

Overview

Certain functionality of xReporter is reusable outside the context of the xReporter server itself. This page will contain those as subprojects.

Feedback about any of the projects listed below is welcome on the xReporter mailing list.

GroupingTransformer

A Cocoon transformer that can perform grouping and summary calculation on table-like data. [Click here for more information...](#)¹

Expression language interpreter

The expression language interpreter included with xReporter can be reused as a standalone library. It is not (yet) available as a separate download. To use it, first check out the "xreporter" module from SVN (see [project details](#)²). Then build it by executing the following command in the xreporter directory:

```
For Windows: build.bat standalone-jars
For Linux:   ./build.sh standalone-jars
```

Now the following jar will be created: build/standalone/xreporter-expression.jar. This is the only jar needed to use the expression library. Sample code of how to use it can be found in the file ExpressionTest.java and other places throughout xReporter.

Grouping library

The code that performs the grouping logic (and the calculation of the summaries) can be reused as a standalone library outside xReporter (as demonstrated by the Cocoon GroupingTransformer). To obtain it, follow exact the same instructions as for the expression language interpreter. After building, the required jars are located in the build/standalone directory: xreporter-expression.jar and xreporter-grouping.jar.

There's no specific documentation yet about how to use this library, but both the GroupingTransformer and the code in xReporter serve as examples. Basically you need to create and fill a "Table" object, then create the grouping structure by using the "Grouper" class and passing it an array of GroupDefinition objects (the first element in this array should be an instance of TopLevelGroupDefinition). This will give a tree of "Group" objects, with each group containing a startIndex and endIndex pointing to where the group starts/ends in the Table structure. Then use the "CalculateSummariesVisitor" to calculate the summaryfields in each Group, and that's it. You could then use a custom visitor to serialize the grouping structure.

-
1. [daisy:69-cd \(Introduction\)](#)
 2. [daisy:66-cd \(Getting involved\)](#)