Using DODS

Using DODS

Together Teamlösungen EDV-Dienstleistungen GmbH

Elmargasse 2-4
A-1190
Vienna
Austria
+43 (0) 5 04 04 - 122
+43 (0) 5 04 04 - 11 122
<office@together.at>
http://www.together.at/together/index.html

Table of Contents

Chapter 1. Data Object Design Studio

The Data Object Design Studio (DODS), shown in Figure 1, is a tool which, for the given doml file, can generate SQL script files for creating tables (for each table separately and one cumulative file for creating all tables), one file for deleting all tables, and/or java code for data objects described in the given doml file. DODS also has possibility to compile generated java classes and to parse SQL files (to split cumulative SQL into more separated SQL files using SQLSplitter tool).

Figure 1: DODS Generator Wizard

Data objects described in the given DOML file correspond to tables in the database. Each data object has attributes, which describe database columns, and reference attributes, which refer to other data objects. Reference attributes let you create a hierarchy of data objects (for example, many-to-one or many-to-many relationships).

For the given DOML, DODS generates all of the code to implement it. For example:

- SQL code to define the database tables
- Java code to create the corresponding application data objects

For each data object, DODS generates a set of source files. For example, if your DOML file includes the definition of an entity named "thing," then DODS would generate the following:

- A file named thing.sql containing the SQL CREATE TABLE command to construct a table in a relational database.
- Java source file defining a data object representing a row in the table.

  This class provides a "set" and "get" method for each attribute, methods to handle caching, and is a subclass of the Enhydra framework class GenericDO. In this example, the class would be named ThingDO.
- Java source file that defines a query class, which provides SQL query access to the database table.

  The query class returns a collection of ThingDO objects that represent the rows found in the table matching criteria passed from the application..

DODS is one part of Enhydra 5.3 and Enhydra 6.3. If Enhydra 5.3(1) is installed, so is DODS. In this case, DODS home directory is: <enhydra_home>/dods.

Since this version, DODS has become independent from Enhydra, which means that can be used without it. In this case, DODS home directory <dods_home> is the directory in which independent DODS is installed.

Note: DODS have methods: getTableName(), getHandle(), getVersion(), getNewVersion(), getOId(), getVersionColumnName(), getOIdColumnName(), getOriginDatabase(), getPrimaryKeyName() and the same set methods. Due to a possible method name collisions, please do not call your columns of tables by names: tableName, handle, version, newVersion, oId, versionColumnName, oIdColumnName, originDatabase, primaryKeyName. These DODS methods are deprecated and will be removed in the future DODS version. New methods have syntax get_xxx() and set_xxx(…), so, the columns names should not

start with "_" sign.

Also, DODS has method getConfigurationAdministration() that is not deprecated, so, column name configurationAdministration also must not be used.

DODS independence is detailly explained in the chapter DODS independence.

Chapter 2. DODS source building

In Enhydra5.3(1) is included binary version of DODS (without source). So, DODS can not be build out of source in Enhydra, only as independent project.

To build independent DODS, it is necessary to do the following actions:

- Unix stile slashes (/) must always be used instead of DOS stile backslashes (\).
- Start Command Promt and go to <DODS_SOURCE> directory.
- To configure DODS you can call configure batch file with the following options:

```
configure
    [-version version_number] [-release release_tag] [-jdkhome jdk_home_dir]
    [-debug on/off] [-optimize on/off] [-instdir installdir]
```

  where:

  - -version sets version_number. Default: 6.3
  - -release sets release_tag. Default: 1
  - -jdkhome sets java jdk_home_dir. Default: Path to system registred (if any) jdk.
  - -debug compiles source with debug information (on/off). Default: off
  - -optimize sets whether the source should be compiled with optimization or not (on/off). Default: on.
  - -instdir the path to your installation directory (see "Make Options" --> make install)

  Configure without parameters sets configuration parameters to default values.

- To build DODS start Command Promt and go to <DODS_SOURCE> directory.

  DODS building is completely Ant based. You can give one of the following options to the make command:

  - make - builds and configures DODS with javadoc and docbook documentation
  - make buildAll - builds and configures DODS with javadoc and docbook documentation
  - make buildOptimize - builds, optimizes and configures DODS with javadoc and docbook documentation
  - make buildNoDoc - builds and configures DODS without documentation building
  - make install - copies and configures DODS without source compiling
  - make distributions - builds and configures DODS with javadoc and docbook documentation and creates distribution; nsis 2.0b should be included in DODS if doesn't exist (files makensis.exe and makensisw.exe in Dods/Install/Windows/install directory)

  - make optimizeDistributions - builds and configures DODS with javadoc and docbook documentation and creates optimized distribution; nsis 2.0b should be included in DODS if doesn't exist (files makensis.exe and makensisw.exe in Dods/Install/Windows/install directory)
  - make clean - removes the output folder (in order to start a new compilation from the scratch)
  - make help - displays all options

  where <DODS_HOME> is directory in which DODS is built.

- After DODS building, you MUST add <DODS_HOME>\bin directory to the begining of the system PATH.

Chapter 3. DOML file syntax

This section describes the syntax of DOML files, which are used by the Data Object Design Studio (DODS) to generate data access code for Enhydra applications.

Structure

The hierarchy of tags in a DOML file is:

```
<doml>

  <author/>

  <projectname/>

  <database>

    <package>

      <package>

        <package>

          <table>

            <column>

              <type/>

              <initialValue/>

            </column>

            <column>

              <type/>

            </column>

            <index>

              <indexColumn/>

            </index>

          </table>

        </package>
```

```
        <package>
          <table>
            <column>
              <type/>
              </column>
            <column>
              <javadoc/>
              <referenceObject/>
              <type/>
            </column>
            <column>
              <javadoc/>
              <type/>
            </column>
          </table>
        </package>
      </package>
    </package>
  </database>
</doml>
```

Tag reference

This chapter contains an alphabetical reference of all the tags allowed in DOML files. Each entry corresponds to an XML tag, and contains the subsections:

- Content - tags that the tag can contain.
- Attributes - attributes the tag can have.
- Context - tags within which the tag can appear, in other words, the tags that can contain it.

So, for a tag <sampleTag>, whose attributes are attribute1, attribute2, and so on, whose context is <contextTag>, and which can contain contentTag, its general syntax would look like:

```
<contextTag>

  <sampleTag attribute1 attribute2 ...>

    <contentTag/>

  </sampleTag>

</contextTag>
```

<author>

Author of doml project.

Content: None

Attributes: None

Context: <doml>

<column>

<column> describes a column in a database table.

Content: <javadoc>, <referenceObject>, <type>, <initialValue>

Attributes:

1. id - The name of the column in the database table.

2. usedForQuery - Specifies whether the values of the column will be used for queries. The possible values for usedForQuery are: true and false.

3. isConstant - Specifies whether the column contains a constant value. The possible values for isConstant are: true and false.

4. generateSecure - Specifies whether secure methods (methods with check of Users access) should be generated for the column. The possible values for generateSecure are: true and false. The default value is false.

5. generateInsecure - Specifies whether insecure methods (methods without check of Users access) should be generated for the column. The possible values for generateInsecure are: true and false. The default value is true.

Context: <table>

<database>

Contains the package hierarchy, and specifies the database.

Content: <package>

Attributes:

1. database - The database attribute specifies the database vendor. The valid types are as follows:

Oracle - DODS will generate SQL optimized for Oracle databases.

Informix - DODS will generate SQL optimized for Informix databases.

MySQL - DODS will generate SQL optimized for MySQL databases.

MSQL - DODS will generate SQL optimized for MSQL databases.

Sybase - DODS will generate SQL optimized for Sybase databases.

PostgreSQL - DODS will generate SQL optimized for PostgreSQL databases.

DB2 - DODS will generate SQL optimized for DB2 databases.

QED - DODS will generate SQL optimized for QED databases.

InstantDB - DODS will generate SQL optimized for InstantDB databases.

HypersonicSQL- DODS will generate SQL optimized for HypersonicSQL databases.

MckoiSQLConf - DODS will generate SQL optimized for HypersonicSQL databases.

2. dirtyDOs - Optionally, "dirty" methods (methods that can create DOs in memory without transactions) can be marked as "deprecated" or even not be generated at all. If set to "Compatible", "dirty" methods will be generated (as before), if set to "Deprecate", "dirty" methods will be generated as deprecated, and if set to "Omit", "dirty" methods will not be generated at all. If parameter is set in <table> tag, this value overrides program default value and value defined in <database> tag.

3. generateSecure - Specifies whether secure methods (methods with check of Users access) should be generated for the database. The possible values for generateSecure are: true and false. The default value is false.

4. generateInsecure - Specifies whether insecure methods (methods without check of Users access) should be generated for the database. The possible values for generateInsecure are: true and false. The default value is true.

5. templateset - The templateset that will be used for java code generation. The possible values for templateset are: "standard" and "<any user defined template>". Default value is "standard".

6. massDeletes - When turned allow you to build data layer including classes xxxDelete. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

7. massUpdates - When turned allow you to build data layer including classes xxxUpdate. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global

and transactions) valid.

Context: <doml>

<doml>

Root element of DOML files. This tag contains a database hierarchy that contains all the packages.

Content: <author>, <projectname>, <database>

Attributes: None

Context: None

<index>

<index> is used to specify index columns.

Content: <indexColumn>

Attributes:

1. id - The name of the index constraint in the database table.

2. unique - Specifies whether the index constraint is unique. The possible values for unique are: true and false.

3. clustered - Specifies whether the index constraint is clustered. The possible values for clustered are: true and false.

Context: <table>

<indexColumn>

<indexColumn> is used to specify each index column in the constraint index.

Content:None

Attributes:

1. id - The name of the column in the database table.

Context: <index>

<initialValue>

<initialValue>is used to specify a default initial value for the column.

Content: None

Attributes: None

Context: <column>

<javadoc>

The <javadoc> tag contains the text for Javadoc entries for the column.

Content: None

Attributes: None

Context: <column>

<package>

Each package can contain a sub-package or a table structure.

Content: <package>, <table>

Attributes:

1. id - The name of the package. The format for the name includes the parent package's id value. For example, if I had a package myPackage, and a sub-package of it called mySubPackage, mySubPackage's id value would be myPackage.mySubPackage.

Context: <database>

<projectname>

Contains information about project.

Content: None

Attributes: None

Context: <doml>

<referenceObject>

If the column is a reference to another table, <referenceObject> specifies the table.

Content: None

Attributes:

1. Constraint - Specifies whether the specified table row must exist. The possible values for constraint are: true and false.

2. Reference - Specifies the ID of the referenced table.

Context: <column>

<table>

<table> describes a table in a database.

Content:<column>, <index>

Attributes:

1. id - Similar to the id attribute in <package>, <table>'s id contains the value of the table name located in the package. For example, if I had a package myPackage, a subpackage mySubPackage, and a table myTable, the id value is myPackage.mySubPackage.myTable.

2. dbTableName - The actual SQL table name. By default this is the same as the id value, minus the package information. For example, myPackage.mySubPackage.myTable's dbTableName is myTable.

3. isView - This attribute is not currently used by DODS.

4. generateSecure - Specifies whether secure methods (methods with check of Users access) should be generated for the table. The possible values for generateSecure are: true and false. The default value is false.

5. generateInsecure - Specifies whether insecure methods (methods without check of Users access) should be generated for the table. The possible values for generateInsecure are: true and false. The default value is true.

6. massDeletes - When turned allow you to build data layer including classes xxxDelete. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

7. massUpdates - When turned allow you to build data layer including classes xxxUpdate. These class provide you QueryBuilder speed in massive update operations, while maintaining caches (both global and transactions) valid.

8. multidb - Specifies whether code for multi database support should be generated. The possible values for multidb are: true and false. The default value is false.

9. dirtyDOs - This parameter is attribute of <database> and <table> tag in DOML file. Optionally, "dirty" methods (methods that can create DOs in memory without transactions) can be marked as "de-

precated" or even not be generated at all. If set to "Compatible", "dirty" methods will be generated (as before), if set to "Deprecate", "dirty" methods will be generated as deprecated, and if set to "Omit", "dirty" methods will not be generated at all. If parameter is set on <table> tag they override default value and value on <database> tag.

Context: <package>

<type>

<type> dictates the form of the data stored in the column. If no <type> is specified, the column contains all default values.

Content: None

Attributes:

1. Size - Specifies the size of data types that are commonly measured in width, like VARCHAR. Size must be an integer.

2. CanBeNull - Specifies whether the column can contain null values. The possible values for canBeNull are: true and false.

3. dbType - Specifies the internal SQL data type the database will use for this column. The default value of dbType is VARCHAR.

4. JavaType - Specifies the Java data type returned by the DO to the user when querying this attribute of the DO. The default value of javaType is String.

Context: <column>

Sample DOML file

The following snippet shows content of a DOML file, discRack.doml, which creates tables containing data person and its discs. This file can be found in discRack example, in <dods_home>/examples/discrack directory.

```
<?xml version="1.0" encoding="UTF-8"?>

<doml>

  <database database="Standard">

    <package id="discRack">

      <package id="discRack.data">

        <package id="discRack.data.person">

          <table id="discRack.data.person.Person" dbTableName="person">
```

```
            <column id="login" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="password" usedForQuery="true" generateSecure="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="firstname" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="lastname" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
          </table>
        </package>
        <package id="discRack.data.disc">
          <table id="discRack.data.disc.Disc">
            <column id="title" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="artist" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="genre" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="owner" usedForQuery="true">
              <javadoc>/**
 *Attribute describing a link to the owner of this disc.
 */</javadoc>
```

```
              <referenceObject constraint="true" reference="discRack.data.person.Person"/>
              <type dbType="none" javaType="discRack.data.person.PersonDO"/>
            </column>
            <column id="isLiked" usedForQuery="true">
              <javadoc>/**
 * A flag indicating whether the user likes this disc
 */</javadoc>
              <type dbType="BIT" javaType="boolean"/>
            </column>
          </table>
        </package>
      </package>
    </package>
  </database>
</doml>
```

Sample of part of DOML file for using indexes

The following snippet shows part of a DOML file, Computers.doml (creates tables containing data about computers and their parts).

```
<table id="firm.computers.hardware.parts.motherboard"  dbTableName="Motherboard">
              <column id="manufacturrer" generateSecure="false"  generateInsecure="false">
                 <type canBeNull="false" dbType="CHAR" javaType="String" size="40"/>
              </column>
              <column id="type" generateSecure="false">
                 <type dbType="CHAR" javaType="String" size="40"/>
              </column>
              <column id="chipSet" generateSecure="false">
                 <type dbType="CHAR" javaType="String" size="40"/>
```

```
               </column>

               <column id="compName" generateSecure="false">

                  <referenceObject constraint="true"
reference="firm.computers.hardware.Computers" />

                  <type dbType="none" javaType="firm.computers.hardware.ComputersDO"/>

               </column>

               <column id="integratedGraphicAdapter">

                  <type dbType="BIT" javaType="boolean"/>

               </column>

               <column id="integratedModem" >

                  <type dbType="BIT" javaType="boolean"/>

               </column>

               <column id="integratedNetworkKard">

                  <type dbType="BIT" javaType="boolean"/>

               </column>

               <column id="integratedMusicKard">

                  <type dbType="BIT" javaType="boolean"/>

               </column>

<!-- Each computer has only one motherboard. (There are all common computers.) -->

            <index id="computerName" unique="true">

              <indexColumn id="compName"/>

            </index>

</table>

. . . . . .

. . . . . .

<table id="firm.computers.hardware.parts.monitor" dbTableName="Monitor">

               <column id="manufacturrer">

                  <type canBeNull="false" dbType="CHAR" javaType="String" size="40"/>

               </column>
```

```
            <column id="type" >
               <type dbType="CHAR" javaType="String" size="40"/>
            </column>
            <column id="maxResolution">
               <type canBeNull="true" dbType="CHAR" javaType="String" size="20"/>
            </column>
            <column id="refreshFrequency" >
               <type canBeNull="true" dbType="INTEGER" javaType="int"/>
            </column>
            <column id="compName" >
               <referenceObject constraint="true"
reference="firm.computers.hardware.Computers"/>
               <type dbType="none" javaType="firm.computers.hardware.ComputersDO"/>
            </column>
<!-- Each computer has only one monitor. (There are all common computers.) -->
         <index id="computerName" unique="true">
            <indexColumn id="compName"/>
         </index>
</table>
```

The whole DOML file Computers.doml in <dods_home>/examples/doml_examples directory.

Chapter 4. Starting dods generator

There are two different ways to run dods generator. If you want to start generator quickly, you can start wizard by typing

```
dods (for Windows)
```

or

```
./dods (for Linux)
```

without any parameter. Those files are located in

- <enhydra_home>/bin folder, for DODS in Enhydra.
- <dods_home>/bin folder, for independent DODS

- Note:

  <enhydra_home>/bin (in the case DODS is used in Enhydra), or <dods_home>/bin folder (for independent DODS) should be added in the system path. Then, DODS can be started from any directory (by typing dods).

This will be described in the section "Quick Compile" of this chapter.

If you want to start generator without wizard, you need to type (in the command line) dods with additional parameters. You can find details in the section "Custom Compile" of this chapter.

File location

After generating, locations of generated files are:

```
<OUTPUT_DIRECTORY>\SQLcreate.sql

<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>DataStruct.java

<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>DataStruct.java

<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>DOI.java

<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>DO.java

<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>Query.java

<OUTPUT_DIRECTORY>\<PACKAGE_0>\..\<PACKAGE_N>\<TableName>.xml
```

where <OUTPUT_DIRECTORY> is base directory of your project, <PACKAGE_0>\..\<PACKAGE_N> is generated from last package id attribute of DOML file, and

<TableName> is the name of the table from your database. For example, if part of your DOML file looks like this:

```
<package id="discRack">

  <package id="discRack.data">

    <table id="discRack.data.Person" dbTableName="Person">

        ...

    </table>
```

you will get file structure as follows:

```
<OUTPUT_DIRECTORY>\discRack\data\PersonDataStruct.java

<OUTPUT_DIRECTORY>\discRack\data\PersonDOI.java

<OUTPUT_DIRECTORY>\discRack\data\PersonDO.java

<OUTPUT_DIRECTORY>\discRack\data\PersonQuery.java

<OUTPUT_DIRECTORY>\discRack\data\Person.xml
```

There are transient XML files that are generated from DOML file, before Java code is generated. The java code, mentioned before, is actually generated from those transient xml files. If you want, you can change these xml files instead of DOML file and generate Java code directly, without using the DOML file. You can find instructions for this in Advanced Custom Compile section.

If you change the DOML file, all java classes will be generated again, but, if you change transient xml files instead of the DOML file, only changed xml files are generated in java files. Other java files (whose xml files are not changed) are left as they are.

Quick Compile

DODS Generator Wizard is a graphical tool that helps you to easily generate Java and SQL files. It is recommended for the first time users.

When you start dods.bat you will get window like on Figure 2.

Figure 2: DODS Generator Wizard

In the Output directory field you should input directory with full path of output directory that will be used.

DOML file field should be used for entering your DOML file.

Config directory field contains path to custom configuration folder (which contains dodsConf.xml file). It is used to generate java source code and SQL scripts. If the path is set to any other path than default (offered), in the application's configuration file shoud be set parameter :

```
DatabaseManager.ConfigurationDir
```

to new path of the custom configuration folder.

example:

```
DatabaseManager.ConfigurationDir=C:\configurations\dods
```

There are four options on the Generator Wizard:

- Generate SQL:

  This field should be checked if you want to generate: SQL files for each table separately, one cumulutave SQL file for creating all tables (SQLcreate.sql), and one file for deleting those tables (SQLdrop.sql).

  - SQL Splitter:

    It is used for creating separated cumulative SQL files (for creating tables, for adding foreign keys, primary keys and for deleting tables). This option enables creating tables without cross references, and after their creation, adding needed references.

    SQL Splitter copies all SQL commands from all SQL files which are situated in the working directory and all its subdirectories into SQL files.

    Original SQL files are created by DODS.

    All SQL commands are copied into file separateCreate.sql except sql commands which reference to foreign and primary key columns.

    In the separateIntegrity.sql file class puts ALTER TABLE sql commands with adding foreign key references.

    In the separatePrimary.sql file class puts ALTER TABLE sql commands with adding primary keys.

    In the separateDropTable.sql file class puts DROP TABLE sql commands for all tables which were created by create table SQL statements in the first file (separateCreate.sql).

    In the separateIndex.sql file class puts CREATE INDEX sql commands for all tables which were created by create table SQL statements in the first file(separateCreate.sql).

    In the separateDropIntegrity.sql file class puts DROP foreign key sql commands for all tables which were created by create table SQL statements in the first file (separateCreate.sql).

    In the separateDropPrimary.sql file class puts DROP primary sql commands for all tables which created by create table SQL statements in the first file (separateCreate.sql).

    In the separateDropIndex.sql file class puts DROP IDEX sql commands for all tables which created by create table SQL statements in the first file (separateCreate.sql).

    All others Sql commands class puts into separate file.

  Unless Generate SQL field is checked, this field can not be checked. If this option is checked, Generator Wizard doesn't create cumulative SQL files.

- Generate Java:

  This field should be checked if you want to generate Java files (DO, Query, DOI and DataStruct ob-

jects).

- Compile Java:

  It is used for compiling generated java files. Compiled files will be located in folder <output_directory>/classes. Unless Generate Java field is checked, this field can not be checked.

If you do not need both Java and SQL generation, you can choose one of them instead of both.

At least one of the Generate fields must be checked.

There are two combo boxes on the Generator Wizard. Template set combo box contains possible template sets:

- standard:

  If this template set is chosen, DODS generates standard code.

- <user_defined_templates>:

  Users can define their own tempate sets.

Selected template set depends on <template_set> tag in doml file. If this tag is not set, defaut template set is "standard". If this tag is set, the value of this tag will be selected in template set combo box.

DB vendor combo box contains list of database vendors. If one of these vendors is selected, this database will overwrite database declared in doml file. Possible database vendors are:

- MSQL
- Oracle
- Informix
- Sybase
- MySQL
- PostgreSQL
- MckoiSQL
- Standard
- DB2
- QED
- HypersonicSQL
- InstantDB

There is a possibility on the Generator Wizard for generating the following types of documentation:

- HTML:

  If you check this field, doml file will be converted into html file.

- PDF:

  If you check this field, doml file will be converted into pdf file.

- XMI:

  If you check this field, doml file will be converted into xmi file.

- PTL:

  If you check this field, doml file will be converted into ptl (Rational Rose) file.

On the Generator Wizard, there is also a check box:

- overwrite

  for code generating (java and sql), no matter if the code already existed.

Custom Compile

In case you want to generate Java and SQL code manually, type dods in the command line with desired parameters.

Command line:

```
        dods  [-?/help] [-a action] [-t templateset] [-b/-database] [-c confPath]

              [-f/force] [-h/html] [-p/pdf] [-x/xmi] [-r/ptl] domlfile outputdir
```

where:

- outputdir is full path to output directory that will be used.
- domlfile is full path to .doml file for generating code.

options:

- [-? -help] shows help.
- [-a action] - ant task parameter for code generation:

  - dods:build_all - to create all sql files and java classes (default).
  - dods:sql - to create only sql files.
  - dods:java - to create only java files and to compile them.
  - dods:javaNoCompile - to create only java files and not to compile them.
  - dods:noCompile - to create SQL files and java files and not to compile them.
  - dods:build_all_split - to create all sql files and java classes and to compile them. SQL files will be divided into separate files using SQLSplitter .
  - dods:sqlsplit - to create only sql files and separate them in different files using SQLSplitter.
  - dods:noCompileSplit - to create SQL files and separate sql commands using SQLSplitter and java files and not to compile them.
  - dods:generatorOff - to disable generating and compiling of java source code, for generating documentation only (you stil need to set documentation property: html, pdf, ptl, xmi).
- [-t templateset] - template set for generating java and sql code:
  - standard - generate standard java code (default).
  - <user defined> - any user defined template set.
- [-b/-database] - sets database vendor for generating sql.
- [-c confPath] - sets folder with dodsConf.xml file.
- [-f/-force] - with this switch, code will be always generated, without it, only changes will be regenerated.
- [-h/-html] - generates DODS html documentation from .doml file.
- [-p/-pdf] - generates DODS pdf documentation from .doml file.
- [-x/-xmi] - generates DODS xmi documentation from .doml file.
- [-r/-ptl] - generates DODS ptl (Rational Rose) documentation from .doml file.

Advanced Custom Compile

In this section you can find information about advanced settings for generation of Java files.

One XML file is generated for every table from DOML file (situated in table folder with other java and sql files). That XML file is used as a base for generating four Java files.

DTD for that file can be found in <dods_home>/dtd/temporaryXML.dtd file. Some tags could be changed, i.e. <AUTHOR>.

- Important: some tags should not be changed, or otherwise generated code will not be compailable.

Structure

The hierarchy of tags in a XML file is, as follows:

```
<TABLE>
        <PACKAGE/>
        <AUTHOR/>
        <PROJECT_NAME/>
        <TABLE_NAME/>
        <CLASS_NAME/>
        <DB_VENDOR/>
        <TEMPLATE_SET/>
        <COLUMN>
                <REFERENCE_OBJECT>
                        <CONSTRAINT/>
                        <IS_ABSTRACT/>
                        <IS_FOREIGN_KEY/>
                        <TABLE_NAME/>
                        <PACKAGE/>
                </REFERENCE_OBJECT>
                <IS_CONSTANT/>
                <JAVADOC/>
                <DB_TYPE/>
                <JAVA_TYPE/>
                <JAVA_DEFAULT_VALUE/>
                <USED_FOR_QUERY/>
                <CAN_BE_NULL/>
```

```
                <IS_PRIMARY_KEY/>
                <SIZE/>
                <GENERATE_SECURE/>
                <GENERATE_INSECURE/>
        </COLUMN>
        <REFERRER>
                <REFATTR/>
        </REFERRER>
        <INDEX>
                <INDEX_COLUMN/>
        </INDEX>
        <DO_IS_OID_BASED/>
        <IS_ABSTRACT/>
        <DELETE_CASCADES/>
        <DO_IS_MULTIDB_BASED/>
        <IS_ANY_COLUMN_SECURE/>
        <GENERATE_DIRTY/>
        <GENERATE_SECURE/>
        <GENERATE_INSECURE/>
        <MASS_UPDATES/>
        <MASS_DELETES/>
</TABLE>
```

Tag reference

This section contains an alphabetical reference of all the XML tags that DODS can generate using given DOML file. Every tag contains the subsections:

- Content - tags that the tag can contain.
- Attributes - attributes the tag can have.

- Context - tags within which the tag can appear, in other words, the tags that can contain it.

<author>

Author of the Java code (name).

Content: None

Attributes: None

Context: <table>

<can_be_null>

Can column be null. Possible values for can_be_null are: true and false.

Content:None

Attributes: None

Context: <column>

<class_name>

The name of the class which represents table in the database, mostly, it is the TABLE_NAME.

Content: None

Attributes: None

Context: <table>

<column>

Represents one column in the table.

Content: <REFERENCE_OBJECT>, <IS_CONSTANT>, <JAVADOC>, <DB_TYPE>, <JAVA_TYPE>, <JAVA_DEFAULT_VALUE>, <USED_FOR_QUERY>, <CAN_BE_NULL>, <IS_PRIMARY_KEY>, <SIZE>, <GENERATE_SECURE>, <GENERATE_INSECURE>

Attributes:

1. name - Name of the column in the table.

Context:<table>

<constraint>

Specifies whether the specified table row must exist. Possible values for constraint are: true and false.

Content: None

Attributes: None

Context: <reference_object>

<db_type>

Data type from database that represents column.

Content: None

Attributes: None

Context: <column>

<db_vendor>

The database type. Possible values are Standard, InstantDB, Oracle, Informix, MySQL, Sybase, PostgreSQL, MSQL, DB2, QED or HypersonicSQL.

Content: None

Attributes: None

Context: <table>

<delete_cascades>

This is value retrieved from the configuration file and is used for sql code generation.

Content: None

Attributes: None

Context: <table>

<do_is_multidb_based>

Contains information about generation code for multi database support. Possible values: true and false.

Content: None

Attributes: None

Context: <table>

<do_is_oid_based>

Is table based on OID keys. Posible values for do_is_oid_based are: true and false.

Content: None

Attributes: None

Context: <table>

<generate_dirty>

Specifies whether 'dirty' methods are to be generated ("Compatible") - as before, deprecated ("Depricate"), or not generated at all ("Omit").

Content: None

Attributes: None

Context: <table>

<generate_secure>

True if secure methods should be generated, otherwisw false.

Content: None

Attributes: None

Context: <table>

<generate_insecure>

True if insecure methods should be generated, otherwise false.

Content: None

Attributes: None

Context: <table>

<index>

Represents table index.

Content: <index_column>

Attributes:

1. id - Id of index.

2. unique - True if index is unique, otherwise false.

3. clustered - True if index is clustered, otherwise false.

Context: <table>

<index_column>

Identifies index column.

Content: None

Attributes:

1. id - Id of index column, same as name of column.

Context: <index>

<is_abstract>

Is generated class abstract. Possible values for is_abstract are: true and false.

Content: None

Attributes: None

Context: <table>, <reference_object>

<is_any_column_secure>

It is true if for any table column are generated secure methods, otherwise, is false.

Content: None

Attributes: None

Context: <table>

<is_constant>

Does column have constant value, that is, does it represent constant class attribute (not taken from database). Possible values for is_constant are: true and false.

Content: None

Attributes: None

Context: <column>

<is_foreign_key>

Is column used as a foreign key. Currently is_foreign_key tag has only one value: false.

Content: None

Attributes: None

Context: <reference_object>

<is_primary_key>

Is column a primary key. Possible values for is_primary_key are: true and false.

Content: None

Attributes: None

Context: <column>

<javadoc>

Text for Javadoc documentation.

Content: None

Attributes: None

Context: <column>

<java_default_value>

Default value for Java data type.

Content: None

Attributes: None

Context: <column>

<java_type>

Data type from Java that represents column.

Content: None

Attributes: None

Context: <column>

<package>

Package that contains Java files.

Content: None

Attributes: None

Context: <table>, <reference_object>

<project_name>

The project name.

Content: None

Attributes: None

Context: <table>

<refattr>

Tag that is used like attribute for tag <referrer>. It represents column of table that references generated class.

Content: None

Attributes:

1. name - Name of the column that references some DO objects. It is object of generated class, mostly.

2. do_name - Name of the DO object that is referenced by attribute.

3. generateSecure - True if secure methods should be generated, otherwise false.

Context: <referrer>

<reference_object>

If the column is a reference to another table, <reference_object> specifies the table.

Content: <CONSTRAINT>, <IS_ABSTRACT>, <IS_FOREIGN_KEY>, <PACKAGE>, <TABLE_NAME>

Attributes:

1. name - Name of the reference object class.

Context: <column>

<referrer>

Outer table that references generated class.

Content: <refattr>

Attributes:

1. name - Name of the outer table that references generated class.

2. package - Name of the outer table package that references generated class.

3. generateSecure - True if secure methods should be generated, otherwise false.

Context: <table>

<size>

Specifies the size of data types that are commonly measured in width, like VARCHAR. size must be an integer.

Content: None

Attributes: None

Context: <column>

<table>

Root element of XML files. It contains one table from database.

Content: <package>, <author>, <project_name>, <table_name>, <class_name>, <db_vendor>, <template_set>, <do_is_oid_based>, <is_any_column_secure>, <is_abstract>, <delete_cascades>, <column>, <referrer>, <index>, <generate_secure>, <generate_insecure>, <do_is_multidb_based>, <generate_dirty>

Attributes: None

Context: None

<table_name>

The name of the table in the database.

Content: None

Attributes: None

Context: <table>, <reference_object>

<template_set>

Template set that will be used for java code generation. The possible values for template_set are: standard (default) and <any user defined template>.

Content: None

Attributes: None

Context: <table>

<used_for_query>

Should column be used for queries. Possible values for used_for_query are: true and false.

Content: None

Attributes: None

Context: <column>

Sample of part of transient XML file

The following snippet shows part of a transient file, Disc.xml (for table Disc from DiscRack example).

```
<TABLE>(1)

        <PACKAGE>discRack.data.disc</PACKAGE>(2)

        <AUTHOR>NN</AUTHOR>(2)

        <PROJECT_NAME>DiscRack</PROJECT_NAME>(2)

        <TABLE_NAME>Disc</TABLE_NAME>(2)

        <CLASS_NAME>Disc</CLASS_NAME>(2)

        <DB_VENDOR>Standard</DB_VENDOR>(2)

        <TEMPLATE_SET>standard</TEMPLATE_SET>(4)

        <GENERATE_SECURE>false</GENERATE_SECURE>(2)

        <GENERATE_INSECURE>true</GENERATE_INSECURE>(2)

        <DO_IS_OID_BASED>true</DO_IS_OID_BASED>(2)
```

```
        <IS_ABSTRACT>false</IS_ABSTRACT>(2)
        <DELETE_CASCADES>false</DELETE_CASCADES>(2)
        <DO_IS_MULTIDB_BASED>false</DO_IS_MULTIDB_BASED>(2)
        <IS_ANY_COLUMN_SECURE>false</IS_ANY_COLUMN_SECURE>(2)
        <GENERATE_DIRTY>Compatible</GENERATE_DIRTY>(2)
        . . . . . .
        . . . . . .
        <COLUMN name="artist">(2)
                <IS_CONSTANT>false</IS_CONSTANT>(2)
                <DB_TYPE>VARCHAR</DB_TYPE>(2)
                <JAVA_TYPE>String</JAVA_TYPE>(2)
                <USED_FOR_QUERY>true</USED_FOR_QUERY>(2)
                <CAN_BE_NULL>false</CAN_BE_NULL>(2)
                <IS_PRIMARY_KEY>false</IS_PRIMARY_KEY>(2)
                <SIZE>32</SIZE>(2)
                <GENERATE_SECURE>false</GENERATE_SECURE>(2)
                <GENERATE_INSECURE>true</GENERATE_INSECURE>(2)
        </COLUMN>(2)
        . . . . . .
        . . . . . .
        <COLUMN name="owner">(2)
                <REFERENCE_OBJECT name="Person">(3)
                        <CONSTRAINT>true</CONSTRAINT>(3)
                        <IS_ABSTRACT>false</IS_ABSTRACT>
                        <IS_FOREIGN_KEY>false</IS_FOREIGN_KEY>(3)
                        <PACKAGE>discRack.data.person</PACKAGE>(3)
                        <TABLE_NAME>person</TABLE_NAME>(3)
                </REFERENCE_OBJECT>(3)
                <IS_CONSTANT>false</IS_CONSTANT>(2)
```

```
                <JAVADOC>/**
                 *Attribute describing a link to the owner of this disc.
                 */</JAVADOC>(4)
                <DB_TYPE>none</DB_TYPE>(2)
                <JAVA_TYPE>discRack.data.person.PersonDO</JAVA_TYPE>(2)
                <JAVA_DEFAULT_VALUE></JAVA_DEFAULT_VALUE>(4)
                <USED_FOR_QUERY>true</USED_FOR_QUERY>(2)
                <CAN_BE_NULL>false</CAN_BE_NULL>(2)
                <IS_PRIMARY_KEY>false</IS_PRIMARY_KEY>(2)
                <IS_ARRAY>false</IS_ARRAY>(2)
                <GENERATE_SECURE>false</GENERATE_SECURE>(2)
                <GENERATE_INSECURE>true</GENERATE_INSECURE>(2)
        </COLUMN>(2)
</TABLE>(1)
```

Tags (1) must exist and must NOT be changed.

Tags (2) must exist and can be changed.

Tags (3) are not required.

Tags (4) can exist and must NOT be changed.

Chapter 5. Structure of modular DODS

History

Since version 6.0, DODS is modular. The structure of modular DODS 6.0 contained the following jars:

Dods runtime:

- *dbmanager-api.jar* - contains DatabaseManager interfaces and exceptions,
- *dbmanager.jar* - DatabaseManager core - standard Database Manager implementation

Implementations:

- *stdconnection.jar* - standard implementations of ConnectionAllocator and DBConnection,
- *stdtransaction.jar* - standard implementation of DBTransaction
- *stdcaches.jar* - standard cache implementations (DataStruct cache, Query caches),
- *dsconnection.jar* - Implementation of ConnectionAllocator and DBConnection which supports connection to the database using DataSource objects and which supports using DataSource connection pool - used in Enhydra 6.0.

Generator:

- *dods.jar*,
- *ejen.jar*

Structure of current version

The current DODS contains the following jars:

Dods runtime:

- *dbmanager.jar* - DatabaseManager core - standard Database Manager implementation

Implementations:

- *stdconnection.jar* - standard implementations of ConnectionAllocator and DBConnection,
- *stdtransaction.jar* - standard implementation of DBTransaction,
- *stdcaches.jar* - standard cache implementations (DataStruct cache, Query caches),
- *extendedtransaction.jar* - implementation of DBTransaction that helps changing order of table modifications based on objects relations (defined in doml file),
- *dsconnection.jar* - implementation of ConnectionAllocator and DBConnection which supports connection to the database using DataSource objects and which supports using DataSource connection pool,
- *xatransaction.jar* - implementation of ObjectIdAllocator and AbstractDBTransactionFactory for creating transactions that are aware of JTA environment

Generator:

- *dods.jar*,
- *ejen.jar*

How to use different implementations:

ObjectIdAllocator

It manages the allocation of unique object ids.

Parameter: ClassName.

Default value: none (DODS will use StandardObjectIdAllocator).

File: configuration file.

Context: ObjectId.

If this parameter is set to full class name of class that implements ObjectIdAllocator interface, DODS will use this class to create ObjectId Allocator. If parameter is not set, DODS will use default ObjectIdAllocator implementation - StandardObjectIdAllocator.

ConnectionAllocator

Parameter: ConnectionAllocator.

Default value: none (DODS will use StandardConnectionAllocator).

File: configuration file.

Context: DatabaseManager, Database.

If this parameter is set to full class name of class that implements ExtendedConnectionAllocator interface, DODS will use this class to create Connection Allocator. If parameter is not, set DODS will use default ExtendedConnectionAllocator implementation - StandardConnectionAllocator.

Connection

Parameter: ConnectionFactory.

Default value: none (DODS will use StandardDBConnectionFactory).

File: configuration file.

Context: Connection.

If this parameter is set to full class name of class that implements AbstractDBConnectionFactory interface DODS will use this class to create database connection factory. If parameter is not set, DODS will use default AbstractDBConnectionFactory implementation - StandardDBConnectionFactory.

Transaction

Parameter : TransactionFactory.

Default value: none (DODS will use StandardTransactionFactory).

File: configuration file.

Context: DatabaseManager, Database.

If this parameter is set to full class name of class that implements AbstractDBTransactionFactory interface DODS will use this class to create database transaction factory. If parameter is not set, DODS will use default AbstractDBTransactionFactory implementation - StandardDBTransactionFactory.

Cache implementations

Parameter: QueryCacheImplClass.

Default value: none (DODS will use QueryCacheImpl class).

File: configuration file.

Context: DatabaseManager, Database.

If this parameter is set to full class name of class that extends abstract class DataStructCache, DODS will use this class to create data struct cache for xxxDO class. If parameter is not set, DODS will use QueryCacheImpl class as a default.

Chapter 6. DODS independence

Since 5.1 version, DODS is independent from Enhydra. This means that it is possible for user to make any application (it doesn't need to be enhydra application) that can use DODS.

DODS works with DatabaseManagers. DatabaseManager is class that provides facilities for work with databases.

There are two modes of using DODS:

- non-threading

  In non-threading mode, only one DatabaseManager is used for the whole application, no matter the application has one or more Threads.

- threading

  In threading mode, there is one DatabaseManager for every Thread. User needs, for every Thread, to define DatabaseManager. If, for any Thread, the DatabaseManager is not defined, the default DatabaseManager is used.

In the following text, the DODS independence is explaned for non-threading mode.

To make non-enhydra application that can use DODS, the following things must be done:

You need to init DODS from your application source code. There are two ways to do this:

- You can startup DODS with method

  ```
  DODS.startup(String fileName)
  ```

  to do this add code that makes new DatabaseManager and registers it in DODS.

  Example:

  ```
  try {

      . . .

      String fileName = "discRack.conf";

      DODS.startup(fileName);

      . . .

  } catch (Exception e) {

      e.printStackTrace();

  }
  ```

In this case 'fileName' is absolute path to application's configuration file. If configuration file can not be found, on given path, then DODS try to find default 'dods/conf/databaseManager.conf' file in application's classpath (folders and jar-files).

- Or you can startup DODS with method

```
DODS.startup(URL confURL, String confFile)
```

where:

- 'confURL parameter' is additional place (given as object of java.net.URL class) where DODS can search for application's configuration file, if confURL is set to 'null' then DODS use application's classpath to search for configuration file.

- 'confFile' is relative path to application's configuration file, relativ to 'confUrl' parameter (if they are not 'null'),and to application's class path (folders and jar-files).

If DODS can not find 'fileName' file relativ to 'confURL' or application's class path, or 'fileName' parameter is set to null , then DODS try to open default configuration file 'dods/conf/databaseManager.conf' searching relatively to application classpath (folders and jar-files).

Example:

```
try {

    . . .

    String fileName = "discRack.conf";

    DODS.startup(null, fileName);    // or DODS.startup(someUrl, fileName);

    . . .

} catch (Exception e) {

    e.printStackTrace();

}
```

Where "discRack.conf" is an example of application's configuration file. This file is the same as the Database Manager section of Enhydra application's configuration file.

This file can look like this:

```
#---------------------------------------------------------------------------

#                   Database Manager Configuration

#---------------------------------------------------------------------------

#

# The databases that are used by CSAM.  Each of these databases
```

```
# has configuration parameters set under DatabaseManager.DB."databaseName".

#

DatabaseManager.Databases[] = "sid1"

#

# The default database used in this application.

#

DatabaseManager.DefaultDatabase = "sid1"

#

# Turn on/off debugging for transactions or queries. Valid values

# are "true" or "false".

#

DatabaseManager.Debug = "false"

#

# The type of database. Normally this is "Standard".

#

DatabaseManager.DB.sid1.ClassType = "Standard"

# DatabaseManager.DB.sid1.ClassType = "Oracle"


#

# The jdbc driver to use.

#

DatabaseManager.DB.sid1.JdbcDriver = "org.enhydra.instantdb.jdbc.idbDriver"

# DatabaseManager.DB.sid1.JdbcDriver = "oracle.jdbc.driver.OracleDriver"

# DatabaseManager.DB.sid1.JdbcDriver = "sun.jdbc.odbc.JdbcOdbcDriver"


#

# Database url.

#

DatabaseManager.DB.sid1.Connection.Url =
"jdbc:idb:C:/DODS_6.0/output/examples/discrack/output/discRack.prp"
```

```
# DatabaseManager.DB.sid1.Connection.Url = "jdbc:oracle:thin:@MyHost:MyPort:MyDBName"

# DatabaseManager.DB.sid1.Connection.Url = "jdbc:odbc:discRack"


#

# Database user name.  All connection are allocated by this user.

#

DatabaseManager.DB.sid1.Connection.User = "scott"

#DatabaseManager.DB.sid1.Connection.User = "Admin"


# Database user password.

#

DatabaseManager.DB.sid1.Connection.Password = "tiger"

#DatabaseManager.DB.sid1.Connection.Password = ""


#

# The maximum number of connections that a connection

# pool will hold.  If set to zero, then connections

# are allocated indefinitly or until the database

# refuses to allocate any new connections.

#

DatabaseManager.DB.sid1.Connection.MaxPoolSize = 30


#

# Maximum amount of time that a thread will wait for

# a connection from the connection pool before an

# exception is thrown.  This will prevent possible dead

# locks.  The time out is in milliseconds.  If the

# time out is <= zero, the allocation of connections

# will wait indefinitely.

#
```

```
DatabaseManager.DB.sid1.Connection.AllocationTimeout = 10000



#

# Used to log database (SQL) activity.

#

DatabaseManager.DB.sid1.Connection.Logging = false



#

# The number of object identifiers that are allocated

# as a group and held in memory.  These identifiers

# are assigned to new data objects that are inserted

# into the database. Parameter NextWithPrefix is implemented to solve problems with some
databases which has NEXT as a reserver word (like Hsql).
    If this parametre set to true following statament will be used for update object id
table:
      update OID_TABLE set OID_TABLE.next = ? where OID_TABLE.next = ?. Default value for
this parameter is false.

# Parameter OidTableName represents name of table where oid numbers are stored. Default value
for this parameter is objectid.

# Parameter NextColumnName represents name of column in objectid table where oid numbers are
stored. Default value for this parameter is next.

#

DatabaseManager.DB.sid1.ObjectId.CacheSize = 20

DatabaseManager.DB.sid1.ObjectId.MinValue = 1000000

DatabaseManager.DB.sid1.ObjectId.NextWithPrefix = true

DatabaseManager.DB.sid1.ObjectId.OidTableName = objectid

DatabaseManager.DB.sid1.ObjectId.NextColumnName = next



#

# User wildcards

#

DatabaseManager.DB.User.userWildcard = "*"

DatabaseManager.DB.User.userSingleWildcard = "_"

DatabaseManager.DB.User.userSingleWildcardEscape = "$"
```

```
DatabaseManager.DB.User.userWildcardEscape = "$"

#
# Default table configuration
#
# DatabaseManager.defaults.AllReadOnly = false
DatabaseManager.defaults.lazyLoading = false
DatabaseManager.defaults.maxExecuteTime = 200
DatabaseManager.defaults.TransactionCaches = true
DatabaseManager.defaults.TransactionCheck = true
DatabaseManager.defaults.AutoSave = true

#
# Default database configuration
#
DatabaseManager.DB.sid1.TransactionCheck = false
DatabaseManager.DB.sid1.AutoSave = false

#
# Default cache configuration
#
DatabaseManager.defaults.cache.maxCacheSize = 500
DatabaseManager.defaults.cache.maxSimpleCacheSize = 100
DatabaseManager.defaults.cache.maxComplexCacheSize = 10
DatabaseManager.defaults.cache.maxMultiJoinCacheSize = 10
DatabaseManager.defaults.cache.reserveFactor = 0.1

#
# Table person - configuration
#
```

```
DatabaseManager.DB.sid1.person.lazyLoading = true

DatabaseManager.DB.sid1.person.cache.initialCondition = *

DatabaseManager.DB.sid1.person.cache.maxCacheSize = 800

DatabaseManager.DB.sid1.person.cache.reserveFactor = 0.25


#

# Table Disc - configuration

#

DatabaseManager.DB.sid1.Disc.maxExecuteTime = 500

DatabaseManager.DB.sid1.Disc.cache.maxSimpleCacheSize = 300

DatabaseManager.DB.sid1.Disc.cache.maxComplexCacheSize = 150

DatabaseManager.DB.sid1.Disc.cache.maxMultiJoinCacheSize = 100
```

The example of non-enhydra application that can use DODS is DiscRack application, explained in next section.

Examples of non-enhydra applications

Examples of DODS non-enhydra applications are included in DODS installation and they are in DODS, in directory:

```
<DODS_HOME>/examples
```

Process of running non-enhydra application will be presented in this section on the example Disc Rack. This example application is in <DODS_HOME>/examples/discrack directory.

To run example , these steps have to be done in Command Promt:

- first, go to wanted example (directory)

  ```
  cd <DODS_HOME>/examples/discrack
  ```

- second, run ant, by typing:

  ```
  ant
  ```

  ant will build this application in its <output_directory>

- then, go to application's output directory:

  ```
  cd <output_directory>
  ```

- then, example will be run with:

```
run
```

The ant, which is used here, must be DODS's ant.bat, which means that path <DODS_HOME>/bin must be included at the beginning of the system path.

Chapter 7. DODS Ant task

Invokes DODS to generate a set of java classes from a doml file. The files will only be regenerated/compiled if the date on the doml file is newer than at least one of the generated files.

This taskdef extends Ant's <javac> task; refer to documentation for parameters that affect compilation.

Typically made visible to an Ant build file with the following declaration:

```
<taskdef name="dods" classname="org.enhydra.ant.taskdefs.Dods"/>
```

Parameters:

domlfile - The doml input file describing data object mapping. Required = Yes.

outputDir - Target for generated classes, expressed as a directory path. Required = Yes.

force - Forces DODS always to regenerate source files. Possible values: ("true", "false"(default)). Required = No.

action - Name of Ant task from generate.xml. Required = No.

templateDir - Name of folder for template set for generating java code, expressed as a directory path. Required = No.

templateSet - Template set for generating java code. Required = No.

confDir - Path to custom configuration folder (If the path is set to any other path than default (offered), in the application's configuration file shoud be set parameter:

```
DatabaseManager.ConfigurationDir
```

to new path of the custom configuration folder). Required = No.

database - Sets database vendor for generating sql. Required = No.

html - Indicates DODS to generate html documentation from .doml file. Possible values: (true, false (default)). Required = No.

pdf - Indicates DODS to generate pdf documentation from .doml file. Possible values: (true, false (default)). Required = No.

xmi - Indicates DODS to generate xmi documentation from .doml file. Possible values: (true, false (default)). Required = No.

ptl - Indicates DODS to generate ptl (Rational Rose) documentation from .doml file. Possible values: (true, false (default)). Required = No.

action parameters:

- without parameters - to create all sql files and java classes and to compile them.
- dods:build_all - to create all sql files and java classes.

- dods:sql - to create only sql files.
- dods:java -to create only java files and to compile them.
- dods:javaNoCompile -to create only java files and not to compile them.
- dods:noCompile -to create SQL files and java files and not to compile them.
- dods:build_all_split - to create all sql files and java classes and to compile them. SQL files will be divided into separate files using SQLSplitter.
- dods:sqlsplit - to create only sql files and separate them in different files using SQLSplitter.
- dods:noCompileSplit - to create SQL files and separate sql commands using SQLSplitter and java files and not to compile them.
- dods:generatorOff - to disable generating and compiling of java source code, for generating documentation only (you stil need to set documentation propertys: html, pdf, ptl, xmi).

templateset parameters:

- standard - generate standard java code.
- <user defined> - any user defined template set.

Example:

```
<dods doml="${basedir}/discRack.doml"

         outputDir="${basedir}/src"

         templateSet="standard"/>
```

Chapter 8. Table configuration

Table configuration is explained on DiscRack example (directory <dods_output>/examples/discrack). The table parameters are defined on three levels.

The first level is DatabaseManager level. On this level can be defined the following parameters (all information are optional):

```
DatabaseManager.defaults.lazyLoading = true
DatabaseManager.defaults.maxExecuteTime = 200
DatabaseManager.defaults.AllReadOnly = false
```

The second level is database level. On this level can be defined the following parameters (all information are optional):

```
DatabaseManager.DB.<database_name>.lazyLoading = false
DatabaseManager.DB.<database_name>.maxExecuteTime = 350
DatabaseManager.DB.<database_name>.AllReadOnly = false
```

The third level is table level. In the case of DiscRack example, there are two tables: Disc and person. The tables can have the following parameters:

```
#
# Table Disc - table configuration
# DatabaseManager.DB.DiscRack.Disc.readOnly = false
DatabaseManager.DB.DiscRack.Disc.lazyLoading = false
DatabaseManager.DB.DiscRack.Disc.maxExecuteTime = 150
```

```
#
# Table Person - table configuration
# DatabaseManager.DB.DiscRack.person.readOnly = true
DatabaseManager.DB.DiscRack.person.lazyLoading = false
# DatabaseManager.DB.DiscRack.person.maxExecuteTime = 150
```

Table defaults on DatabaseManager and Database are default values for all application's tables. If any of these parameters is defined on the Database level, that value is used as a default for all tables. If any of the parameters is not defined on the Database level, then, if it is defined on the DatabaseManager level, this value is used. If any of these parameters is not defined neither on the Database, nor on DatabaseManager level, DODS uses its own program defaults. For lazyLoading, program default is false, for maxExecuteTime 0 and for readOnly and AllReadOnly false.

If any of parameters lazyLoading or maxExecuteTime is defined on the table level, that value is used. If not, the default value for all tables is used (explained in previous paragraph).

Table parameter readOnly is true if the table is read-only, otherwise is false. If read-only is true, the operations: insert, update or delete on the tables are not possible.

If parameter AllReadOnly is defined and set to true (it can be defined on DatabaseManager or Database level), all applications will be read-only. In that case, table parameter readOnly is ignored. Only, If AllReadOnly is set to true and readOnly attribute of the table is set to false, warning is written to log during table initialization. In runtime exception is thrown on attempt of writing to that table.

Parameter lazyLoading is true if table supports lazy-loading, otherwise is false.

Parameter maxExecuteTime is time for query execution. Every query that is executed longer than maxExecuteTime is printed (SQL statement, execution time and maxExecutionTime) in application's log file.

Chapter 9. Caching

Caching affects the behaviour of the DO class. If checked, all DO instances (their original DataStruct objects) are stored in the cache inside the DO class. Subsequent queries of the table use the Query class for queries. The results of all Queries are complete.

Cache transformation

Since DODS 5.1 final, the DO cache is transformed into DataStruct cache. Instead of the whole DOs, only their original DataStructs are added to new DataStruct cache.

DO has had only one data (DataStruct object) and all transformations were done on this object. DataStruct object contains values of columns of one table row. Now, DO holds 2 DataStruct-references:

- originalData
- data

The originalData holds original data (that was read from the database). This is never modified till commit, and this DataStruct object is added to DataStruct cache, if this cache exists.

The second, data, is only created (by copying the first one) if data is modified. If the second DataStruct exists, the DO's attribute isDirty is set to true. Even if after some modifications the new DataStruct holds exactly the same values as the original one, the DO is still dirty. So there is no way back from isDirty=true to isDirty=false (except during commit of the transaction). If the transaction is committed, the new DataStruct is moved in the place of the original one. The new DataStruct is NULL again, so the attribute isDirty becomes false again.

A newly created DO (in memory, not from the database) will just have a DataStruct object data. Data values in DataStruct object originalData is null before the commit().

The oid and the version attributes are moved from DO to DataStruct object.

New attributes added in DataStruct object are:

- isEmpty

- type: boolean

- default value: true

Since originalData is being constructed for every DO, this flag "knows" if DataStruct has any useful content. If there is no data in DataStructs - except oid, this attribute is true, otherwise false.

- databaseName

- type: String

- default value: null

The logical database to which this DataStruct belongs to.

New methods added in DataStruct object are:

- getOId()

Returns DataStruct's identifier.

- setDatabase(String dbName)

Sets attribute databaseName.

- getDatabase()

Returns attribute databaseName.

- getHandle()

Returns this DataStruct's handle (identifier as a string).

- getCacheHandle()

Returns this DataStruct's cache handle (String in the form: "<database_name>.<indentifier_as_String>").

- get and set methods for every table column

In DO class are added new methods that work with originalData:

- originalData_get<column_name>()

Returns the row value of the column <column_name> of the DO's originalData object.

- originalData_set(Object data)

Sets the DO's originalData object.

- getData()

Returns DO's DataStruct object. If DO's data object exists, returns that object, otherwise returns DO's originalData object.

- originalData_get()

Returns DO's originalData object.

- getOriginalVersion()

Returns the current version of DO's originalData object.

Introduction

DODS provides the possibility for every table to have its cache.

The possible cache types are:

1.None

No caching is available.

2.LRU

The size of the cache is limited by the maximal number of objects that can be stored in it. When the cache is full, the objects in it are being replaced by new objects according to LRU (least recently used) algorithm. This algorithm says that the object which had been used the least recently (in the scale of time, the object to which had been accessed the longest time ago, which is on the end of LRU list) is removed from list and new one is put in front of the LRU list. If maximal number of objects is set to 0, it means that caching is not available (None type) at the moment.

3.Complete

This cache extends HashMap and is unbounded. This cache type is defined by the negative number of maximal cache size.

4.Full (special case of complete caching)

This is a complete cache (HashMap), for which is the entire table queried and cached when the application starts (initial condition is "*"). This is appropriate for tables of "static" data which are accessed frequently.

There is a method, *isComplete()*, in the cache class that checks if the cache (DataStruct cache) is complete or not. If the cache was not complete at the start, it is not checked if it becomes complete or not. But, if the cache was complete, it is than calculated whether the cache is still complete. The method for setting max cache size (in the situation when cache is not null and new cache size is not zero) for DataStruct cache changes cache implementation (form complete to LRU), only if the cache was complete and the new maxCacheSize is positive. In all other cases, the implementation stays as it was.

It is a little bit different with query caches. They don't define the global caching type, so any change form negative to positive max cache size (and vice versa) changes the cache implementation (Complete

or LRU).

When any of the caches (DataStruct or any of query caches) is created from scratch, the procedure is the same. Based on max cache size, the proper implementation is used. The same goes for methods for cache refreshing and enabling.

DODS has two levels of caching:

1.Data Caching level

There is only one LRU cache: cache with DataStruct objects. The keys of this cache are cache handles - Strings in the following form:

"<DataStruct_database_name>.<Table_name>.<String_presentation_of_DataStruct_oid>"

and cache values are, as mentioned before, DataStruct objects.

2.Query caching level

Beside DataStruct object cache, there is a possibility of using three query caches (simple, complex and multi-join). Multi-join cache is included since DODS 6.0. All query caches are also LRU caches. The keys of these caches are Strings in the following form:

"<query_database_name>.<String_presentation_of_query>",

and cache values are Query objects. Query objects are objects of the org.enhydra.dods.cache.QueryCacheItem class.

The QueryCacheItem object stores one query and its necessary data:

- Database of the query
- List of oids of DataStruct objects that are results of the query. This list can contain all query results, or just some of them.
- Number of cached query results
- Information whether all results are in result list or not
- Information whether the query results are modified (if there have been performed inserts, updates or deletes, the results are modified)
- Time needed for query execution
- Array of conditions declared in WHERE part of the query (array of org.enhydra.dods.cache.Condition objects). This is needed only for simple queries.

Queries that were created with the query's and QueryBuilder's methods that support joins between tables are stored in multi-join cache. Queries that are supported by DataStruct cache are simple queries. Simple query is query that is not multi-join query and for which cache mechanisms can determine whether DataStruct object is query result or not (and query). Other queries (that are not multi-join queries) are complex queries.

The default values for maximal DataStruct cache size, simple, complex and multi-join query cache are 0 (no caching).

Cache configuration

Cache configuration is explained on DiscRack example (directory <dods_output>/examples/discrack). The cache parameters are defined on three levels.

The first level is DatabaseManager level. On this level can be defined the following parameters (all information are optional):

```
  # DatabaseManager.defaults.cache.maxCacheSize = 100
  DatabaseManager.defaults.cache.maxSimpleCacheSize = 20
  DatabaseManager.defaults.cache.maxComplexCacheSize = 5
  DatabaseManager.defaults.cache.maxMultiJoinCacheSize = 3
  DatabaseManager.defaults.cache.reserveFactor = 0.1
  DatabaseManager.defaults.cache.CachePercentage = -1
  # DatabaseManager.defaults.cache.initAllCaches = true
  DatabaseManager.defaults.cache.asynchLoadThreadNum = 2
  DatabaseManager.defaults.cache.simpleCacheRowCountLimit = 300
  DatabaseManager.defaults.cache.synchLoadRowCountLimit = 5000
  DatabaseManager.defaults.cache.maxExecuteTimeCacheInit = 300
  DatabaseManager.defaults.cache.queryTimeoutCacheInit = 10
  DatabaseManager.defaults.cache.queryTimeLimitCacheInit = 12000
```

The second level is database level. On this level can be defined the following parameters (all information are optional):

```
  DatabaseManager.DB.<database_name>.cache.maxCacheSize = 1100
  # DatabaseManager.DB.<database_name>.cache.maxSimpleCacheSize = 10
  # DatabaseManager.DB.<database_name>.cache.maxComplexCacheSize = 5
  # DatabaseManager.DB.<database_name>.cache.maxMultiJoinCacheSize = 3
  DatabaseManager.DB.<database_name>.cache.reserveFactor = 0.1
  DatabaseManager.DB.<database_name>.cache.CachePercentage = -1
  DatabaseManager.DB.<database_name>.cache.initAllCaches = true
  DatabaseManager.DB.<database_name>.cache.simpleCacheRowCountLimit = 400
  DatabaseManager.DB.<database_name>.cache.synchLoadRowCountLimit = 6000
  DatabaseManager.DB.<database_name>.cache.maxExecuteTimeCacheInit = 400
  DatabaseManager.DB.<database_name>.cache.queryTimeoutCacheInit = 15
  DatabaseManager.DB.<database_name>.cache.queryTimeLimitCacheInit = 12000
```

The third level is table level. In the case of DiscRack example, there are two tables: Disc and person. The tables can have the following parameters:

```
#
  # Table Disc - cache configuration
  # DatabaseManager.DB.DiscRack.Disc.cache.maxCacheSize = 10000
  DatabaseManager.DB.DiscRack.Disc.cache.maxSimpleCacheSize = 2000
  DatabaseManager.DB.DiscRack.Disc.cache.maxComplexCacheSize = 250
  DatabaseManager.DB.DiscRack.Disc.cache.maxMultiJoinCacheSize = 100
  DatabaseManager.DB.DiscRack.Disc.cache.reserveFactor = 0.1
  DatabaseManager.DB.DiscRack.Disc.cache.CachePercentage = 0.5
  DatabaseManager.DB.DiscRack.Disc.cache.asynchLoadPriority = 2
  DatabaseManager.DB.DiscRack.Disc.cache.simpleCacheRowCountLimit = 1000
  DatabaseManager.DB.DiscRack.Disc.cache.synchLoadRowCountLimit = 10000
  DatabaseManager.DB.DiscRack.Disc.cache.maxExecuteTimeCacheInit = 500
  DatabaseManager.DB.DiscRack.Disc.cache.queryTimeoutCacheInit = 10
  DatabaseManager.DB.DiscRack.Disc.cache.queryTimeLimitCacheInit = 12000
```

```
#
  # Table Person - cache configuration
  DatabaseManager.DB.DiscRack.person.cache.maxCacheSize = -1
  DatabaseManager.DB.DiscRack.person.cache.maxSimpleCacheSize = 2000
  DatabaseManager.DB.DiscRack.person.cache.maxComplexCacheSize = 250
  DatabaseManager.DB.DiscRack.person.cache.maxMultiJoinCacheSize = 75
  DatabaseManager.DB.DiscRack.person.cache.initialCondition = *
  DatabaseManager.DB.DiscRack.person.cache.asynchLoadPriority = 1
  DatabaseManager.DB.DiscRack.person.cache.simpleCacheRowCountLimit = 100
  DatabaseManager.DB.DiscRack.person.cache.synchLoadRowCountLimit = 5000
  DatabaseManager.DB.DiscRack.person.cache.maxExecuteTimeCacheInit = 200
  DatabaseManager.DB.DiscRack.person.cache.queryTimeoutCacheInit = 5
  DatabaseManager.DB.DiscRack.person.cache.queryTimeLimitCacheInit = 7000
```

Cache defaults on DatabaseManager and Database are default values for all application's table caches. If,

any of these parameters is defined on the Database level, that value is used as a default for all tables. If any of the parameters is not defined on the Database level, then, if it is defined on the DatabaseManager level, this value is used. If any of these parameters is not defined neither on the Database, nor on DatabaseManager level, DODS uses its own program defaults. For maxCacheSize, maxSimpleCacheSize, maxComplexCacheSize, maxMultiJoinCacheSize, reserveFactor, asynchLoadThreadNum, simpleCacheRowCountLimit, synchLoadRowCountLimit program default value is 0, for CachePercentage is -1.0, for initAllCaches is false and for maxExecuteTimeCacheInit, queryTimeoutCacheInit and queryTimeLimitCacheInit program default value is value defined for parameters maxExecuteTime, QueryTimeout and QueryTimeLimit.

If any of table level parameters maxCacheSize, maxSimpleCacheSize, maxComplexCacheSize, maxMultiJoinCacheSize, reserveFactor, CachePercentage, simpleCacheRowCountLimit, synchLoadRowCountLimit, maxExecuteTimeCacheInit, queryTimeoutCacheInit or queryTimeLimitCacheInit is defined on the table level, that value is used. If not, the default value for all tables is used (explained in previous paragraph).

The parameter initialCondition can be defined only on the table level. It contains "where" part of select clause. With this select clause is DataStruct cache of specified table initialized. If initialCondition = '*', the entire table will be added to the DataStruct cache in DataStruct cache initialization. If the parameter is NULL or not defined, no objects are added to the Data cache during the cache initialization.

If, for any table parameter initialCondition is not defined and the initAllCaches parameter is set to 'true' (on DatabaseManager or Database level, as explained before), the default value of initialCondition parameter for the table is "*".

Parameter maxCacheSize contains information about maximal size of DataStruct cache. Parameter maxSimpleCacheSize contains information about maximal size of simple query cache. Parameter maxComplexCacheSize contains information about maximal size of complex query cache. Parameter maxMultiJoinCacheSize contains information about maximal size of multi-join query cache.

Parameter CachePercentage is used for query to make decision what type of query will be executed: select t.* or select t.oid. If no lazy loading and caching is turned on and value of CachePercentage is less then currently used cache (in percents), t1.* is used for query statement. Otherwise select t.oid. Parameter value 0 means use always t1.oid if cache is turned on, -1 (default) means never if not lazyloading but cached. If lazy loading is on always is used t1.oid query.

In <table_name>Query.java class are added new methods:

- setLoadData(boolean newValue)

  If parameter newValue set to true, query select t.* will be executed no matter what are the values of parameters lazyLoading and CachePercentage.

- getLoadData()

  Returns true if query select t.* will be executed, otherwise false.

Reserve factor is constant used in query caching. It is percent of how many more object are taken for evaluation. If num is number of needed results, then it is used

```
num + reserveFactor * num
```

objects for estimating what is quicker: go to database for all object that are not in the cache, or run again query on database. This value is given in percents, as number between 0 and 1 (0.25 means 25%).

For example, if reserveFactor is 0.5, and wanted number of results is 50, the estimation will be done on 75 (50 + 0.5 * 50) objects.

In the following text are explained maximal cache sizes (for DataStruct cache and query caches). The parameters maxCacheSize, maxSimpleCacheSize, maxComplexCacheSize and maxMultiJoinCacheSize of application's configuration file define these sizes.

- maxCacheSize > 0

This cache is limited. The maximal number of elements in the cache is maxCacheSize. This is LRU cache type.

- maxCacheSize = 0

This means that there is no cache available. This value excludes cache from use.

- maxCacheSize < 0

This cache is unlimited. This is complete type of cache (HashMap).

The parameter asynchLoadThreadNum is only defined on DatabaseManager level. This is the number of threads used for asynchronous cache load during application startup. The default value is 0 (asynchronous cache load is not used).

The parameter asynchLoadPriority is only defined on table level. It is the priority of asynchronous cache load for the table. The table that has the lowest value for this parameter will be first asynchronous loaded during application startup. When a thread finishes cache load of a table, it takes the next table from the priority list and loads its cache, and so on. The default value for this parameter is -1. This means that the cache for that table will not be asynchronous loaded.

The parameter simpleCacheRowCountLimit defines max number of rows in the table for which simple cache is still used. If the table has more rows than defined by this parameter, complex cache is used for simple queries. The default value is 0 (simple cache is used for all simple queries).

The parameter synchLoadRowCountLimit defines the max number of rows in the table for which the synchronous cache load is performed if defined by configuration. If the number of rows is greater, the table's cache will be loaded asynchronous and this number will be taken for the asynchLoadPriority . The default value is 0 (asynch cache load is not performed if configuration parameters for asynch cache load are not defined).

The parameter maxExecuteTimeCacheInit is similar to table parameter maxExecuteTime, but defined for cache initialization. It defines the max time for which the query is not printed in application's log file during the cache initialization. If the time is greater, query (SQL statement, execution time and maxExecutionTime) is printed. The default value is value defined for parameter maxExecuteTime (whose default is 0 - nothing is printed).

The parameter queryTimeoutCacheInit is similar to table parameter QueryTimeout, but defined for cache initialization. It defines max number of seconds for which the query for cache initialization should be executed. If the limit is exceeded, an exception is thrown. The default value is value defined for parameter QueryTimeout (whose default is 0 - no limit).

The parameter queryTimeLimitCacheInit is similar to table parameter QueryTimeLimit, but defined for cache initialization. It defines max number of milliseconds for which the query for cache initialization should be executed and the resulset read from ResultSet. If the limit is exceeded, an Exception is thrown. The default value is value defined for parameter QueryTimeLimit (whose default is 0 - no limit).

In the previous mentioned DiscRack example for cache configuration, DataStruct cache for table person has type full, because maxCacheSize is negative and initialCondition is "*". This combination of parameters values forms special case of complete cache: *full* cache.

DODS has class org.enhydra.dods.cache.UpdateConfigurationAdministration. This class has public synchronized methods that provide possibility of run-time setting some cache and table parameters. This class is used by Enhydra application CacheAdmin. It is not recommended to be used by user applications.

Table and cache statistics

DODS has the possibility of providing table and cache statistics.

The public method

```
get_statistics()
```

of the <table_name>DO.java class returns the statictics object (statistics object must implement org.enhydra.dods.statistics.Statistics interface). This object provides the following methods for the table statistics and one method for retrieving cache statistics:

- getStatisticsType()

  Returns type of the statistics. It returns 0 if statistics is for table that has no caching, 1 if statistics is for table with only Data caching, and 2 if statistics is for table with Query caching.

- getInsertNum()

  Returns number of insert statements performed on the table.

- setInsertNum(int newInsertNum)

  Sets number of insert statements performed on the table to value newInsertNum.

- incrementInsertNum()

  Increases number of insert statements performed on the table for one.

- getUpdateNum()

  Returns number of update statements performed on the table.

- setUpdateNum(int newUpdateNum)

  Sets number of update statements performed on the table to value newUpdateNum.

- incrementUpdateNum()

  Increases number of update statements performed on the table for one.

- getDeleteNum()

  Returns number of delete statements performed on the table.

- setDeleteNum(int newDeleteNum)

  Sets number of delete statements performed on the table to value newDeleteNum.

- incrementDeleteNum()

  Increases number of delete statements performed on table for one.

- getDMLNum()

  Returns number of DML operations (inserts, updates and deletes) performed on the table.

- getLazyLoadingNum()

  Returns number of lazy loadings performed on the table.

- setLazyLoadingNum(int newLazyLoadingNum)

  Sets number of lazy loadings performed on the table to value newLazyLoadingNum.

- incrementLazyLoadingNum()

  Increases number of lazy loadings performed on the table for one.

- getStartTime()

  Returns time when the statistics was started.

- setStartTime(Date startTime)

  Sets time when the statistics starts to value startTime.

- getStopTime()

  Returns time when the statistics was stopped.

- setStopTime(Date stopTime)

  Sets time when the statistics stops to value stopTime.

- stopTime()

  Sets stop time to current time.

- getQueryNum()

  Returns total number of non-oid queries performed on the table. Query by oid is query which "where" clause contains request for DO with specified oid. Non-oid query is any other query.

- setQueryNum(int newQueryNum)

  Sets total number of non-oid queries performed on the table to value newQueryNum.

- incrementQueryNum()

  Increases total number of non-oid queries performed on the table for one.

- getQueryByOIdNum()

  Returns total number of queries by oid performed on the table.

- setQueryByOIdNum(int newQueryByOIdNum)

  Sets total number of queries by oid performed on the table to value newQueryByOIdNum.

- incrementQueryByOIdNum()

  Increases total number of queries by oid performed on the table for one.

- getQueryAverageTime()

  Returns average time needed for executing non-oid query.

- updateQueryAverageTime(int newTime)

  Updates average time needed for executing non-oid queries to value newTime.

- getQueryByOIdAverageTime()

  Returns average time needed for executing query by oid.

- updateQueryByOIdAverageTime(int newTime, int no)

  Updates average time for executing OId queries with time newTime and increments number of them by paramether no.

- clear()

  Clears DO, simple query and complex query statistics.

- getCacheStatistics(int type)

  Returns cache statistics (objects must implement interface org.enhydra.dods.statistics.CacheStatistics) for :

  - DataStruct cache when parameter type equals 0
  - simple query cache when parameter type equals 1
  - complex query cache when parameter type equals 2
  - multi-join query cache when parameter type equals 3

Cache statistics objects have the following methods:

- getCacheAccessNum()

  Returns total number of times the cache was accessed.

- setCacheAccessNum(int num)

  Sets total number of times the cache was accessed to value num.

- incrementCacheAccessNum(int num)

  Increases total number of times the cache was accessed for value num.

- getCacheHitsNum()

Returns number of cache accesses that were successful.

- setCacheHitsNum(int cacheHitsNum)

  Sets number of of cache accesses that were successful to value cacheHitsNum.

- incrementCacheHitsNum(int num)

  Increases number of cache accesses that were successful for value num.

- getUsedPercents()

  Returns how much cache is currently used. This value is given in percents. If cache is unbounded, method returns 100%.

- getCacheHitsPercents()

  Returns how many cache accesses were successful. This value is given in percents.

- clearStatistics()

  Clears statistics.

Select statement

For query by oid (query by oid is query which "where" clause contains request for DO with specified oid), first is checked in the DataStruct cache if there is DataStruct object with desired oid. If DataStruct object is not find in the cache, hitting the database is performed, and the retrieved DataStruct object is added to the DataStruct cache. Queries by oid are not added in the query cache (they are trivial).

For full caching also, for query by oid, first is checked in the DataStruct cache if there is DataStruct object with desired oid. If DataStruct object is not find in the cache, hitting the database is not performed (all rows from the table are in the cache, so there is no result of this query).

For non-oid queries, for full caching, if the query is simple query, the query's result can be retrieved from the DataStruct cache, so there is no need to retrieve results from the database. In any other case of full caching, everything is done the same as for any other query (this is explained in the next paragraph).

For all other queries, it is checked if the query is already in the Query cache (simple, complex or multi-join). Query object has one attribute called "orderRelevant" which is true if query results must not be modified (no DO can be inserted, updated or deleted from cached query results). With the method isOrderRelevant() is checked whether the results of select can be modified or not.

If query is in the cache and the isOrderRelevant() returns false, result oids are retrieved from QueryCache. If query is in the cache and the isOrderRelevant() returns true, and the result oids are not modified, the result oids are also retrieved from query cache. But, if query is in the cache and the isOrderRelevant() returns true, but the result oids are modified, the result oids from the QueryCache are not used. Instead of that, hitting the database is performed.

If the result is found in the query cache, for every result oid, it is checked whether there is that object is in the DataStruct cache. Then, when is counted number of results that are not in the DataStruct cache, the time needed for performing queries by oid on database for all oids from the result that are not in the

cache is compared against the time needed for performing the whole query.

If the time needed for performing queries by oid on database is less or equal to query execution time, results are retrieved from the cache, and those that are not there, from database (using queries by oid).

If the time is longer, or the query is not in the query cache, or the query supports joins with other tables, or cached query results are modified but for this query is order relevant, the query is performed on the database.

If the results are retrieved from the database, the query and its necessary data are put in the Query cache (simple, complex or multi-join).

If there was already that query in the query cache, but the query was executed again (because there were not enough result oids in the result list, or because the old query was modified, and for the new query isOrderRelavant is true), the old query is replaced by the new one (this query is not modified).

Insert statement

Data object is inserted in the database and first time the data is moved to original DataStruct, it is added to the DataStruct cache, after successful commit.

All complex and multi-join queries of the table that are for the database of inserted DO, are removed from the query caches.

For every simple query of the table (with the inserted DO's database) from query cache it is checked whether inserted DO is query result or not.

If new DO is query result, in the query cache is this query marked as "modified".

If its cached results are complete (all are in the query cache), oid of this inserted DO is added to query cached result list. If cached results are not complete oid is not added to the list.

Update statement

Data object is updated in the database and first time the data is moved to original DataStruct, it is added to the cache if commit was successful (the old DataStruct object is removed from the DataStruct cache if it was there).

All complex and multi-join queries of the table that are for the database of inserted DO are removed from the query caches.

For every simple query of the table (with the inserted DO's database) from query cache it is checked whether updated DO is the query result or not.

If yes, this query is marked as "modified" in the query cache, and the DO is included in query results only if it wasn't in the cache and the cached result list is complete.

If no, if DO's oid exists in the query results, it is removed from there and because of this change of the results, this query is marked as "modified" in the query cache.

Delete statement

Deletes DO from the database and removes its original DataStruct object originalData from the Data-Struct cache (if it is there).

Goes through the query cache (simple, complex and multi-join) and wherever finds this DO, removes it from the query results and marks that query as "modified".

Cache Initialization

For every DataStruct cache, it is possible to define initial query statement which contains "where" clause which is used during DataStruct cache initialization. When cache is created, query with this "where" condition is performed on the database, and the results are put in the DataStruct cache.

Before the query is executed, parameter maxDBrows is set to maxCacheSize using method

setMaxRows(int max)

of <table_name>Query.java class.

Using this method, maximum maxCacheSize DOs will be retrieved from the database and their original DataStruct objects (originalData) will be added to DataStruct cache.

If a table is fully cached, simple queries are done in the memory (even the first time this is done in the cache and not in the Database)

If initial query statement is set to "*", all rows of the table from the database (up to maxCacheSize) will be put in the DataStruct cache.

If initial query statement is set to null, no rows from the table in database will be put in the DataStruct cache during the initialization (cache would be empty).

The parameter ClassList defines the absolute path to "DODSClassList.xml" file that contains the list of data layer classes names whose caches should be initialized. There are two types of cache initialization: synchronous and asynchronous.

Synchronous cache initialization is performed during the application startup.

Beside ClassList parameter, for aynchronous cache initialization, two more parameters must be defined: *asynchLoadThreadNum* and *asynchLoadPriority*.

If parameter *asynchLoadThreadNum* is defined and has positive value, the asynchronous cache load would be performed with the number of threads defined by this parameter. If the parameter is not defined, or has negative or 0 value, the asynchronous cache initialization will not be performed during the application startup.

The parameter *asynchLoadPriority* (must be number equal or greater than zero) defines the priority (order) for the table's asynchronous cache load. The lower the number, the sooner the cache for the table will be loaded.

As mentioned, if parameter *asynchLoadThreadNum* is not defined (or negative) asynchronous cache load will not be performed for any of tables. Caches of the tables that have defined *asynchLoadPriority*,

will not even be initialized synchronous because of this parameter. But, class com.lutris.appserver.server.sql.StandardDatabaseManager has method:

```
public void asynchInitCaches(int threadNum)
```

that can be call later. It defines the number of threads for asynchronous cache initialization and calls the initialization. This method can have effect only on tables whose caches were not loaded before.

Beside this main parameters for cache initialization, there are more parameters that can be used during the cache initialization (all are explained in the chapter *Caching*, in the section about *Cache configuration*):

- *synchLoadRowCountLimit* - max number of rows in the table for which the synchronous cache load is performed if defined by configuration. If the number of rows is greater, the table's cache will be loaded asynchronous and this number will be taken for the *asynchLoadPriority*. The default value is 0 (asynch cache load is not performed if configuration parameters for asynch cache load are not defined).
- *maxExecuteTimeCacheInit* - max time for which the query is not printed in application's log file during the cache initialization. If the time is greater, query (SQL statement, execution time and maxExecutionTime) is printed. The default value is value defined for parameter *maxExecuteTime* (whose default is 0 - nothing is printed).
- *queryTimeoutCacheInit* - max number of seconds for which the query for cache initialization should be executed. If the limit is exceeded, an exception is thrown. The default value is value defined for parameter QueryTimeout (whose default is 0 - no limit).
- *queryTimeLimitCacheInit* - max number of milliseconds for which the query for cache initialization should be executed and the resulset read from ResultSet. If the limit is exceeded, an Exception is thrown. The default value is value defined for parameter QueryTimeLimit (whose default is 0 - no limit).

When synchronous cache load is performed, the application waits for it to be over before it starts its work. The other case is with asynchronous cache load: the application is running while the caches are being asynchronous loaded.

Chapter 10. User wildcards

Like cache size, user wildcards are also defined in application's configuration file.

Example:

For file discRack.conf part of code for user wildcards can look like:

```
#

# User wildcards

#

DatabaseManager.DB.User.userWildcard = "*"

DatabaseManager.DB.User.userSingleWildcard = "_"

DatabaseManager.DB.User.userSingleWildcardEscape = "$"

DatabaseManager.DB.User.userWildcardEscape = "$"
```

Chapter 11. Database Independency

DODS generates java code that is database independent. This means that java code is the same no matter which database is used.

When you want to change the database, the only thing you need to do is to change <App_name>.conf file (update it with information considering new database). This change is necessary for connection to the database.

Chapter 12. Using multi databases in DODS

DODS has the possibility of working with more than one database at the same time. This means that, when the application is started, it doesn't have to be stopped in order to change the database the application uses.

The table supports multi databases if the attribute multidb of <table> tag in doml file is set to true for that table.

To take advantage of simultaneous use of multiple table DODS requires separate doml file for every distinct database.

Example:

```
<doml>
  <database database="Standard">
    .........
    <package id="multibase.data.employee">
        <table id="multibase.data.employee.Employee"  multidb="true">
             <column id="firstName" usedForQuery="true">
                 <type dbType="VARCHAR" javaType="String"/>
             </column>
             ........
        </table>
        .........
    </package>
    .........
</doml>
```

```
<doml>
  <database database="Standard">
        ............
        <package id="multibase.data.employee.programer">
          <table id="multibase.data.employee.programer.Programer"  multidb="true">
            <column id="firstName" usedForQuery="true">
              <type dbType="VARCHAR" javaType="String"/>
            </column>
            ............
          </table>
          ...........
        </package>
        ...........
  </database>
</doml>
```

For that kind of table, in <App_name>.conf file must be defined all logical databases the application will use on this table. For each of these databases must be configured all needed parameters.

Example:

```
#-------------------------------------------------------------------------------
#                   Database Manager Configuration
#-------------------------------------------------------------------------------

DatabaseManager.Databases[] = "programer", "employee"
DatabaseManager.DefaultDatabase = "employee"

DatabaseManager.DB.programer.Connection.User = ""
DatabaseManager.DB.employee.Connection.User = ""

DatabaseManager.DB.programer.Connection.Password = ""
DatabaseManager.DB.employee.Connection.Password = ""

.................................................

DatabaseManager.DB.programer.Connection.Logging = false
```

```
DatabaseManager.DB.employee.Connection.Logging = false

DatabaseManager.DB.programer.ObjectId.CacheSize = 20
DatabaseManager.DB.programer.ObjectId.MinValue = 1000000

DatabaseManager.DB.employee.ObjectId.CacheSize = 20
DatabaseManager.DB.employee.ObjectId.MinValue = 1000000
```

If the table doesn't support multi databases, the default database will be used for this table.

When the <App_name>.conf file (with information about all databases) is updated, and the application is started, it uses the default database. The definition of the new (desired) database is being done in the stage of creation of DO and Query objects.

When a Query object is created for a database (given or default), the results of this Query are only DOs from that database, not from any other database.

If caching is used, there is only one cache for all <table_name>DO's original data originalData (these DataStruct objects can belong to different databases, but are all placed in the same DataStruct cache).

DODS takes care of referential integrities within the database which means that DODS searches referenced object in the same database in which the object that referenced it is. If you want to use referenced objects from any other database, you must yourself take care of referential integrities.

In the <table_name>DO class public constructors and methods (query, createVirgin, createCopy, createExisting) are now defined and with the database parameter.

Here are some examples of using these constructors and methods.

Query example:

```
ProgramerDO[] programers;
ProgramerQuery pQuery = new ProgramerQuery("programer");
programers = pQuery.getDOArray();
```

Create example:

```
EmployeeDO newE=EmployeeDO.createVirgin("employee");
newE.setFirstName(employees[i][0]);
.................................
newE.save();
```

Example of transferring data from one database to another:

```
ProgramerDO[] programers;
ProgramerQuery pQuery = new ProgramerQuery("programer");
programers = pQuery.getDOArray();
for(int i=0; i< programers.length; i++) {
   EmployeeDO newEmployee=EmployeeDO.createVirgin("employee");
   newEmployee.setFirstName(programers[i].getFirstName());
   newEmployee.setLastName(programers[i].getLastName());
   newEmployee.setOccupation("programer");
   newEmployee.setDepartment("IT");
   newEmployee.save();
}
```

You can use them with this parameter in which case the object will be created for the given logical database, or you can use these constructors and methods without database parameter. In this case, they will be created for default database. If the methods with database parameter are used, and the parameter is set to null, the default database is used.

Chapter 13. Conversion of doml file

DODS has the possibility of converting doml file, release 5.*. As described in the section "Quick Compile" in this document, Generator Wizard has a possibility of converting doml file into html, pdf, xmi and ptl document types. The name of the target (html, pdf, xmi, ptl) files will be the same as the name of doml file that is being converted, and they would be located in output directory.

The doml file can also be converted manually. For this purpose are used files in <dods_root>/bin folder, and they are:

- doml2html - converts doml 5.* file into html file.

  doml2html is used with the following parameters:

  ```
  doml2html [-help] [doml5*-file] [html-file]
  ```

  where:

  - help - prints message for usage and exits.
  - doml5*-file - doml file relese 5.*.
  - html-file - desired target html file.

- doml2pdf - converts doml 5.* file into pdf file.

  doml2pdf is used with the following parameters:

  ```
  doml2pdf [-help] [doml5*-file] [pdf-file]
  ```

  where:

  - help - prints message for usage and exits.
  - doml5*-file - doml file relese 5.*.
  - pdf-file - desired target pdf file.

- doml2xmi - converts doml 5.* file into xmi file.

  doml2xmi is used with the following parameters:

  ```
  doml2xmi [-help] [doml5*-file] [xmi-file]
  ```

  where:

  - help - prints message for usage and exits.
  - doml5*-file - doml file relese 5.*.
  - xmi-file - desired target xmi file.

- doml2ptl - converts doml 5.* file into ptl file.

doml2ptl is used with the following parameters:

```
doml2ptl [-help] [doml5*-file] [ptl-file]
```

where:

- help - prints message for usage and exits.
- doml5*-file - doml file relese 5.*.
- ptl-file - desired target ptl (Rational Rose) file.

- olddoml_2_doml60 - converting doml 3.1 and 5.0 into doml 6.0 file.

  olddoml_2_doml60 is used with the following parameters:

  ```
  olddoml_2_doml60  [-help] [olddoml-file] [doml60-file]
  ```

  where:

  - help - prints message for usage and exits.
  - olddoml-file - doml file relese 3.1. or 5.0
  - doml60-file - desired target doml file relese 6.0.

- doml31_2_conf60 - converting old style (doml 3.1) propertys of doml elements, from doml 3.1 file, to configuration file properties parameter setings in output file.

  doml31_2_conf60 is used with the following parameters:

  ```
  doml31_2_conf60 [-help] doml31-file confProperty60-file [defaultLazyLoading]
  [defaultCaching]
  ```

  - help - prints message for usage and exits.
  - doml31-file - Original doml 3.1 file
  - confProperty60-file - Output file where configuration properties will be written.
  - defaultLazyLoading - If LazyLoading is not defined in the doml file (for that table) then this value is used (values: true/false).
  - defaultCaching - If Caching is not defined in the doml file (for that table) then this value is used (values: true/false).

  If LazyLoading (or Caching) is not defined in the doml file, and there is no default value defined (defaultLazyLoading or defaultCaching), the following line is written to the application configuration file (no value) :

  ```
  DatabaseManager.DB.<database_name>.<table_name>.cache.maxCacheSize=
  DatabaseManager.DB.<database_name>.<table_name>.cache.initialCondition=
  DatabaseManager.DB.<database_name>.<table_name>.lazyLoading=
  ```

Chapter 14. Template sets

User can make its own template sets. All template sets (standard and users) are placed in one directory. The name and location of this directory is defined in the dodsConf.xml file. This file is detailly explained in the next section.

Also, this directory can be set from DODS Ant task (parameter templateDir).This was explained before in this chapter, in the DODS Ant task section.

Within this template set directory, every template set is placed in its own subdirectory.

Chapter 15. Custom Configuration

To configure DODS, use dodsConf.xml file, located in <dods_root>/build/conf directory. This file contains the following information:

- Location of templates - tag <TemplateDir>

  example:

```
<TemplateDir>C:/DODS/build/template</TemplateDir>
```

- For each database vendor, location of its configuration file (in xml format) - tag <Database>. The paths are relative to dodsConf.xml file folder.

  example:

```
<Database>

  <Vendor name="Standard">StandardConf.xml</Vendor>

  <Vendor name="InstantDB">InstantDBConf.xml</Vendor>

  <Vendor name="Oracle">OracleConf.xml</Vendor>

  <Vendor name="Informix">InformixConf.xml</Vendor>

  <Vendor name="MSQL">MSQLConf.xml</Vendor>

  <Vendor name="Sybase">SybaseConf.xml</Vendor>

  <Vendor name="PostgreSQL">PostgreSQLConf.xml</Vendor>

  <Vendor name="HypersonicSQL">HypersonicSQLConf.xml</Vendor>

  <Vendor name="DB2">DB2Conf.xml</Vendor>

  <Vendor name="QED">QEDConf.xml</Vendor>

  <Vendor name="MySQL">MySQLConf.xml</Vendor>

</Database>
```

Database Vendor's configuration file contains information about that database (type of ObjectId, column name for oid and version, information about DeleteCascade, constraints, quotes,comments, characters for like and wildcard, mapping JDBC types to vendor-specific data types,...).

If tags <ClassPath>, <ClassName> in database vendor's configuration file are not mentioned, standard code is generated for that database vendor.

If database is specific, path to jar file for that database vendor is in tag <ClassPath>, and its main class is in tag <ClassName>.

example, for database Informix:

```
<ClassPath>C:/DODS/lib/dbvendors/informix.jar</ClassPath>

<ClassName>com.lutris.appserver.server.sql.informix.InformixLogicalDatabase</ClassName>
```

Chapter 16. Logging in DODS

DODS suport loggig through concepts of *Logger* and *LogChannel* (from *com.lutris.logging* package).

In this version, DODS ships with two: *com.lutris.logging.StandardLogger*, and *com.lutris.logging.Log4jLogger* implementation of *Logger*.

To support logging in DODS application it's necessary to set parameters in application configuration file:

- *DatabaseManager.LogClassName* - Fully qualified name of class implementing Logger (for now *com.lutris.logging.StandardLogger* or *com.lutris.logging.Log4jLogger*)

Then if you using *com.lutris.logging.Log4jLogger* Logger implementation class you need to set parameter :

- *DatabaseManager.Log4j* - Pathname of Log4j XML configuration file.

If *DatabaseManager.LogClassName* is set to *com.lutris.logging.StandardLogger* you need to set :

- *DatabaseManager.LogFile* - This is the file where the server log is written.
- *DatabaseManager. LogToFile[]* - This is a comma separated list of message types to send to the log file specified in server.logFile.
- *DatabaseManager.LogToStderr[]* - This is a comma separated list of message types to send to standard error.

Example:

```
DatabaseManager.LogClassName = com.lutris.logging.StandardLogger
DatabaseManager.LogFile = discRack.log
DatabaseManager.LogToFile[] = EMERGENCY, ALERT, CRITICAL, ERROR, WARNING, INFO
DatabaseManager.LogToStderr[] = EMERGENCY, ALERT, CRITICAL, ERROR, WARNING, INFO, DEBUG
```

or

```
DatabaseManager.LogClassName = com.lutris.logging.Log4jLogger
DatabaseManager.Log4j = log4j.xml
```

If you use DODS in Enhydra, Enhydra logger will be used as a default logger.

Chapter 17. Stored procedures used as an oid allocator

DODS generator creates file with stored procedures which can be used as an oid allocator. Currently DODS creates stored procedures for MSQL (MSQL 2000), Oracle and PostgreSQL. DODS generator generates two procedures : create_oid (for oid per database logic) and create_oid_per_table (for oid per table logic).

How to use procedures?

Stored procedures can be used if you want to insert data in oid based database, without DODS based application. In that case, usage of stored procedures will guarantee that oid based mechanism for primary keys will be valid.

Stored procedures return next available oid value. You can use it in SQL statements on following way:

```
insert into tabla_x(oid, ...) values( create_oid() , ... values for other columns...)
```

In case of MSQL database:

```
declare @x decimal(19,0)
exec create_oid, @returnOid = @x OUTPUT
insert into tabela_x(oid,...) values(@x, ... values for other columns...)
```

Chapter 18. More Information

DODS, especially caching and transactions, is more detailly explained in the document "Inside DODS" (html [../inside_dods/index.html] , pdf [../inside_dods/inside_dods.pdf]).

Chapter 19. License

Development components The Data Object Design Studio (DODS) development components (generator, templates,...) are licensed under the GNU General Public License (GPL).

Runtime components The Data Object Design Studio (DODS) runtime components (necessary to run generated Java code) are licensed under the GNU Lesser General Public License (LGPL).

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).