MIB Explorer Pro - Slim DAO

SlimDAO

Instead of using a static object relational database mapping, MIB Explorer uses its own lightweight and easy configurable database mapping mechanism called **SlimDAO**.

Basically a SlimDAO DB mapping configuration file contains SQL statements and their fragments in an XML file. Because the SQL dialects of different databases often differ only for special statements, like sequence and constraint creation, SlimDAO supports statement inheritance.

A SQL statement (or a complete script) can be defined in the default context „." and overwritten in a specific context whose name matches the JDBC driver class name for the database whose SQL dialect differs from the default context's dialect (e.g., org.postgresql.Driver for a PostgreSQL database).

Each statement and script in the mapping configuration has a unique name within its context. MIB Explorer will lookup a statement by its name. The name is hard-coded. The supported statement names are defined by the default context in the Monitor2DB.xml file from the db-monitor directory.

A statement is defined for example by the following construct:

```
    <statement name="select.monitor.by.id">
      SELECT * FROM @INSERT[SCHEMA].MXP_MONITOR
                WHERE ID = @VALUE[ID]
    </statement>
```

During runtime, MIB Explorer uses the statement to get monitor data from the database. It replaces the constructs of the pattern

```
@<TAG>[<PARAM_LIST>]
```

with the corresponding values as described by the table following this section. The resulting SQL statement would then be executed on the DB server as follows:

```
SELECT * FROM MXP.MXP_MONITOR WHERE ID = ?
```

The question mark ? is a placeholder for the value within the prepared statement. The value will be inserted by the JDBC driver when the prepared statement is being executed. MIB Explorer sets the value for the statement by replacing the @VALUE[ID] reference with the ? in the statement and setting the statements first parameter value to the value of ID (for example 2).

Tag	Description
@VALUE[<param>]	Replace the tag by Example:? in the statement and set the value of the prepared parameter to the value of the MIB Explorer variable with name <param>. ``` SELECT ID FROM MXP.MXP_MONITOR WHERE NAME = @VALUE\[NAME\] ``` will be rendered to ``` SELECT ID FROM MXP.MXP_MONITOR WHERE NAME = ? ``` where ? is assigned with the value of the MIB Explorer variable NAME when the statement is being executed.
@INSERT[<param>]	Replace the tag by the value of MIB Explorer variable with name <param> Example: ``` SELECT ID FROM MXP.MXP_MONITOR WHERE NAME = '@INSERT[NAME]' ``` will be rendered to ``` SELECT ID FROM MXP.MXP_MONITOR WHERE NAME = 'MyMonitor' ``` if the variable NAME has the string value MyMonitor.

@USE[<statementName>]	Replace the tag with the rendered content of the statement with name <statementName> of the same context.
	Example:
| | ```
<statement name="select.id"> \\
SELECT ID \\
</statement> \\
<statement name="select.monitor.id.by.name"> \\
@USE[select.id] FROM MXP.MXP_MONITOR WHERE NAME = @VALUE[NAME] \\
</statement>
``` |
| | will be rendered to |
| | ```
SELECT ID FROM MXP.MXP_MONITOR WHERE NAME = 'MyMonitor'
``` |
	if the variable NAME has the string value MyMonitor.
@IF[<stmt>,<test>, <value>,..]	Replace the tag with the statement with name <stmt> if the condition <test> is true for the MIB Explorer variable references <value1> through <valueN>.
	Example:
| | ```
<statement name="monitor.dataset.by.id"> \\
SELECT * FROM MXP.MXP_MONITORDATASET WHERE MONITOR_ID = @VALUE\[ID\] \\
@IF\[and.nameOrDataType,not-null,NAME,DATATYPE\] \\
ORDER BY MONITOR_ID,DATATYPE \\
</statement> \\
<statement name="and.nameOrDataType"> \\
AND (NAME = @VALUE\[NAME\] OR DATATYPE = @VALUE\[DATATYPE\]) \\
</statement>
``` |
| | will be rendered to |
| | ```
SELECT * FROM MXP.MXP_MONITORDATASET WHERE MONITOR_ID = ? AND (NAME = ? OR DATATYPE
= ?) ORDER BY MONITOR_ID,DATATYPE
``` |
| | if the MIB Explorer variables NAME and DATATYPE are not null. |

`@IFELSE[<stmtTrue>, <stmtFalse>,<test>, <value>..]`	Replace the tag with the statement with name `<stmtTrue>` if the condition `<test>` is true for the MIB Explorer variable references `<value1>` through `<valueN>`. Otherwise, replace the tag with the statement with name `<stmtFalse>`.

Example:

```
<statement name="monitor.dataset.by.id"> \\
SELECT * FROM MXP.MXP_MONITORDATASET WHERE MONITOR_ID = @VALUE\[ID\] \\
@IFELSE\[dataTypeNotNull,dataTypeNull,not-null,DATATYPE\] \\
</statement> \\
<statement name="dataTypeNull"> \\
AND DATATYPE IS NULL \\
</statement> \\
<statement name="dataTypeNotNull"> \\
AND DATATYPE IS NOT NULL \\
</statement>
```

will be rendered to

```
SELECT * FROM MXP.MXP_MONITORDATASET WHERE MONITOR_ID = ? AND DATATYPE IS NOT NULL
```

if the MIB Explorer variables `DATATYPE` is not `null`. Otherwise, the statement would be rendered as:

```
SELECT * FROM MXP.MXP_MONITORDATASET WHERE MONITOR_ID = ? AND DATATYPE IS NULL
```

The tags `@IF` and `@IFELSE` insert a statement depending on the truth of a condition. The supported condition rules are listed by the following table.

Test Condition	Description
`exists`	Is `true` if all parameters exists. A parameter value might still be `null` though.
`not-exists`	Is `true` if at least one of the parameters does not exists.
`is-null`	Is `true` if at least one parameter is `null` or does not exists.
`not-null`	Is `true` if all parameters are not `null` and exist.
`=`	Is `true` if all parameters are equal to the first parameter.
`!=`	Is `true` if all parameters (starting from the second one) are not equal to the first parameter.
`<`	Is `true` if the first parameter is *less* than all the following parameters.
`>`	Is `true` if the first parameter is *greater* than all the following parameters.
`<=`	Is `true` if the first parameter is *less or equal* than all the following parameters.
`>=`	Is `true` if the first parameter is *greater or equal* than all the following parameters.