Display tag library v.1.0

Project Documentation

Table of Contents

1.1 Introduction

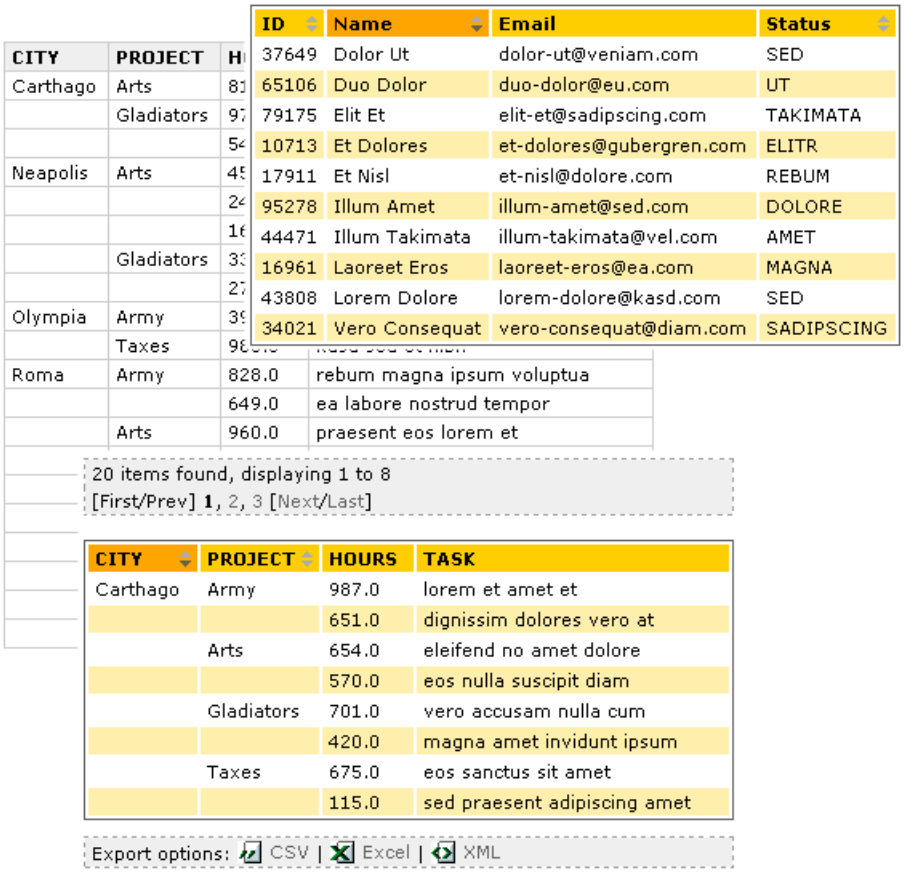
Overview

The display tag library is an open source suite of custom tags that provide high-level web presentation patterns which will work in an MVC model. The library provides a significant amount of functionality while still being easy to use.

What can I do with it?

Actually the display tag library can just... display tables! Give it a list of objects and it will handle column display, sorting, paging, cropping, grouping, exporting, smart linking and decoration of a table in a customizable XHTML style.

The tables in the sample images below were generated from lists using the <display:table> tag:



1.2 Dependencies

Dependencies

The following is a list of dependencies for this project. These dependencies are required to compile or run the application. Most of the libraries listed here are actually needed only at build time or during tests (mainly for framework integration tests) and you should not worry about them.

Required at runtime

Artifact ID	Type	Version	URL/comments
commons-beanutils	jar	1.6.1	http://jakarta.apache.org/commons/beanutils
commons-collections	jar	2.1.1	http://jakarta.apache.org/commons/collections also 3.0 is supported.
commons-lang	jar	2.0	http://jakarta.apache.org/commons/lang commons-lang 1.0 will not work
commons-logging	jar	1.0.4	http://jakarta.apache.org/commons/logging

Required for the EL version

Artifact ID	Type	Version	URL/comments
jstl	jar	1.0.2	http://jakarta.apache.org/taglibs/doc/standard-1.0-doc/
standard	jar	1.0.4	http://jakarta.apache.org/taglibs/

Optional (and needed for compiling)

Artifact ID	Type	Version	URL/comments
itext	jar	0.99	http://prdownloads.sourceforge.net/itext/ Needed at build time to compile classes for the PDF export. Needed at runtime to enable PDF export.

Only required to build or to run unit tests

Artifact ID	Type	Version	URL/comments
portlet-api	jar	unknown	http://jakarta.apache.org/jetspeed/site/install.html Required to build jetspeed/websphere portal server support classes. Not needed at runtime.
log4j	jar	1.2.8	http://logging.apache.org/log4j/docs/index.html optional: you can use any logging framework supported by commons-logging
maven-taglib-plugin	plugin	1.2	http://maven-taglib.sourceforge.net maven plugin needed to generate tag reference documentation page and to generate the 1.1 version of the tld
servletapi	jar	2.3	Servlet 2.3 support is required to build the library. At runtime only servlet 2.2 (tomcat 3, websphere 4) is strictly needed, servlet 2.3 support (tomcat 4, websphere 5) is needed only for the EL version of the taglib
httpunit	jar	1.6	http://httpunit.sourceforge.net Needed to compile and run HttpUnit tests.
jtidy	jar	4aug2000r7-dev	http://jtidy.sourceforge.net Needed to run HttpUnit tests.
nekohtml	jar	0.9.1	Needed to run HttpUnit tests.
js	jar	1.5R4.1	Needed to run HttpUnit tests.
jasper-compiler	jar	4.0.4	Needed to run HttpUnit tests.
jasper-runtime	jar	4.0.4	Needed to run HttpUnit tests.
xerces	jar	2.4.0	Needed to run HttpUnit tests.
xml-apis	jar	1.0.b2	Needed to run HttpUnit tests.
tools	jar	1.3	http://java.sun.com/j2se/1.4.2/download.html Needed to run HttpUnit tests. This is the tools.jar from sun jdk. Jar in not in maven repository but is set to JAVA_HOME/../lib/tools.jar in project.properties
Struts	jar	1.2.4	http://struts.apache.org/ Required to build Struts i18n adapter. Required at runtime only if you use Struts (if you wish to use it you should already have Struts in your project)
Spring	jar	1.1.1	http://www.springframework.org/ Required to build Spring i18n adapter. Required at runtime only if you use Spring (if you wish to use it you should already have Spring in your project)
commons-digester	jar	1.4.1	http://jakarta.apache.org/commons/digester Struts dependency needed to run Struts integration tests.
webwork	jar	2.1.5	http://www.opensymphony.org/ Required to build Webwork i18n adapter. Required at runtime only if you use Webwork (if you wish to use it you should already have Webwork in your project)
xwork	jar	1.0.3	http://www.opensymphony.org/ Required to build Webwork i18n adapter. Required at runtime only if you use Webwork (if you wish to use it you should already have Webwork in your project)
oscore	jar	2.2.4	http://www.opensymphony.org/ Required to build Webwork i18n adapter. Required at runtime only if you use Webwork (if you wish to use it you should already have Webwork in your project)
ognl	jar	2.6.5	http://www.ognl.org/ Required to build Webwork i18n adapter. Required at runtime only if you use Webwork (if you wish to use it you should already have Webwork in your project)

2.1 Download

Download

releases

You can download source and binary distributions from the SourceForge Server .

development snapshot

The latest snapshot build (usually uploaded in sync with this website) can be downloaded directly from here . Get this one only if you are a developer or you absolutely need a feature/fix added before the latest release (see changes for the full change log).

source code from CVS

You can also obtain the source from the SourceForge CVS Server, see Source Repository .

Files

The "bin" versions include the following:

- **displaytag.jar** - the jar file that contains the taglib classes, you need to drop this file into your web application WEB-INF/lib directory.
- **displaytag.tld** - the tld file that contains the tag library descriptor, you need to drop this file into your web application WEB-INF directory.
- **displaytag.war** - the documentation and the example applications that appear on this web site, optionally drop this into your web container deployment directory.

The "src" versions includes the complete source code to the tag library, the various example pages, and the documentation.

2.2 Install

Installation Guide

This package comes with pre-built binaries located in the dist directory. Those distribution files are:

file	description
displaytag.war	documentation and examples
displaytag.jar	the taglib jar
displaytag.tld	the taglib tld file

To quickly view the documentation and examples showing the features and functionality of the display taglib, just deploy the displaytag.war file to your application server (the details of how differ from server to server) or servlet container.

If you would like to make use of the display taglib in your own application, do the following:

STEP 1 : Drop the displaytag-{version}.jar file in your application `WEB-INF/lib` directory

STEP 2 : Make sure that following libraries are in your `WEB-INF/lib` directory (or made available via the classpath to your application server). Refer to the dependencies document for the correct version of these libraries. You can download a copy of everything from jakarta or you can grab them from the example webapp in the bin distribution.

- commons-logging.jar
- commons-lang.jar
- commons-collections.jar
- commons-beanutils.jar
- log4j.jar

STEP 3 : *Needed only for JSP 1.1 containers* . Drop the displaytag-{taglibversion}.tld file in your application `WEB-INF/` directory. Refer to the tlds page for the available tlds.

STEP 4 : *Needed only for JSP 1.1 containers*. Define a taglib element like the following in your /WEB-INF/web.xml file

```
<taglib>
    <taglib-uri>http://displaytag.sf.net</taglib-uri>
    <taglib-location>/WEB-INF/displaytag-{taglibversion}.tld</taglib-location>
</taglib>
```

STEP 5 : *Optional*. Depending on your architecture, you may need to configure a filter to make export work. See the export filter page for the details about how to do it and when you could need it.

DONE : Define the tag extension in each JSP page that uses the display taglib. The uri directives must match what you defined in the web.xml file above OR the URI defined in one of the tlds in the jar file. With JSP 1.2 containers, the jar file is automatically scanned and you don't need to define an entry in your web.xml file. The prefix identifies the tags in the tag library within the JSP page.

```
<%@ taglib uri="http://displaytag.sf.net" prefix="display" %>
```

The declaration, if you are using a JSP XML syntax, looks like:

```
<jsp:root version="1.2" xmlns:jsp="http://java.sun.com/JSP/Page"
     xmlns:display="urn:jsptld:http://displaytag.sf.net">
```

For more help with general taglib use, please see: http://jakarta.apache.org/taglibs/tutorial.html

2.3 Tlds

Tlds

Displaytag comes with 3 different tlds at the moment. Look at this table to understand which is the right one for you.

tld	URI	description
`displaytag-11.tld`	http://displaytag.sf.net	Jsp 1.1 version of the tld, you will need to use this one if you plan to install your application on container only supporting j2ee 1.2 (Tomcat 3, Websphere 4, WebLogic 6...).
`displaytag-12.tld`	http://displaytag.sf.net	Jsp 1.2 version of the tld: requires j2ee 1.3 (Tomcat 4, WebSphere 5, WebLogic 7...). Use this version if you are not looking for j2ee 1.2 compatibility and don't need EL support.
`displaytag-el-12.tld`	http://displaytag.sf.net/el	EL version of the tag library. It offers the same features as the standard 1.2 version, plus Expression Language Support. It will require a couple of addictional libraries, see the dependencies page. **Don't use this one if you are looking for EL support on jsp 2.0 containers (Tomcat 5).** In Jsp 2.0 compatible servers expressions are evaluated directly by the container, so you can use the standard 1.2 tld and still have EL support (the EL tld will not work, since expressions wil be evaluated twice).

2.4 Tag reference

Display *: Tag Library

The display tag library is an open source suite of custom tags that provide high level web presentation patterns which will work in a MVC model, and provide a significant amount of functionality while still being simple and straight-forward to use. The primary tag in the library is the Table tag. This is version 1.0 .

- caption Simple tag which mimics the html caption tag .
- column Displays a property of a row object inside a table .
- footer Tag wich should be nested into a table tag to provide a custom table footer .
- setProperty Sets the indicated property on the enclosing Table tag .
- table Displays a list in an html table, formatting each item in the list according to the Column tags nested inside of this tag .

*Required attributes are marked with a * .*

table

Displays a list in an html table, formatting each item in the list according to the Column tags nested inside of this tag. Use the list attribute to indicate the Collection of data, in some scope, that the tag should operate on. Supports the export of the list data to alternative formats such as CSV, Excel, and XML. The contents of the list can be sorted, and the list can be broken into individual pages for display. If you use this tag in Struts, or in some other framework where the page is included via a jsp:include, you should use the requestURI attribute to indicate where tag generated links should point.

Can contain: JSP

Example

```
<display:table name="someList" export="true" id="row" requestURI="MyAction.do">
  <display:column sortable="true" title="ID"> <c:out value="${row.id}"/>
</display:column>
  <display:column property="email" autolink="true"/>
  <display:column property="description" title="Comments"/>
</display:table>
```

Attributes

Name	Description	Type
cellpadding	html pass through attribute. Better using "padding" css attribute in style or class	`String`
cellspacing	html pass through attribute	`String`
class	html pass through attribute	`String`
decorator	Fully qualified class name for a TableDecorator. Use a TableDecorator to provide custom operations against the whole list, such as computing totals. Must extend org.displaytag.decorator.TableDecorator.	`String`
defaultorder	The default order for the sorted column. Valid values are "ascending" (default) or "descending"	`String`
defaultsort	The index of the column that will be used by default for sorting (starting from 1)	`int`
excludedParams	Whitespace separated list containg the name of parameters which should NOT be forwarded during paging or sorting. You can use excludedParams="*" to match (exclude) any parameter.	`String`
export	enable/disable export. Valid values are true or false	`boolean`
frame	html pass through attribute.	`String`
htmlId	html "id" pass through attribute	`String`
id	See "uid". The id attribute can't be a runtime expression in jsp 1.0 compliant containers, while uid will allow it.	`String`
length	number of records to be shown	`int`
list	Reference to the object used as source for the table. Can be an expression like requestScope.object.property . You must define either the name attribute or the list attribute. Using "Name" is suggested.	`String`
name	reference to the object used as source for the table. Can be an expression like requestScope.object.property. In the EL version of the taglibrary this must be an EL expression which points to the source object.	`String`
offset	index of the first record to be shown	`int`
pagesize	number of records in a page	`int`
requestURI	When the present, links for sorting, exports, and paging are formed by adding any tag generated parameters to the value of requestURI attribute.	`String`
requestURIcontext	Enable/disable prepending of application context to generated links. Default is true, you can set it to false in order to generate cross-context links.	`boolean`

Name	Description	Type
rules	html pass through attribute.	`String`
sort	Use 'page' if you want to sort only visible records, or 'list' if you want to sort the full list	`String`
style	html pass through attribute	`String`
summary	html pass through attribute	`String`
uid	Unique id used to identify this table. The object representing the current row is also added to the pageContext under this name and the current row number is added using the key uid_rowNum. Two tables in the same page can't have the same uid (paging and sorting will affect both). If no "htmlId" is specified the same value will be used for the html id of the generated table.	`String`
styleClass	@deprecated: use "class"	`String`
align	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
background	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
bgcolor	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
height	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
hspace	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
vspace	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
width	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
border	@deprecated html pass through attribute. Use css "border"	`String`
scope	@deprecated in displaytag 1.0. Use "pageScope.", "requestScope.", "sessionScope.", "applicationScope." prefixes in name. Not supported in the EL version of the tag.	`String`
property	@deprecated in displaytag 1.0. Use list.property in "name" attribute. Not supported in the EL version of the tag.	`String`

column

Displays a property of a row object inside a table. MUST be nested inside of a Table tag. The value displayed will be the results of a decorator (if any); else the property named by the 'property' attribute; or

if the 'property' attribute is null, then the results of evaluating the JSP body of the tag.

Can contain: JSP

Attributes

Name	Description	Type
autolink	Automatically hyperlink URLs and email addresses that appear in the column. Defaults to 'false'.	`boolean`
class	html pass through attribute; use this instead of directly coding presentational atttributes.	`String`
decorator	The fully qualified class name of a class that should be used to "decorate" the underlying object being displayed. The class should implement org.displaytag.decorator.ColumnDecorator. If a decorator is specified for the entire table, then this decorator will decorate that decorator.	`String`
group	The grouping level (starting at 1 and incrementing) of this column (indicates if successive contain the same values, then they should not be displayed). The level indicates that if a lower level no longer matches, then the matching for this higher level should start over as well. If this attribute is not included, then no grouping is performed.	`int`
headerClass	"class" html attribute added only for header cells.	`String`
href	The base URL used to construct the dynamic link. If this attribute is provided, then the data that is shown for this column is wrapped inside a <a href> tag with the url provided through this attribute. Typically you would use this attribute along with one of the struts-like param attributes (param*) to create a dynamic link so that each row creates a different URL based on the data that is being viewed. An empty href value will generate a link to the current page, preserving parameters just like for paging links.	`String`
maxLength	If this attribute is provided, then the column's displayed is limited to this number of characters. An elipse (...) is appended to the end if this column is linked, and the user can mouseover the elipse to get the full text. Be careful on using this attribute for String which can contain html tags or entities, or together with the autolink attribute turned on: displaytag will do its best trying to avoid leaving unclosed tags or broken entities in the output, but a complex or bad input could lead to unattended results.	`int`

Name	Description	Type
maxWords	If this attribute is provided, then the column's displayed is limited to this number of words. An elipse (...) is appended to the end if this column is linked, and the user can mouseover the elipse to get the full text. Be careful on using this attribute for String which can contain html tags or entities, or together with the autolink attribute turned on: displaytag will do its best trying to avoid leaving unclosed tags or broken entities in the output, but a complex or bad input could lead to unattended results.	`int`
media	Use this attribute to keep a column from being output during an export. The column will only render for the named media type(s) - it won't be added to the table if the current request media is not supported. Can be any space separated combination of 'html', 'csv', 'xml', 'all', or 'excel'. Defaults to 'all'. See the export page in the example webapp for more details.	`String`
nulls	By default, null values don't appear in the list. By setting 'nulls' to 'true', then null values will appear as "null" in the list (mostly useful for debugging). Defaults to 'false'.	`boolean`
paramId	The name of the request parameter that will be dynamically added to the generated href URL. The corresponding value is defined by the paramProperty and (optional) paramName attributes, optionally scoped by the paramScope attribute.	`String`
paramName	The name of a JSP bean that is a String containing the value for the request parameter named by paramId (if paramProperty is not specified), or a JSP bean whose property getter is called to return a String (if paramProperty is specified). The JSP bean is constrained to the bean scope specified by the paramScope property, if it is specified. If paramName is omitted, then it is assumed that the current object being iterated on is the target bean.	`String`
paramProperty	The name of a property of the current object being iterated on, whose return value will be used as the value of the parameter (named by the paramId attribute) that will be dynamically added to this href URL. If paramName is also specified the property will not be fetched from the object being iterated on, but from the bean specified by paramName. The support of paramProperty in conjunction with paramName will be probably removed in future: use paramProperty only if you need a property in the iterated object, elsewhere use only paramName (you can select a property using an expression name.property).	`String`
property	name of the property in the bean specified in the parent table tag (via the "name" attribute) mapped to this column	`String`
sortable	Set to 'true' to make the column sortable. Defaults to 'false'.	`boolean`

Name	Description	Type
sortProperty	name of the property in the bean specified in the parent table tag (via the "name" attribute) which will be used to sort values in the column. This can be used when the column body is filled or a decorator is used and column should sort on undeorated values.	`String`
style	html pass through attribute.	`String`
title	title of the column (text for the th cell)	`String`
titleKey	Resource key used to lookup the title value. Only works if "title" is not defined. Works together with a configured I18nResourceProvider, specified via the displaytag.properties file. By default, if JSTL is available, the JSTL provider is used, which makes this attribute work the same as fmt:message's key property.	`String`
url	The base URL used to construct the dynamic link. This attribute has the same functionality as the href attribute, but it pre-pends the contextPath.	`String`
width	@deprecated; html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
styleClass	@deprecated: use "class"	`String`
headerStyleClass	@deprecated: use "headerClass"	`String`
sort	@deprecated: use "sortable"	`boolean`
align	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
background	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
bgcolor	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
height	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
nowrap	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
valign	@deprecated html attribute. Use "style" or "class" to set presentational attributes using css.	`String`
paramScope	@deprecated - use Expressions in paramName. The scope within which to search for the bean specified by the paramName attribute. If not specified, all scopes are searched. If paramName is not provided, then the current object being iterated on is assumed to be the target bean.	`String`

setProperty

Sets the indicated property on the enclosing Table tag. MUST be nested within a Table tag. As an alternative, you may create a property file that holds sitewide defaults; see the configuration documentation or the DisplayPropertiesLoaderServlet javadoc for information.

Can contain: JSP

Example

```
<display:setProperty name="paging.banner.placement" value="bottom" />
or
<display:setProperty name="paging.banner.placement">bottom</display:setProperty>
```

Attributes

Name	Description	Type
*name	The name of the property to set on the enclosing Table tag.	`String`
value	The value to which the property is set. You can also set the property value in the tag body.	`String`

footer

Tag wich should be nested into a table tag to provide a custom table footer. The body of the tag is outputted as is in the generated table in the tfoot section.

Can contain: JSP

Example

```
<display:table name="someList">
  <display:column property="mail"/>
  <display:column property="total"/>
  <display:footer>
          <tr>
                  <td>total:</td>
                  <td><c:out value="${something}" /></td>
          <tr>
  </display:footer>
```

```
</display:table>
```

Attributes

caption

Simple tag which mimics the html caption tag. Use it inside a table tag to display a caption.

Can contain: JSP

Example

```
<display:table name="someList">
  <display:column property="mail"/>
  <display:column property="total"/>
  <display:caption>This is the table caption</display:caption>
</display:table>
```

Attributes

Name	Description	Type
class	html pass through attribute.	`String`
dir	html pass through attribute.	`String`
id	html pass through attribute.	`String`
lang	html pass through attribute.	`String`
style	html pass through attribute.	`String`
title	html pass through attribute.	`String`

2.5 Configuration

Configuration properties

This table lists all the configurable properties for the tag libraries. The default properties are defined in the `TableTag.properties` file included in the library jar.

There are 2 ways to override default property settings:

- For the whole web application, create a custom properties file named "displaytag.properties" and place it in the application classpath. Displaytag will use the locale of the request object to determine the locale of the property file to use; if the key required does not exist in the specified file, the key will be loaded from a more general property file.
- For a single table instance, using the <display:setProperty> tag

Include in your custom properties file only the properties you need to change. If a property is not defined in the user file, the default from the TableTag.properties included in the jar is used.

I18n

Some properties contain messages you may wish to display according to the user Locale. To do that first add a default `displaytag.properties` file where you set all the locale independent entries and default messages. Then you can add localized properties file (for example `displaytag_IT.properties`).

Generic

Property	Default	Valid Values	Description	Can be set using file/setProperty
basic.show.header	true	true, false	Indicates if you want the header to appear at the top of the table, the header contains the column names, and any additional action banners that might be required (like paging, export, etc...)	yes/yes
basic.empty.showtable	false	true, false	Indicates if you want the table to show up also if the list is empty	yes/yes

Property	Default	Valid Values	Description	Can be set using file/setProperty
basic.msg.empty_list	Nothing found to display	Any string	The message that is displayed if the list that this table is associated with is either null, or empty. Used only if `basic.empty.showtable` is `false` .	yes/yes
basic.msg.empty_list_row	<tr class="empty"><td colspan="{0}">Nothing found to display.</td></tr>	Any string	The message that is displayed into the first table row if the list that this table is associated with is either null, or empty. {0} is replaced with the total column number to generate a correct colspan. Used only if `basic.empty.showtable` is `true` .	yes/yes
sort.amount	page	page, list	Indicates if the full list should be sorted before paging or if the sorting only affects items in the current page. Default behaviour is to sort only items in the current page (first paging, then sorting).	yes/no
export.banner	Export options: {0}	Any string in a message format with 1 placeholder	Contains the string that is displayed in the table footer when the user indicates that they want to enable the export function. The placeholder is replaced with links to the various export formats that are support.	yes/yes
export.banner.sepchar	|	Any string	Used to separate the valid export type (typically would be a bar, a comma, or a dash).	yes/yes
paging.banner.placement	top	top, bottom, both	When the table tag has to show the header for paging through a long list, this option indicates where that header should be shown in relation to the table	yes/yes
paging.banner.item_name	item	Any string	What the various objects in the list being displayed should be referred to as (singular)	yes/yes
paging.banner.items_name	items	Any string	What the various objects in the list being displayed should be referred to as (plural)	yes/yes
paging.banner.no_items_found	No {0} found.	Any string in a message format with 1 placeholder	What is shown in the pagination header when no objects are available in the list to be displayed. The single placeholder is replaced with the name of the items in the list (plural)	yes/yes

Property	Default	Valid Values	Description	Can be set using file/setProperty
paging.banner.one_item_found	One {0} found.	Any string in a message format with 1 placeholder	What is shown in the pagination header when one object is available in the list to be displayed. The single placeholder is replaced with the name of the items in the list (singular)	yes/yes
paging.banner.all_items_found	{0} {1} found, displaying all {2}.	Any string in a message format with 3 placeholders	What is shown in the pagination header when all the objects in the list are being shown. {0} and {2} are replaced with the number of objects in the list, {1} is replaced with the name of the items {plural}	yes/yes
paging.banner.some_items_found	{0} {1} found, displaying {2} to {3}.	Any string	What is shown in the pagination header when a partial list of the objects in the list are being shown. Parameters: • {0}: total number of objects in the list • {1}: name of the items (plural) • {2}: start index of the objects being shown • {3}: end index of the objects being shown • {4}: current page • {5}: total number of pages	yes/yes
paging.banner.group_size	8	Any reasonable number	The number of pages to show in the header that this person can instantly jump to	yes/yes
paging.banner.full	[First/ Prev] {0} [Next/ Last]		What is shown in the pagination bar when there are more pages and the selected page is not the first or the last one. Parameters: • {0}: numbered pages list • {1}: link to the first page • {2}: link to the previous page • {3}: link to the next page • {4}: link to the last page • {5}: current page • {6}: total number of pages	yes/yes
paging.banner.first	[First/Prev] {0} [Next/ Last]		What is shown in the pagination bar when the first page is being shown. Placeholders are the same as for `paging.banner.full`.	yes/yes

Property	Default	Valid Values	Description	Can be set using file/setProperty
paging.banner.last	[First/ Prev] {0} [Next/Last]		What is shown in the pagination bar when the last page is being shown. Placeholders are the same as for `paging.banner.full`.	yes/yes
paging.banner.onepage	{0}		What is shown in the pagination bar when only one page is returned. Placeholders are the same as for `paging.banner.full`.	yes/yes
paging.banner.page.selected	{0}		selected page. {0} is replaced with the page number, {1} with the page url.	yes/yes
paging.banner. page.link	{0}		link to a page. {0} is replaced with the page number, {1} with the page url.	yes/yes
paging.banner.page.separator	,		separator between pages	yes/yes
factory.requestHelper	org.displaytag.util.DefaultRequestHelperFactory	Class name for any valid RequestHelperFactory implementation	RequestHelperFactory to be used. You can replace the default one if you need to generate links with a different format (for example in portal applications).	yes/no

Exporting

Displaytag supports exporting to excel, csv, and xml formats. Some configurable properties are specific for one of these format. Replace " {*export name*} " in the property name with "excel", "csv" or "xml". Some of the properties won't work in any export format.

Property	Default	Valid Values	Description	Can be set using file/setProperty
export.types	csv excel xml pdf	whitespace separated list of configured export types	Holds the list of configured export types.	yes/no
export. *{export name}*	true	true, false	Should the tag present the option to export data in this specific format.	yes/yes
export. *{export name}*.class		Any valid class which implements the `org.displaytag.export.ExportView` interface	Fully qualified class name for the class which will be used for exporting.	yes/no
export. *{export name}*.label	*{export name}*	Any string	The label on the link that the user clicks on to export the data in a specific format.	yes/yes

Property	Default	Valid Values	Description	Can be set using file/setProperty
export. *{export name}*.include_header	false	true, false	If set to true, the first line of the export will contain column titles as displayed on the HTML page. The header by default is not included in when exporting.	yes/yes
export. *{export name}*.filename	*none*	any valid file name	When saving exported files the user will be prompted to use this file name.	yes/yes
export.amount	list	page, list	Indicates how much data should be sent down to the user when they ask for a data export. *By default, it sends the entire list*. But, you can instruct the table tag to only send down the data that is currently being shown on the page	yes/yes
export.decorated	true	true, false	Should the data be "decorated" as it is exported. The default value is true, but you might want to turn off any decoration that is for example HTML specific when exporting the data.	yes/yes

CSS

Property	Default	Valid Values	Description	Can be set using file/setProperty
css.tr.even	even	any valid css class name	css class automatically added to even rows	yes/yes
css.tr.odd	odd	any valid css class name	css class automatically added to odd rows	yes/yes
css.th.sorted	sorted	any valid css class name	css class automatically added to the header of sorted columns	yes/yes
css.th.ascending	order1	any valid css class name	css class automatically added to the header of a column sorted is ascending order	yes/yes
css.th.descending	order2	any valid css class name	css class automatically added to the header of a column sorted is descending order	yes/yes
css.table	*none*	any valid css class name	css class automatically added to the main table tag	yes/yes
css.th.sortable	*none*	any valid css class name	css class automatically added to any sortable column	yes/yes

2.6 Export filter

Export filter? What's that?

When displaytag exports data in any non-html format, it needs to change the content type returned to the browser and reset any other content generated by the surrounding page.

Sometimes this can't be done: if content has already been sent back to the user, the response can't be reset and you get an error. This could happen because:

- Too many chars have been already written to the response, so that the response buffer was full and response has been automatically flushed.
- Something (tags? java snippets?) before the `display:table` tag has explicitely flushed the response (`response.flushBuffer()`).
- Your page is dinamically included into another page. This happens for example using Struts tiles.

Another problem is related to exporting binary files. The output of binary data is not supported in jsps: it may work on some application server, but it may end up with errors in others. Because of this restriction an "external help" may be required. Cvs, xml and excel formats don't require a binary output, but if you want to try a pdf export or other custom binary formats you will have to face some problems.

The solution

The first attempt can be using a larger page buffer in your jsp pages, for example:

```
<%@ page buffer = "16kb" %>
```

However, this can work only if you are in the first situation listed above.

In j2ee 1.3/jsp 1.2 containers you can take advantage of filters to solve the problem. Displaytag ships with a filter which works together with the table tag during export, disallowing the response to be flushed when an export has been requested.

Installing the export filter

Configure the Filter in your web.xml:

```
<filter>
    <filter-name>ResponseOverrideFilter</filter-name>
    <filter-class>org.displaytag.filter.ResponseOverrideFilter</filter-class>
</filter>
```

And add mappings for the pages that you will intercept, for example:

```
<filter-mapping>
    <filter-name>ResponseOverrideFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>ResponseOverrideFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

3.1 Basic usage

Simplest case, no columns

```
<% request.setAttribute( "test", new TestList(10, false) ); %>

<display:table name="test" />
```

Amount	Project	Task	City
587.0	Arts	et gubergren ut et	Olympia
72.0	Gladiators	duo sit erat justo	Neapolis
277.0	Gladiators	justo tempor consetetur consetetur	Roma
598.0	Gladiators	magna erat tempor justo	Roma
953.0	Gladiators	voluptua nonumy et sadipscing	Olympia
7.0	Taxes	et est tempor eirmod	Roma

The simplest possible usage of the table tag is to point the table tag at a java.util.List implementation and do nothing else. The table tag will iterate through the list and display a column for each property contained in the objects.

Typically, the only time that you would want to use the tag in this simple way would be during development as a sanity check. For production, you should always define at least a single column.

Basic, columns

```
<% request.setAttribute( "test", new TestList(10, false) ); %>

<display:table name="test">
  <display:column property="id" title="ID" />
  <display:column property="name" />
  <display:column property="email" />
  <display:column property="status" />
  <display:column property="description" title="Comments"/>
</display:table>
```

ID	Name	Email	Status	Comments
42109	Diam Ipsum	diam-ipsum@eirmod.com	STET	sed amet...
41756	Elitr Sed	elitr-sed@eirmod.com	ERAT	no Lorem...
75830	Ipsum Sit	ipsum-sit@eirmod.com	DIAM	duo takimata...
72151	Tempor Sadipscing	tempor-sadipscing@sea.com	IPSUM	dolores clita...
91649	Ipsum Aliquyam	ipsum-aliquyam@sit.com	IPSUM	duo ut...
47277	Et Lorem	et-Lorem@duo.com	ET	sed sanctus...
15618	Ea Sanctus	ea-sanctus@dolor.com	TEMPOR	accusam ipsum...
28021	Ipsum Dolor	ipsum-dolor@magna.com	MAGNA	sed takimata...
32369	Dolores Est	dolores-est@consetetur.com	AT	erat invidunt...
53096	Ut Labore	ut-labore@sadipscing.com	SADIPSCING	est dolore...

This example starts to show you how to use the table tag. You point the table tag at a datasource (a List), then define a number of columns with properties that map to accessor methods (getXXX) for each object in the List.

Note that you have one column tag for every column that you want to appear in the table. And, the column specifies what property is shown in that particular row.

You can define the content of a column by adding a `property` attribute to the column tag or adding a content to the tag.

- <display:column property="email" />
- <display:column title="email">email@it.com</display:column>

There are two ways to define the content of a column. Of course, in the tag body you can use scriptlets or other custom tags. Using the `property` attribute to define the content of a column is usually faster and works better with sorting. If you add a `property` attribute the tag body is ignored.

Adding content in the column body you can easily concatenate or "decorate" fields available in objects in the list. See the implicit objects chapter for more details.

The `property` attribute specifies what `getXXX` method is called on each item in the list. So for the second column, `getName` is called. By default the property name is used as the header of the column unless you explicitly give the column a title.

3.2 Implicit objects

Implicit objects created by table

```
<display:table name="test" id="testit">
    <display:column property="id" title="ID" />
    <display:column property="name" />
    <display:column title="static value">static</display:column>
    <display:column title="row number
(testit_rowNum)"><%=pageContext.getAttribute("testit_rowNum")%></display:column>
    <display:column title="((ListObject)testit).getMoney()">
      <%=((ListObject)pageContext.getAttribute("testit")).getMoney()%>
    </display:column>
</display:table>
```

If you add and `id` attribute the table tag makes the object corresponding to the given row available in the page context so you could use it inside scriptlet code or some other tag. Another implicit object exposed by the table tag is the row number, named *`id`*`_rowNum` .

These objects are saved as attributes in the page scope (you can access it using `pageContext.getAttribute("id")`). They are also defined as nested variables (accessible using `<%=id%>`), but only if the value of the id atribute is not a runtime expression. The preferred way for fetching the value is to always use pageContext.getAttribute().

If you do not specify the `id` attribute no object is added to the pageContext by the table tag

This is a simple snippet which shows the use of the implicit objects created by the table tag with JSTL.

```
<display table id="row" name="mylist">
  <display:column title="row number" >
    <c:out value="${row_rowNum}"/>
  </display:column>
  <display:column title="name" >
    <c:out value="${row.first_name}"/>
    <c:out value="${row.last_name}"/>
  </display:column>
</display:table>
```

3.3 Data sources

Data sources

Expressions

Up until this point, we have simply had a List object available to us under the name "list" in the request scope that has driven the display of the tables shown. We have been setting up that bean with the following scriptlet, but presumably you would be doing something similar in your Action class rather then raw on this jsp page.

```
<% request.setAttribute( "test", new TestList( 10 ) ); %>
```

This table is called with the following attributes:

```
<display:table name="test">
```

You can also acquire a handle to the list you want to display by specifying not only a bean name, but also a bean property (a getter method), and the table tag will call that property to fetch the list to display.

Actually there are two "flavors" of displaytag: an EL version, which requires j2ee 1.3 and a jsp 1.1 (j2ee 1.2 compatible) version

In the EL version you can obviously use an EL expression, like name="${pageScope.name.property}"

In the non-EL version you can define the "name" attribute using a really similar sintax, just without the `${}`:

You can define the scope of the bean adding one of the following suffix:

- pageScope
- requestScope (default)
- sessionScope
- applicationScope

You can also access javabean style properties, mapped properties or indexed properties in the bean, also nested. The syntax for accessing a javabean property is `.property` . You can read a mapped property specifying it between `()` and an indexed property using `[]`.

So the following:

`sessionScope.list.value.attribute(name).item[1]`

is equivalent to:

```
session.getAttribute("list").getValue().getAttribute("name").getItem(1)
```

Supported data

The table tag actually supports the following kind of objects:

- a Collection
- an Enumeration
- a Map (values are displayed in row)
- a Dictionary (values are displayed in row)
- an array
- an Iterator
- any Object with an iterator() method
- ... any other object will be displayed on a single row

From a db?

Displaytag will never support retrieving data from a db directly. Displaytag is here to help you in displaying data, not to retrieve them.

Anyway, there are a couple of easy methods to get records from a db and display them using displaytag:

1) Using jstl:

just use the sql:query tag and pass the result to the table tag in this way

```
<sql:query var="results">
  select * from table
</sql:query>

<display:table name="${results.rows}" />

(or
<display:table name="pageScope.results.rows" />
if not using the EL version)
```

2) Using dynabeans

see
http://jakarta.apache.org/commons/beanutils/api/org/apache/commons/beanutils/RowSetDynaClass.html

```
<%
      Connection con = ...; // just open a connection

      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * from table");
```

```
      RowSetDynaClass resultSet = new RowSetDynaClass(rs, false);
      stmt.close();
      con.close();
      request.setAttribute("results", resultSet);
%>

      <display:table name="requestScope.results.rows" />
```

3.4 Decorators

Decorators

A "decorator" is a design pattern where one object provides a layer of functionality by wrapping or "decorating" another object.

Table decorators

```
<display:table name="test" decorator="org.displaytag.sample.Wrapper" >
  <display:column property="id" title="ID" />
  <display:column property="email" />
  <display:column property="status" />
  <display:column property="date" />
  <display:column property="money" />
</display:table>
```

Let's assume you have list of business objects that you want to display, and the objects contain properties that don't return native Strings, and you want control over how they get displayed in the list (for example, Dates, money, numbers, etc...). I would be bad form to put this type of formatting code inside your business objects, so instead create a Decorator that formats the data according to your needs.

Notice the following 4 key things (and refer to the TableDecorator javadoc for some of the other details).

- The Wrapper class must be a subclass of TableDecorator. There is various bootstrapping and API methods that are called inside the TableDecorator class and your class must subclass it for things to work properly (you will get a JspException if your class does not subclass it).
- Be smart and create your formatters just once in the constructor method - performance will be a lot better...
- Notice how the getDate() and getMoney() methods overload the return value of your business object contained in the List. They use the `TableDecorator.getCurrentRowObject()` method to get a handle to the underlying business object, and then format it accordingly.
- We do not have to overload each of the other business object properties (like getID, getEmail, etc...). The decorator class is called first, but if it doesn't implement the method for the property called, then the underlying business class is called.

The way this works is that a single decorator object is created right before the table tag starts iterating through your List, before it starts processing a particular row, it gives the object for that row to the decorator, then as the various properties getXXX() methods - the decorator will be called first and if the

decorator doesn't implement the given property, the method will be called on the original object in your List.

Column Decorators

You can also specify decorators that work on individual columns, this would allow you to come up with data specific formatters, and just reuse them rather then coming up with a custom decorator for each table that you want to show a formatted date for.

```
<display:table name="test">
  <display:column property="id" title="ID" />
  <display:column property="email" />
  <display:column property="status" />
  <display:column property="date" decorator="org.displaytag.sample.LongDateWrapper"
/>
</display:table>
```

Table decorators, column decorators or code in the column body?

As a rule of thumb, a decorator is faster than using scriptlet or custom tags in the column body when using paging. When the column body is filled and full list is sorted, all the records need to be "prepared" by the table tag iterating on the whole list. If the column body is used the content will be evaluated for any row, also for the non displayed ones; using `property` , on the other hand, will cause the decorator only to be called for displayed rows.

A table decorator have the power to add extra properties to your objects: for example you can add a `getFullAddress()` method to your table decorator and then use `property="fullAddress"` in a column. A table decorator can also provide custom html code added at the beginning/end of rowss and table.

A column decorator is rather limited in its funcionality: it simply format an available value, and has actually no access to the page context or other properties. However, it is the simplest and most reusable block if you simply need to format dates, number or custom strings.

Leaving decorators out and filling the column body is the simplest solution if you don't have to worry too much about paging and performance and it is optimal in a small, non paged, table. During sorting, though, if the column body is used, the result will be always sorted as a String.

3.5 Links

Generating links

Smart linking of column data

If you have email addresses or web URLs in the data that you are displaying in columns of your table, then you can set the `autolink="true"` attribute in your display:column tag, and that will tell the display:table to automatically display those pieces of data as hyperlinks, you will not have to take any action to convert that data.

- Email addresses will be wrapped with a `<a href="mailto:xxx">xxx</a>` tag, where "xxx" is the email address that was detected.
- Web URLs will be wrapped with a `<a href="xxx">xxx</a>` tag, where "xxx" is the URL that was detected (it can be any valid URL type, http://, https://, ftp://, etc...)

If your column data has additional text, only the data that appears to be an email address or a URL will be linked (not the entire column).

Turning on autolink does carry a performance penalty, as each string has to be scanned for patterns and updated if it matches on an address or URL.

Dynamic links

There are two ways to create dynamic links that should appear in a column. The first method is a "struts-like" approach which works well if the link you want to create is based on just a single property of the object being displayed (like a primary key value). The second approach makes use of decorators as described on the previous example. A decorator should be used when the dynamic link being created relies on multiple pieces of information, relies on the index of the object in the list, relies on some other data around it, or you want to change the text that is linked (ie you want it to say "edit", instead of showing the primary key of the object). Below I show how to use both examples.

Struts-like approach

The column tag provides 5 struts-like attributes that can be set to create a dynamic linke (href, paramID, paramName, paramProperty, paramScope). See the display:column documentation, and the struts documentation for a complete description of their usage, but basically:

href

the base URL used to construct the dynamic link

paramId

the name of the parameter that gets added to the URL specified above

paramName

name of the bean that contains the data we want to tack on the the URL (typically null, indicating the current object in the List)

paramProperty

property to call on the object specified above to return the value that gets tacked onto the URL.

paramScope

specific scope where the databean lives, typically null

Of these params, you typically would not use paramName and paramScope. Leaving each as null indicates that you want to use the object corresponding to the current row being processed.

```
<display:table name="sessionScope.details">
  <display:column property="id" title="ID" href="details.jsp" paramId="id" />
  <display:column property="email" href="details.jsp" paramId="action"
paramName="testparam" paramScope="request" />
  <display:column property="status" href="details.jsp" paramId="id"
paramProperty="id" />
</display:table>
```

Using a decorator

The previous example page introduced the decorator to format dates, money, etc... It can also be used to create dynamic links on the fly so that you can either click on a particular column value and "drill down" for more information, or you can create a column of text labels which are hyperlinks that perform some action on the object in that row.

These dynamic links can be created based on some primary key of the object, or they can make use of the object List index.

Below is a table that has two columns that have hyperlinks created on the fly, the first makes use of the object's "ID" field to show additional details about that object, while the second makes use of the object's row index value to do basically the same thing.

Here you can see the details of the getLink1() and getLink2() methods in the sample TableDecorator

```
        public String getLink1()
        {
                ListObject lObject= (ListObject)getCurrentRowObject();
                int lIndex= getListIndex();
                return "&lt;a href=\"details.jsp?index=" + lIndex + "\">" +
lObject.getId() + "&lt;/a>";
        }
```

```
	public String getLink2()
	{
		ListObject lObject= (ListObject)getCurrentRowObject();
		int lId= lObject.getId();

		return "&lt;a href=\"details.jsp?id=" + lId
			+ "&amp;action=view\">View&lt;/a> | "
			+ "&lt;a href=\"details.jsp?id=" + lId
			+ "&amp;action=edit\">Edit&lt;/a> | "
			+ "&lt;a href=\"details.jsp?id=" + lId
			+ "&amp;action=delete\">Delete&lt;/a>";
	}
```

```
<display:table name="sessionScope.details" decorator="org.displaytag.sample.Wrapper"
>
  <display:column property="link1" title="ID" />
  <display:column property="email" />
  <display:column property="link2" title="Actions" />
</display:table>
```

3.6 Style

Style

You actually have a lot of flexibility in how the table is displayed, but of course you should probably stay close to the defaults in most cases. You adjust the look of the table via two methods:

1. pass through table and column attributes
2. style sheets

Html attributes

You can assign to the <display:table> tag any standard html **strict** attribute (es. style, class, cellspacing, cellpadding), and it will be included in the rendered table.

Likewise, you can assign to the <display:column> tag any standard html attribute and it will be included in any <td> tag of the rendered table. You can also specify a class to be used only for the column header (<th>) using the `headerClass` attribute.

Html **transitional** attributes are also supported in version 1.0 but they will be removed soon, so you are encoraged to avoid any html presentational attribute (such as border, background, bgcolor, width, height...): there are replaced by an appropriate use of css rules. See the tag reference page for the full list of supported/ deprecated attributes.

Style Sheets

The <display:table> tag produces well formed html tables with <thead> and <tbody> sections. Css classes are also automatically added to rows/cells when needed.

You can easily customize the generated table simply specifing appropriate css rules in your stylesheet, based on these classes/selectors.

All the automatically added css classes can be customized in displaytag.properties. You can also add a css class to any generated display:table if needed. This is the list of css classes added by default:

class	assigned to
odd	assigned to the tr tag of all odd numbered data rows
even	assigned to the tr tag of all even numbered data rows
sorted	assigned to the th tag of the sorted column
order1	assigned to the th tag of the sorted column if sort order is ascending

class	assigned to
order2	assigned to the th tag of the sorted column if sort order is descending
sortable	assigned to the th tag of a sortable column

3.7 Export

Exporting data

When you set the Table Tag's **export** attribute to "true", a footer will appear below the table which will allow you to export the data being shown in various formats.

Displaytag includes a few ready made *export views* which allow you to export data in CSV, excel, and XML format. A simple PDF export view is also available. The following table lists the predefined export options included in displaytag distribution.

Media	Export View Class	Description
CSV	org.displaytag.export.CsvView	Export to comma separated list
Excel	org.displaytag.export.ExcelView	Export to excel - ascii format, tab separated
XML	org.displaytag.export.XmlView	Simple xml output
PDF	org.displaytag.export.PdfView	Sample PDF export view. This is not enabled by default in the distribution but can be enabled by setting `export.pdf=true` in displaytag.properties and including the required IText dependency (see displaytag dependencies). Since you probably want to tweak the layout of your pdf output, this is probably more useful as a base reference for creating your own PDF export view.

Configuring export and export views

The `export.types` parameter contains the list of registered export views. For each export type you can configure other parameters: see the `export.` *`exportname`*`.*` parameters in configuration .

You can enable/disable a specific export type using the `export.` *`exportname`*`.enabled` parameter.

If you don't want some column to show during export (or you only want them to show during export) you can use the column `media` attribute (see tag reference for more details).

Adding a new Export view

1. Write your own exportView class. You need to implement the `org.displaytag.export.TextExportView` or `org.displaytag.export.BinaryExportView` interface. You can look at the sample binary PDF export view or to the base text export view used by displaytag .
2. Add a `displaytag.properties` file in your application classpath (if you already don't have one)

and add the name of your export media along with the default ones to the export type parameter:

```
export.types=csv excel xml [mymedia]
```

3. Always in `displaytag.properties` , add the following properties:

```
export.[mymedia]=true

export.[mymedia].class=fully.qualified.class.name

export.[mymedia].label=Click here to try my export

# include header parameter is forwarded to your export view
export.[mymedia].include_header=true

# if set, file is downloaded instead of opened in the browser window
export.[mymedia].filename=
```

4. Try it. You should see a new link with the text you added to `export.` *`mymedia`* `.label` . Clicking on it will invoke your Export view. You should see the results in your browser.

Text/Binary export views

Common displaytag export options (CSV, Xml and Excel) output a simple text-based format. Other file formats require binary content, like the sample PDF included with the distribution.

Exporting binary data from a JSP is a bit tricky, since JSPs are only designed to output characters: as a starting point keep in mind that **binary export is not assured to work on every application server** , at least without the help of an external filter (see export filter).

It may work without a filter if your application server allows JSPs to call `response.getOutputStream()` , but this method really shouldn't be used in JSPs. Using an the export filter, especially in buffered mode, could solve the problem, since the output stream is requested by the filter outside the JSP.

3.8 I18n

i18n - Internationalization

If you use displaytag in a multi-language application, you will probably need to translate html generated by displaytag as well. Displaytag supports i18n for html snippets used in the paging and export banner and for the title of column headers.

i18n for displaytag resources

Using the `displaytag.properties` file you can set all the messages handled by displaytag.

Configuring:

```
paging.banner.one_item_found=One item found
```

Will make displaytag output `One item found` when only a row is displayed.

In order to support other languages you can add any number of additional files named `displaytag_LANGUAGE.properties` . For example, you can add a `displaytag_IT.properties` for italian users with the following content:

```
paging.banner.one_item_found=Un solo elemento trovato
```

You don't need to copy all the properties in any internationalized file (some of them are also configuration properties which don't need to be translated at all). Configure displaytag properly in the main `displaytag.properties` file and then replicate only the strings you want to translate.

i18n for column title

There are a few different ways to define the content for a column header in the `column` tag:

1. specifying a `title` attribute: the content of the title attribute will be used as is for the column header
2. specifying a `titlekKey` attribute: the content of the titlekey will be used to lookup a resource in a resource bundle and the value will be used in the column header. If the specified key can't be found

Advanced

Displaytag will probably be used in an application where content is already internationalized using a specific framework, which should provide a way to resolve the current locale and to lookup properties in a resource bundle.

Displaytag provides a way to plug-in different adapters to use the same i18n support you are using in your application.

Locale resolution

By default displaytag will use the locale specified in the request (i.e. the locale set in the user browser). This can be fine for a basic use, but you could need a way to override this selection and to force a different locale.

Here comes the problem: if you already use Jstl, Struts or other frameworks you will know that there is no standard way to specify the locale to use: each framework works in a different way.

Displaytag provides an interface `LocaleResolver` with a few ready to use implementations which match the behaviour of common frameworks. The locale resolver is specified in the `displaytag.properties` file using the `locale.resolver` key.

If nothing is specified the locale from the request is used, as specified above. However, you are free to configure here any custom implementation of the `org.displaytag.localization.LocaleResolver` interface with a simple method `resolveLocale(HttpServletRequest)`

Displaytag provides by default these ready to use implementations:

class name	behaviour
org.displaytag.localization.I18nJstlAdapter	Mimic JSTL, looking for a locale specified in session with the `Config.FMT_LOCALE` key.
org.displaytag.localization.I18nStrutsAdapter	Struts adapter, will look for the locale specified by `Globals.LOCALE_KEY`
org.displaytag.localization.I18nWebworkAdapter	Webwork2 adapter, will look for the locale specified by the fist LocaleProvider action in the stack
org.displaytag.localization.I18nSpringAdapter	Spring adapter, will use `RequestContextUtils.getLocale()` for locale resolution (which will in turn delegate to the Spring locale resolver)

Resources lookup

Other than resolving the currently used locale, your framework will probably provide a standard way to store i18n resources. Just like for the locale resolution, displaytag will allow you to plug in different implementations.

Displaytag provides an interface `I18nResourceProvider` with a few ready to use implementations which match the behaviour of common frameworks. This is configured in `displaytag.properties`

using the `locale.provider` key. By default the JSTL implementation is used.

The ready to use locale resolvers are (yes, these are the same classes used for locale resolution, since they implement both interfaces):

class name	behaviour
org.displaytag.localization.I18nJstlAdapter	JSTL implementation, works in the same way as fmt:message. Note that this depends from the jakarta jstl implementation: it will also work with Resin jstl support, but you will still need standard.jar in the classpath.
org.displaytag.localization.I18nStrutsAdapter	Struts adapter, will use `TagUtils.message()`
org.displaytag.localization.I18nWebworkAdapter	Webwork2 adapter, will look for the first TextProvider action in the stack and will obtain a message for the given key.
org.displaytag.localization.I18nSpringAdapter	Spring adapter, will look for the configured `messageSource` and use it to obtain a value for the given key.

4.1 FAQ

Frequently Asked Questions

General

1. When I was trying to copy and paste the samples from the website I would get compile time errors looking for TestList, ReportList, etc. This would happen even if I included the display tag binary distribution jar file.
2. One thing that I really wanted to do was create tables that contained things other than text fields. Drop Downs, CheckBoxes and Input Fields were stuff that I was looking for. On the examples page there wasn't an example that showed how to do this.
3. What is the c tag library? In the examples there is a tag library called the c tag library which was used in a lot of places (c:if, c:out, etc.).
4. Can I use a java variable as the value of the id attribute?

Displaytag and Struts

1. How do I use this tag with Struts? The links that it creates for sorts and exports all point back to the JSP page, not my Action!

Rendering

1. How can I display static headers in a table, so that headers remain visible while user scrolls the table body?

App Servers specific problems

1. After deployed the displaytag war in JRun 4 I get a `java.lang.NoClassDefFoundError: org/apache/log4j/Layout`
2. The <example> elements in the `displaytag-el.tld` and `displaytag-12.tld` are invalid according to JRun4. When adding displaytag.jar to a JRun4 web application I get the following stacktrace: `jrun.jsp.compiler.JRunTagLibraryInfo$InvalidTLDElementException: The tag example on line 244 is not a valid TLD element at jrun.jsp.compiler.JRunTagLibraryInfo$TLDParser.startElement(JRunTagLibraryInfo.java:`

General

General

When I was trying to copy and paste the samples from the website I would get compile time errors looking for TestList, ReportList, etc. This would happen even if I included the display tag binary distribution jar file.

Within the display tag binary distribution the "org.displaytag.sample" class files are not included. So if you are trying to use TestList, ReportList which are part of the org.displaytag.sample package

you'll get a compile error telling you they can't find the classes. The solution I came up with was to download the source file distribution and copy the org.display.sample source files into your build. If you do this you probably need to copy over files from the `org.displaytag.decorator` and `org.displaytag.exception` packages since there are dependencies between them all.

One thing that I really wanted to do was create tables that contained things other than text fields. Drop Downs, CheckBoxes and Input Fields were stuff that I was looking for. On the examples page there wasn't an example that showed how to do this.

There is a really good example of how to use displaytag to create an editable row of data located at http://demo.raibledesigns.com/appfuse/demos/users-edit.jsp . It contains the source code for how to do it.

What is the c tag library? In the examples there is a tag library called the c tag library which was used in a lot of places (c:if, c:out, etc.).

The c tag library is actually the JavaServer Pages Standard Tag Library (JSTL). To import it you use the line `<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>` . You also have to download the library from sun and include the jstl.jar, standard.jar within your war, in the appropriate places.

Can I use a java variable as the value of the id attribute?

Yes, from version 1.0rc2 displaytag will allow you to use a java variable as the value of the id attribute. But you have to be aware that, while usually displaytag declares a variable named with the value of the id attribute, this is not possible when using a runtime expression. You can however fetch the value of the row object from the pagecontext, as showed in the following example.

```
<% String myId = "row">
<display:table id="<%=myId%>" ...>
  <display:column><%=row%></display:column> <-- will not work
  <display:column><%pageContext.getAttribute("row")></display:column> <-- works
```

In the EL version of the tag library you are also allowed to do:

```
<c:set var="myId" value="row" />
<display:table id="${myId}" ...>
  <display:column><c:out value="${row}" /></display:column>
```

Note that `pageContext.getAttribute()` is the preferred way for accessing the object in the current row and the declared variable will probably be removed in future (it is already not used in the EL version of the tag library).

Some containers don't allow the id attribute to be a runtime expression at all (this has been reported in earlier versions of Tomcat 5 and ATG Dynamo 5), so an attribute with the name `uid` has been added. Simply use `uid` instead of `uid` in the table tag.

Displaytag and Struts

Displaytag and Struts

How do I use this tag with Struts? The links that it creates for sorts and exports all point back to the JSP page, not my Action!

Use the requestURI attribute of the column tag. When the requestURI attribute is present, links for sorting, exports, and paging are formed by adding any tag generated parameters to the value of requestURI attribute. So if your page is served under `/ManageLatin.do` , you should have `requestURI="/ManageLatin.do"` on your display:table.

Using the requestURI attribute with an empty "" attribute is another strategy. Sometimes you don't know what the uri is because the table might be part of an included tile. The resulting URL will be a link that is based upon the original requestURI + the appended display tag parameters.

Rendering

Rendering

How can I display static headers in a table, so that headers remain visible while user scrolls the table body?

You can do that simply using css, adding an height and the `overflow: scroll` property to the `tbody` element. Unfortunately this will work perfectly in Netscape/Mozilla, but not in Internet Explorer.

A table with a scrollable body can be made in Internet Explorer using javascript or a more complex css. Here you can find a good css example.

App Servers specific problems

App Servers specific problems

After deployed the displaytag war in JRun 4 I get a `java.lang.NoClassDefFoundError: org/apache/log4j/Layout`

This is due to a known bug in JRun.

To make the sample webapp working you will need to:

- remove `commons-logging-*.jar` from `displaytag.war/WEB-INF/lib`
- move `log4j-1.2.8` to `$JRUN/servers/lib/` (create this directory if it doesn't exist)

You can find more info related to this jrun bug on google: searching jrun "org/apache/log4j/Layout"

The <example> elements in the `displaytag-el.tld` and `displaytag-12.tld` are invalid according to JRun4. When adding displaytag.jar to a JRun4 web application I get the following stacktrace:

```
jrun.jsp.compiler.JRunTagLibraryInfo$InvalidTLDElementException: The
tag example on line 244 is not a valid TLD element at
jrun.jsp.compiler.JRunTagLibraryInfo$TLDParser.startElement(JRunTagLibraryInfo.java:6
```

Again, this appears to be a JRun fault.

The <example> elements are absolutely valid according to the sun dtd `http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd` but JRun doesn't accept them. A bug report has already been submitted to Macromedia hoping they will fix this.

In the meanwhile, you have two available solutions to make displaytag work pacefully with JRun:

- If you don't need to use the EL version, simply delete the `displaytag-el.tld` and `displaytag-12.tld` files from displaytag.jar and use the 1.1 version. This one works without problems.
- If you need the EL version, extract `displaytag-el.tld` from the jar and remove all the <example> tags. Replace the original tld in the jar with this modified one.

We are sorry for this extra step needed with JRun, but at the moment we decided to not to remove the example tags in the main tld since they are needed for the generated documentation.

4.2 Reporting bugs

Issue Tracking

Go to Displaytag bug and feature request tracker (JIRA) to see open bugs or to submit new requests.

Please note that the issue tracker on sourceforge is not used anymore and any remaining item will be moved to the new tracker.

Bug reporting

Before submitting a bug to the JIRA above, please be sure you have done your homework. This includes:

1. Reading the FAQ to understand the correct behavior
2. Searching the JIRA for a comparable issue; you may not be the first person to find this bug
3. Searching the user list archives if you are not sure the behavior you are experiencing is a bug

Post help requests to the user mailing list.

Please do not post bug reports or patches directly to the user or development mailing lists ! Always submit them to the JIRA before. This will help us track and review them. After submitting a bug, you are free to discuss it in the developer mailing list (without attaching code or patches to mails).

Feature requests and patches for new features

As always, use the JIRA to submit feature requests (in the RFE category).

If you have modified your local version of the display tag library by adding a new feature and would like to see in the main distribution, open a new RFE (Request For Enhancement) in the JIRA and attach the patch to it (**don't send it to the mailing list**).

If you want to see your patch quickly applied by committers, you should be able to provide the following items:

1. A CVS diff against the latest CVS version. **No, a zip file with all modified sources is not OK**.
2. One or more junit tests related to the new feature. Also, be sure to run all of the existing testcases to verify that you are not breaking existing code. Look in the src/test directory to understand how to write simple JSP test cases.
3. Update the documentation (xdoc files and/or the sample application).

If all of these requirements are met, your patch will likely be accepted soon. If you only provide a zip with the modified source files; keep in mind that before your addition can be committed to CVS, a developer must complete all of the other tasks *for you*. This could take a long time.

4.3 Mailing Lists

Mailing Lists

These are the mailing lists that have been established for this project. For each list, there is a subscribe, unsubscribe, and an archive link.

List Name	Subscribe	Unsubscribe	Archive
Display tag library User Mailing list	Subscribe	Unsubscribe	Archive
Display tag library Developer Mailing list	Subscribe	Unsubscribe	Archive
Display tag library Cvs Mailing list	Subscribe	Unsubscribe	Archive

5.1 Source Repository

Web Access

http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/displaytag/displaytag/

Anonymous Access with Maven

This project's CVS repository can be checked out through anonymous (pserver) CVS with the following instruction on a single line.

```
          maven scm:checkout-project
            -Dmaven.scm.method=cvs
            -Dmaven.scm.cvs.module=displaytag
-Dmaven.scm.cvs.root=:pserver:anonymous@cvs.sourceforge.net:/cvsroot/displaytag
            -Dmaven.scm.checkout.dir=.
```

Anonymous CVS Access

This project's CVS repository can be checked out through anonymous (pserver) CVS with the following instruction set. When prompted for a password for anonymous, simply press the Enter key.

cvs -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/displaytag login
cvs -z3 -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/displaytag co displaytag

Updates from within the module's directory do not need the -d parameter.

Developer Access with Maven

Only project developers can access the CVS tree via this method. Substitute **username** with the proper value.

```
          maven scm:checkout-project
            -Dmaven.scm.method=cvs
            -Dmaven.scm.cvs.module=displaytag
-Dmaven.scm.cvs.root=:ext:username@cvs.sourceforge.net:/cvsroot/displaytag
            -Dmaven.scm.checkout.dir=.
            -Dmaven.scm.cvs.rsh=ssh
```

Remember to replace 'username' with your actual username on cvs.sourceforge.net.
Also change `ssh` in `maven.scm.cvs.rsh=ssh` to the name of your ssh executable.

Developer CVS Access via SSH

Only project developers can access the CVS tree via this method. SSH1 must be installed on your client machine. Substitute **username** with the proper value. Enter your site password when prompted.

```
export CVS_RSH=ssh
cvs -z3 -d :ext:username@cvs.sourceforge.net:/cvsroot/displaytag co displaytag
```

Remember to replace 'username' with your actual username on cvs.sourceforge.net.

CVS Access behind a firewall

For those developers who are stuck behind a corporate firewall, CVSGrab can use the viewcvs web interface to checkout the source code.

5.2 Building from sources

Building from Source

Getting the source

Refer to the download section for how to download a source release of a snapshot from CVS.

Installing Maven

Displaytag uses maven for build and site generation. First of all, you need to download and install maven (you will need at least Maven 1.0) from maven.apache.org . Follow the instructions on the Maven site on how to do this.

Run it!

It couldn't be more simple: go to the folder containing the (unzipped) source, type **maven** and watch the show. For this you need a working internet connection since maven will first try to download all the needed libraries (see dependencies) from a remote repository.

Running maven without arguments will execute the default goal, which will build the library, the sample webapp, and the documentation site (this one). You can run **maven -g** to see the available goals. Running **maven jar** will simply create the library jar without the sample site and documentation.

5.3 Directory Organization

Overview

The display tag library uses Maven for build and documentation and tries to conform to general Maven project layout. If you are used to working on a maven-driven project you should feel comfortable with it. Here is a description of the file and folder organization. The **strong** names refer to project specific setting/files.

Common Directory Layout

Directory or file name	Content	Comment
LICENSE.txt	The license for the project.	This file contains the license that applies to the project.
project.xml	Maven project descriptor	This file contains the basic project configuration for maven (project name, developers, urls, dependencies, etc).
project.properties	A file defining project specific properties.	This file can be used to override maven default properties for the core and properties for the various plugins. It can also be used to define any maven properties used by a project. The properties defined in this file should be applicable to most users of your project, as opposed to custom properties for a specific build/user which should be defined in `${project.home}/build.properties` .
maven.xml	Maven configuration for defining build goals	This file contains the default maven goals for the project, plus added pre-post operations to be performed.
src/	Source code	This is the main directory for all the source code, subdivided in subdirectory to separate different "kind" of code.
src/java	Java source code	This directory contains all the java source code both for the library and the example webapp.
src/webapp	Example webapp source	This directory contains all the code (JSP, images, web.xml configuration) for the example webapp.
src/tld	tld	This directory contains the tag library definition file (tld), which is added in the library jar and in the examples war during the build.

Directory or file name	Content	Comment
xdocs/	Documentation files in XML format.	Maven projects use Jelly/JSL to transform documentation files in XML into HTML. Project documentation should be placed in this directory. Maven converts all XML files in this directory using JSL. Non-XML files (and directories) are copied without modification to permit the inclusion of "other" types of documentation (including images). The generated HTML files automatically inherit the Maven look-and-feel by default.
xdocs/navigation.xml	Navigation links for site.	Maven projects use Jelly stylesheets to transform documentation files in XML into HTML (XHTML for the most part). This file includes the navigation links that are added to each xdoc transformed in the `xdocs` directory.
checkstyle.xml	Checkstyle configuration file	Configuration file containing checkstyle settings used during generation of reports.

Maven-Generated Layout

Directory name	Content	Comment
target/	Contains compiled classes and JARs.	The contents of the target/ directory should be enough to use the project. This directory contains the final JAR and WAR that are generated.
target/classes	Contains compiled classes.	The target/classes directory contains all compiled classes. This directory is used when packaging the final JAR for a project.
target/generated-docs	Contains Maven generated xdocs.	The target/generated-docs directory contains all of the Maven-generated xdocs. All content generated by Maven is first converted to xdoc format, so the same stylesheet used to transform the rest of the site can be used on generated content. The contents of this directory are transformed and stored in the docs/ directory.
target/docs	Documentation files intended for the website publication.	The docs/ directory contains only generated documentation that is intended to be published as the project's website. This directory includes the Velocity/DVSL generated HTML files, JavaDocs, cross-referenced sources, and various generated reports. Generally, all documentation is stored in the xdocs/ directory and then "transformed" into this directory. The specific documents that Maven generates are described below.
target/docs/index.html	Starting point for browsing the documentation.	Browsing the documentation locally should yield the same results as browsing the documentation on the project's home page.
target/docs/apidocs	API documentation.	Maven automatically generates JavaDocs for projects using the JavaDoc utility. Placing the API documentation under docs/apidocs/ makes it slightly easier for other documentation files under docs/ to reference API documentation and vice versa.

Directory name	Content	Comment
target/docs/xref	Cross-referenced source code.	Maven automatically generates cross-referenced source code that enables easy browsing of an entire source tree. Placing the cross-referenced sources under docs/xref/ makes it slightly easier for other documentation files under docs/ to reference API documentation and vice versa.
target/docs/mail-lists.html	Mailing list documentation.	Maven automatically generates a list of mailing lists based on the information provided in the project descriptor.
target/docs/team-list.html	The list of project team members.	Maven automatically generates a list of project team members based on the information provided in the project descriptor.
target/docs/dependencies.html	The list of dependencies.	Maven automatically generates a list of dependencies based on the information provided in the project descriptor.
target/docs/changelog.html	The CVS change log.	Maven automatically generates a change log from CVS log messages. This log is currently limited to the past 30 days (but will be configurable in the future).
target/docs/file-activity-report.html	The File Activity Report.	Maven automatically generates a log from your SCM listing file changes in the last 30 days.
target/docs/developer-activity-report.html	The Developer Activity Report.	Maven automatically generates a log from your SCM listing changes per developer in the last 30 days.
target/docs/jdepend-report.html	Metric report.	Maven automatically generates a report on various source code metrics. This report can provide further insight into a project.
target/docs/checkstyle-report.html	Checkstyle report.	Maven automatically generates a report on the results of Checkstyle. This report provides assurance that the coding conventions for your project are being followed.