

WINDWARD STUDIOS

---

Windward Reports 6.0

# Bean Guide

WINDWARD REPORTS

# Bean 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
BeanProvider	4
The Architecture	5
The Functionality	6
Properties	6
Implementation	7
Stateful and Stateless Methods	7
Everything Else	9
Package Naming	9
Technical Support	9
Appendix A: Implementing the Bean Provider Interface	10
Appendix B: Using a Bean in a Template	12
Appendix C: Using Beans when Calling WR	13
Index	16

# Windward Reports

*An introduction to Windward Reports.*

**W**indward 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. For details, refer to the *Programmer's Guide* for information on incorporating Windward Reports into your system, and the *Template Creator's Guide* for details on how to insert the report tags.

## About This Guide

This guide is for the programmer writing a Windward Reports bean.

---

A bean provides you with the capability to add two sets of functionality to Windward Reports:

- To provide the final text and formatting for any *out* or *function* tag. Note that this text can be totally independent of the text the *out* or *function* tag would normally produce.
- To override any *if* tag's true/false value and to end a *forEach* loop early.

Please note that beans are not a means to provide data to the report. If you wish to do that, write a data source (please see the *Data Source Guide*). Beans are designed as a very lightweight method of providing some commonly-needed functionality.

Windward Reports includes two beans available for your use. Their source code is included in the WindwardReports.jar file. The beans are:

- RedNegative which will display positive numbers in black, and negative numbers in red.
- RunningTotal which will keep a running total of numbers passed through it, and that total can then be displayed at any point in the report.

Finally, there are some special methods in each bean which exist solely for programs that access the Windward Reports Report Server over the network.

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](mailto: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

---

## BeanProvider

*Fine tune the output.*

Windward Reports gives you substantial control over the data source when formatting your report. However, you may find that you need additional control not provided with the standard tags. In this case, you may find that writing a Windward bean (a BeanProvider implementation) will give you the functionality you need.

The BeanProvider package was designed to be very lightweight and to ride on top of the existing tags. This means that writing a bean is usually easy – but it also means there are limits to what you can accomplish with a bean. If you find a bean too limiting, you should probably be writing a DataSourceProvider.

This interface, classes, 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. Your code is yours. But the interfaces we have defined are ours.

## The Architecture

### *The Big Picture.*

There are two types of beans—stateful and stateless. If you are calling Windward Reports directly from Java, this difference is usually not important. However, if your beans are loaded by the Report Server to be used for reports generated by calls from a client program over the network, then this is usually an important decision.

A single stateless bean instance will be called while creating multiple reports. These multiple calls may be intermixed between reports. For example, it may call the bean twice for report A, once for report B, two more times for report A, then again for report B.

A stateless bean is much more efficient. It works well when all necessary information for the bean is passed in on the call to the bean. For example, the bean `RedNegative` is stateless because its actions are based on the string passed in to it. Therefore it can handle calls from numerous reports simultaneously.

The calls `StatelessBeanProvider.serverSetup()` and `StatelessBeanProvider.serverTeardown()` are only called when a stateless bean is used by the report server. They are not called if a bean is used when directly calling Windward Reports from Java.

A stateful bean will be created when the generation of a single report begins. It will be called for that single report only. Therefore if two reports both need an instance of the same stateful bean at the same time, two objects will be created and each report will use its bean instance. `RunningTotal` is an example of a stateful bean.

The `StatefulBeanProvider` methods are called both if the bean is called via the report server and if the bean is used via direct access from Java.

In the case of a direct call from Java, a bean exists only for the life of generating that one report. The bean is viewed in this case as being “owned” by the calling program and after Windward Reports returns the generated report, it will not touch that bean again.

---

You must always implement either a `StatefulBeanProvider` or a `StatelessBeanProvider`. The `BeanProviderImpl` based classes are provided as a do nothing implementation of each interface, but do not need to be used.

If you need to create a bean based on an existing class of yours, that class should implement the appropriate `BeanProvider` interface. If your bean does not need to inherit from a class of yours, then you should create your bean as extending from one of the `BeanProviderImpl` classes.

## The Functionality

You have two basic functionalities available in beans. The first is when using a bean in combination with an *out* or *function* tag. In these cases, you are passed the text these tags would place in the report, if the bean did not exist. The bean can then change the text and apply some basic formatting to the text.

The *out* and *function* methods return a `BeanResult` object. This object has numerous values that you can set. Any value that is null is ignored upon return. So null is the means by which you state that the field in the `ReturnResult` is not being set by your bean.

The second is for the *if* and *forEach* tags. In this case, the bean can return a true or false value. In the case of the *if* tag, it overrides the boolean return of the tag. In the case of a *forEach*, it functions as an early end to the loop.

## Properties

If you have a tag with a `bean='name'` attribute, then Windward Reports accepts any number of additional attributes in that tag. These additional attributes can then be accessed in the bean via the passed in tag object. In this manner you can have your beans perform differently based on bean defined attributes set in the tag. The `RunningTotal` bean makes use of this capability.

---

## Implementation

### *Writing the Code.*

Several coding examples are provided in the appendices:

- Appendix A illustrates how to implement the Bean Provider interface. This code is in the `BeanProviderImpl.java` file, located in the `WindwardReports.jar` file. This is a do-nothing bean. If you set this as a bean in a tag, it would be identical to not having the bean there.
- Appendix B illustrates how to use a bean in a template.
- Appendix C illustrates how to use beans when calling WR directly.
- Appendix D illustrates how to use beans when running WR as a server.

### Stateful and Stateless Methods

The stateful and stateless methods are totally dependent on what you need when running from the report server. Unless you are 100% certain you will never use your bean via the report server, make sure you have written and tested your beans via the report server.

The best rule of thumb for the bean interface is this – it's simple. The code inside your bean may be complex, but the interface comes down to four methods called, depending on the tag they are used within, and two to four methods that are called as the report is started and ended.

#### **StatelessBeanProvider**

The method `StatelessBeanProvider.serverSetup()` is called on an instantiated bean once, right after the Report Server creates it. It is passed a set of properties that are set in the Report Server's configuration file. The information for this file is in the *System Administrator's Guide*.

---

The method `StatelessBeanProvider.serverTeardown()` is called when the Report Server is shutdown. If the Report Server is not shut down but is instead terminated (i.e. the process is just killed), this method will not be called.

These methods are not called when the bean is used via direct Java calls to Windward Reports.

### **StatefulBeanProvider**

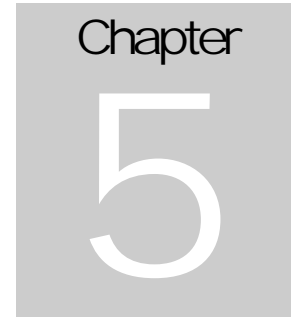
The method `StatefulBeanProvider.reportSetup()` is called on an instantiated bean once, right after the bean is created. At this point in time, the passed in `ProcessReport` object has completed the `ProcessReport.processSetup()` call but has not yet processed any data sources.

The method `StatefulBeanProvider.reportTeardown()` is called after the report has been generated. This is the last call that will be made to the bean. In the case of a bean used by the Report Server, the bean will be available for garbage collection after this call. In the case of a bean passed in direct Java access, the bean belongs to the calling program at this time.

The method `StatefulBeanProvider.datasourceSetup()` is called before each data source is applied to the report. The method `StatefulBeanProvider.datasourceTeardown()` is called after each data source is applied to the report.

Therefore, these methods will be called once, in a pair, for each data source applied to the report. If the same data source is applied multiple times, these methods will be called multiple times for that data source.

---



## Everything Else

### *In Closing...*

First of all, if there is anything you think we could add to make writing a bean easier – or functionality that should be added to the bean API, please let us know.

### Package Naming

Please do not use the `net.windward.*` namespace for your beans. There is no need for it to be in the same 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](mailto:support@windward.net).
  - Technical Support can be reached from 8:00 AM to 5:00 PM, Mountain Time, at 1-303-499-2544.
-

## Appendix A: Implementing the Bean Provider Interface

This code illustrates how to implement the Bean Provider interface. This code is in the `BeanProviderImpl.java` file, located in the `WindwardReports.jar` file. This is a do-nothing bean. If you set this as a bean in a tag, it would be identical to not having the bean there.

```

/*
 * Copyright (c) 2004 by Windward Studios, Inc. All rights reserved.
 *
 * This software is the confidential and proprietary information of
 * Windward Studios ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Windward Studios, Inc.
 */
package net.windward.bean;
import net.windward.tags.*;
import java.util.Locale;
/**
 * This implements a do nothing implementation of BeanProvider. It is equivalent to
 * not having a bean. To implement a bean you can override this class and then
 * implement all methods that need to perform non-default actions.
 * Note: you do not need to inherit from this class to create a bean. You only need
 * to implement the BeanProvider interface. This class exists as a convenience for
 * developers who wish to use it.
 *
 * @version 1.0 Jun 18, 2004
 */
public abstract class BeanProviderImpl implements BeanProvider {
    /**
     * Process a <wr:function ... /> tag. Return the string and formatting
     * to display.
     *
     * @param tag The tag that is being printed.
     * @param value The final string value for the tag.
     * @param locale The locale in effect for the document.
     * @return The string and formatting to write at this point in the report.
     */
    public BeanResult functionTag (FunctionTag tag, String value, Locale locale)
    throws BeanProviderException {
        return new BeanResult(value);
    }
    /**
     * Process a <wr:out ... /> tag. Return the string and formatting to
     * display.
     *
     * @param tag The tag that is being printed.
     * @param value The final string value for the tag.
     * @param locale The locale in effect for the document.
     * @return The string and formatting to write at this point in the report.
     */
    public BeanResult outTag (OutTag tag, String value, Locale locale) throws
    BeanProviderException {

```

---

```
        return new BeanResult(value);
    }
    /**
     * Process a <wr:forEach ... /> tag. This is called before each iteration
     * of a forEach loop. This provides the ability to end a loop early. Return true
     * to continue in the loop and false to end the loop.
     *
     * @param tag The tag that is being iterated.
     * @return true to continue in the loop and false to end the loop.
     */
    public boolean forEachTag (ForEachTag tag) {
        return true;
    }
    /**
     * Process a <wr:if ... /> tag. This provides the ability to set the value
     * of the if. Return true to include the text inside the if and false to skip the
     * if (or process the else if there is one).
     *
     * @param tag The tag that is being evaluated.
     * @param value The evaluated value of the tag.
     * @return true to include the text inside the if and false to skip the if
     * (or process the else if there is one).
     */
    public boolean ifTag (IfTag tag, boolean value) {
        return value;
    }
}
```

---

## Appendix B: Using a Bean in a Template

This example illustrates how to use a bean in a template, and may be found in Sample 14.

### A simple bean sample – list sales

This sample does not use the field format for the tags so they are easier to read.

<pre>&lt;wr:forEach select = '/order/item' &gt;   &lt;wr:out select = '/quantity' /&gt;</pre>	<pre>&lt;wr:out select = './description' /&gt;</pre>	<pre>&lt;wr:out select = './unit' type = 'CURRENCY' bean = "RedNegative" /&gt;</pre>	<pre>&lt;wr:out select = './total' type = 'CURRENCY' bean = 'RunningTotal' /&gt;</pre>
<pre>&lt;/wr:forEach &gt;</pre>		<pre>total</pre>	<pre>&lt;wr:out select = "" type = 'CURRENCY' bean = 'RunningTotal' total = 'display' /&gt;</pre>

Please note that the final total has a `select=""`. This is because the `select` attribute is required but because there is a bean, it can have an empty value (and should in this case). You can only do `select=""` for the out and function tags.

## Appendix C: Using Beans when Calling WR

This code illustrates how to use beans when calling WR directly. This code is located in Sample 14.

```

/*
 * Copyright (c) 2002 - 2004 by Windward Studios, Inc. All rights reserved.
 *
 * This software is the confidential and proprietary information of
 * Windward Studios ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Windward Studios, Inc.
 */

import java.io.*;
import java.util.*;

import net.windward.xmlreport.*;
import net.windward.format.htm.*;

/** Process a xml and rtf file into a report. This is a sample program that runs a report
 * from the command line. The usage is:<br>
 * java net.windward.xmlreport.RunReport order.xml template.rtf report.htm<br>
 * The three filenames can be anything you want. They are:<ul>
 * <li>order.xml - the xml data to use for the report.</li>
 * <li>template.rtf - the report template which must be a rtf file.</li>
 * <li>report.htm which is the report created. This file can have an extension of rtf, htm,
 * html, xhtml, pdf, sml(SpreadsheetML), xml(WordML), or txt and will create a report in the
 * appropriate format.</li>
 * </ul>
 *
 * @author David Thielen
 * @version 1.0
 */

public class RunReportBeans {

    /**
     * Process a xml and rtf file into a report. This is a sample program that runs a report
     * from the command line.
     *
     * @param args Must be three filenames like order.xml template.rtf report.htm where:<ul>
     * <li>order.xml - the xml data to use for the report.</li>
     * <li>template.rtf - the report template which must be a rtf file.</li>
     * <li>report.htm which is the report created. This file can have an extension of rtf,
     htm(no css),
     * html(with css), xhtml, pdf, sml(SpreadsheetML), xml(WordML), or txt and will create a
     report in the appropriate format.</li>
     * </ul>
     */
    public static void main(String[] args) {

        ProcessReport.init();

        if (args.length != 3) {
            System.out.println("RunReportBeans xmlFile, rtfFile, outFilename");
            return;
        }
        System.out.println("Processing xml file " + new File(args[0]).getAbsolutePath() + "\n" +

```

---

PROGRAMMER'S GUIDE

```

        "\ttemplate file " + new File(args[1]).getAbsolutePath() + "\n" +
        "\treport file " + new File(args[2]).getAbsolutePath());
File fil = null;

try {

    fil = new File(args[0]);
    FileInputStream xml = new FileInputStream(args[0]);
    fil = new File(args[1]);
    FileInputStream rtf = new FileInputStream(args[1]);
    fil = new File(args[2]);
    FileOutputStream out = new FileOutputStream(args[2]);

    // pick the output type
    ProcessReportAPI report;
    if (args[2].endsWith(".rtf")) {
        report = new ProcessRtf(xml, rtf, out);
    } else if (args[2].endsWith(".pdf")) {
        report = new ProcessPdf(xml, rtf, out);
    } else if (args[2].endsWith(".xls")) {
        report = new ProcessXls(xml, rtf, out);
        report.setTitle(fil.getName());
    } else if (args[2].endsWith(".sml")) {
        report = new ProcessExcelML(xml, rtf, out);
        report.setTitle(fil.getName());
    } else if (args[2].endsWith(".xml")) {
        report = new ProcessWordML(xml, rtf, out);
    } else if (args[2].endsWith(".txt")) {
        report = new ProcessTxt(xml, rtf, out);
    } else if (args[2].endsWith(".htm")) {
        // base html report
        report = new ProcessHtml(xml, rtf, out);
        ((ProcessHtmlAPI) report).setCss(ProcessHtmlAPI.CSS_NO, null, null);
    } else if (args[2].endsWith(".html")) {
        // html 4.01 with css report
        report = new ProcessHtml(xml, rtf, out);
        ((ProcessHtmlAPI) report).setCss(ProcessHtmlAPI.CSS_INCLUDE, null, null);
    } else if (args[2].endsWith(".xhtml")) {
        // xhtml
        report = new ProcessHtml(xml, rtf, out);
        ((ProcessHtmlAPI) report).setCss(ProcessHtmlAPI.CSS_INCLUDE, null, null);
        ((ProcessHtmlAPI) report).setSpec(ProcessHtmlAPI.XHTML);
    } else {
        System.err.println("output file must end with rtf, pdf, xls, eml, xml, txt, htm,
or html");
        return;
    }

    // add the beans
    report.addBean("RedNegative", new net.windward.bean.impl.RedNegative());
    report.addBean("RunningTotal", new net.windward.bean.impl.RunningTotal());

    report.process();

    // images?
    if (report.getReportType() == ProcessReport.TYP_HTML) {
        ArrayList images = ((ProcessHtmlAPI) report).getImageNames();
        for (int ind = 0; ind < images.size(); ind++) {
            HtmlImage img = (HtmlImage) images.get(ind);
            System.out.println(WindwardReports.getResource("report.run.image",
img.getName()));
            FileOutputStream file = new FileOutputStream(img.getName());
            ((ByteArrayOutputStream) img.getStream()).writeTo(file);
            file.close();
        }
    }

    xml.close();
    rtf.close();
    out.close();
}

```

---

## PROGRAMMER'S GUIDE

```
        System.out.println(args[2] + " built, " + Integer.toString(report.getNumPages()) + "
pages long");
    } catch (FileNotFoundException fnf) {
        System.err.println("File not found: " + fil.getAbsolutePath());
        fnf.printStackTrace();
    } catch (Throwable t) {
        System.err.println("Exception occurred: " + t.getMessage());
        t.printStackTrace();
    }
}
}
```

---

# Index

- Architecture, 5
  - attributes, 6
  - bean
    - sample, 7, 10
  - bean sample, 12
  - BeanProvider implementation, 4
  - beans
    - stateful, 5
    - stateless, 5
  - DataSourceProvider., 4
  - existing classes, 6
  - forEach* tag, 6
  - function* method, 6
  - Functionality, 6
  - HTML, 2
  - if* tag, 6
  - Implementation, 7
  - Implementing the Bean Provider Interface, 10
  - inheritance, 6
  - J2EE, 2
  - multiple reports
    - creating, 5
  - out* method, 6
  - output formats, 2
  - Package Naming, 9
  - PDF, 2
  - Properties, 6
  - Providing Feedback, 3
  - RedNegative, 3
  - RTF, 2
  - running from the report server, 7
  - RunningTotal, 3
  - sample bean, 7, 10
  - SpreadsheetML, 2
  - SQL, 2
  - Stateful and Stateless Methods, 7
  - stateful bean, 5
  - StatefulBeanProvider, 8
  - stateless bean, 5
  - StatelessBeanProvider, 7
  - Technical Support, 9
  - TXT, 2
  - Using a Bean in a Template, 12
  - Using Beans when Calling WR Directly, 13
  - Windward Studios copyright restrictions, 4
  - WordML, 2
  - XLS, 2
  - XML, 2
-