

WINDWARD STUDIOS

Windward Reports 6.0

Data Source Guide

WINDWARD REPORTS

Data Source Guide

© 2004 – 2008 Windward Studios, Inc.
2945 Center Green Court South
Boulder, CO 80301
Phone 303.499.2544 • Fax 303.554.5636

Last revised 28 January 2008
Revised through version 6.0.2

This document Copyright © 2004 – 2008 by Windward Studios, Inc.
All Rights Reserved

Table of Contents

Windward Reports	2
About This Guide	2
Providing Feedback	3
DataSourceProvider	4
The Architecture	6
Your Basic Design	7
Step 1	8
Handling a <i>forEach</i>	8
DataSourceProvider	8
parse and SimpleEvaluator	9
DataSourceNode	9
Handling an <i>if</i>	10
Handling an <i>out</i>	11
Testing	11
Step 2	12
Everything Else	13
Package Naming	13
Technical Support	13
Index	15

Windward Reports

An introduction to Windward Reports.

Windward Reports™ is the only .NET report engine that uses Microsoft Word™ or Excel™ as its layout tool – so creating reports is fast & easy, for both technical and non-technical users.

With the robust, plug 'n' play Windward Reports engine, you've got an elegant, flexible and very affordable reporting solution.

It's as easy as 1-2-3:

1. Design your report template using MS-Word (or any RTF-capable editor) or Excel. Templates can be created in RTF WordML (XML), DOCX, or XLSX.
2. Send your XML and/or SQL data source and the report template to the Windward Reports engine.
3. Specify your desired output format: PDF, DOCX, XLSX, RTF, HTML, TXT, WordML (Word's XML format), SpreadsheetML (Excel's XML format), and XLS.

That's it! There's no learning curve or added expenses for your layout tool, and the intuitive JSP-like formatting syntax lets you create everything from simple everyday reports to complex, enterprise-wide analyses..

About This Guide

This guide is for the programmer writing a custom data source. A custom data source provides data to Windward Reports from a proprietary system. This document assumes the reader is an experienced Java programmer. It also assumes the reader is familiar with Microsoft Word.

Providing Feedback

We have tried to make our documentation as clear and complete as possible. But only you can tell us what we missed. So if you are new to Windward Reports, please email docs@windward.net with your suggestions for improvement... especially when you find yourself wishing for something that would save you a lot of time.

Thank you for your feedback, - The Windward Documentation Team

DataSourceProvider

Your own data.

Windward Reports provides the ability to supply data for a template from virtually any source. Included with Windward Reports are classes that supply data from an XML stream or a SQL database. However, if you need to generate reports and the necessary data is not available as an XML stream or from a SQL database, then you can write your own data source.

The `DataSourceProvider` and associated classes provide all data source specific methods necessary for substituting data into a report. It has an XML-like flavor but does not require the data source to be XML or SQL.

The `net.windward.datasource` package contains the interfaces that must be implemented by a data provider. There is no code in this package other than a single exception class.

You will find that the data provider implementation requires some code that generally could be handled by Windward Reports and not pushed down to the provider. However, for the rare cases where the data provider needs to handle an issue in a non-standard way, all work was pushed down to the data provider level. For example, while the `out` tag has a default value, it's up to the data provider to use that value if a node does not exist. This way the default value can be a key to the actual default value, instead of being limited to being the literal string.

This interface and all implementations provided by Windward Studios are copyrighted by Windward Studios. You can write your own implementation of these interfaces with two restrictions:

- This interface can only be used with Windward Reports. It cannot be used with other programs.
 - Any implementation of this interface must be made publicly available for free to anyone else who wishes to use it. This includes posting the code at
-

www.windwardreports.com. This restriction is not required for code that interfaces with a company's internal API.



The Architecture

The Big Picture.

Originally, Windward Reports' data source only used xpath to access an XML file. Although a SQL data source has been added, the basic architecture is best thought of in terms of an XML DOM (Document Object Model).

The key class is `DataSourceNode`. `DataSourceNode` represents a node in the DOM or a `ResultSet` in the case of SQL/JDBC. Everything that occurs in the data source is in relation to a `DataSourceNode`. If no specific node is set, then it is the root node, or in the case of a SQL data source, a `ResultSet` composed of all records from all tables in the database.

An XML file always has the concept of the current node. This is not the case with a SQL database - there is no current `ResultSet`. A `ResultSet` is only generated by the *query* and the *foreach* tags. Therefore, all other tags only make sense if they use the **var** attribute from a *query* or *forEach* to identify the `ResultSet` they are to be applied to. A tag not identifying a `ResultSet` will throw an exception. Refer to the *Template Creator's Guide* for specific information on the report tags.

The second key concept is nested *forEach* loops. On every call in the `DataSourceNode` method, a stack of `DataSourceNode`'s are passed in. There is one node for each nesting level of *forEach* loops that the method is called from (within the loop).

There is always at least one element in the stack as `stack[0]` is `DataSourceProvider.getRootNode()`. So you can always use `stack[stack.length-1]` as the context you are presently operating in. The stack is provided as there are **select** commands that are specified to operate on a level other than the innermost level.

A `DataSourceNode` is only created in one of two ways. First, `DataSourceProvider.getRootNode()` returns the root node for the data. Second, when a

forEach loop is hit, `DataSourceNode.iterator()` is called. `DataSourceIterator().next()` returns the `DataSourceNode` for that iteration. Therefore, the only `DataSourceNodes` that exist are the stack of nodes for the nested *forEach* loops the program is presently processing.

In addition, a given node will only have a single `DataSourceIterator` in use at a time. A node will never have two iterators in use at the same time. If you have a stack of four nodes, each node (except the root) will have an iterator for a total of 3 iterators in use. But no single node will have more than 1 iterator.

Finally, your methods will only be called as required by the tags in a template. If you have no *out* tag, the `getOut` method will never be called. The only method that is not mapped directly to a tag is the `DataSourceNode.isExistingNode()` method. This can be called before any tag method is called. (Note: the *if* tag calls `isIf()`, not `isExistingNode()`.)

Your Basic Design

You have two key questions you must decide before writing any code. First, what is a node? Generally this is easy to answer. For XML it is a DOM node. For SQL it is a `ResultSet`. Your entire design, your tag **select** attributes— everything revolves around this question.

Your second key question is how to define a value, for example, what to display for an *out* tag. For XML, this is the text representation of a node. For SQL, this is a field in a row in a `ResultSet`. This has a substantial effect on your implementation and what is allowed in your tags. For example, an XML tag can have `select="/item/name"` but a SQL tag can only have `select="{name}"`; it would make no sense (the way `JdbcDataSource` is implemented) to have `select="select name from people"` as the **select** for an *out*.

This second question affects your initial implementation. If you require `{name}` for the *out* tag, then you need to implement that mapping as part of step one. However, most of the code for this can be copied from `JdbcDataSource.java`.

Step 1

Getting the basics in place.

To implement the basic functionality, all you want to handle is *out*, *if*, and *forEach*. Most of the work is handling *forEach*.

Handling a *forEach*

It is tempting to handle just *out* and *if* in step 1, but *forEach* affects your entire design and skipping it now means a lot of repeat work later.

Your best bet is to look at `Dom4jDataSource.java` (723 lines) or `JdbcDataSource.java` (850 lines), depending on which model is closer to your proprietary data. Complete source for each is in `WindwardReports.jar`, as are all Java files for `net.windward.datasource` and all sub-packages. They are well commented, and the code at the end of each for substituting variables can often be copied across to your own implementation.

Do not worry in step 1 about implementing `#{variables}`, unless you have a JDBC-type implementation where you must have it for the *out* tag.

DataSourceProvider

Create a constructor that connects to your data. The parameters passed to this and how it connects are totally dependent on your data source and is up to you. You can create several constructors if you wish. You will use these constructors to pass `DataSourceProvider` object(s) to Windward Reports to create a report.

Make sure that the only exceptions that your constructor and any other methods in your data source code throw are run time exceptions and `DataSourceException`.

`DataSourceException` can be passed an inner exception, but the exception thrown must be `DataSourceException`.

Implement the `close()` method as necessary for your data source. The `dump()` method is for your own use only, so implement it as you see fit. For the `setMap()` method, just save the map inside your `DataSourceProvider` object.

parse and SimpleEvaluator

In both `Dom4jDataSource.java` and `JdbcDataSource.java` you will find the private method `parse`. This method and the sub-methods it uses are about 150 lines of code and you should copy one of them to your implementation.

This code (which also uses `SimpleEvaluator`) could have been placed outside of the data source package as it is common to both the XML and SQL implementations, despite slight differences. This code was left in the data source implementation in case you need to handle the **select** attribute parsing in a different manner.

The best case scenario is that you can copy it across. And even if you do have to write your own parse code (or add to the existing code), you should use the `SimpleEvaluator` class to handle any evaluations needed of **select** attributes.

DataSourceNode

Your `DataSourceNode` class will generally work best if it is an inner class inside your `DataSourceProvider` class. This gives it access to all member variables in the `DataSourceProvider` class.

First add do nothing methods for:

```
addQuery( )
getBitmap( )
getEscape( )
getHtml( )
getImport( )
getLink( )
```

Second, implement `getIterator()`. This requires implementing the `DataSourceIterator` class. This should be an inner class of `DataSourceNode`. Your `DataSourceNode` class can hold the iterator it is presently using as a variable, because it will only be asked for one iterator at a time. This is returned by the method `getIterator()`.

All you really need to implement at first is the constructor, `hasNext()`, and `next()`. The rest can have hard-coded values. The `index`, `count`, `first`, & `last` properties are only called if the associated `status.index`, etc. values are in a **select**.

Also do not worry about implementing steps (only returning every Nth row). You do not ever need to implement steps unless you are going to use them. But if you don't, make sure you throw an exception if you get a step value other than 1.

Keep in mind that `DataSourceIterator` has a very simple job.

- Determine if there is another row to return.
- Return the next row as a `DataSourceNode`.

Handling an *if*

The good news is you now have the hardest part behind you – you are handling *forEach* loops and you are handling variable substitution. Handling an *if* is the only remaining part of any difficulty and is easier.

You now need to implement `isIf()` and `isExistingNode()`. They are very similar but there are differences. First implement `isExistingNode()`. This returns if a node exists and has no rules or attributes that impose special conditions. This method is called a lot so you want it to be fast. This is also the one method that can be called before any tag, is not mapped to any specific tag, and can be called at any time.

Keep in mind that if multiple data sources are applied to a template, `isExistingNode()` is used to determine if a tag's node is for the data source it is on. If the node does not exist, and it is not the last data source to be applied, then the tag is left as is for the next data source.

This means `isExistingNode()` can get passed `select` statements that are totally illegal for your data source and may cause it to throw an exception or have unwanted side effects. You must return false in these cases. (Test `isExistingNode()` by passing it `xpath` and `SQL select=""` tags.)

Next, implement the `isIf()` tag. Keep in mind that this can be either a `test=""` or `select=""` `if`, and they are generally two very different tests. You need to implement both. You also need to implement the empty test appropriately on a `select` test. What constitutes empty in the case of your data source is a decision you need to make.

If you are dying to test at this point, you can now handle a template that has just *forEach* and *if* tags in it. However, we strongly recommend that you get through the *out* tag first.

Handling an *out*

Handling an *out* is relatively easy. Return the value of the data as identified by the **select**. The only issue worth noting is to return null if you have no value for the tag and there is no default (as opposed to returning an empty string).

There is one other critical point for the *out* tag. If the *out* is a value="foobar" instead of a select="foobar", there is no way to determine if the value is for the data set presently being processed. Therefore, the rule is if after parsing and formatting, there is still a "\${" in the string, then the assumption is that this tag is not for this data source and a null is returned.

You also will want to use `OutTag.formatTag()` which takes care of all formatting of the output string (used primarily for numbers and dates). This method also handles using the default value if the passed in value is empty.

Testing

You are now ready to test. First create a simple template with a single *forEach*, a single *if*, and two or three *out* tags. Run it through and debug as needed.

Next, create a nested test with an inner and outer loop. Test where each loop has 0, 1, 2 and 3 iterations. It is critical that you perform this test, as this is where you are most likely to find subtle problems.

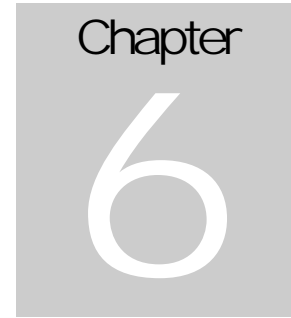
Finally create a template with SQL and xpath tags in it, too. Run the test where your data source is not the lastData. It is fine if there is no lastData passed in and the SQL and xpath tags remain. This test is to ensure your `isExistingNode()` can handle SQL and xpath queries.

Step 2

Finishing the job.

The rest is easy. If you did not implement `${variable}` substitution in step 1, do this first. Then implement the remaining methods, which will all closely resemble the `getOut()` method.

Finally, test, test, test. There are unit tests in `net.windward.datasource.dom4j.test` and `net.windward.datasource.jdbc.test` and you are encouraged to either borrow from them and/or implement similar tests. If your data source code has a bug in it, it will affect all of Windward Reports.



Everything Else

In Closing...

First of all, if there is anything that you think we could add to make writing a data source provider easier, please let us know.

Package Naming

If your data source is for proprietary data, please do not use the `net.windward.*` namespace for your class. There is no need for it to be in the same namespace.

If you are creating a data source for a public data API, please check with us and we will assign you a package in the `net.windward.datasource.*` namespace.

Technical Support

Online technical support can be reached 24 hours a day on our support forums. Please go to <http://ohana.windwardreports.com/> to access them. The following two support options are only available to users who purchased a support contract:

- E-mail technical support can be reached from 8:00 AM to 5:00 PM, Mountain Time. E-mail questions will be replied to within 30 minutes. Please send e-mail to support@windward.net.
 - Technical Support can be reached from 8:00 AM to 5:00 PM, Mountain Time, at 1-303-499-2544.
-



Index

- Architecture, 6
 - Basic Design, 7
 - close() method, 9
 - constructor, 8
 - custom data source, 2
 - DataSourceNode, 9
 - DataSourceProvider, 4, 8
 - dump() method, 9
 - getIterator method, 9
 - Handling a *forEach*, 8
 - Handling an *if*, 10
 - Handling an *out*, 11
 - HTML, 2
 - implementing *\$(variables)*, 8
 - implementing *out, if, forEach* functionality, 8
 - isExistingNode method, 10
 - isIf method, 10
 - J2EE, 2
 - output formats, 2
 - Package Naming, 13
 - parse method, 9
 - PDF, 2
 - Providing Feedback, 3
 - RTF, 2
 - setMap() method, 9
 - SimpleEvaluator, 9
 - single exception class, 4
 - SpreadsheetML, 2
 - SQL, 2
 - SQL data source, 4
 - Technical Support, 13
 - Testing, 11
 - TXT, 2
 - Windward Studios copyright restrictions, 4
 - WordML, 2
 - XLS, 2
 - XML, 2
 - XML data source, 4
-