style have to be obeyed.

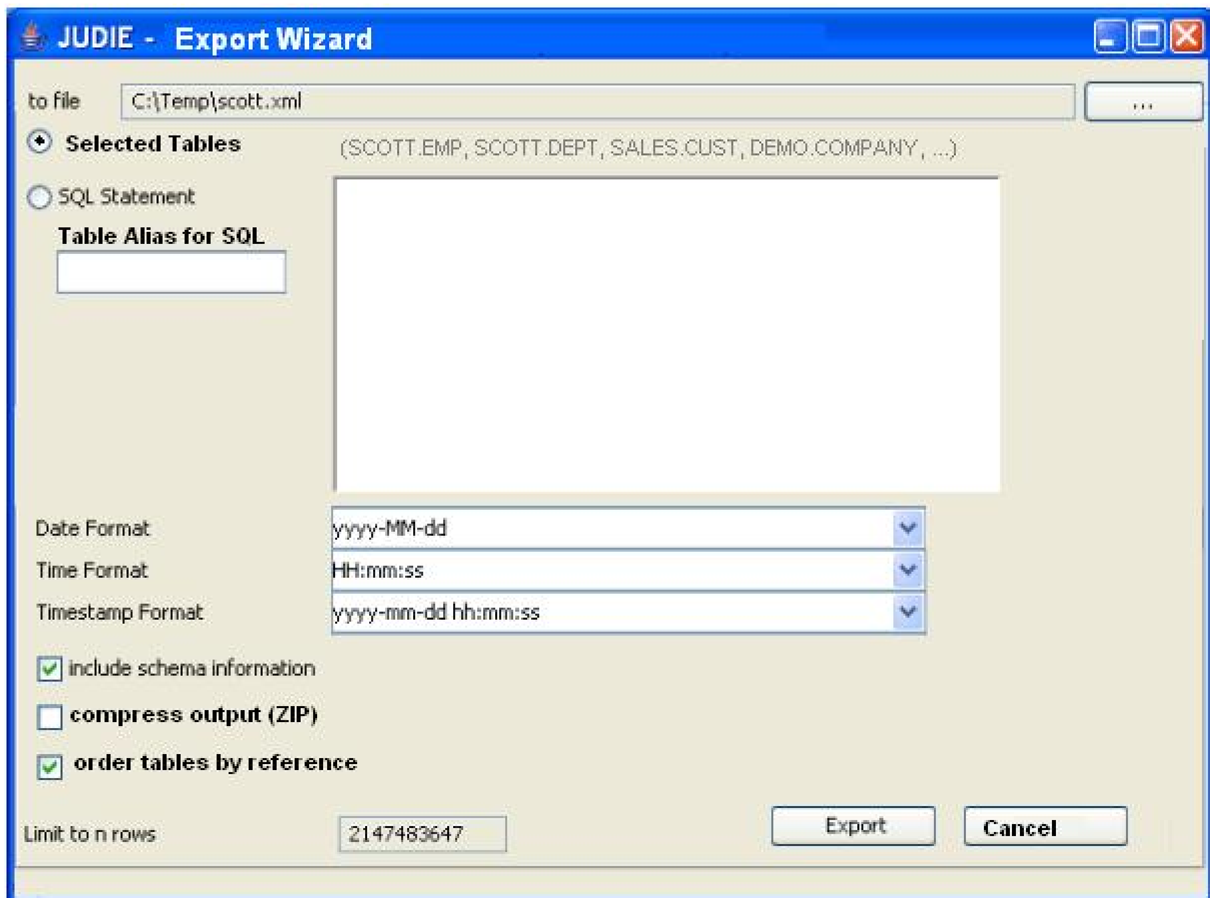
The wizards get called by the adapter which retrieves the selection and provides a data transfer object to the wizards. There is one DTO for export which holds the connection object and a list of selected tables (already expanded by the adapter) and one for import which holds the connection object and either a list of tables (including schema information, already expanded) or the schema override name, if importing into one schema with schema override and a bit to indicate if override GUI element is greyed out. Example: user selected two schema nodes for import, adapter queries all tables in both schemas, builds list and calls import wizard.

Export Wizard

This wizard appears after the user has selected a number of tables and applied JUDIE Export menu item. Tables from multiple schemas can be in this selection (but not from different connections, see above).

The user also can select on or more schema nodes resulting in exporting all the tables within these schemas or he could select the database node exporting all tables in this database.

Selected tables get displayed as label, cut to fitting length, if too long, tool tip should show all select tables including schema.



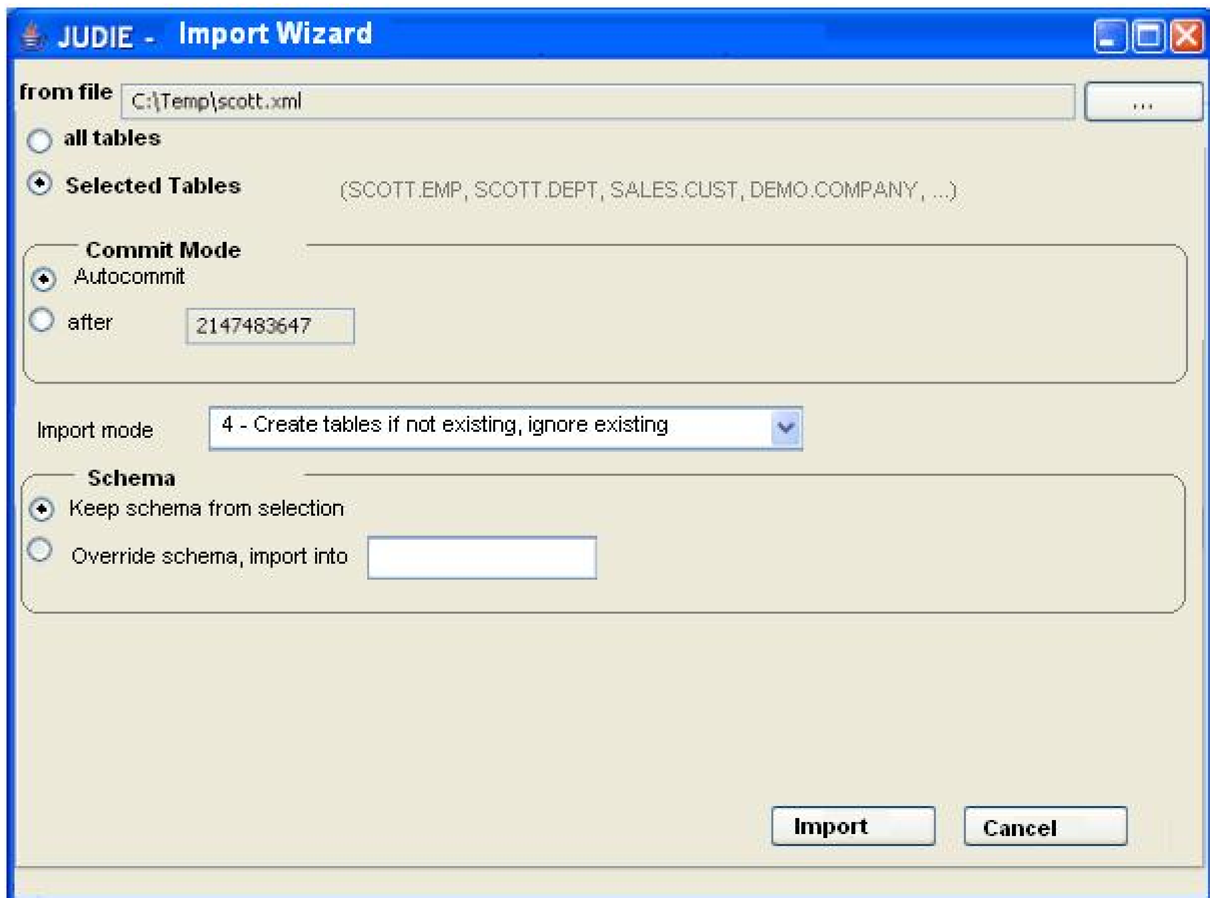
When user clicks export the export starts as worker thread in background. There should be some feedback about the tables exported (schema, table name, number of rows exported).

Import Wizard

This wizard appears after the user has selected a number of tables and applied JUDIE Import menu item.

Depending on kind of selection the wizard uses different settings for schema import:

- if only table nodes are selected the wizard implies “Keep schema”, meaning tables will be imported into the schema of selected table and exported table must exactly match schema plus table name. Of course, user can change radio button to override schema to FOO (but unexpected results when override and SCOTT.EMP and TEST.EMP exist in XML file – both tables imported into FOO.EMP)
- if single schema node is selected, “Keep schema” is selected, user can select override schema, importing all tables from XML into this schema.
- If more than one schema node is selected, “Keep schema” is selected, user cannot choose override (greyed out).
- if database node selected, “Keep schema” is selected, user can select override schema, importing all tables from XML into this schema.
- Selecting mixed type nodes (table, schema database is not supported and will result in error message (“Please only select nodes of same type for import”).



When user clicks import the export starts as worker thread in background. There should be some feedback about the tables imported (schema, table name, number of rows exported, status like table created, was existing, …).

Important: override mode imports all tables in the XML file into target schema! If user wants to import only specific tables into differing schema, he has to export only those or edit the XML file.

Features postponed

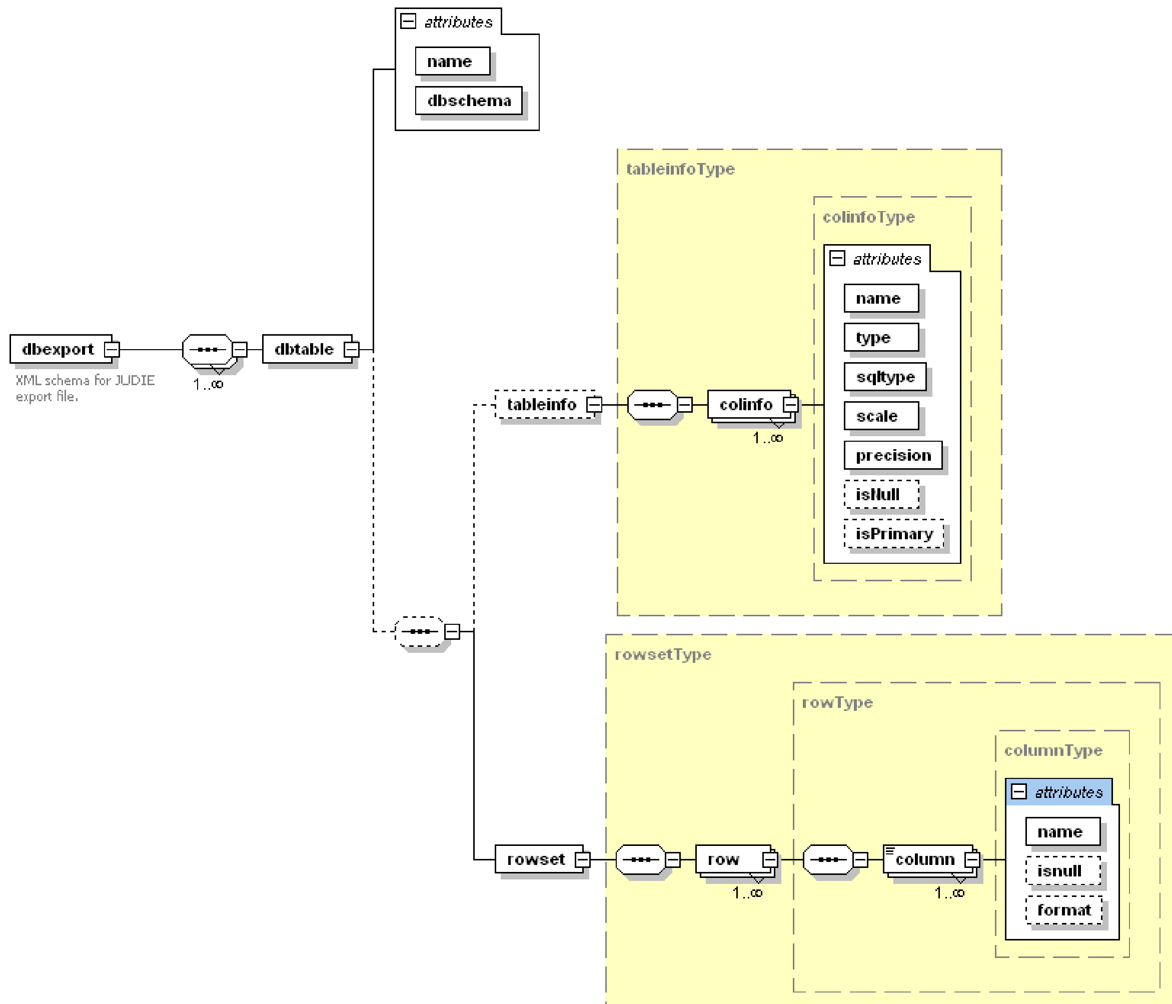
Pre-/post import SQL

Some vendor specific features (like `SET IDENTITY INSERT ON` in SQL Server)could be made available by issuing custom SQL before and after importing a table. This will be part of Milestone 2.

SQL dialects

While ANSI SQL92 should be sufficient for most use cases, the is much vendor specific issues. A concept of SQL dialects like in Hibernate could help to support more SQL features. Not planned yet, not even for Milestone 2.

Appendix A – DTD/ XML Schema




```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
 <xs:element name="dbexport">
  <xs:annotation>
   <xs:documentation>XML schema for JUDIE export file.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence maxOccurs="unbounded">
    <xs:element name="dbtable">
     <!-- dbtable -->
     <xs:complexType>
      <xs:sequence minOccurs="0">
       <xs:element name="tableinfo" type="tableinfoType" minOccurs="0"/>
       <xs:element name="rowset" type="rowsetType"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="dbschema" type="xs:string" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
```

```
  </xs:complexType>
 </xs:element>
 <!-- Definition of Types -->
 <!-- tableinfoType -->
 <xs:complexType name="tableinfoType">
  <xs:sequence>
   <xs:element name="colinfo" type="colinfoType" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <!-- rowsetType -->
 <xs:complexType name="rowsetType">
  <xs:sequence>
   <xs:element name="row" type="rowType" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <!-- colinfoType -->
 <xs:complexType name="colinfoType">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="required"/>
  <xs:attribute name="sqltype" type="xs:integer" use="required"/>
  <xs:attribute name="scale" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="precision" type="xs:nonNegativeInteger"
use="required"/>
  <xs:attribute name="isNull" type="xs:boolean" use="optional"/>
  <xs:attribute name="isPrimary" type="xs:boolean" use="optional"/>
 </xs:complexType>
 <!-- rowType-->
 <xs:complexType name="rowType">
  <xs:sequence>
   <xs:element name="column" type="columnType" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <!-- columnType -->
 <xs:complexType name="columnType">
  <xs:simpleContent>
   <xs:extension base="xs:string">
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="isnull" type="xs:boolean" use="optional"/>
    <xs:attribute name="format" type="xs:string" use="optional"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
 <!-- fkConstraintType, not part of Milestone 1! -->
 <xs:complexType name="fkConstraintType">
  <xs:annotation>
   <xs:documentation>Milestone 1 does not support creation of referential
constraints in DDL nor
does it export foreign key information to XML. So don't worry, if you never
find such elements
in your XML file. Also refcatalog and refschema elements are missing here.
Remember, we are
in Milestone 1.
  </xs:documentation>
  </xs:annotation>
  <xs:sequence>
   <xs:element name="col" type="fkColType" maxOccurs="unbounded"/>
   <xs:element name="reftable" type="xs:string" maxOccurs="unbounded"/>
   <xs:element name="refcol" type="refColType" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <!-- fkColType -->
 <xs:complexType name="fkColType">
  <xs:simpleContent>
```

```
   <xs:extension base="xs:string">
    <xs:attribute name="order" type="xs:integer" use="required"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
 <!-- refColType -->
 <xs:complexType name="refColType">
  <xs:simpleContent>
   <xs:extension base="xs:string">
    <xs:attribute name="order" type="xs:integer" use="required"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:schema>
```

DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XML schema for JUDIE export file.-->
<!ELEMENT dbexport (dbtable)+>
<!ELEMENT dbtable (tableinfo?, rowset)?>
<!ATTLIST dbtable
      name CDATA #REQUIRED
      dbschema CDATA #REQUIRED
>
<!ELEMENT tableinfo (colinfo+)>
<!ELEMENT rowset (row+)>
<!ELEMENT colinfo EMPTY>
<!ATTLIST colinfo
      name CDATA #REQUIRED
      type CDATA #REQUIRED
      sqltype NMTOKEN #REQUIRED
      scale NMTOKEN #REQUIRED
      precision NMTOKEN #REQUIRED
      isNull NMTOKEN #IMPLIED
      isPrimary NMTOKEN #IMPLIED
>
<!ELEMENT row (column+)>
<!ELEMENT column (#PCDATA)>
<!ATTLIST column
      name CDATA #REQUIRED
      isnull NMTOKEN #IMPLIED
      format CDATA #IMPLIED
>
<!ELEMENT col (#PCDATA)>
<!ATTLIST col
      order NMTOKEN #REQUIRED
>
<!ELEMENT reftable (#PCDATA)>
<!ELEMENT refcol (#PCDATA)>
<!ATTLIST refcol
      order NMTOKEN #REQUIRED
>
```

Appendix B - Ant Task Description

Taskname judieexport

Description

Exports data from database to XML file. You can specify the database connection and which tables/views to export. You can also choose to export result of SQL query instead.

Make sure that the JDBC driver jar is included in the classpath of your ant build.

Parameters

Attribute	Description	***Required***
driver	JDBC driver class	yes
file	File for export	yes

Examples

TODO

Sample ant files

The following samples show how to export or import using the JUDIE ant task.

Sample 1 – export two tables

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="AntSample" basedir="." default="sample">

    <target name="sample" description="Sample for Ant Task">
        <taskdef name="judieexport"
classname="de.onoffshoresoft.judie.ant.ExportTask"
classpath="dist/judie.jar"/>

        <judieexport driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:XE"
            user="hibernate" password="hibernate" file="myexport.xml">
         <table name="EMP" schema="SCOTT" />
         <table name="CUST" schema="SCOTT" />
        </judieexport>

    </target>

</project>
```

Sample 2 – export by custom SQL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="AntSample" basedir="." default="sample">

    <target name="sample" description="Sample for Ant Task">
        <taskdef name="judieexport"
classname="de.onoffshoresoft.judie.ant.ExportTask"
classpath="dist/judie.jar"/>

        <judieexport driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:XE"
            user="hibernate" password="hibernate" file="myexport.xml">
         <sql query="SELECT * FROM EMP WHERE DEP = 10"/>
        </judieexport>

    </target>

</project>
```

Sample 3 – import all from XML file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="AntSample" basedir="." default="sample">

    <target name="sample" description="Sample for Ant Task">
        <taskdef name="judieimport"
classname="de.onoffshoresoft.judie.ant.ImportTask"
classpath="dist/judie.jar"/>

        <judieimport driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:XE"
            user="hibernate" password="hibernate" file="myexport.xml">
        </judieimport>

    </target>

</project>
```

Sample 4 – import some tables from XML file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="AntSample" basedir="." default="sample">

    <target name="sample" description="Sample for Ant Task">
        <taskdef name="judieimport"
classname="de.onoffshoresoft.judie.ant.ImportTask"
classpath="dist/judie.jar"/>

        <judieimport driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:XE"
            user="hibernate" password="hibernate" file="myexport.xml">
            <table name="EMP" schema="TEST" />
            <table name="CUST" schema="TEST" />
        </judieimport>

    </target>

</project>
```