

PageList

```
package com.forio.common.util;

import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.ListIterator;
import java.util.Collection;

/**
 * An implementation of list that can be used to present paged information.
 * Includes getters and setters for use in templates to format page controls.
 * Can be used to manage paged behavior directly, or can store a page at a time
 * from an external source such as a database.
 *
 * This code is shared "as-is" and is intended as a guide for other
 * Velocity users rather than as a polished module.
 * Permission to use, modify, redistribute freely granted.
 *
 * To implement paging, put this list in the page. Then include
 * control statements like:
 *
 * #if($list.hasNext)
 *     <a href="?next=$list.nextOffset">Next</a>
 * #end
 *
 * It is the application responsibility to read the "next" parameter and
 * call setRowOffset to move the page forward.
 *
 * No developer documentation is available, but some useful User docs
 * from an actual application using this code are located at:
 *
 * http://broadcast.forio.com/broadcastdocs/creating_interfaces/lists.htm#paged_lists
 *
 * @copyright (c) Will Glass-Husain, Forio Business Simulations
 * @author <a href="mailto:wglass@forio.com">Will Glass-Husain</a>
 */
public class PageList implements List
{
    // properties from caller
    private List sourcelist;
    private int rowoffset;
    private int displayrows;
    private int totalrows;

    // calculated page properties
    private int pageindex;
    private int pagecount;
    private boolean hasNext;
    private boolean hasprevious;
    private int nextoffset;
    private int previousoffset;

    // misc state
    private boolean executed = false;
    private boolean one_page_only;
    private List sublist;
    private boolean display_all = false;

    // constants
    private static int DEFAULT_DISPLAY_ROWS = 10;

    /**
     * Use this constructor when the caller is providing the list and
     * PageList is responsible for the sublist.
     */
    public PageList (List sourcelist)
```

```

{
    this.sourcelist = sourcelist;
    one_page_only = false;

    this.displayrows = DEFAULT_DISPLAY_ROWS;
    this.totalrows = sourcelist.size();

    setPageIndex(1);
}

/**
 * Use this constructor when the caller is providing the list and PageList is responsible
 * for the sublist.
 */
public PageList (List sourcelist, int displayrows)
{
    this.sourcelist = sourcelist;
    one_page_only = false;
    this.totalrows = sourcelist.size();

    if (displayrows == -1)
        display_all = true;

    this.displayrows = displayrows;

    setPageIndex(1);
}

/**
 * Use this constructor when the caller is providing just the single page
 * to be echoed back by PageList.
 */
public PageList (List sourcelist, int rowoffset, int displayrows, int totalrows)
{
    this.sourcelist = sourcelist;
    this.rowoffset = rowoffset;
    this.displayrows = displayrows;
    this.totalrows = totalrows;
    one_page_only = true;

    calculatePageInfo();
}

////////// getters / setters

public List getSourceList() {return sourcelist;}
public void setSourceList(String val) {calculatePageInfo();}

public int getRowOffset() {return rowoffset;}
public void setRowOffset(int val) {rowoffset = val;calculatePageInfo();}

public int getFirstRowIndex() {return rowoffset + 1;}
public int getLastRowIndex() {return rowoffset + size();}

public int getDisplayRows() {return displayrows;}
public void setDisplayRows(int val) {
    displayrows = val;
    if (displayrows == -1)
        display_all = true;
    calculatePageInfo();
}

public int getTotalRows() {return totalrows;}
public void setTotalRows(int val) {totalrows = val;calculatePageInfo();}

public int getPageIndex() {return pageindex;}
public int getPageCount() {return pagecount;}

```

```
public int getNextOffset() {return nextoffset;}
public int getPreviousOffset() {return previousoffset;}

public boolean getHasNext() {return hasnext;}
public boolean getHasPrevious() {return hasprevious;}
```

```
////////// paging methods
```

```
public void nextPage()
{
    setPageIndex(pageindex + 1);
}

public void previousPage()
{
    setPageIndex(pageindex - 1);
}

public void setPageIndex(int pageindex)
{
    if (pageindex > pagecount)
        pageindex = pagecount ;

    if (pageindex < 1)
        pageindex = 1;

    showRow ((pageindex - 1) * displayrows);
}

public void showRow(int rowoffset)
{
    if (rowoffset >= totalrows)
        rowoffset = totalrows - 1;

    if (rowoffset < 0)
        rowoffset = 0;

    this.rowoffset = rowoffset;

    calculatePageInfo();
}

private void calculatePageInfo()
{
    int endrow;
    if (display_all) {
        rowoffset = 0;
        endrow = totalrows;
    } else {
        endrow = rowoffset + displayrows;
    }

    if (endrow > totalrows)
        endrow = totalrows;

    if (sourcelist != null) {
        sublist = sourcelist.subList(rowoffset,endrow);
    }

    if (display_all) {
        pageindex = 1;
        pagecount = 1;
        nextoffset = 0;
    } else if (displayrows == 0) {
        pageindex = 1;
        pagecount = 0;
    }
}
```

```

        nextoffset = 0;

    } else {
        pageindex = rowoffset / displayrows + 1;
        pagecount = (int) Math.ceil(totalrows / (displayrows + 0.0));
        nextoffset = (rowoffset / displayrows + 1) * displayrows;
        previousoffset = (int) Math.ceil(rowoffset / (displayrows + 0.0) - 1) * displayrows;
    }

    if (previousoffset < 0)
        previousoffset = 0;

    // determine whether there should be a next or previous button
    hasprevious = (rowoffset > 0);
    hasNext = (endrow < totalrows);

}

////////// list methods

public int size() {return sublist.size();}
public boolean isEmpty() {return sublist.isEmpty();}
public boolean contains(Object o) {return sublist.contains(o);}
public Iterator iterator() {return sublist.iterator();}
public Object[] toArray() {return sublist.toArray();}
public Object[] toArray(Object a[]) {return sublist.toArray(a);}
public boolean containsAll(Collection c) {return sublist.containsAll(c);}
public Object get(int index) {return sublist.get(index);}
public boolean equals(Object o) {return sublist.equals(o);}
public int hashCode() {return sublist.hashCode();}

public int indexOf(Object o) {return sublist.indexOf(o);}
public int lastIndexOf(Object o) {return sublist.lastIndexOf(o);}
public ListIterator listIterator() {return sublist.listIterator();}
public ListIterator listIterator(int index) {return sublist.listIterator(index);}
public List subList(int fromIndex, int toIndex) {return sublist.subList(fromIndex,toIndex);}

public boolean add(Object o) {throw new UnsupportedOperationException();}
public void add(int index, Object element) {throw new UnsupportedOperationException();}
public boolean remove(Object o) {throw new UnsupportedOperationException();}
public boolean addAll(Collection c) {throw new UnsupportedOperationException();}
public boolean addAll(int index, Collection c) {throw new UnsupportedOperationException();}
public boolean removeAll(Collection c) {throw new UnsupportedOperationException();}
public boolean retainAll(Collection c) {throw new UnsupportedOperationException();}
public void clear() {throw new UnsupportedOperationException();}
public Object set(int index, Object element) {throw new UnsupportedOperationException();}
public Object remove(int index) {throw new UnsupportedOperationException();}

public String toString() {return sublist.toString();}

//////////

}

```