ESQL Taglib (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Description

The ESQL logicsheet is an XSP logicsheet that performs sql queries and serializes their results as XML. This allows you to work with data from a wide variety of different sources when using Apache Cocoon.

It has a number of important advantages over the old (deprecated) SQL logicsheet and SQL processor. For example, it allows you to mix esql with other logicsheets. It also supports prepared statements (which gives you automatic parameter escaping), multiple encodings in a single query and even multiple resultsets on one statement (if supported from database)!

The name was chosen merely to emphasise the fact that this is an extended version of the old sql logicsheet - esql still uses standard SQL syntax. In fact, it is just a conversion wrapper around your JDBC database driver, so it supports no more and no less SQL syntax than your JDBC driver supports.

2. Installation

Check your cocoon.xconf for this line and add it if it's not already there:

<builtin-logicsheet> <parameter name="prefix" value="esql"/> <parameter name="uri" value="http://apache.org/cocoon/SQL/v2"/> <parameter name="href" value="resource://org/apache/cocoon/components/language/markup/xsp/java/esql.xsl"/> </builtin-logicsheet>

3. Configuration

Map the

http://apache.org/cocoon/SQL/v2

namespace to the esql prefix. Elements in the esql taglib namespace will be interpreted as input to the esql taglib and will be stripped from the output.

This is typically done like this:

<xsp:page language="java" xmlns:xsp="http://apache.org/xsp" xmlns:esql="http://apache.org/cocoon/SQL/v2" > . . . </xsp:page>

3.1. Connection

Esql can use connection pools configured in cocoon.xconf or individually set up connections.

esql:pool gives the name of the connection pool to use.

Individually configured connections use the esql:driver, esql:dburl, esql:username, esql:password tags. Their meaning should be obvious.

3.1.1. Connection Options

Per default, esql will try to switch a connection to *autocommit* mode. This is because it prevents hanging transactions that hold locks and disturb further database accesses. Esql can be forced to not use autocommit, by giving the <esql:autocommit>false</esql:autocommit> nested element to

esql:connection.

Even if a connection is configured with autocommit off in cocoon.xconf, esql will switch autocommit on if not instructed to do otherwise.

Other options like limiting the size of the resultset are discussed below.

4. Usage and Examples

At the moment documentation on esql is quite thin on the ground - however, it should be enough to get you started. In the docs/samples/xsp directory you will find esql.xsp, which is an example of two esql queries, demonstrating "nested" queries and dynamic prepared statements. However, much more comprehensive is the **schema** in esql.xsd which is a formal specification, written in the W3C standard language XML Schema, of every single esql element and attribute. It is fairly human-readable and includes comments for the purpose of each tag.

A fairly common example is to list a query result in a table. Notice that esql:results and esql:no-results are mutual exclusive. So only one of them will be in your XML tree. This example takes a connection from a datasource defined in cocoon.xconf:

<esql:connection> <esql:pool>connectionName</esql:pool> <esql:execute-query> <esql:query>SELECT mycolumn1,mycolumn2 FROM table</esql:query> <esql:results> <table> <esql:row-results> <tr> <td><esql:get-string column="mycolumn1"/></td> <td><esql:get-string column="mycolumn2"/></td> </tr> </esql:row-results> </table> </esql:results> <esql:no-results> Sorry, no results! </esql:no-results> </esql:execute-query> </esql:connection>

4.1. Dynamic Queries

When a query contains dynamic parts, e.g. a value that is to be matched, esql offers two different possibilities to achieve that. First, as the query is really a string, it can be constructed like any other string by concattenation.

<xsp:logic> String orderBy = null; switch(type) { case 1: orderBy = "order by name"; break; case 2: orderBy = "order by salary"; break; default: orderBy = ""; } </xsp:logic> <!-- ... --> <esql:query><xsp:expr>"SELECT name, salary FROM employee "+orderBy</xsp:expr></esql:query>

Note, however, that here any string will be part of the actual statement. In this example it does no harm as the value for the orderBy variable is completely under the control of your code. Any malicious attacker could not inject his or her own code. Thus this technique should not be used when values returned from the client have to be used.

The second variant is to use a PreparedStatement for dynamic parameters. Since the driver is supposed to keep parameters distinct from the statement, no code can be injected this way. In addition, your DBMS puts more effort into optimizing the statement. PreparedStatements are created whenever a <esql:parameter/> tag appears in a query.

<esql:query>SELECT name, salary FROM employee WHERE name=<esql:parameter><xsp:expr>name</xsp:expr></esql:parameter></esql:query>

4.2. Refering to Results

A select query usually returns one ResultSet. This case is handled by the esql:results tag and its content. However, many special cases exist, e.g. an error occurs or an update query is used. Esql provides different tags for these cases.

If an empty result set is returned, the esql:no-results block is used.

4.2.1. Errors

In case of an error, usually signalled by an Exception during setup or execution of a query, the esql:error-results block is evaluated. If no such tag exists, the exception is rethrown and processing is stopped. Withing the tag, esql:get-message, esql:get-stacktrace, and esql:to-string allow access to the error message.

4.2.2. Limiting the number of rows returned

Esql allows to display only a part of the result set using the esql:use-limit-clause. If your DBMS is supported, the DBMS generates only the indicated rows, otherwise a number of rows are skipped and retrieval is stopped after a given number of rows. It works like a fixed-size window to the result set, paging through it.

These parameters are set for a connection.

If the esql:use-limit-clause is empty or set to "auto", esql tries to determine automatically which method to use, depending on the connection URL.

esql:skip-rows and esql:max-rows tags specify how many rows should be skipped at the beginning and how many rows should be retrieved at maximum.

In this context the esql:previous-results and esql:more-results blocks hold code and content that is only used if this sliding window has previous or following windows.

```
<esql:connection> <esql:pool>connectionName</esql:pool> <esql:execute-query>
<esql:query>SELECT mycolumn1,mycolumn2 FROM table</esql:query>
<esql:use-limit-clause>auto</esql:use-limit-clause>
<esql:skip-rows><xsp:expr>skiprows</xsp:expr></esql:skip-rows>
<esql:max-rows>10</esql:max-rows> <esql:results> <table> <esql:row-results>
<esql:previous-results>previous rows available</esql:previous-results>
<esql:more-results>more rows available</esql:more-results> <tr> <td><esql:get-string
column="mycolumn1"/></td> <td><esql:get-string column="mycolumn2"/></td> </tr>
</esql:row-results> </table> </esql:results> <esql:error-results>An error
occurred</esql:error-results> <esql:no-results> <p>Sorry, no results!</p> </esql:no-results>
</esql:execute-query> </esql:connection>
```

4.2.3. Updates

In JDBC, updates, inserts, and deletes are "update queries". For those, no results are available but an update count is returned, indicating, how many rows were affected.

Code or content that depends on this has to be placed inside the esql:update-results tag. It is used whenever at least one row was affected. The update count can be accessed through the esql:get-update-count tag.

If no rows where affected, the esql:no-results block is used.

```
<esql:connection> <esql:pool>connectionName</esql:pool> <esql:execute-query>
<esql:query>update table set price=price*1.17</esql:query> <esql:error-results>An error
occurred</esql:error-results> <esql:update-results> <esql:get-update-count/> prices
adjusted. </esql:update-results> <esql:no-results> <p>Sorry, no prices adjusted!</p>
</esql:no-results> </esql:execute-query> </esql:connection>
```

4.3. Groups

For more complex lists, often nested queries are needed. Esql allows arbitrary nesting of queries. However, you can do table joins and then insert a header whenever a "watched" column value changes using the <esql:group/> and <esql:member/> tags. It follows the nesting ideology of <xsp:logic> ... <xsp:content></></>You can nest <esql:group> and <esql:member> indefinately. group-on can be an attribute of group or a text node. The value of the text node has precedence over the attribute. The value can be the column name or the column number.

<esql:execute-query> <esql:query> select committeeName, title, firstName, middleName, lastName, suffix, status from committeeMember left join directoryInformation using(userid) left join committee on committee.id=committeeMember.committeeid order by committeeName asc </esql:query> <esql:results> <esql:row-results> <esql:group group-on="committeeName"> <h2><esql:get-string column="committeeName"/></h2> <esql:member> <esql:get-string column="title"/> <esql:get-string column="firstName"/> <esql:get-string column="middleName"/> <esql:get-string column="lastName"/> <esql:get-string column="suffix"/> </esql:member> </esql:group> </esql:row-results> </esql:results> </esql:execute-query>

One important limitation of the grouping feature is, that *no access to a column may appear after closing a group.* The value will belong to the following row or cause an error if no next row exists. If this is needed, consider swapping columns using XSLT or embedded JAVA. Hence the following example is illegal:

<esql:execute-query> <esql:query> select committeeName, committeeTitle, title, firstName, middleName, lastName, suffix, status from committeeMember left join directoryInformation using(userid) left join committee on committee.id=committeeMember.committeeid order by committeeName asc </esql:query> <esql:results> <esql:row-results> <esql:group group-on="committeeName"> <h2><esql:get-string column="committeeName"/></h2> <esql:member> <esql:get-string column="title"/> <esql:get-string column="firstName"/> <esql:get-string column="middleName"/> <esql:get-string column="lastName"/> <esql:get-string column="suffix"/> </esql:member> </esql:group> <esql:get-string column="committeeTitle"/><!-- illegal !! --> </esql:row-results> </esql:results> </esql:execute-query>

4.4. Stored Procedure Support

In order to use stored procedures replace <esql:query/> with <esql:call/>, use either DBMS specific syntax or JDBC escape syntax {? = foo(?)}. If your jdbc driver requires to use the executeQuery() method instead of the execute() method (like e.g. INFORMIX does), set needs-query="true" attribute.

If a result set is returned through the (only) return parameter of a stored procedure, e.g. resultset-from-object="1" as attribute to <esql:call/>to automatically use this result set. For a more general alternative see further below.

Parameters for a stored procedure call may be of direction="in|out|inout" with the usual JDBC meaning. In addition a type needs to be supplied for "out" and "inout" parameters. This would be the same "XXX" as used in a get-XXX JDBC-method call. Alternatively, you can use a fully qualified field name, e.g. "java.sql.Types.CHAR"

<esql:call-results/> (child of <esql:execute-query/>) may contain code that will always be executed whether the query returned a result or not. For example most stored procedures will not return a result set but several out parameters.

All <esql:get-xxx/> tags accept a new attribute from-call="yes" to indicate that the value is retrieved from the CallableStatement rather than the current ResultSet. Obviously, this only works after a call to a stored procedure.

Retrieve a ResultSet from any column and use it like the result of a nested query with the esql:use-results tag. It behaves exactly like nesting queries. Thus the ancestor attribute can be used to access e.g. the original query.

Example:

<esql:call>{? = foo(<esql:parameter direction="in" type="Int"><xsp:expr>1</xsp:expr></esql:parameter>)} </esql:call> <esql:call-results> <esql:use-results> <esql:result><xsp:expr>(ResultSet)<esql:get-object column="1" from-call="true"/></xsp:expr></esql:result> <esql:results> <esql:row-results> <esql:get-string column="1"/> </esql:row-results> </esql:results> </esql:use-results> </esql:call-results>

Example:

<esql:query>select name, list_of_aliases from table</esql:query> <esql:results> <esql:row-results> <esql:get-string column="name"/>: <esql:use-results> <esql:result><xsp:expr><esql:get-array column="list_of_aliases"/>.getResultSet()</xsp:expr></esql:result> <esql:results> <esql:row-results> <esql:get-string column="1"/> </esql:row-results> </esql:results> </esql:use-results> </esql:row-results> </esql:results>

4.5. Multiple Results

If multiple results are returned from a stored procedure or a query, the esql:results block is reused. However, it is supported to have different blocks for each result. Since a result can either be a ResultSet or an UpdateCount, both are counted independently. The nth ResultSet will be handled by the nth esql:results block, or - if there are fewer blocks - the last one.

The same holds true for esql:update-results and esql:no-results blocks as well.

Support for multiple results is not widely available with DBMSs. Therefore support is disabled by default. Use the <esql:allow-multiple-results>yes</esql:allow-multiple-results> parameter to the <esql:connection/>.

Example: Suppose stored procedure bar returns an update count, another update count, a result set, an update count, and a last result set.

<esql:call>{? = bar(<esql:parameter direction="in" type="Int"><xsp:expr>1</xsp:expr></esql:parameter>)} </esql:call> <esql:results> <!-- this is used for the first result set --> </esql:results> <esql:results> <!-- this is used for the second and all following result sets --> </esql:results> <esql:update-results> <!-- this is used for the first update count --> </esql:update-results> <esql:no-results> <!-- this is used for the first update count --> </esql:no-results> <esql:update-results> <!-- this is used for the second and all following update counts --> </esql:update-results> <esql:no-results> <!-- this is used for the second and all following update counts --> </esql:no-results>

The ultimate reference, is of course the source code, which is an XSLT logicsheet contained in the file src/org/apache/cocoon/components/language/markup/xsp/java/esql.xsl

Of course, we would be very grateful for any improvements on this documentation or further examples - please send them to users.at.cocoon.apache.org!

5. Template Descriptions

Tag	Description
esql:row-results//esql:get-columns	results in a set of elements whose names are the names of the columns. the elements each have one text child, whose value is the value of the column interpreted as a string. No special formatting is allowed here. If you want to mess around with the names of the elements or the value of the text field, use the type-specific get methods and write out the result fragment yourself. For Cocoon 2 only, this outputs structured types as well. Here sql-list or sql-set contains several sql-list-item or sql-set-item element that again contain the actual data.
esql:row-results//esql:get-string	returns the value of the given column as a string
esql:row-results//esql:get-date	returns the value of the given column as a date. if a format attribute exists, its value is taken to be a date format string as defined in java.text.SimpleDateFormat, and the result is formatted accordingly.
esql:row-results//esql:get-time	returns the value of the given column as a time. if a format attribute exists, its value is taken to be a date format string as defined in java.text.SimpleDateFormat, and the result is formatted accordingly.
esql:row-results//esql:get-timestamp	returns the value of the given column as a timestamp. if a format attribute exists, its value is taken to be a date format string as defined in java.text.SimpleDateFormat, and the result is formatted accordingly.
esql:row-results//esql:get-boolean	returns the value of the given column as true or false
esql:row-results//esql:get-double	returns the value of the given column as a double. if a format attribute exists, its value is taken to be a decimal format string as defined in java.text.DecimalFormat, and the result is formatted accordingly.
esql:row-results//esql:get-float	returns the value of the given column as a float. if a format attribute exists, its value is taken to be a decimal format string as defined in java.text.DecimalFormat, and the result is formatted accordingly.
esql:row-results//esql:get-int	returns the value of the given column as an integer
esql:row-results//esql:get-long	returns the value of the given column as a long
esql:row-results//esql:get-short	returns the value of the given column as a short
esql:row-results//esql:get-ascii	returns the value of the given column as a clob

esql:row-results//esql:get-object	returns the value of the given column as an object
esql:row-results//esql:get-array	returns the value of the given column as an java.sql.Array. This is frequently used for collection datatypes like lists, sets, bags etc.
esql:row-results//esql:get-struct	returns the value of the given column as a java.sql.Struct. This is frequently used for row types.
esql:row-results//esql:get-xml	returns the value of the given column interpreted as an xml fragment. The fragment is parsed by the default xsp parser and the document element is returned. If a root attribute exists, its value is taken to be the name of an element to wrap around the contents of the fragment before parsing.
esql:results//esql:get-column-count	returns the number of columns in the resultset.
esql:row-results//esql:get-row-position|esql:results	returns the position of the current row in the result set
esql:row-results//esql:get-column-name	returns the name of the given column. the column must be specified by number, not name.
esql:row-results//esql:get-column-label	returns the label of the given column. the column must be specified by number, not name.
esql:row-results//esql:get-column-type-name	returns the name of the type of the given column. the column must be specified by number, not name.
esql:row-results//esql:is-null	allows null-column testing. Evaluates to a Java expression, which is true when the referred column contains a null-value for the current resultset row
esql:error-results//esql:get-message	returns the message of the current exception
esql:error-results//esql:to-string	returns the current exception as a string
esql:error-results//esql:get-stacktrace	returns the stacktrace of the current exception
esql:results/esql:get-metadata	returns the metadata associated with the current resultset
esql:results/esql:get-resultset	returns the current resultset
esql:group	Allows header elements around groups of consecutive records with identical values in column named by @group-on. Facilitates a single query with joins to be used in lieu of some nested queries.
esql:member	Used in conjunction with and nested inside esql:group. Formatting for individual records goes within esql:member. Header stuff goes in between group and member.
@*|node()	used internally to determine which column is the given column. if a column attribute exists and its

	value is a number, it is taken to be the column's position. if the value is not a number, it is taken to be the column's name. if a column attribute does not exist, an esql:column element is assumed to exist and to render as a string (after all of the xsp instructions have been evaluated), which is taken to be the column's name.

1. Comments

add your comments