

Data Sources

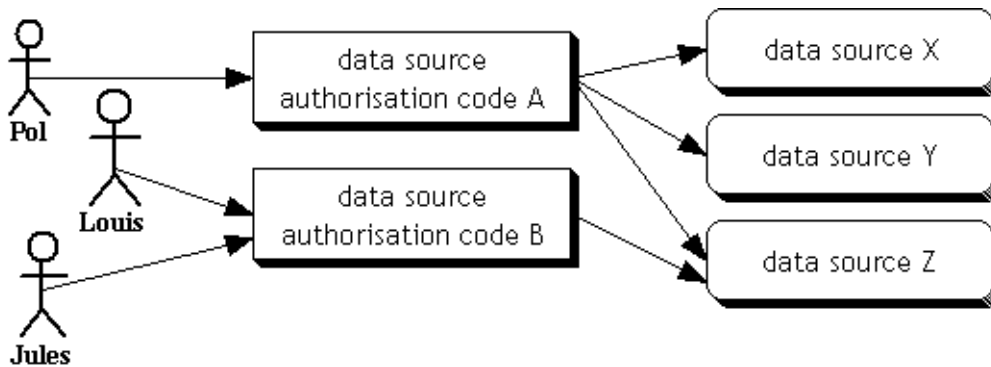
Concepts

A data source in xReporter describes both the connection parameters and some metadata concerning the database such as a name and description. (The word " data source" in xReporter thus covers somewhat more than a Java data source).

Multiple data sources can be defined. xReporter can handle large numbers of different data sources.

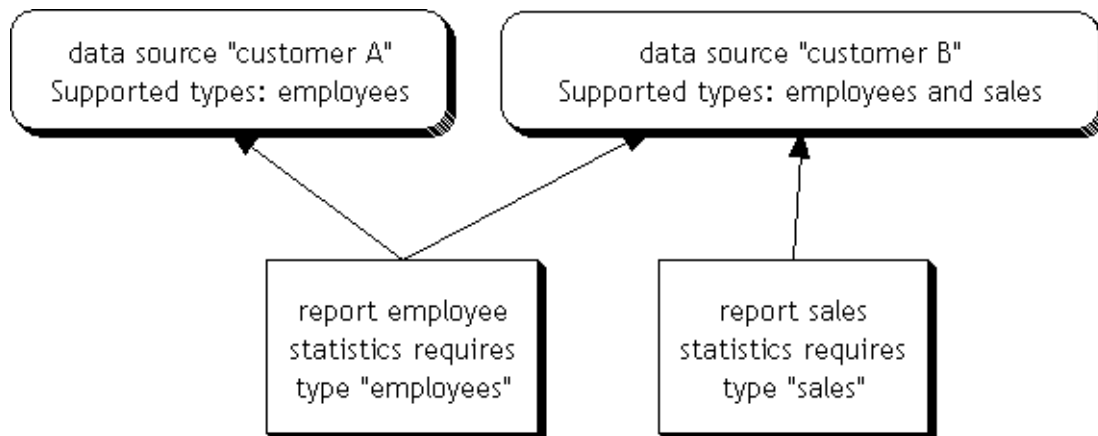
Authorisation codes

The data sources a user is allowed to access are limited using a so-called "data source authorisation code" (DAC). Each user is assigned one DAC, and each DAC is associated with a set of data sources. The figure below illustrates this principle.



Types

To each data source one or more types can be assigned. A type identifies the kind of data present in the data source. Each report will require one specific type of data source, and will hence only be executable on data sources that implement this type. The figure below illustrates this principle.



Practically this means that after a user has selected a data source, only reports compatible with the selected data source will be shown (i.e. reports that require a type supported by the data source).

Dialects

In xReporter's report definitions you are free to define whatever SQL you can imagine, and thus you can also make use of database-specific features. In some cases you might want to use a specific report with different kinds of databases (Oracle, MySQL, ...) and still use database specific features. This is possible by assigning a "dialect" to each data source. In the report definition you can then specify different SQL statements for each dialect.

Sort codes

To each data source two sort codes can be assigned. The sort codes determine the order in which the data sources will be shown to the user. This provides more flexibility than simply sorting on the name of the data sources.

Configuration of the data sources

The data sources are configured in an XML file. The name and location of this file can be freely chosen and must be configured in the config.xml. Below is an example of a data source configuration file. It contains just one data source.

```
<datasources>
  <datasource id="president">
    <name>presidents.name</name>
    <description>presidents.description</description>
    <sortcode1>president</sortcode1>
    <sortcode2></sortcode2>
    <catalogs>
      <catalog>foo/bar</catalog>
    </catalogs>
    <supported-types>
      <type>president</type>
    </supported-types>
    <dialect-name>mysql</dialect-name>
    <connection-params>
      <url>jdbc:mysql://example.com/presidents</url>
      <user>xreporter</user>
      <password>xreporter</password>
    </connection-params>
  </datasource>
</datasources>
```

The element `datasources` is the document element. It contains one or more `datasource` elements. Each `datasource` element must have an `id` attribute, whose value must be a unique identification for that data source. `id`'s should always start with a letter, and can then contain letters, numbers, and the characters `.` (dot), `-` (dash) and `_` (underscore).

Using the `name` and `description` elements you can specify a short and a long text which will be shown to the user when selecting a data source. These elements do not really contain this text, instead they contain resource bundle keys. Such a key is used, together with the user's language, to lookup the text from a resource bundle. A resource bundle key can be chosen freely, but may not contain spaces, or the characters `'=`' and `'.'`.

The `sortcode1` and `sortcode2` elements contain the the sort codes we talked about earlier. These elements are optional. As default an empty string will then be used as sort code.

The `catalogs` element allows grouping `datasources` into catalogs, similar to the report catalogs. The `catalogs` element is completely optional. The `catalogs` element can contain one or more `catalog` child elements. The

catalog element should contain a slash-separated path specifying the catalog. The strings in the path are used as resource bundle keys to lookup the label for the catalog.

The connection-params element contains the JDBC connection parameters for the data source. This element should always contain a child element called 'url'. Usually there will also be child elements for user and password. Other elements can be added according to the properties supported by the driver.

Configuration data source authorisation codes (DAC's)

The DAC's are configured in an XML file. The name and location of this file can be chosen freely and must be configured in the config.xml. Below is an example of a DAC configuration file.

```
<datasource-authorisation-codes>
  <datasource-authorisation-code id="A">
    <datasource id="ds1"/>
    <datasource id="ds2"/>
    <datasource id="ds3"/>
  </datasource-authorisation-code>

  <datasource-authorisation-code id="B">
    <datasource id="ds1"/>
    <datasource id="ds4"/>
  </datasource-authorisation-code>
</datasource-authorisation-codes>
```

The format of this file does not need much explanation: the datasource authorisation code A allows access to the datasources ds1, ds2 and ds3. The DAC B allows access to ds1 and ds4. The id's ds1, ds2, ds3 and ds4 should of course correspond to id's that are assigned to data sources.

Connection pools

By default xReporter will pool database connections. The connection pool implementation is specific for xReporter. A special feature is that the connection pools for all datasources are managed by one management thread, while other implemenations typically use one thread for each connection pool. Since xReporter has been developed to use in a situation where hundreds of datasources could be used, this was unacceptable. Other properties of the connection pool are that it has no specific upper or lower boundaries: the upper boundary is theoretically unlimmited, while the lower boundary is zero. This means that if connections are not used for a while, the pool becomes empty again.

To disable the pooling of connections you can edit the file assembly.xml (found at the same location as config.xml) to change the implementation of the connectionprovider to the "NonPoolingConnectionProviderManager" (which is commented out in that file).